A biased introduction to neural networks and old school machine learning

Michael Biehl

The Shallow and the Deep

University of Groningen Press

The Shallow and the Deep

The Shallow and the Deep

A biased introduction to neural networks and old school machine learning

Michael Biehl

University of Groningen Press

Published by University of Groningen Press
Broerstraat 4
9712 CP Groningen
The Netherlands

Comments, corrections and suggestions are welcome, contact: m.biehl@rug.nl

Please cite as: Biehl, M. (2023). *The Shallow and the Deep: A biased introduction to neural networks and old school machine learning.* University of Groningen Press.

This book has been published open access thanks to the financial support of the Open Access Textbook Fund of the University of Groningen.

Cover design: Bas Ekkers
Coverphoto: Michael Biehl
Production: LINE UP boek en media bv

ISBN (print) 9789403430287
ISBN (ePDF) 9789403430270
DOI https://doi.org/ 10.21827/648c59c1a467e

Preface

Stop calling everything AI.

— Michael I. Jordan, in [Pre21]

The subtitle of these lecture notes is "A **biased** introduction to neural networks and **old school** machine learning" for good reasons. Although the aim was to give an accessible introduction to the field, it has been clear from the beginning that it would not end up as a comprehensive, complete overview. The focus is on *classical* machine learning, many recent developments cannot be covered.

Personally, I first got in touch with neural networks in my early life as a physicist. At the time, it was sufficient to read a handful of papers and perhaps a little later *the good book* [HKP91] to be up-to-date and able to contribute a piece to the big puzzle. Am I exaggerating and somewhat nostalgic? Probably. But the situation has definitely changed a lot. Nowadays, an overwhelming flood of publications makes it difficult to filter out the relevant information and keep up with the developments.

The selection of topics in these notes has been determined to a large extent by my own research interests and early experiences. This is definitely true for the initial focus on the simple perceptron, the *hydrogen atom of neural network research* as Manfred Opper put it [Opp90]. Moreover, the bulk of this text deals with shallow systems for supervised learning, in particular classification, which reflects my main interest in the field.

The notes may be perceived as old school, certainly by some *dedicated followers of fashion* [Dav66]. Admittedly, the text does not address the most recent developments in e.g. Deep Learning and its applications. However, in my humble opinion it is invaluable to have a solid background knowledge of the basics before exploring the world of machine learning with an ambition that goes beyond the application of *some* software package to *some* data set.

Therefore, the emphasis is on basic concepts and theoretical background, with specific aspects selected from a personal and clearly biased viewpoint. In a sense, the goal is to de-mystify machine learning and neural networks without

losing the appreciation for their fascinating power and versatility. Very often, this involves a look into the history and *pre-history* of neural networks, where the foundations for most of the recent developments were laid.

I have aimed at pointing the interested reader to many resources for further exploration of the area. Therefore, the list of references in the bibliography, although by no means complete, is slightly more extensive than initially envisioned.

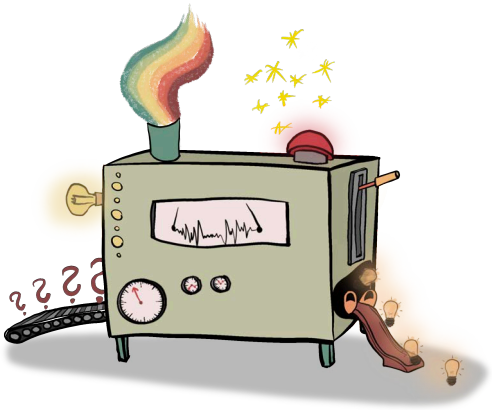
The starting point for these notes was the desire to provide more comprehensive material than the presentation slides in the MSc level course *Neural Networks* (renamed *Neural Networks and Computational Intelligence* later) which I have been giving at the University of Groningen. A thorough archeological investigation of the text and figures would also reveal traces of the courses *Theorie Neuronaler Netzwerke* and *Unüberwachtes Lernen*, that I taught way back when in the Physics program at the University of Würzburg.

My writing activity was greatly boosted on the occasion of the wonderful *30th Canary Islands Winter School* in 2018, devoted to *Big Data analysis in Astronomy,* where I had the honor to give a series of lectures on supervised learning, see [MSK19, Bie19] for course materials and video-recorded lectures.

Last not least I would like to acknowledge constructive feedback from several "generations" of students who followed the course and from many colleagues and collaborators. In particular, I thank Elisa Oostwal and Janis Norden for a critical reading of the manuscript and many suggestions for improvements.

Groningen, June 2023

Michael Biehl



The mysterious machine learning machine

Contents

---

1. Abandon the idea that you are ever going to finish.

— John Steinbeck, *Six Writing Tips*

Chapter 1

From neurons to networks

Reality is overrated anyway.

— Unknown

To understand and explain the brain's fascinating capabilities[1] remains one of the greatest scientific challenges ever. This is particularly true for its plasticity, i.e. the ability to learn from experience, to adapt to and to survive in ever-changing environments.

Ultimately, the performance of the brain must rely on its *hardware* (or *wetware* [LB89]) and emerges from the cooperative behavior of its many, relatively simple, yet highly interconnected building blocks: the neurons. The human cortex, for instance, comprises an estimated number of 10^{12} neurons and each individual cell can be connected to thousands of others.

In this introduction to *Neural Networks and Computational Intelligence* we will study artificial neural networks and related systems, designed for the purpose of adaptive information processing. The degree to which these systems relate to their biological counterparts is, generally speaking, quite limited. However, their development was greatly inspired by key aspects of biological neurons and networks. Therefore, it is useful to be aware of the conceptual connections between artificial and biological systems, at least on a basic level.

Quite often, technical solutions are inspired by natural systems without copying all their properties in detail. Due to biological constraints, nature (i.e. evolution) might have found highly complex solutions to certain problems that could be dealt with in a simpler fashion in a technical realization. A somewhat overused example in this context is the construction of efficient aircraft, which by no means requires the use of moving wings in order to imitate bird flight faithfully.

Of course, it is unclear *a priori* which of the details are essential and which ones can be left out in artificial systems. Obviously, this also depends on the

[1] (including the capability of being fascinated)

specific task and context. Consequently, the interaction between the neurosciences and machine learning research continues to play an important role for the further development of both.

In this introductory text we will consider learning systems, which draw on only the most basic mechanisms. Therefore, this chapter is only meant as a very brief overview, which should allow to relate some of the concepts in artificial neural computation to their biological background. The reader should be aware that the presentation is certainly over-simplifying and probably not quite up-to-date in all aspects.

Recommended textbooks and other sources

In most of this introductory chapter, detailed citations concerning specific topics will not be provided. Instead, the following list points the reader to selected textbooks, reviews or lecture notes. They range from brief and superficial to very comprehensive and detailed reviews of the biological background. The same is true for the discussion of the different conceptual levels on which biological systems can be modelled. References point to the full citation information in the bibliography. Note that the selection is certainly incomplete and biased by personal preferences.

- K. Guerney's *(Neural Networks)* gives a very basic overview and provides a glossary of biological or biologically inspired terms [Gue97].
- The first sections of *Neural Networks and Learning Machines* by S. Haykin cover the relevant topics in slightly greater depth [Hay09].
- The classical textbook *Neural Networks: An Introduction to the Theory of Neural Computation* by J.A. Hertz, A. Krogh and R.G. Palmer discusses the inspiration from biological neurons and networks in the first chapters. It also provides a thorough analysis of the Hopfield model from a statistical physics perspective [HKP91].
- H. Horner and R. Kühn give a brief general introduction in *Neural Networks* [HK98], including a basic discussion of the biological background.
- Models of biological neurons, their bio-chemistry and bio-physics are in the focus of C. Koch's comprehensive monograph on the *Biophysics of computation* [Koc98]. It discusses the different modelling approaches and relates them to experimental data obtained from real world neurons.
- T. Kohonen has introduced important prototype-based learning schemes. An entire chapter of his seminal work *Self-Organizing Maps* is devoted to the *Justification of Neural Modeling* [Koh97].
- H. Ritter, T. Martinetz and K. Schulten give an overview and also discuss some aspects of the organization of the brain in terms of *maps* in their monograph *Neural Computation and Self-Organizing Maps* [RMS92].
- M. van Rossum's lecture notes on *Neural Computation* provide an overview of biological information processing and models of neural activity, synaptic interaction and plasticity. Moreover, modelling approaches are discussed in some detail [Ros16].

1.1 Spiking neurons and synaptic interactions

The physiology and functionality of the biological systems is highly complex, already on the single neuron level. Sophisticated modelling frameworks have been developed that take into account the relevant electro-chemical processes in great detail in order to represent the biology as faithfully as possible. This includes the famous Hodgkin-Huxley model and variants thereof.

They describe the state of cell compartments in terms of an electrostatic potential, which is due to varying ion concentrations on both sides of the cell membrane. A number of ion channels and *pumps* control the concentrations and, thus, govern the membrane potential. The original Hodgkin-Huxley model describes its temporal evolution in terms of four coupled ordinary differential equations, the parameters of which can be fitted to experimental data measured in real world neurons.

Whenever the membrane potential reaches a threshold value, for instance triggered by the injection of an external current, a short, localized electrical pulse is generated. The term action potential or the more sloppy *spike* will be used synonymously. The neuron is said to *fire* when a spike is generated.

The action potential discharges the membrane locally and propagates along the membrane. As illustrated in Figure 1.1 (left panel), a strongly elongated extension is attached to the soma, the so-called axon. From a purely technical point of view, it serves as a cable along which action potentials can travel.

Of course, the actual electro-chemical processes are significantly different from the flow of electrons in a conventional copper cable, for instance. In fact, action potentials jump between short gaps in the myelin sheath, an insulating layer around the axon. By means of *saltatory conduction*, action potentials spread along the axonic branches of the firing neuron and eventually reach the points where the branches connect to the dendrites of other neurons. Such a connection, termed synapse, is shown schematically in Fig. 1.1 (right panel). Upon arrival of a spike, so-called neuro-transmitters are released into the synaptic cleft, i.e. the gap between pre-synaptic axon branch and the post-synaptic dendrite. The transmitters are received on the post-synaptic side by substance specific receptors. Thus, in the synapse, the action potential is not transferred directly through a physical contact point, but chemically.[2] The effect that an arriving spike has on the post-synaptic neuron depends on the detailed properties of the synapse:

- If the synapse is of the *excitatory* type, the post-synaptic membrane potential increases upon arrival of the pre-synaptic spike,
- When a spike arrives at an *inhibitory* synapse, the post-synaptic membrane potential decreases.

Both excitatory and inhibitory synapses can have varying strengths, as reflected

[2] Note that also so-called *gap junctions* exist which can function as bi-directional *electrical synapses*, see e.g. [CL04] for further information and references.

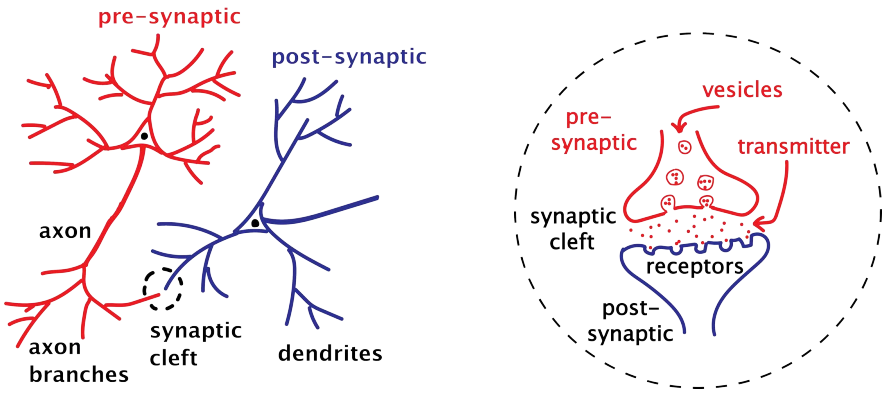


Figure 1.1: Schematic illustration of neurons (pyramidal cells) and their connections. **Left:** Pre-synaptic and post-synaptic neurons with soma, dendritic tree, axon, and axonic branches. **Right:** The synaptic cleft with vesicles releasing neuro-transmitters and corresponding receptors on the post-synaptic side. Redrawn after [Kat66].

in the magnitude of the change that a spike imposes on the post-synaptic membrane potential.

Consequently, the membrane potential of a particular cell will vary over time, depending on the actual activities of the neurons it receives spikes from through excitatory and inhibitory synapses. When the threshold for spike generation is reached, the neuron fires itself and, thus, influences the potential and activity of all its post-synaptic neighbors. All in all, a set of interconnected neurons forms a complex dynamical system of threshold units which influence each other's activity through generation and synaptic transmission of action potentials.

The origin of a very successful approach to the modelling of neuronal activity dates back to Louis Lapicque in 1907. In the framework of the so-called Integrate-and-Fire (IaF) model, electro-chemical details accounted for in the Hodgkin-Huxley type of models are omitted (and were probably not known at the time). The membrane is simply represented by its conductance and ohmic resistance. All charge transport phenomena are combined in one effective electric current, which summarizes the individual contributions of changing ion concentrations as well as leak currents through the membrane. Similarly, the precise form of spikes and details of their generation and transport are ignored. Instead, the firing is modelled as an *all-or-nothing* threshold process, which results in an instantaneous discharge. A spike is represented by a structureless Dirac delta function which defines the time point of the event. Despite its simplicity compared to more realistic electro-chemical models, the IaF model can be fitted to physiological data and yields a fairly realistic description of neuronal activity.

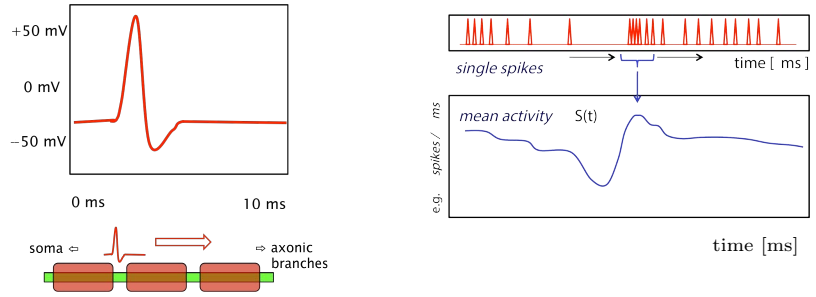


Figure 1.2: Left (upper): Schematic illustration of an action potential, i.e. a short pulse on mV- and ms-scale. **Left (lower):** Spikes travel along the axon through saltatory conduction via gaps in the insulating myelin sheath. **Right:** Schematic illustration of how mean firing rates are derived from a temporal spike pattern.

1.2 Firing rate models

In another step of abstraction, the description of neural activity is simplified by taking into account only the mean *firing rate*, e.g. obtained as the average number of spikes per unit time; the concept is illustrated in Fig. 1.2 (right panel).

The implicit assumption is that most of the information in neural processing is contained in the mean activity and frequency of spikes of the neurons. Hence, the precise timing of individual action potentials is completely disregarded. While the role of individual spike timing appears to be the topic of ongoing debate in the neurosciences[3], the simplification clearly facilitates efficient simulations of very large networks of neurons and can be seen as the basis of virtually all artificial neural networks and learning systems considered in this text.

1.2.1 Neural activity and synaptic interaction

The firing rate picture allows for a simple mathematical description of neural activity and synaptic interaction. Consider the mean activity S_i of neuron i, which receives input from a set $\mathbb{J}$ of neurons with $j \neq i$. Taking into account the fact that the firing rate of a biological neuron cannot exceed a certain maximum due to physiological and bio-chemical constraints, we can limit S_i to a range of values $0 \leq S_i$ where the upper limit 1 is given in arbitrary units. The *resting state* $S_i = 0$ obviously corresponds to the absence of any spike generation.

The activity of neuron i is given as a (non-linear) response of incoming spikes, which are - however - also represented only by the mean activities S_j: in

$$S_i \;=\; h(x_i) \quad \text{with} \quad x_i = \sum_{j \in \mathbb{J}} w_{ij}\, S_j. \tag{1.1}$$

[3]See, e.g., http://romainbrette.fr/category/blog/rate-vs-timing/ for further references.

Here, the quantities $w_{ij} \in \mathbb{R}$ represent the strength of the synapse connecting one neuron $j \in \mathbb{J}$ with neuron i. Positive $w_{ij} > 0$ increase the so-called local potential x_i if neuron j is active ($S_j > 0$), while $w_{ij} < 0$ contribute negative terms to the weighted sum. Note that real world chemical synapses are strictly uni-directional: even if connections w_{ij} and w_{ji} exist for a given pair of neurons, they would be physiologically separate, independent entities.

1.2.2 Sigmoidal activation functions

A variety of different activation functions $h(x)$ have been employed in artificial neural networks. A few specific types of functions will be introduced in a later chapter. Here we restrict the discussion to the by now classical *sigmoidal activation* which arguably captures important characteristics of biological systems.

It is plausible to assume the following mathematical properties of the activation function $h(x)$ of a given neuron (subscript i omitted) with local potential x as in Eq. (1.1):

$$\begin{aligned} \lim_{x\to-\infty} h(x) &= 0 \quad && \text{(resting state, absence of spike generation)} \\ h'(x) &\geq 0 \quad && \text{(monotonic increase of the excitation)} \\ \lim_{x\to+\infty} h(x) &= 1 \quad && \text{(maximum possible firing rate),} \end{aligned} \tag{1.2}$$

which takes into account the limitations of individual neural activity discussed in the previous section.

Various activation or transfer functions have been suggested and considered in the literature. In the context of feed-forward neural networks, we will discuss several options in Sec. 5.4. A very important class of plausible activations is given by so-called sigmoidal functions, one prominent[4] example being

$$h(x) = \frac{1}{2}\Big(1 + \tanh\big[\gamma(x-\theta)\big]\Big) \tag{1.3}$$

which clearly satisfies the conditions given above. The two important parameters are the threshold θ, which localizes the steepest increase of activity, and the gain parameter γ, which quantifies the slope. It is important to note that θ does not directly correspond to the previously discussed threshold of the *all-or-nothing* generation of individual spikes: it marks the characteristic value of h at which the activation function is centered.

[4]Its popularity is partly due to the fact that the relation $\tanh' = 1 - \tanh^2$ facilitates a very efficient computation of the derivative, see also Chapter 5.

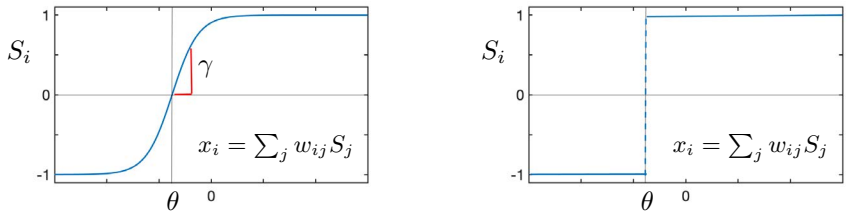


Figure 1.3: Schematic illustration of symmetrized activation functions. **Left:** A sigmoidal transfer function with gain γ and threshold θ in the symmetrized representation, cf. Eq. (1.6). **Right:** The binary McCulloch Pitts activation as obtained in the limit $\gamma \to \infty$.

Symmetrized representation of activity

We will frequently consider a symmetrized description of neural activity in terms of modified activation functions:

$$\begin{aligned} \lim_{x\to-\infty} g(x) &= -1 && \text{(resting state, absence of spike generation)} \\ g'(x) &\geq 0 && \text{(monotonic increase of the excitation)} \\ \lim_{x\to+\infty} g(x) &= 1 && \text{(maximum possible firing rate).} \end{aligned} \tag{1.4}$$

An example activation analogous to Eq. (1.3) is

$$g(x) \;=\; \tanh\big[\gamma(x-\theta)\big]. \tag{1.5}$$

At first sight, this appears to be just an alternative assignment of a value $S = -1$ to the resting state.

Note that in the original description with $0 < S_j < 1$, a quiescent neuron does not influence its postsynaptic neurons explicitly. However, keeping the form of the activation as

$$S_i \;=\; g(x_i) \quad \text{with} \quad x_i = \sum_{j\in\mathbb{J}} w_{ij}\, S_j \tag{1.6}$$

implies that the absence of activity ($S_j = -1$) in neuron j can now increase the firing rate of neuron i if connected through an inhibitory synapse $w_{ij} < 0$. This and other mathematical subtleties are clearly biologically implausible which is due to the somewhat artificial introduction of – in a sense – *negative* and *positive activities* which are treated in a symmetrized fashion.

However, as we do not aim at describing biological reality, the above discussed symmetrization can be justified. In fact, it simplifies the mathematical and computational treatment, and has contributed to, for instance, the fruitful popularization of neural networks in the statistical physics community in the 1980s and 1990s.

McCulloch Pitts neurons

Quite frequently, an even more drastic modification is considered: for infinite gain $\gamma \to \infty$ the sigmoidal activation becomes a step function, see Fig. 1.3 (right panel) for an illustration. Eq. (1.5) for instance yields in this limit

$$g(x) = \text{sign}(x-\theta) = \begin{cases} +1 & \text{if } x \geq \theta \\ -1 & \text{if } x < \theta. \end{cases} \tag{1.7}$$

In this symmetrized version of a binary activation function, only two possible states are considered: either the model neuron is totally quiescent ($S = -1$) or it fires at maximum frequency, which is represented by $S = +1$.

The extreme abstraction to binary activation states without the flexibility of a graded response was first discussed by McCulloch and Pitts in 1943, who originally denoted the quiescent state by $S = 0$. The persisting popularity of this model is due to its simplicity as well as its similarity to Boolean concepts in conventional computing. In the following, we will frequently resort to binary model neurons in the symmetrized version (1.7). In fact, the so-called perceptron, as discussed in Chapter 3, can be interpreted as a single McCulloch Pitts unit which is connected to N *input neurons*.

1.2.3 Hebbian learning

Probably the most intriguing property of biological neural networks is their ability to learn. Instead of realizing only *pre-wired* functionalities, brains adapt to their environment or - in higher level terms - they can learn from experience. Many potential forms of plasticity and memory representation have been discussed in the literature, including the chemical storage of information or learning through neurogenesis, i.e. the growth of new neurons.

A very popular and plausible paradigm of learning is synaptic plasticity. A key mechanism, Hebbian Learning, is named after psychologist Donald Hebb, who published his work *The Organization of Behavior* in 1949 [Heb49]. The original hypothesis was formulated in terms of a pair of neurons, which are connected through an excitatory synapse:

> *"When an axon of cell A is near enough to excite cell B and repeatedly or persistently takes part in firing it, some growth process or metabolic change takes place in one or both cells such that A's efficiency, as one of the cells firing B, is increased."*

This is known as Hebb's law and sometimes rephrased as *"Neurons that fire together, wire together."* Hebbian Learning results in a memory effect which favors the simultaneous activity of neurons A and B in the future. Hence, it constitutes a form of learning through synaptic plasticity.

The question to which extent the Hebbian paradigm reflects the biological reality of learning is subject of on-going debate. Alternative or complementing mechanisms have been suggested, see [SVG+18] for a recent example. In the

context of artificial neural networks, Hebbian synaptic plasticity provides a very plausible basis for the representation of learning in the models.

In the mathematical framework of firing rate models presented in the previous section, we can express Hebbian Learning quite elegantly, assuming that the synaptic change is simply proportional to the pre- and post-synaptic activity:

$$\Delta w_{AB} \propto S_A S_B. \tag{1.8}$$

Hence, the change Δw_{AB} of a particular synapse w_{AB} depends only on locally available information: the activities of the pre-synaptic (S_B) and the post-synaptic neuron (S_A). For $S_A, S_B > 0$ this is quite close to the actual Hebbian hypothesis.

The symmetrization with $-1 < S_{A,B} < +1$ adds some biologically implausible aspects to the picture. For instance, an excitatory synapse connecting A and B would also be strengthened according to Eq. (1.8) if both neurons are quiescent at the same time, since in this case $S_A S_B > 0$. Similarly, high activity in A and low activity in B (or vice versa) with $S_A S_B < 0$ would weaken an excitatory or strengthen an inhibitory synapse. In Hebb's original formulation, however, only the presence of simultaneous activity should trigger changes of the involved synapse. Moreover, the mathematical formalism in (1.8) facilitates the possibility that an individual excitatory synapse can become inhibitory or vice versa, which is also questionable from the biological point of view.

Many learning paradigms in artificial neural networks and other adaptive systems can be interpreted as Hebbian Learning in the sense of the above discussion. Examples can be found in a variety of contexts, including supervised and unsupervised learning, see Sec. 2 for working definitions of these terms.

Note that the actual interpretation of the term *Hebbian Learning* varies a lot in the literature. Occasionally, it is employed only in the context of unsupervised learning, since feedback from the environment is quite generally assumed to constitute non-local information. Here, we follow the wide-spread, rather relaxed use of the term for learning processes which depend on the states of the pre- and post-synaptic units as in Eq. (1.8).

Frequently, learning can be seen as the optimization of suitable costs which are interpreted as a function of the network parameters, i.e. the synaptic strengths or weights. As we will see, in many cases numerical optimization procedures, which are for instance based on gradient descent, lead to update rules for the weights that resemble Hebbian Learning to a large extent.

1.3 Network architectures

In the previous section we have considered types of model neurons which retain certain aspects of their biological counterparts and allow for a mathematical formulation of neural activity, synaptic interactions, and learning. This enables us to construct networks from, for instance, sigmoidal or McCulloch Pitts neurons, and model or simulate the dynamics of neurons and/or learning processes concerning the synaptic connections.

In the following, only the most basic and clear-cut types of network architectures are introduced and discussed, namely fully connected recurrent networks and feed-forward layered networks. The possibilities for modifications of these networks, as well as for hybrid and intermediate types are nearly endless. Some more specific architectures will be introduced briefly later; in Section 5.5 and Chapter 6 various *shallow* and *deep* networks will be addressed.

1.3.1 Attractor networks and the Hopfield model

Networks with very high or unstructured connectivity form dynamical systems of neurons which influence each other through synaptic interaction. In a network as shown in Figure 1.4 (left panel) the activity of a particular neuron depends on its synaptic input. Considering discrete timesteps t one obtains an update of the form

$$S_i(t+1) \;=\; g\left(\sum_{j\in\mathbb{J}} w_{ij}\, S_j(t)\right), \tag{1.9}$$

where the sum is taken over all units $j \in \mathbb{J}$ which neuron i receives input from through a synapse $w_{ij} \neq 0$. Eq. (1.9) can be interpreted as an update of all neurons in parallel. Alternatively, units could be visited in a deterministic or randomized sequential order. We will not discuss the subtle, yet important, differences between parallel and sequential dynamics here and refer the reader to the literature, e.g. [HKP91].

From an initial configuration $\mathbf{S}(0)$ at time $t = 0$ which comprises the individual activities $\mathbf{S}(0) = (S_1(0), S_2(0), \ldots, S_N(0))^\top$, the dynamics generates a sequence of states $\mathbf{S}(t)$ which can be considered the system's response to the initial *stimulus*. The term *recurrent networks* has been coined for this type of dynamical system.

One of the most extreme, clear-cut example of a recurrent architecture is the fully connected Hopfield or Little-Hopfield model [Hop82, Lit74, HKP91]. A Hopfield network comprises N neurons of the McCulloch Pitts type which are fully connected by bi-directional synapses

$$w_{ij} = w_{ji} \in \mathbb{R} \quad (i, j = 1, 2, \ldots N) \;\text{ with } w_{ii} = 0 \;\text{ for all } i. \tag{1.10}$$

While the exclusion of explicit, non-zero self-interactions w_{ii} appears plausible, the assumption of symmetric, bi-directional interactions clearly constitutes yet another serious deviation from biological reality.

The dynamics of the binary units is given by

$$S_i(t+1) = \text{sign}\left(\sum_{\substack{j=1\\ j\neq i}}^{N} w_{ij} S_j(t)\right). \tag{1.11}$$

John Hopfield [Hop82] realized that the corresponding random sequential update can be seen as a *zero temperature* Metropolis Monte Carlo dynamics which is

Figure 1.4: Recurrent neural networks. **Left:** A network of $N = 5$ neurons with partial connectivity and uni-directional synapses. **Right:** Pattern retrieval from a noisy initial configuration in a Hopfield network of 2500 units, storing 100 activity patterns. Activities $S_j = \pm 1$ are shown as black and white 'pixels', respectively. Initially, 40% of the S_j are flipped with respect to the pattern. The rightmost figure displays the system shortly before perfect retrieval is achieved.

governed by an energy function of the form

$$H(\mathbf{S}(t)) = -\sum_{\substack{i,j=1 \\ i<j}}^{N} w_{ij}\, S_i(t)\, S_j(t). \tag{1.12}$$

The mathematical structure is analogous to the so-called Ising model in Statistical Physics, see [HKP91, Kob97] for background and further references. There, the degrees of freedom $S_i = \pm 1$ are typically termed *spins* and they represent microscopic magnetic moments. Ising-like systems have been considered in a variety of scientific contexts ranging from the formation of binary alloys to abstract models of segregation in the social sciences. Positive *weights* w_{ij} obviously favor pairs of *aligned* $S_i = S_j$ which reduce the total energy of the system. For the modelling of magnetic materials one considers specific couplings w_{ij} as motivated by the physical interactions. For instance, constant positive $w_{ij} = 1$ are assumed in the so-called Ising ferromagnet, while randomly drawn interactions are employed to model disordered magnetic materials, so-called *spin glasses* [HKP91].

In the actual Hopfield model, however, synaptic weights w_{ij} are constructed or *learned* in order to facilitate a specific form of information processing [Hop82]. From a given set of uncorrelated, N-dimensional *activity patterns* $\mathbb{P} = \{\boldsymbol{\xi}^\mu\}_{\mu=1}^{P}$ with $\xi_i^\mu \in \{-1, +1\}$, a weight matrix is constructed according to

$$w_{ij} = w_{ji} = \frac{1}{P}\sum_{\mu=1}^{P} \xi_i^\mu\, \xi_j^\mu \quad \text{for } i \neq j \quad \text{and } w_{ii} = 0 \ \text{ for all } i, \tag{1.13}$$

where the constant pre-factor $1/P$ follows the convention in the literature. According to Eq. (1.13), we can interpret the weights as empirical averages over

the data set. Improved versions of the weight matrix for correlated patterns are also available. In principle, all perceptron training algorithms discussed later could be applied (per neuron) in the Hopfield network as well.

The Hopfield network can operate as an *auto-associative* or *content addressable* memory: If the system is prepared in an initial state $\mathbf{S}(t=0)$ which differs from one of the patterns $\boldsymbol{\xi}^\nu \in \mathbb{P}$ only for a limited fraction of neurons with $S_i(0) = -\xi_i^\nu$, the dynamics can retrieve the original $\boldsymbol{\xi}^\mu$ from corrupted or noisy information. Ideally, the temporal evolution under the updates (1.11) restores the pattern nearly perfectly and $\mathbf{S}(t)$ approaches $\boldsymbol{\xi}^\nu$ for large t. The retrieval of a stored pattern from a noisy initial state is illustrated in Fig. 1.4 (right panel). Note that the two-dimensional arrangement of neurons serves purely illustrative purposes. Since every neuron is connected to every other neuron, neighborhood relations do not define a low-dimensional topology in the Hopfield model.

Successful retrieval of a stored pattern is only possible if the initial deviation of $\mathbf{S}(0)$ from $\boldsymbol{\xi}^\nu$ is not too large. Moreover, only a limited number of patterns can be stored and retrieved successfully. For random patterns with zero mean activities $\xi_j^\mu = \pm 1$, the statistical physics based theory of the Hopfield model (valid in the limit $N \to \infty$) shows that $P \leq \alpha_r N$ must be satisfied. The value $\alpha_r \approx 0.14$ marks the so-called capacity limit of the Hopfield model[5].

Note that the weight matrix construction (1.13) can also be interpreted as Hebbian Learning: Starting from a tabula rasa state of the synaptic strengths with *zero* weights, a single term of the form $\xi_i^\mu \xi_j^\mu$ is added for each activity pattern, representing the neurons that are connected by synapse w_{ij} (and w_{ji}). Hence, Eq. (1.13) can be written as an iteration

$$w_{ij}(0) = 0, \quad w_{ij}(\mu) = w_{ij}(\mu - 1) + \frac{1}{P}\, \xi_i^\mu \xi_j^\mu, \tag{1.14}$$

where the incremental change of w_{ij} depends only on locally available information and is of the form *"pre-synaptic × post-synaptic activity."*

The Hopfield model serves as a prototypical example of highly connected neural networks. Potential applications include pattern recognition and image processing tasks. Perhaps more importantly, the model has provided many theoretical and conceptual insights into neural computation and continues to do so.

More general recurrent neural networks are applied in various domains that require some sort of temporal or sequence-based information processing. This includes, among others, robotics, speech or handwriting recognition.

1.3.2 Feed-forward layered neural networks

Throughout this reader, we will mainly deal with another clear-cut network architecture: layered feed-forward networks. In these systems, neurons are arranged in layers and information is processed in a well-defined direction.

[5]This critical value is often referred to as α_c in the literature, but it should not be confused with the storage capacity of feed-forward networks in, e.g., Sec. 4.4

The left panel of Fig. 1.5 shows a schematic illustration of a feed-forward architecture. A specific, single layer of units (the top layer in the illustration) represents external input to the system in terms of neural activity. In the biological context, one might think of the photoreceptors in the retina or other sensory neurons which can be activated by external stimuli.

The state of the neurons in all other layers of the network is determined via synaptic interactions, and activations of the form

$$S_i^{(k)} = g\left(\sum_j w_{ij}^{(k)} S_j^{(k-1)}\right). \tag{1.15}$$

Here, the activity $S_i^{(k)}$ of neuron i in layer k is determined from the weighted sum of activities in the previous layer $(k-1)$ only: information contained in the input is processed layer by layer. Ultimately, the last layer in the structure (bottom layer in the illustration) represents the network's output, i.e. its response to the input or stimulus in the first layer. The illustration displays only a single output unit, but the extension to a layer of several outputs is straightforward.

The essential property of the feed-forward network is the directed information processing: neurons receive only input from units in the previous layer. As a consequence, the network can be interpreted as the parameterization of an input/output relation, i.e. a mathematical function that maps the vector of input activations to a single or several output values. This interpretation still holds if nodes receive input from several previous layers, or in other words: connections may "skip" layers. For the sake of clarity and simplicity, we will not consider this option in the following.

The feed-forward property and interpretation as a simple input/output relation is lost as soon as any form of feedback is present: inter-layer synapses or backward connections feeding information into previous ("higher") layers introduce feedback loops, making it necessary to describe the system in terms of its full dynamics.

Neurons that do not communicate directly with the environment, i.e. all units that are neither input nor output nodes, are termed hidden units (nodes, neurons) which form hidden layers in the feed-forward architecture.

The right panel of Fig. 1.5 displays a more concrete example. The network comprises one layer of hidden units, here with activities $\sigma_k \in \mathbb{R}$, and a single output S. The response of the system to an input configuration $\boldsymbol{\xi} = (\xi_1, \xi_2, \ldots, \xi_N) \in \mathbb{R}^N$ is given as

$$S(\boldsymbol{\xi}) = g\left(\sum_{k=1}^{K} v_k\ \sigma_k\right) = g\left(\sum_{k=1}^{K} v_k\ g\Big(\sum_{j=1}^{N} w_{kj}\,\xi_j\Big)\right). \tag{1.16}$$

Here we assume, for simplicity, that all hidden and output nodes employ the same activation function $g(\ldots)$. Obviously, this restriction can be relaxed by defining layer-specific or even individual activation functions. In Eq. (1.16) the

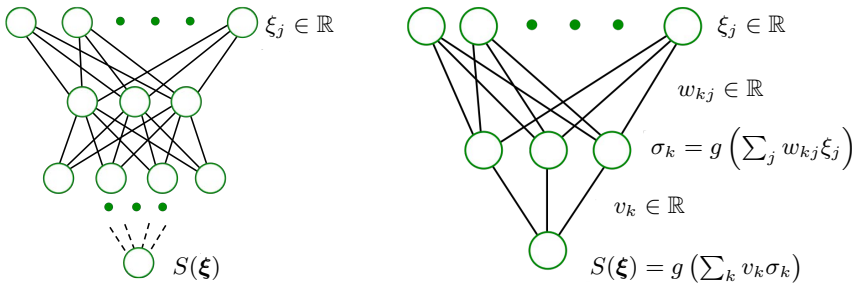


Figure 1.5: Feed-forward neural networks.[6] **Left:** A multilayered architecture with varying layer-size and a single output unit. **Right:** A convergent feed-forward network with a layer of input neurons, one hidden layer, and a single output unit.

quantities w_{kj} denote the weights connecting the j-th input component to the k-th hidden unit with $k = 1, 2, \ldots K$, where K is the total number of hidden units (in the example $K = 3$). These weights can be grouped in vectors $\mathbf{w}_k \in \mathbb{R}^N$, while the hidden-to-output connections are denoted by $v_k \in \mathbb{R}$.

Altogether, the architecture and connectivity, the activation function and its parameters (gain, threshold etc.), and the set of all weights determine the actual input/output function $\boldsymbol{\xi} \in \mathbb{R}^N \to S(\boldsymbol{\xi}) \in \mathbb{R}$ parameterized by the feed-forward network. Again, the extension to several output units, i.e. multi-dimensional function values, is conceptually straightforward.

Without going into details yet, we note that we control the function that is actually implemented by setting the weights and other free parameters in the network. If their determination is guided by a set of example data representing a target function, the term *learning* is used for this adaptation or fitting process. To be more precise, this situation constitutes an example of supervised learning as discussed in the next section.

To summarize, a feed-forward neural network represents an adaptive parameterization of a, in general non-linear, functional dependence. Under rather mild conditions, feed-forward networks with suitable, continuous activation functions are universal approximators. Loosely speaking, this means that a network can approximate any "non-malicious", continuous function to arbitrary precision, provided the network comprises a sufficiently large (problem dependent) number of hidden units in a suitable architecture, see Chapter 5. This property clearly motivates the use of feed-forward nets in quite general regression tasks.

If the response of the network is discretized, for instance due to a thresholding operation that yields

$$S(\boldsymbol{\xi}) \in \{1, 2, \ldots C\}\,, \tag{1.17}$$

the system performs the assignment of all possible inputs $\boldsymbol{\xi}$ to one of C categories

[6] Following the author's personal preference, layered networks are drawn from top (input) to bottom (output). Alternative orientations can be achieved by rotating the page.

or classes. Hence, the feed-forward network constitutes a classifier which can be adapted to example data by choice of weights and other free parameters.

The simplest feed-forward classifier, the so-called perceptron, will serve as a very important example system in the following. The perceptron is defined as a linear threshold classifier with response

$$S(\boldsymbol{\xi}) = \text{sign}\left(\sum_{j=1}^{N} w_j \xi_j - \theta\right) \tag{1.18}$$

to any possible input $\boldsymbol{\xi} \in \mathbb{R}^N$, corresponding to an assignment to one of two classes represented as $S = \pm 1$. Comparison with Eq. (1.7) shows that it can be interpreted as a single McCulloch Pitts neuron which receives input from N real-valued units.

The perceptron will be discussed in detail in Chapter 3 as a basic architecture that provides valuable insights into the fundamentals of machine learning.

1.3.3 Other architectures

Apart from the clear-cut, fully connected attractor neural networks and strictly feed-forward layered nets, a large variety of network types have been considered and designed, often with specific application domains in mind, see Chapter 5.

Combinations of feed-forward structures with, for instance, layers of highly interconnected units are employed in the context of *Reservoir Computing*, see e.g. [SVC07, LJ09] for overviews and references.

Recently, the use of feed-forward architectures has re-gained significant popularity in the context of *Deep Learning* [GBC16, Hue19]. Specific designs and architectures of Deep Neural Networks including e.g. so-called *convolutional* or *pooling* layers will be discussed very briefly in Chapter 5.

The framework of prototype-based learning is introduced in Chapter 6. Systems like Learning Vector Quantization can also be interpreted as layered networks with specific, distance-based activation functions in the hidden units (the prototypes) and a *winner-takes-all* or *softmax* output layer for classification or regression, respectively. Prototype-based systems in unsupervised learning will be discussed briefly in Chapter 6.

Chapter 2

Learning from example data

You live and learn. At any rate, you live.

— Douglas Adams

Different forms of machine learning were already briefly presented in Sec. 1. In the following section, we focus on the most clear-cut scenarios: supervised learning and unsupervised learning. In addition, we will briefly discuss the interesting and fruitful relations of machine learning with statistical modelling.

2.1 Learning scenarios

The main chapters of these notes deal with supervised learning, with emphasis on classification and regression. Several of the introduced concepts and methods can however be transferred to unsupervised settings. This is also the case for the prototype-based methods discussed in Chapter 6.

2.1.1 Unsupervised learning

Unsupervised learning is an umbrella term comprising various methods for the analysis of unlabeled data. Such data sets do not contain label information associated with some pre-defined target as it would be the case in classification or regression. Moreover, there is no direct feedback available from the environment or a *teacher* that would facilitate the evaluation of the system's performance. A comparison of its response with a given ground truth or approximate representation thereof is not available or possible.

For more about the background of unsupervised learning, the reader is referred to the literature, e.g. [Hay09, Bis95a, Bis06]. An introduction, specific algorithms and applications can also be found in lecture notes by Dalya Baron [Bar19]. Here we only briefly discuss the framework of unsupervised data

analysis in contrast to supervised learning. We briefly revisit some key methods of unsupervised learning in the context of preprocessing in Chapter 8.

Potential aims of unsupervised learning are quite diverse, a few examples being:

- **Data Reduction**
  Frequently it makes sense to represent large amounts of data by fewer *exemplars* or *prototypes*, which are of the same form and dimension as the original data and capture the essential properties of the original, larger data set. One important framework is that of *Vector Quantization*.

- **Compression:**
  Another form of unsupervised learning aims at replacing original data by lower-dimensional representations without reducing the actual number of data points, see also Chapter 8. The mapping to lower dimension should obviously preserve information to a large extent. Compression could be done by explicitly selecting a reduced set of features, for instance. Alternative techniques provide explicit projections to a lower-dimensional space or representations that are guided by the preservation of relative distances or neighborhood relations.

- **Visualization**
  Two or three-dimensional representations of a data set can be used for the purpose of visualizing a given data set. Hence, it can be viewed as a special case of *compression* and many techniques can used in both contexts. In addition, more specific tools have been devised for visualization tasks only.

- **Density Estimation:**
  Often, an observed data set is interpreted as being generated in a stochastic process according to a model density. In a training process, parameters of the density are optimized, for instance aiming at a high *likelihood* as a measure of how well the model explains the observations, see Sec. 8.5.

- **Clustering**
  One important goal of unsupervised learning is the grouping of observations into clusters of similar data points which jointly display properties from the other groups or clusters in the data set. Most frequently, clustering is formulated in terms of a specific (dis-)similarity or distance measure, which is used to compare different feature vectors.

- **preprocessing**
  The above mentioned and other unsupervised techniques can be employed to identify representations of a data set suitable for further processing. Consequently, unsupervised learning is frequently considered a useful preprocessing step also for supervised learning tasks.

Note that the above list is by far not complete. Furthermore, the goals mentioned here can be closely related and, often, the same methods can be applied

to several of them. For instance, density estimation by means of *Gaussian Mixture Models* (GMM) could be interpreted as a probabilistic clustering method and the obtained centers of the GMM can also serve as prototypes in the context of Vector Quantization.

Several relevant techniques of unsupervised learning are discussed in Chapter 8, where additional references can also be found.

In a sense, in unsupervised learning there is no "right" or "wrong". This can be illustrated in the context of a toy clustering problem: If we sort a number of fruit according to shape and taste, we would most likely group pears and apples and oranges in three corresponding clusters. Alternatively, we can sort according to color only and end up with clusters of objects with like colors, e.g. combining green apples with green pears vs. yellowish and red fruit. Without further information or requirements defined by the environment, many clustering strategies and outcomes can be plausible. The example also illustrates the fact that the choice of how the data is represented and which types of properties/features are considered important can determine the outcome of an unsupervised learning process to the largest extent.

The important point to keep in mind is that, ultimately, the *users* define the goal of the unsupervised analysis themselves. Frequently this is done by formulating a specific cost function or objective function which reflects the task and guides the training process. The selection or definition of a cost function can be quite subjective and, moreover, its optimization can even completely fail to achieve the implicitly intended goal of the analysis, see Sec. 8.4.3 for the discussion of Vector Quantization as an example.

As a consequence, the identification of an appropriate optimization criterion and objective function constitutes a key difficulty in unsupervised learning. Moreover, a suitable model and mathematical framework has to be chosen that serves the purpose in mind.

2.1.2 Supervised learning

In supervised learning, available data comprises feature vectors[1] together with target values. The data is analysed in order to tune parameters of a model, which can be used to predict the (hopefully correct) target values for novel data that was not contained in the training set.

Generally speaking, supervised machine learning is a promising approach if the target task is difficult or impossible to define in terms of a set of simple rules, while example data is available that can be analysed.

We will consider the following major tasks in supervised learning:

- **Regression**
  In regression, the task is frequently to assign a real-valued quantity to each observed data point. An illustrative example could be the estimation of the weight of a cow, based on some measured features like the animal's height and length.

[1] The discussion of non-vectorial, relational or other data structures is excluded here.

- **Classification**
  The second important example of supervised problems is the assignment of observations to one of several categories or classes, i.e. to a discrete target value. A currently somewhat overstrained example is the discrimination of cats and dogs based on photographic images.

A variety of problems can be formulated and interpreted as regression or classification tasks, including time series prediction, risk assessment in medicine, or the pixel-wise segmentation of an image, to name only a few.

Because target values are taken into account, we can define and evaluate clear quality criteria, e.g. the number of misclassifications for a given test set of data or the expected mean square error (MSE) in regression. In this sense, supervised learning appears well defined in comparison to unsupervised tasks, generally speaking. The well-defined quality criteria suggest naturally meaningful objective functions which can be used to guide the learning process with respect to the given training data.

However, also in supervised learning, a number of issues have to be addressed carefully, including the selection of a suitable model. Mismatched, too simplistic, or overly complex systems can hinder the success of learning. This will be discussed from a quite general perspective in Chapter 7. Similarly, details of the training procedure may influence the performance severely. Furthermore, the actual representation of observations and the selection of appropriate features is essential for the success of supervised training as well.

In the following, we will mostly consider a prototypical workflow of supervised learning where

a) a model or hypothesis about the target rule is formulated in a *training phase* by means of analysing a set of labeled examples. This could be done, for instance, by setting the weights of a feed-forward neural network.

and

b) the learned hypothesis, e.g. the network, can be applied to novel data in the *working phase*, after training.

Frequently, an intermediate *validation phase* is inserted after (a) in order to estimate the expected performance of the system in phase (b) or in order to tune model (hyper-)parameters and compare different setups. In fact, validation constitutes a key step in supervised learning, see Chapter 7.

It is important to keep in mind that many realistic situations deviate from this idealized scenario. Very often, the examples available for training and validation are not truly representative of the data that the system is confronted with in the working phase. The statistical properties and the actual target may even change while the system is trained. This very relevant problem is addressed in the context of so-called *continual* or *life-long* learning.

A clear-cut strategy for the supervised training of a classifier is based on selecting only hypotheses that are consistent with the available training data and perfectly reproduce the target labels in the training set. As we will discuss at length in the context of the perceptron classifier, this strategy of *learning*

in version space relies on the assumption that (a) the target can be realized by the trained system in principle and that (b) the training data is perfectly reliable and noise-free. Although these assumptions are hardly ever realized in practice, the consideration of the idealized scenario provides insight into how learning occurs by elimination of hypotheses when more and more data becomes available.

This can be illustrated in terms of a toy example. Assume that integer numbers have to be assigned to one of two classes denoted as "A" or "B". Assume furthermore that the following example assignments are provided

$$\boxed{4 \to A} \quad \boxed{13 \to B} \quad \boxed{6 \to A} \quad \boxed{8 \to A} \quad \boxed{11 \to B}$$

as a *training set.* From these observations we could conclude, for instance, that A is the class of even integers, while B comprises all odd integers. However, we could also come to the conclusion that all integers $i < 11$ belong to class A and all others to B. Both hypotheses are perfectly consistent with the available data and so are many others. It is in fact possible to formulate an infinite number of consistent hypotheses based on the few examples given.

As more data becomes available, we might have to revise or extend our analysis accordingly. An additional example $\boxed{2 \to B}$ for instance, would rule out the above mentioned concepts, while the assignment of all prime numbers to class B would (still) constitute a consistent hypothesis now.

We will discuss *learning in version space* in greater detail in the context of the perceptron and other networks with discrete output. Note that the strategy only makes sense if the example data is reliable and noise-free; the data itself has to be consistent with the unknown rule that we want to infer, obviously.

The simple toy example also illustrates the fact that the space of allowed hypotheses has to be limited in order to facilitate learning at all. If possible hypotheses may be arbitrarily complex, we can always construct a consistent one by, for instance, simply taking over the given list of examples and claiming that *"all other integers belong to class A"* (or just as well *"...to class B"*). Obviously this approach would not infer any useful information from the data, and such a largely arbitrary hypothesis cannot be expected to *generalize* to integers outside the training set. This is a very simple example for an insight that could be summarized as as[2]

Generalization begins where storage ends.

Merely storing the example set by training a very powerful system may completely miss the ultimate goal of learning, which is inference of useful information about the underlying rule. We will study this effect more formally with respect to neural networks for classification.

The above arguments are particularly clear in the context of classification. In regression, the concept of consistent hypotheses has to be softened, since agreement with the data set is in general quantified by a continuous error measure.

[2] (rephrasing "Generalization begins where learning ends" — T.M. Cover)

However, the main idea of supervised learning remains the same: additional data provides evidence for some hypotheses while others become less likely.

2.1.3 Other learning scenarios

A variety of specific, relevant scenarios can be considered which deviate from the clear-cut simple cases of supervised learning and unsupervised learning. The following examples highlight just some tasks or practical situations that require specific training strategies to cope with. Citations merely point to just one selected review, edited volume or monograph for further reference.

- **Semi-supervised Learning** [CSZ06]
  Frequently, only a subset of the available data is labeled. Strategies have been developed which, in a sense, combine supervised and unsupervised techniques in such situations.

- **Reinforcement Learning** [WO12]
  In various practical contexts, feedback on the performance of a learning system only becomes available after a sequence of decisions has been taken, for instance in the form of a cumulative *reward.* Examples would be the reward received only after a number of steps in a game or in a pathfinding problem in robotics.

- **Transfer Learning** [YCC18]
  If the training samples are not representative for the data that the system is confronted with in the working phase, adjustments might be necessary in order to maintain acceptable performance. Just one example could be the analysis of medical images which were obtained by using similar, yet not identical technical platforms.

- **Lifelong or Continual Learning** [DRAP15]
  Drift processes in non-stationary environments can play an important role in machine learning. The statistics of the observed example data and/or the target itself can change while the system is being trained. A system that learns to detect *spam* e-mail messages, for instance, has to be adapted constantly to the ever-changing strategies of the senders.

- **Causal Learning** [PJS17]
  Regression systems and classifiers typically reflect correlations they have inferred from the data, which allow to make some form of prediction based on future observations. In general, this does not explicitly take *causal relations* into account. The reliable detection of causalities in a data set is a highly non-trivial task and requires specifically designed, sophisticated methods of analysis.

In this material, we will focus almost exclusively on well-defined problems of supervised learning in stationary environments. In most cases, we will assume that training data is representative of the problem at hand and that it is complete and reliable to a certain extent.

2.2 Machine Learning vs. Statistical Modelling

In the sciences it happens quite frequently that the same or very similar concepts and techniques are developed or rediscovered in different (sub-)disciplines, either in parallel or with significant delay.

While it is – generally speaking – quite inefficient to *re-invent the wheel*, a certain level of redundancy is probably inevitable in scientific research. The same questions can occur and re-occur in very different settings, and different communities will come up with specific approaches and answers. Moreover, it can be beneficial to come across certain problems in different contexts and to view them from different angles.

It is not at all surprising that this is also true for the area of machine learning, which has been of inter-disciplinary nature right from the start, with contributions from biology, psychology, mathematics, physics and others.

2.2.1 Differences and commonalities

An area, which is often viewed as competing, complementary, or even superior to machine learning is that of inference in statistical modelling. A simple *web-search* for, say, "Statistical Modelling versus Machine Learning" will yield numerous links to discussions of their differences and commonalities. Some of the statements that one may very likely come across are[3]:

- *The short answer is that there is no difference*
- *Machine learning is just statistics, the rest is marketing*
- *All machine learning algorithms are black boxes*
- *Machine learning is the new statistics*
- *Statistics is only for small data sets, machine learning is for big data*
- *Statistical modelling has lead to irrelevant theory and questionable conclusions*
- *Whatever machine learning will look like in ten years, I'm sure statisticians will be whining that they did it earlier and better.*

These and similar opinions reflect a certain level of competition, which can be counterproductive at times, to put it mildly. In the following, we will refrain from choosing sides in this on-going debate. Instead, the relation between machine learning and statistical modelling will be highlighted in terms of an illustrative example.

One of the most comprehensive, yet accessible presentations of statistical modelling based learning is given in the excellent textbook *The Elements of Statistical Learning* by T. Hastie, R. Tibshirani, and J. Friedman [HTF01]. A view on many important methods, including density estimation and *Expectation Maximization* algorithms is provided in *Neural Networks for Pattern Recognition* [Bis95a] and the more recent *Pattern Recognition and Machine Learning* [Bis06] by C. Bishop.

[3]Exact references and links are not provided in the best interest of the originators.

In both, machine learning and statistical modelling, the aim is to extract information from observations or data and to formalize it. Most frequently, this is done by generating a mathematical model of some sort and *fitting* its parameters to the available data.

Quite often, machine learning and statistical models have very similar or identical structures and, frequently, the same mathematical tools or algorithms are used. The differences lie in the emphasis that is usually put on different aspects of the modelling or learning:

Generally speaking, the main aim of statistical inference is to describe, but also explain and understand the observed data in terms of models. These usually take into account explicit assumptions about statistical properties of the observations. This includes the possible goal of confirming or falsifying hypotheses with a desired significance or confidence.

In Machine Learning, on the contrary, the main motivation is to make predictions with respect to novel data, based on *patterns* detected in the previous observations. Frequently, this does not rely on explicit assumptions in terms of statistical properties of the data but employs heuristic concepts of inference.[4] The goal is not so much the faithful description or interpretation of the data, but rather, it is the application of the derived hypotheses to novel data that is in the center of interest. The corresponding performance, for instance quantified as an expected error in classification or regression, is the ultimate guideline.

Obviously, these goals are far from being really disjoint in a clear-cut way. Genuine statistical methods like Bayesian classification can clearly be used with the exclusive aim of accurate prediction in mind. Likewise, sophisticated heuristic machine learning techniques like *relevance learning* are designed to obtain insight into mechanisms underlying the data, see Chapter 6.

Very often, both perspectives suggest very similar or even identical methods which can be used interchangeably. Frequently, it is only the underlying philosophy and motivation that distinguishes the two approaches.

In the following section, we will have a look at a very basic, illustrative problem: linear regression. It will be re-visited as a prototypical supervised learning task a couple of times. Here, however, it serves as an illustration of the relation between machine learning and statistical modelling approaches and their underlying concepts.

2.2.2 An example case: linear regression

Linear regression constitutes one of the earliest, most important and clearest examples of inference.

As a, by now, historical application, consider the theory of an expanding universe according to which the velocity v of far away galaxies should be directly proportional to their distance d from the observer [Hub29]:

$$v = H_o\, d. \tag{2.1}$$

[4] However, it is very important to realize that implicit assumptions are always made, for instance when choosing a particular machine learning framework to begin with.

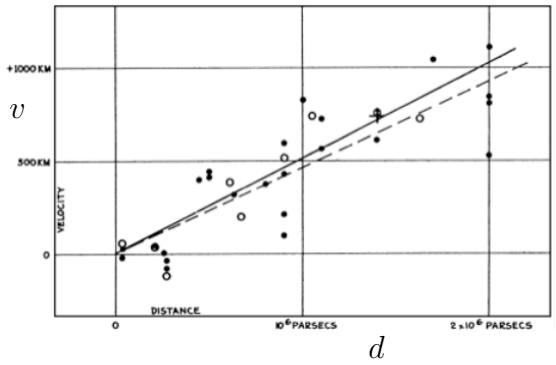


Figure 2.1: Hubble diagram: the velocity v of galaxies as a function of their distance d, taken from [Hub29]. Note that the correct units of v should be km/s. According to PNAS, figure and article [Hub29] are in the public domain.

Here, H_o is the so-called *Hubble constant* which is named after Edwin Hubble, one of the key figures in modern astronomy. Hubble fitted an assumed linear dependence of the form (2.1) to observational data in 1929 and obtained as a rough estimate $H_o \approx 500\frac{km/s}{Mpc}$, see Figure 2.1. The interested reader is referred to the astronomy literature for details, see e.g. [Huc18] for a quick start.

Two major lessons can be learnt from this example: (a) simple linear regression has been and continues to be a highly useful tool, even for very fundamental scientific questions, and (b) the predictive power of a fit depends strongly on the quality of the available data. The latter statement is evidenced by the fact that more recent estimates of the Hubble constant, based on more data of better quality, correspond to much lower values $H_o \approx 73.5\frac{km/s}{Mpc}$ [Huc18].

Obviously, a result of the form (2.1) summarizes experimental or observational data in a *descriptive* fashion and allows us to formulate conclusions that we have drawn from available data. At the same time, it makes it possible to apply the underlying hypothesis on novel data. By doing so, we can test, confirm or falsify the model and its assumptions and detect the need for corrections. The topic of validating a given model will be addressed in greater detail in Chapter 7.

Note that the following discussion is by no means intended to claim that linear regression is a genuine machine learning method. It goes back to at least Legendre and Gauss, see https://en.wikipedia.org/wiki/Linear_regression#History. Here, it merely serves as a relatively simple, illustrative example problem.

A heuristic machine learning approach

Equation (2.1) represents a simple linear dependence of a target function $v(d)$ on a single variable $d \in \mathbb{R}$. In the more general setting of multiple linear regression, a target value $y(\boldsymbol{\xi})$ is assigned to a number of arguments which are concatenated in an N-dimensional vector $\boldsymbol{\xi} \in \mathbb{R}^N$.

In the standard setting of multiple linear regression, a set of examples

$$\mathbb{D} = \{\boldsymbol{\xi}^\mu, y^\mu\}_{\mu=1}^P \quad \text{with} \quad \boldsymbol{\xi}^\mu \in \mathbb{R}^N, y^\mu \in \mathbb{R} \tag{2.2}$$

is given. A hypothesis of the form

$$f_H(\boldsymbol{\xi}) = \sum_{i=1}^{N} w_i\,\xi_i \;=\; \mathbf{w}^\top\boldsymbol{\xi} = \;\mathbf{w}\cdot\boldsymbol{\xi} \;\;\text{with}\;\; \mathbf{w}\in\mathbb{R}^N \tag{2.3}$$

is assumed to represent or approximate the dependence $y(\boldsymbol{\xi})$ underlying the observed data set $\mathbb{D}$. In analogy to other machine learning scenarios considered later, we will refer to the coefficients w_j also as *weights* and combine them in a vector $\mathbf{w}\in\mathbb{R}^N$. Depending on the context, any of the equivalent notations for the scalar product in Eq. (2.3) will be used.

Note that a constant term could be incorporated formally without explicit modification of Eq. (2.3). This can be achieved by decorating every input vector with an additional *clamped* dimension $\xi_{N+1} = -1$ and introducing an auxiliary weight $w_{N+1} = \theta$:

$$\widetilde{\boldsymbol{\xi}} = (\xi_1,\xi_2,\xi_3,\ldots,\xi_N,-1)^\top\;,\;\; \widetilde{\mathbf{w}} = (w_1,w_2,w_3,\ldots w_N,\theta)^\top\in\mathbb{R}^{N+1}$$

$$\Rightarrow \widetilde{\mathbf{w}}^\top\widetilde{\boldsymbol{\xi}} \;=\; \mathbf{w}^\top\boldsymbol{\xi} - \theta. \tag{2.4}$$

Any inhomogeneous hypothesis $f_H(\boldsymbol{\xi}) = \mathbf{w}^\top\boldsymbol{\xi} - \theta$ including a constant term can be written as a homogeneous function in $N+1$ dimensions for an appropriately extended input space, formally. Hence, we will not consider constant contributions to the the hypothesis f_H explicitly in the following. A similar argument will be used later in the context of linearly separable classifiers.

A quite intuitive approach to the selection of the model parameters, i.e. the weights $\mathbf{w}$, is to consider the available data and to aim at a small deviation of $f_H(\boldsymbol{\xi}^\mu)$ from the observed values y^μ. Of the many possibilities to define and quantify this goal, the quadratic deviation or *Sum of Squared Error* (SSE) is probably the most frequently used one:

$$E^{SSE} \;=\; \frac{1}{2}\sum_{\mu=1}^{P}\Big(f_H(\boldsymbol{\xi}^\mu) - y^\mu\Big)^2 \;\;\text{where}\; f_H(\boldsymbol{\xi}^\mu) = \mathbf{w}^\top\boldsymbol{\xi}^\mu \tag{2.5}$$

and the sum is over all examples in $\mathbb{D}$.

The quadratic deviation disregards whether $f_H(\boldsymbol{\xi}^\mu)$ is greater or lower than y^μ. Note that the pre-factor $1/2$ conveniently cancels out when taking derivatives with respect to the weights but is otherwise irrelevant. We will frequently replace the SSE by the *Mean Squared Error* (MSE) which is defined as $E^{MSE} = E^{SSE}/P$. The constant factor $1/P$ is, of course, also irrelevant for the minimization and the properties of the optima.

Necessary and sufficient conditions for the presence of a (local) minimum in a differentiable cost function are briefly summarized and discussed in Appendix A.2.2, where we also point to additional literature. Here, we consider only the necessary first order condition for a weight vector $\mathbf{w}^*$ that minimizes E^{SSE} :

$$\nabla_w E^{SSE}\big|_{\mathbf{w}=\mathbf{w}^*} \stackrel{!}{=} 0 \qquad \text{with}\;\; \nabla_w E^{SSE} \;=\; \sum_{\mu=1}^{P}\left[\mathbf{w}^\top\boldsymbol{\xi}^\mu - y^\mu\right]\,\boldsymbol{\xi}^\mu. \tag{2.6}$$

Note that the SSE is also a popular objective function in the context of regression in multi-layered networks, see Chapter 5 and e.g. [Bis95a,Bis06,HTF01, HKP91,EB01]. With the convenient matrix and vector notation[5]

$$Y = \left(y^1, y^2, \ldots, y^P\right)^\top \in \mathbb{R}^P, \qquad \chi = \left[\boldsymbol{\xi}^1, \boldsymbol{\xi}^2, \ldots, \boldsymbol{\xi}^P\right]^\top \in \mathbb{R}^{P\times N} \tag{2.7}$$

we can rewrite Eq. (2.6) and solve it formally:

$$\chi^\top (\chi \mathbf{w}^* - Y) \stackrel{!}{=} 0 \Rightarrow \mathbf{w}^* = \underbrace{\left[\chi^\top \chi\right]^{-1} \chi^\top}_{\chi^+_{left}} Y \tag{2.8}$$

where χ^+_{left} is the (left) Moore-Penrose pseudoinverse of the rectangular matrix χ [PP12,BH12]. Note that the solution can be written in precisely this form only if the $(N \times N)$ matrix $[\chi^\top \chi]$ is non-singular and, thus, $[\chi^\top \chi]^{-1}$ exists. This can only be the case for $P > N$, i.e. when the system of P equations $\left\{\mathbf{w}^\top \boldsymbol{\xi}^\mu = y^\mu\right\}$ in N unknowns is *over-determined* and cannot be solved exactly.

For the precise definition of the Moore-Penrose and other generalized inverses (also in the case $P \leq N$) see e.g. the *Matrix Cookbook* [PP12] as a comprehensive source of information in the context of matrix manipulations.

Heuristically, in the case of singular matrices $[\chi^\top \chi]$, one can enforce the existence of an inverse by adding a small contribution of the N-dimensional identity matrix I_N:

$$\mathbf{w}^*_\gamma = \left[\chi^\top \chi + \gamma\, I_N\right]^{-1} \chi^\top\, Y. \tag{2.9}$$

Since the symmetric $[\chi^\top \chi]$ has only non-negative eigenvalues, the matrix on the r.h.s. of (2.9) is guaranteed to be non-singular for any $\gamma > 0$.

We will re-visit the problem of linear regression in the context of perceptron training and more general regression later, see also Appendix A.3.2. There, we will also discuss the case of under-determined systems of solvable equations $\left\{\mathbf{w}^\top \boldsymbol{\xi}^\mu = y^\mu\right\}_{\mu=1}^P$.

In analogy to the above, it is straightforward to show that the resulting weights $\mathbf{w}^*_\lambda$ correspond to the minimum of the modified objective function

$$E^{SSE}_\lambda \;=\; \frac{1}{2} \sum_{\mu=1}^{P} \left(f_H(\boldsymbol{\xi}^\mu) - y^\mu \right)^2 \;+\; \frac{1}{2} \gamma\, \mathbf{w}^2. \tag{2.10}$$

Hence, we have effectively introduced a *penalty term*, which favors weight vectors with smaller norm $|\ \mathbf{w}\ |^2$. The concept is known as *weight decay*, see Sec. 7.2. Note that nearly singular matrices $[\chi^\top \chi]$ would lead to large magnitude weights according to Eq. (2.8).

This is our first encounter of *regularization*, i.e. the restriction of the search space in a learning problem with the goal of improving the outcome of the training process. In fact, weight decay is applied in a variety of problems and

[5]For better readability, χ is used instead of Ξ (the properly capitalized version of ξ).

is by no means restricted to linear regression. Other methods of regularization will be discussed in the context of overfitting in neural networks in Sec. 7.2.

We will revisit linear regression again in later chapters and show that it can also be formulated as the minimization of $\mathbf{w}^2$ under suitable constraints. This approach circumvents the problem of having to choose an appropriate weight decay parameter γ in Eqs. (2.9, 2.10).

The statistical modelling perspective

In a statistical modelling approach, we aim at explaining the observed data $\mathbb{D}$ in terms of an explicit model. To this end, we have to make and formalize certain assumptions. For instance, we can assume that the labels y^μ are generated independently according to a conditional density of the form

$$p(y^\mu \mid \boldsymbol{\xi}^\mu, \mathbf{w}) = \mathcal{N}(y^\mu \mid \mathbf{w}^\top \boldsymbol{\xi}, \sigma^2) = \frac{1}{\sqrt{2\pi}\sigma} \exp\left[-\frac{1}{2\sigma^2}\left(y^\mu - \mathbf{w}^\top \boldsymbol{\xi}^\mu\right)^2\right]. \quad (2.11)$$

Hence, we assume that the observed targets essentially reflect a linear dependence but are subject to Gaussian noise:

$$y^\mu \;=\; \mathbf{w}^\top \boldsymbol{\xi}^\mu \;+ \sigma\, \eta^\mu \quad (2.12)$$

with independent, random quantities η^μ with $\langle \eta^\mu \rangle = 0$ and $\langle \eta^\mu \eta^\nu \rangle = \delta_{\mu\nu}$. In contrast to the previous, heuristic treatment, we start from an explicit assumption for how and why the observed values deviate from the linear dependence.

In the following, we consider only $\mathbf{w}$ as parameters of our model, while σ is fixed. Extensions that include σ as an adaptive degree of freedom are very well possible but not essential for the comparison with the heuristic machine learning approach.

In the simplest case we assume that example inputs $\boldsymbol{\xi}^\mu$ are generated independently. Then, for a given model with weights $\mathbf{w}$, the likelihood of observing a particular set of target values $Y = \left(y^1, y^2, \ldots y^P\right)^\top$ factorizes:

$$p(Y \mid \mathbf{w}, \chi) = \prod_{\mu=1}^{P} p(y^\mu \mid \boldsymbol{\xi}^\mu, \mathbf{w}). \quad (2.13)$$

The corresponding log-likelihood reads

$$\log p(Y \,|\, \chi, \mathbf{w}) = \sum_{\mu=1}^{P} \log p(y^\mu \,|\, \boldsymbol{\xi}^\mu, \mathbf{w}) = -\frac{P}{2}\log(2\pi\sigma^2) - \frac{1}{\sigma^2}\,\frac{1}{2}\,\sum_{\mu=1}^{P}\left(y^\mu - \mathbf{w}^\top \boldsymbol{\xi}^\mu\right)^2, \quad (2.14)$$

where we inserted the Gaussian model (2.11). Now we note that the first term on the r.h.s. is constant with respect to $\mathbf{w}$. Furthermore, the second term is proportional to $-E^{SSE}$ as given in Eq. (2.5).

We conclude that the weights $\mathbf{w}^*$ that explain the data with *Maximum Likelihood* under the assumption of model (2.11) are exactly those that minimize the SSE. Hence, we arrive at the same formal solution as given in (2.8).

This correspondence of the Maximum Likelihood solution in the Gaussian model with a quadratic error measure is of course due to the specific mathematical form of the normal distribution and can be rediscovered in various other contexts. The assumption of Gaussian noise is rarely strictly justified, but it is very popular and appears natural in absence of more concrete knowledge. Frequently, it yields practical methods and can be seen as the basis of popular techniques like Principal Component Analysis, mixture models for clustering, or Linear Discriminant Analysis [HTF01, Bis06].

Note, however, that the statistical approach is more flexible in the sense that we could, for instance, replace the conditional model density in (2.11) by an alternative assumption and proceed along the same lines to obtain a suitable objective function in terms of the associated likelihood.

Moreover, it is possible to incorporate prior knowledge, or *prior beliefs*, into the formalism. If we had reason to assume that weights with low magnitude are more likely to occur, even before any data is observed, we could express this in terms of an appropriate prior density, for instance

$$p_o(\mathbf{w}) \propto \exp\left[-\frac{1}{2\tau_o^2}\mathbf{w}^2\right]. \tag{2.15}$$

Exploiting Bayes Theorem, $P(A|B)P(B) = P(B|A)P(A)$, we obtain from the data likelihood $P(\mathbb{D}|\mathbf{w})$:

$$p(\mathbf{w}|\mathbb{D}) \propto p(\mathbb{D}|\mathbf{w})\, p_o(\mathbf{w}). \tag{2.16}$$

With the proper normalization this represents the *posterior probability* of weights $p(\mathbf{w}|\mathbb{D})$ after having seen a data set $\mathbb{D} = \{\chi, Y\}$, and taking into account the data independent prior $p_o(\mathbf{w})$.

Assuming the independent generation of $\boldsymbol{\xi}^\mu$ again and inserting the particularly convenient Gaussian prior (2.15) we can write the logarithm of the posterior as

$$\log\left[p(\mathbf{w}|\mathbb{D})\right] \;\propto\; -E^{SSE} \;-\; \frac{1}{2}\gamma\,\mathbf{w}^2 \;+\; \text{const.} \tag{2.17}$$

with a suitable parameter γ that depends on τ_o and is obtained easily by working out the logarithm of $p(\mathbf{w}|\mathbb{D})$ from Eq. (2.16).

The important observation is that maximizing the posterior probability with respect to the set of weights $\mathbf{w}$ is equivalent to minimizing the objective function given in Eq. (2.10). Hence, the *Maximum A Posteriori* (MAP) estimate of the parameters $\mathbf{w}$ is formally identical with the MSE estimate when amended by an appropriate weight decay term. Not surprisingly, many different names have been coined for this form of regularization and its variants, including L_2-regularization, Tikhonov-regularization, and ridge-regression [HTF01, Bis95a, Bis06, DHS00].

The above discussed Maximum Likelihood and MAP results are examples of so-called *point estimates*: one particular set of model parameters (here: $\mathbf{w}$) is selected according to the specific criterion in use. The statistical modelling idea allows us to go even further: In the framework of *Bayesian Inference* [HTF01]

we can consider all possible model settings at a time, yielding the *posterior predictive probability*

$$p(y|\boldsymbol{\xi},\mathbb{D}) \propto \int p(y|\boldsymbol{\xi},\mathbf{w}) \underbrace{p(\mathbf{w}|\mathbb{D})}_{\propto p(\mathbb{D}|\mathbf{w})\, p_o(\mathbf{w})} d^N w. \tag{2.18}$$

Properly normalized, this defines the probability of response $y(\boldsymbol{\xi})$ to an arbitrary (novel) input $\boldsymbol{\xi}$ after having seen the data set $\mathbb{D}$. It is obtained as an integral over the specific model responses $p(y|\boldsymbol{\xi},\mathbf{w})$ given a particular $\mathbf{w}$, but integrated over all possible models with the posterior $p(\mathbf{w}|\mathbb{D})$ as a weighting factor.

The formalism yields a probabilistic assignment of the target y, which also makes it possible to quantify the associated uncertainty due to the data dependent variability of the model parameters. On the one hand, f this constitutes an appealing advantage over the simpler point estimates. On the other hand, the full formalism can be quite involved in practice. Frequently, one resorts to convenient parametric forms of model and prior densities and/or derives easy to handle (e.g. Gaussian) approximations of the posterior predictive distribution (2.18).

2.2.3 Conclusion

Heuristic machine learning and statistical modeling based approaches differ significantly in terms of their conceptual foundations. However, in practice, these differences often blur or play a minor role. Very often, seemingly purely heuristic methods of machine learning can be derived from the statistical inference perspective under suitable model assumptions. Frequently, training algorithms and methods are very similar if not identical. Many statistics based methods can be used for the main goals of machine learning, i.e. prediction and generalization. At the same time, sophisticated machine learning techniques can also aim at understanding and explaining the observed data, a goal which is frequently attributed to statistical learning.

In summary, the author's recommendation is to acquire knowledge of both, statistical modeling and heuristically motivated machine learning. We should use the best of both complementary frameworks without being *dogmatic* or *religious* in preferring one over the other. The never-ending debates and claims of superiority by both communities are essentially useless and even counterproductive. Instead, efforts should be combined in order to achieve a better understanding of data analysis and learning.

Chapter 3

The Perceptron

The perceptron has shown itself worthy despite (and even because of!) its severe limitations. It has many features to attract attention: its linearity; its intriguing learning theorem; its clear paradigmatic simplicity as a kind of parallel computation.

— Marvin Minsky and Seymour Papert in [MP69]

3.1 History and literature

The term perceptron is used in a variety of meanings. Throughout this text, however, it will exclusively refer to a system representing inputs $\boldsymbol{\xi} \in \mathbb{R}^N$ in a layer of units, which are connected to a single binary output unit of the McCulloch Pitts type. Its activation $S(\boldsymbol{\xi}) \in \{-1, +1\}$ is taken to represent two possible responses.[1] It corresponds to a linear threshold classifier or – in feed-forward network jargon – to an N–1 architecture with a single binary output, that does not comprise hidden units or layers.

Other well-known linear threshold classifiers are Linear Discriminant Analysis (LDA), the Naive Bayes classifier, and the popular Logistic Regression (LR). Overviews and comparisons can be found in e.g. [DHS00,HTF01,Bis95a,Mur22]. These classical methods are based on concepts of statistical modelling and differ mostly in the way their parameters are determined or constructed from a given data set.

In the literature, more general architectures with several layers and/or continuous output are often referred to as (multilayer, soft, ...) perceptrons. We will later consider layered networks which are constructed from perceptron-like

[1] Of course, any binary output, e.g. $S \in \{0, 1\}$, could serve the same purpose.

units, but in these lecture notes the term perceptron always refers to the single layer, binary classifier.

Even the very simple, limited perceptron architecture is of interest for a multitude of reasons:

- Pioneered by Frank Rosenblatt [Ros58, Ros61], the perceptron has been one of the earliest, very successful machine learning concepts and devices, and it was even realized in hardware, see Figure 3.1.

- Rosenblatt also suggested an algorithm for perceptron training, which is guaranteed to converge, provided a suitable solution exists. The corresponding Perceptron Convergence Theorem is one of the most fundamental results in machine learning and has contributed largely to the initial popularity and success of the field. It will be presented and proven in Sec. 3.3.3.

- It serves as a prototypical model system that provides theoretical, mathematical and intuitive insights into the basic mechanisms of machine learning. At the same time it is a building block from which to construct more powerful systems. As Manfred Opper [Opp90] put it: "The perceptron is the hydrogen atom of neural network research."

- In its modern, conceptually extended re-incarnation, the Support Vector Machine (SVM) [SS02, CST00, STC04, Her02, DFO20], the perceptron persists to be used successfully in a large variety of practical applications. The precise relation of the SVM to the simple perceptron will be discussed in great detail in Section 4.3.

- The history of the perceptron provides insights into how the scientific community deals with high expectations and disillusionments leading to the extreme over-reaction of stalling an entire field of research [Ola96].

Several original texts from the early days of the perceptron are available in the public domain. This includes an original article from 1958 [Ros58], the highly interesting official *Manual of the Perceptron Mark I* hardware [HRM$^+$60] and Rosenblatt's monograph *Principles of Neurodynamics* [Ros61]. An interesting TV documentation is available at [You07].

The so-called *Perceptron Controversy* and its perception and long-lasting impact on the machine learning community is analysed in an article entitled *A Sociological Study of the Official History of the Perceptron Controversy* by M. Ozaran [Ola96].

Clear presentations of the Rosenblatt algorithm can be found in virtually all texts that cover the perceptron. Discussions which are quite close to these lecture notes (apart from notation issues) are given in the monographs by J.A. Hertz, A. Krogh and R.G. Palmer [HKP91] and by S. Haykin in [Hay09], for instance.

The counting argument for the number of linearly separable functions, see Sec. 3.4, is presented in e.g. [HKP91]. In this context, it should be useful to

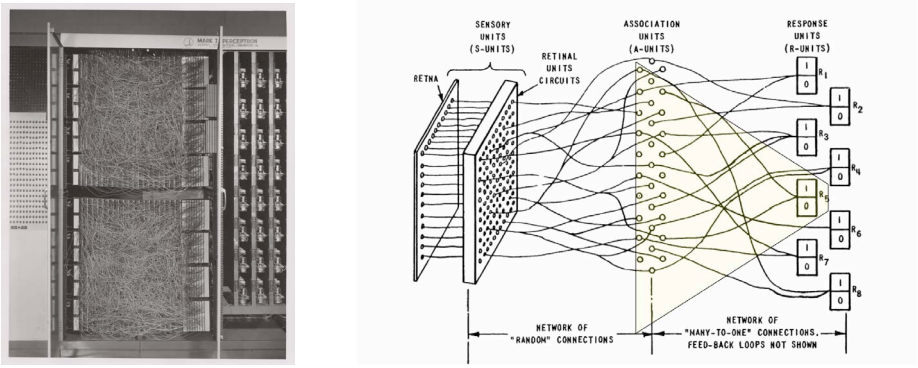


Figure 3.1: The Mark I Perceptron. **Left:** Hardware realization at Cornell Aeronautical Laboratory. Photo reproduced with kind permission from Cornell University Library.[2] The input of the Mark I was realized via a *retina* of 400 photosensors. Adaptive weights were represented by potentiometers that could be tuned by electric motors. **Right:** Schematic outline of the Mark I architecture, based on a figure taken from [HRM$^+$60]. The triangular, shaded region (added to the original illustration) marks a subset of units that is commonly referred to as *the perceptron* in this text.

consult the original publications by R. Winder [Win61], T.M. Cover [Cov65] and G.J. Mitchison and R.M. Durbin [MD89a]. The latter work also presents the extension of the counting argument to two-layered networks *(machines)* with K hidden units.

3.2 Linearly separable functions

The perceptron can be viewed as the simplest feed-forward neural network. It responds to real-valued inputs $\boldsymbol{\xi} \in \mathbb{R}^N$ in terms of a binary output $S \in \{-1, +1\}$. The response of a perceptron with weight vector $\mathbf{w}$ is obtained by applying a threshold operation to the weighted sum of inputs:

$$S_{\mathbf{w},\theta}(\boldsymbol{\xi}) = \text{sign}\,(\mathbf{w} \cdot \boldsymbol{\xi} \, - \, \theta) \, = \, \pm 1, \tag{3.1}$$

where the N-dim. weight vector and the threshold θ parameterize the specific input/output relation. Its mathematical structure suggests an immediate geometrical interpretation of the perceptron which is illustrated in Fig. 3.3: The set of points in feature space

$$\left\{ \widetilde{\boldsymbol{\xi}} \in \mathbb{R}^N \;\middle|\; \left(\mathbf{w} \cdot \widetilde{\boldsymbol{\xi}} - \theta \right) = 0 \right\} \tag{3.2}$$

[2] Cornell University News Service records, #4-3-15. Division of Rare and Manuscript Collections, Cornell University Library. See https://digital.library.cornell.edu/catalog/ss:550351.

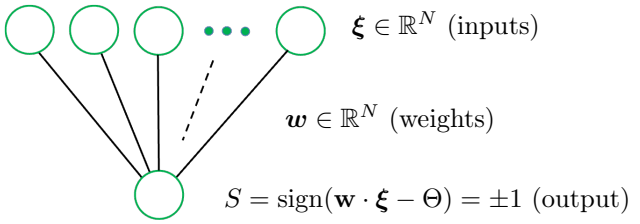


Figure 3.2: Illustration of the single layer perceptron with N-dimensional inputs and a binary output of the McCulloch Pitts type.

corresponds to a (hyper-)plane orthogonal to to $\mathbf{w}$ with an off-set $\theta\,|\mathbf{w}|$ from the origin[3]. Inputs with $\mathbf{w} \cdot \boldsymbol{\xi} > \theta$ result in perceptron output $+1$, while vectors $\boldsymbol{\xi}$ with $\mathbf{w} \cdot \boldsymbol{\xi} < \theta$ yield the response -1. Hence, the perceptron realizes a linearly separable (lin. sep.) function: feature vectors with perceptron output $+1$ are separated by the hyperplane (3.2) from those with output -1.

Two cases can be distinguished: input/output relations of the form (3.1) with $\theta \neq 0$ are called inhomogeneously lin. sep., while homogeneously lin. sep. functions can be written as

$$S_{\mathbf{w}}(\boldsymbol{\xi}) = \text{sign}\,(\mathbf{w} \cdot \boldsymbol{\xi}\,)\,. \tag{3.3}$$

For the latter, the corresponding hyperplane, cf. Fig. 3.3, has no offset ($\theta = 0$) and includes the origin.

In the following, we will focus on homogeneously linearly separable functions, mostly. This does not constitute an essential restriction because any inhomogeneously lin. sep. function can be interpreted as a homogeneous one in a higher dimensional space: Consider the function $S_{\mathbf{w},\theta}(\boldsymbol{\xi}) = \text{sign}\,(\mathbf{w} \cdot \boldsymbol{\xi} - \theta)$ with $\mathbf{w}, \boldsymbol{\xi} \in \mathbb{R}^N$. Now let us define the modified $(N+1)$-dimensional weight vector

$$\widetilde{\mathbf{w}} = (w_1, w_2, \ldots, w_N, \theta)^\top$$

and augment all feature vectors by an auxiliary, "clamped" input dimension:

$$\widetilde{\boldsymbol{\xi}} = (\xi_1, \xi_2, \ldots, \xi_N, -1) \in \mathbb{R}^{N+1}.$$

We observe that

$$\widetilde{\mathbf{w}} \cdot \widetilde{\boldsymbol{\xi}} = \mathbf{w} \cdot \boldsymbol{\xi} - \theta \;\text{ and, thus, }\; \text{sign}\left(\widetilde{\mathbf{w}} \cdot \widetilde{\boldsymbol{\xi}}\right) = \text{sign}\,(\mathbf{w} \cdot \boldsymbol{\xi} - \theta)\,. \tag{3.4}$$

As a consequence, a non-zero threshold θ in N-dimensions can always be rewritten as an additional weight in a trivially augmented feature space and, formally, the two cases can be treated on the same grounds. Note that the argument is analogous to the formal inclusion of a constant term in multiple linear regression, see Eq. (2.4). Later, we will encounter subtleties which require more precise considerations, but for now we will restrict ourselves to homogeneous functions and simply refer to them as linearly separable for brevity.

[3]The distance of the plane from the origin is exactly θ in case of normalized $\mathbf{w}$ with $|\mathbf{w}| = 1$.

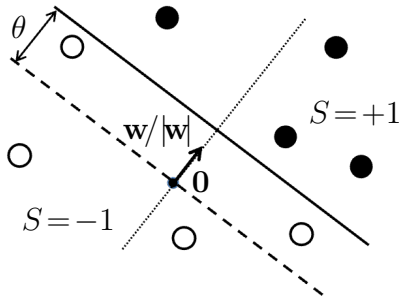


Figure 3.3: Geometrical interpretation of the perceptron. The hyperplane orthogonal to $\mathbf{w}$ with off-set θ from the origin separates feature vectors with output $S = +1$ and $S = -1$, respectively.

In the following, we will also call a set of P input/output pairs

$$\mathbb{D} = \{\boldsymbol{\xi}^\mu, S_T^\mu\}_{\mu=1}^P \tag{3.5}$$

(homogeneously) linearly separable, if at least one weight vector $\mathbf{w}$ exists with

$$S_{\mathbf{w}}^\mu = \text{sign}\,(\mathbf{w}\cdot\boldsymbol{\xi}^\mu) = S_T^\mu \quad \text{for all} \quad \mu = 1, 2, \ldots, P, \tag{3.6}$$

where we use the shorthand notation $S_{\mathbf{w}}^\mu \equiv S_{\mathbf{w}}(\boldsymbol{\xi}^\mu)$. The labels in $\mathbb{D}$ are denoted by $S_T = \pm 1$, with the subscript T for *target* or *training*.

A number of interesting and important questions related to linear separability come to mind:

(Q1) Given a linearly separable data set, (how) can we find a perceptron weight vector $\mathbf{w}$ that satisfies (3.6)?

(Q2) Given an arbitrary data set with binary target labels, can we determine whether it is linearly separable without (before) attempting to train a perceptron? Are there efficient iterative algorithms which indicate (at an early stage) that a data set is not linearly separable?

(Q3) How serious is the restriction to linear separability in the space of binary target functions? How many linearly separable functions exist, i.e. in how many lin. sep. ways can we label P input vectors in N dimensions?

(Q4) Can we learn an underlying linearly separable rule from the examples contained in $\mathbb{D}$? How does the realization or "storage" of the given labels in $\mathbb{D}$ relate to the learning of the unknown rule?

(Q5) If – for a lin. sep. $\mathbb{D}$ – several or many vectors $\mathbf{w}$ satisfy the conditions (3.6), which one is *the best*? What is a meaningful measure of quality and how can the corresponding optimal weight vector be found?

(Q6) If $\mathbb{D}$ is not separable, can we still approximate the target classification by means of a perceptron? Which alternatives or extensions exist?

Most of these questions (Q1–Q5) will be addressed and answered in the forthcoming sections, while Chapter 4 deals with the realization or approximation of classification schemes beyond linear separability (Q6).

3.3 The Rosenblatt perceptron

To a large extent, the success of the perceptron has been due to the existence of a training algorithm and the associated convergence theorem, both presented by Frank Rosenblatt [Ros58, Ros61]. In the following, we first precisely define the basic goal of the training process, outline the general form of iterative perceptron algorithms and present Rosenblatt's algorithm. As a key result, we reproduce the corresponding proof of convergence, eventually.

3.3.1 The perceptron storage problem

Here we address question (Q1) of the list given in the previous section. First we consider the task of reproducing the labels of a given data set of form (3.5) through a perceptron. We define the so-called perceptron storage problem (PSP) as

PERCEPTRON STORAGE PROBLEM (I) (3.7)

For a given $\mathbb{D} = \{\boldsymbol{\xi}^\mu, S_T^\mu\}_{\mu=1}^P$ with $\boldsymbol{\xi}^\mu \in \mathbb{R}^N$ and $S_T^\mu \in \{-1, +1\}$,
find a vector $\mathbf{w} \in \mathbb{R}^N$ with $\mathrm{sign}(\mathbf{w} \cdot \boldsymbol{\xi}^\mu) = \mathrm{S}_\mathrm{T}^\mu$ for all $\mu = 1, 2, \ldots, P$.

The term *storage* refers to the fact that we are not (yet) aiming at the application of the function $S_\mathbf{w}(\boldsymbol{\xi})$ to vectors $\boldsymbol{\xi} \notin \mathbb{D}$. We are only interested in reproducing the correct assignment of labels within the data set by means of a perceptron network. Alternatively, this aim could be achieved by storing $\mathbb{D}$ in a memory table and look up the correct S_T^μ when needed.

In order to rewrite the PSP we note that $\mathrm{sign}(\mathbf{w} \cdot \boldsymbol{\xi}) = \mathrm{S} \;\Leftrightarrow\; \mathbf{w} \cdot \boldsymbol{\xi}\, \mathrm{S} > 0$. Defining the so-called local potentials (the term indicates a vague relation to the membrane potentials, cf. Sec. 1.1.)

$$E^\mu = \mathbf{w} \cdot \boldsymbol{\xi}^\mu \; S_T^\mu \quad \text{for } \mu = 1, 2, \ldots, P, \tag{3.8}$$

we obtain an equivalent formulation of the PSP in terms of a set of inequalities:

PERCEPTRON STORAGE PROBLEM (II) (3.9)

For a given $\mathbb{D} = \{\boldsymbol{\xi}^\mu, S_T^\mu\}_{\mu=1}^P$ with $\boldsymbol{\xi}^\mu \in \mathbb{R}^N$ and $S_T^\mu \in \{-1, +1\}$,
find a vector $\mathbf{w} \in \mathbb{R}^N$ with $E^\mu \geq c > 0$ for all $\mu = 1, 2, \ldots, P$.

Here, we have introduced a constant $c > 0$ as a margin in terms of the conditions $E^\mu > 0$. Note that the actual value of c is essentially irrelevant: Consider vectors $\mathbf{w}_1$ and $\mathbf{w}_2 = \lambda \mathbf{w}_1$ with $\lambda > 0$. Due to the linearity of the scalar product

$$E_1^\mu = \mathbf{w}_1 \cdot \boldsymbol{\xi}^\mu \, S_T^\mu \geq c > 0 \;\text{ implies }\; E_2^\mu = \mathbf{w}_2 \cdot \boldsymbol{\xi}^\mu \, S_T^\mu \geq \lambda\, c > 0. \tag{3.10}$$

The existence of weights $\mathbf{w}$ with all $E^\mu \geq c > 0$ also implies that a solution for any other positive constant can be constructed. This is a consequence of the fact that the function $S_{\mathbf{w}}(\boldsymbol{\xi}) = \text{sign}(\mathbf{w} \cdot \boldsymbol{\xi})$ only depends on the direction of $\mathbf{w}$ in N-dim. feature space while it is invariant under changes of the norm $|\mathbf{w}|$.

3.3.2 Iterative Hebbian training algorithms

In order to answer question (Q1) in Sec. 3.2, we consider iterative learning algorithms which present a single $\left\{\boldsymbol{\xi}^{\nu(t)}, S_T^{\nu(t)}\right\}$ at time step t of the training process. Often, the sequence of examples corresponds to repeated cyclic presentation, i.e.[4]

$$\nu(t) = 1, 2, 3, \ldots, P, 1, 2, 3, \ldots \tag{3.11}$$

where each loop through the examples in $\mathbb{D}$ is called an *epoch* in the literature. A frequently used alternative is random sequential presentation, where at each time step t one of the examples in $\mathbb{D}$ is selected with equal probability $1/P$.

The specific form of perceptron updates we consider in the following is:

GENERIC ITERATIVE PERCEPTRON UPDATES (WEIGHTS)

at discrete time step t

- determine the index $\nu(t)$ of the current training example
- compute the local potential $E^{\nu(t)} = \mathbf{w}(t) \cdot \boldsymbol{\xi}^{\nu(t)} S_T^{\nu(t)}$
- update the weight vector according to

$$\mathbf{w}(t+1) = \mathbf{w}(t) + \frac{1}{N} f(E^{\nu(t)})\, \boldsymbol{\xi}^{\nu(t)} S_T^{\nu(t)}. \tag{3.12}$$

The scaling of the update with N is arbitrary and is used here to achieve consistency with the literature. In order to turn (3.12) into a practical training algorithm, the prescription has to be completed by specifying initial condition $\mathbf{w}$, and by defining a stopping criterion, obviously.

Together with the definition of the sequence $\nu(t)$, the so-called modulation function $f(\ldots)$ determines the actual training algorithm and we assume here that it depends only on the local potential of the actual training example. Note that (3.12) constitutes a realization of Hebbian learning: the change of a component w_j of the weight vector is proportional to the"pre-synaptic" input ξ_j^μ and the "post-synaptic" output S_T^μ.

As a consequence, the weight vector accumulates Hebbian terms $\boldsymbol{\xi}^\mu S_T^\mu$ starting from the given initialization $\mathbf{w}(0)$. Most frequently, we will consider a so-called *tabula rasa* initialization, i.e. $\mathbf{w}(0) = 0$. In this case, after performing

[4] Formally, this can be represented by the function $\nu(t) = \text{mod}[(\text{t}-1), \text{P}] + 1$ for $t \in \mathbb{Z}^+$

updates at time steps $t = 1, 2, \ldots \tau$ the weight vector is bound to have the form

$$\mathbf{w}(\tau) \;=\; \frac{1}{N} \sum_{\mu=1}^{P} x^{\mu}(\tau)\;\boldsymbol{\xi}^{\mu}\, S_T^{\mu}. \tag{3.13}$$

This implies that the resulting perceptron weight vector is a linear combination of the vectors $\boldsymbol{\xi}^{\mu} \in \mathbb{D}$ and the so-called embedding strengths $x^{\mu}(\tau) \in \mathbb{R}$ quantify their specific contributions.

Assuming that $\mathbf{w}(0) = 0$, it is also possible to rewrite the update (3.12) in terms of the embedding strengths. At the end of the training process, the actual weight vector can be constructed according to Eq. (3.13). Hence, the following formulation is equivalent to (3.12):

GENERIC ITERATIVE PERCEPTRON UPDATES (EMBEDDING STRENGTHS)

at discrete time step t

- determine the index $\nu(t)$ of the current training example
- compute the local potential $E^{\nu(t)} = \mathbf{w}(t) \cdot \boldsymbol{\xi}^{\nu(t)}\, S_T^{\nu(t)}$
- update the embedding strength $x^{\nu(t)}$ according to

$$x^{\nu(t)}(t+1) \;=\; x^{\nu(t)}(t) \;+\; f(E^{\nu(t)}) \tag{3.14}$$

(all other embedding strengths remain unchanged at time t).

Many perceptron algorithms can be formulated directly in the weights, cf. (3.12), or in terms of the embedding strengths as in (3.14). While in principle equivalent, we note that the number of variables used to represent the perceptron during training is N in the weight vector formulation and P if embedding strengths are updated. Thus, the computational efficiency of training and the corresponding storage needs will depend on the ratio P/N, in practice.

Note that the simple correspondence between (3.12) and (3.14) can be lost if the structure of the actual updates is modified. Constraints of the form $x^{\mu} \geq 0$ are imposed, for instance, in the AdaTron algorithm [AB89, BAK91] for the perceptron of optimal stability, see Sec. 3.6. This prevents a straightforward formulation of the training scheme as an iteration in weight space. Of course, one can always construct the weight vector via relation (3.13) if needed.

3.3.3 The Rosenblatt perceptron algorithm

In terms of the generic algorithm (3.12, 3.14), the Rosenblatt perceptron algorithm is specified by

- *tabula rasa* initial conditions: $\mathbf{w}(0) = 0$ or, equivalently, $\{x^\mu(0) = 0\}_{\mu=1}^{P}$
- deterministic, cyclic presentation of the examples in $\mathbb{D}$ according to (3.11)
- and the modulation

$$f(E^\mu) \,=\, \Theta\left[c - E^\mu\right] \,=\, \begin{cases} 0 & \text{if } E^\mu > c \\ 1 & \text{if } E^\mu \leq c \end{cases} \tag{3.15}$$

  with the Heaviside function $\Theta[x] = 1$ for $x \geq 0$ and $\Theta[x] = 0$ else.

In summary, the prescriptions (3.12) and (3.14) become:

ROSENBLATT PERCEPTRON ALGORITHM (3.16)

at discrete time step t

- determine the index $\nu(t)$ of the current example according to (3.11)
- compute the local potential $E^{\nu(t)} = \mathbf{w}(t) \cdot \boldsymbol{\xi}^{\nu(t)}\, S_T^{\nu(t)}$
- update the weight vector according to

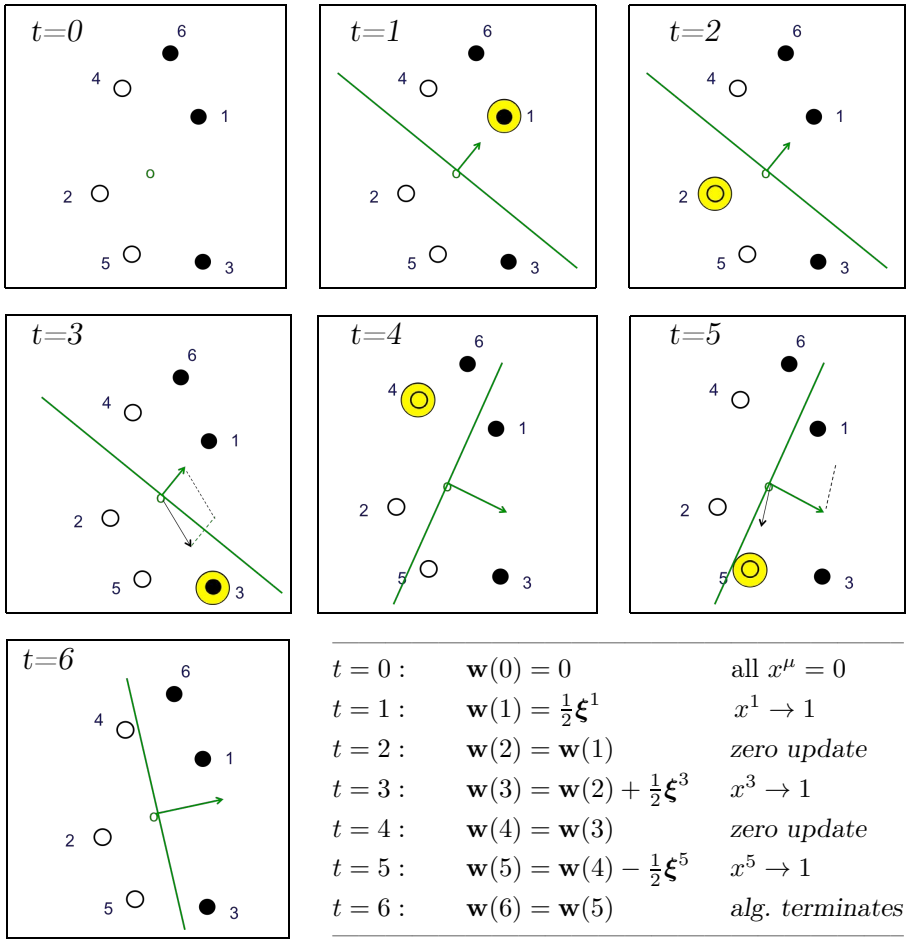
$$\mathbf{w}(t+1) = \mathbf{w}(t) + \frac{1}{N}\,\Theta[c - E^{\nu(t)}]\boldsymbol{\xi}^{\nu(t)}\, S_T^{\nu(t)} \tag{3.17}$$

$$\left[\begin{array}{c} \text{or, equivalently, increment the corresponding embedding strength} \\ x^{\nu(t)}(t+1) = x^{\nu(t)}(t) + \Theta[c - E^{\nu(t)}] \\ \text{(all other embedding strengths remain unchanged at time } t\text{)} \end{array}\right] \tag{3.18}$$

The update (3.16) modifies $\mathbf{w}(t)$ only if the example input is misclassified by the current weight vector or correctly classified with $E^{\nu(t)} < c$. In this case, a Hebbian term is added. The quantity $x^{\nu(t)}$ remains unchanged or increases by 1 in every update step (3.18). Consequently, for tabula rasa initialization, the resulting embedding strengths are non-negative integers.

Frequently, the simple setting $c = 0$ is considered and the resulting scheme is often called the (Rosenblatt) perceptron algorithm. The underlying principle of adding a Hebbian term for misclassified examples is referred to as "learning from mistakes". It is the basis of several other training algorithms discussed in forthcoming sections.

The $(c = 0)$-algorithm stops as soon as all examples in $\mathbb{D}$ are correctly classified: the evaluation of the modulation function yields $\Theta(-E^\mu) = 0$ in all forthcoming update steps. The algorithm is illustrated in terms of a simple two-dimensional feature space and a data set $\mathbb{D}$ comprising six labeled inputs, see Fig. 3.4. In this specific case, the perceptron classifies all feature vectors in



$t = 0:$	$\mathbf{w}(0) = 0$	all $x^\mu = 0$
$t = 1:$	$\mathbf{w}(1) = \frac{1}{2}\boldsymbol{\xi}^1$	$x^1 \to 1$
$t = 2:$	$\mathbf{w}(2) = \mathbf{w}(1)$	*zero update*
$t = 3:$	$\mathbf{w}(3) = \mathbf{w}(2) + \frac{1}{2}\boldsymbol{\xi}^3$	$x^3 \to 1$
$t = 4:$	$\mathbf{w}(4) = \mathbf{w}(3)$	*zero update*
$t = 5:$	$\mathbf{w}(5) = \mathbf{w}(4) - \frac{1}{2}\boldsymbol{\xi}^5$	$x^5 \to 1$
$t = 6:$	$\mathbf{w}(6) = \mathbf{w}(5)$	*alg. terminates*

Figure 3.4: Rosenblatt perceptron algorithm.
Illustration of the training scheme with $c = 0$ in (3.16) in a two-dimensional feature space ($N = 2$). A set of six examples is presented sequentially. Empty circles represent feature vectors labeled with $S_T = -1$, filled circles mark data from class $S_T = +1$. Initial conditions correspond to $\mathbf{w}(0) = 0$ *(tabula rasa)*. Only one epoch of training is considered with $\nu(t) = t = 1, 2, \ldots, 6$. At each time step, $\boldsymbol{\xi}^t$ is marked by a shaded circle. The current weight vector is either updated by adding a Hebbian term if example t is misclassified (time steps $t = 1, 3, 5$) or it remains unchanged if the current classification is correct already (time steps $t = 2, 4, 6$). We refer to the latter as *zero* updates. Actual *non-zero* updates are given by the addition ($S_T^t = +1$) or subtraction ($S_T^t = -1$) of $\frac{1}{N}\boldsymbol{\xi}^t$ which is displayed as an arrow in the illustration. The resulting weight vector is shown in the next time step. For the specific data set considered here, all examples are correctly classified after one epoch already. In general, the data set has to be presented several times before the Rosenblatt algorithm terminates.

$\mathbb{D}$ correctly, and the algorithm stops already after one sweep through the data set, i.e. one epoch of training.

We will show in Sec. 3.3.5 that the Rosenblatt algorithm (3.15) converges in a finite number of steps and finds a weight vector that solves the perceptron storage problem (3.7,3.9), provided the given data set is indeed linearly separable.

3.3.4 The perceptron algorithm as gradient descent

We have introduced the Rosenblatt algorithm more or less intuitively as a form of iterative Hebbian learning. Alternatively, we can motivate it as the minimization of the cost function

$$E(\mathbf{w}) = \sum_{\mu=1} e^{\mu}(\mathbf{w}) = \frac{1}{N} \sum_{\mu=1}^{P} \Theta\left[c - \mathbf{w}\cdot\boldsymbol{\xi}^{\mu} S_T^{\mu}\right] \left(c - \mathbf{w}\cdot\boldsymbol{\xi}^{\mu} S_T^{\mu}\right), \tag{3.19}$$

which is written as a sum over contributions e^{μ} of the given examples in the data set. The example specific term is *zero* if the data point is classified correctly with $\mathbf{w}\cdot\boldsymbol{\xi}^{\mu} S_T^{\mu} > c$ and it contributes the linear costs $(c - \mathbf{w}\cdot\boldsymbol{\xi}^{\mu} S_T^{\mu}) > 0$ otherwise. Note that the objective function (3.19) is frequently referred to as the *hinge loss*, see for instance [HTF01].

Strictly speaking, the function E is not differentiable wherever $\mathbf{w}\cdot\boldsymbol{\xi}^{\mu} = 0$ for one or several examples. In order to cope with this difficulty one can resort to the consideration of the sub-derivative or sub-gradient method, see [Bot04] for a more detailed discussion. From a practical point of view, we can ignore this subtlety here and devise a gradient based minimization as outlined in Appendix A.4 and Sec. 5.2.3 by setting

$$\nabla_w\, e^{\mu} \;=\; -\frac{1}{N}\, \Theta\left[c - \mathbf{w}\cdot\boldsymbol{\xi}^{\mu} S_T^{\mu}\right]\; \boldsymbol{\xi}^{\mu} S_T^{\mu} \tag{3.20}$$

for a single example. Here ∇_w denotes the gradient with respect to the weight vector $\mathbf{w} \in \mathbb{R}^N$. Hence the update (3.17) can be written as

$$\mathbf{w}(t+1) = \mathbf{w}(t) - \left.\nabla_w\, e^{\nu(t)}\right|_{\mathbf{w}=\mathbf{w}(t)} \tag{3.21}$$

where the currently considered example is given by $\nu(t)$ according to (3.11). An update step along the negative gradient tends to decrease $e^{\nu}(t)$ and, thus, the total cost function E.

We will encounter a number of learning algorithms in quite diverse settings which are guided by the gradient of an objective function with respect to the entire data set or single example contributions as above, see Appendix A.4 for a general discussion.

The Rosenblatt algorithm resembles the minimization of $E(\mathbf{w})$ by stochastic gradient descent (SGD), which is presented and discussed more generally in Appendix 5.2.3 and in Chapter 5. In contrast to SGD, however, the original prescription as suggested by Rosenblatt and discussed here presents examples

in the deterministic sequential order (3.11). The formulation in terms of embedding strengths has the structure of *coordinate descent* as discussed in Appendix A.5.1: at every step, only one x^μ is updated. However, its potential increment (3.18) is not given by the derivative $\partial/\partial x^\mu$ of E in Eq. (3.19).[5]

Another important difference to the generic SGD is that, due to the specific structure of the problem, it is not necessary to introduce and fine-tune a learning rate in order to achieve convergence. In fact, one can show that for tabula rasa initialization $\mathbf{w}(0) = 0$, a constant learning rate η as in

$$\mathbf{w}(t+1) = \mathbf{w}(t) - \frac{\eta}{N}\ \nabla_w\, e^{\nu(t)}\Big|_{\mathbf{w}=\mathbf{w}(t)} = \mathbf{w}(t) + \frac{\eta}{N}\,\Theta[c - E^{\nu(t)}]\boldsymbol{\xi}^{\nu(t)}\, S_T^{\nu(t)}$$

would simply rescale the resulting embedding strengths and thus also the weight vector by a factor η as compared to (3.21). This would affect neither the convergence behavior nor the resulting classification scheme. Note, however, that the influence of a time-dependent $\eta(t)$ and its interplay with a potential normalization of the weight vector is non-trivial, see for instance [BSS94].

While the interpretation as a gradient based method helps to relate the Rosenblatt algorithm to other machine learning schemes, we will not make use of it explicitly in the following section. For linearly separable data sets, convergence can be shown explicitly without referring to gradient descent.

3.3.5 The Perceptron Convergence Theorem

The Perceptron Convergence Theorem is one of the most important, fundamental results in the field. The guaranteed convergence of the Rosenblatt perceptron algorithm for linearly separable problems has played a key role for the popularity of the perceptron framework:

PERCEPTRON CONVERGENCE THEOREM (PCT) (3.22)

For linearly separable data $\mathbb{D} = \{\boldsymbol{\xi}^\mu, S_T^\mu\}_{\mu=1}^P$, the Rosenblatt perceptron algorithm stops after a finite number of update steps (3.17) or (3.18) and yields a weight vector $\mathbf{w}$ with $\mathbf{w}\cdot\boldsymbol{\xi}^\mu S_T^\mu \geq c > 0$ for all $\mu = 1, 2, \ldots P$.

In the following we outline the proof of convergence. We consider a linearly separable $\mathbb{D} = \{\boldsymbol{\xi}^\mu, S_T^\mu\}_{\mu=1}^P$, which implies that at least one solution $\mathbf{w}^*$ of the Perceptron Storage Problem (3.7,3.9) exists with

$$\left\{\mathrm{sign}\,(\mathbf{w}^*\cdot\boldsymbol{\xi}^\mu) = \mathrm{S}_\mathrm{T}^\mu\right\}_{\mu=1}^{\mathrm{P}} \quad \text{or, equivalently,} \quad \left\{\mathrm{E}^{\mu *} = \mathbf{w}^*\cdot\boldsymbol{\xi}^\mu\,\mathrm{S}_\mathrm{T}^\mu \geq \mathrm{c} > 0\right\}_{\mu=1}^{\mathrm{P}} \tag{3.23}$$

for some positive constant c.

[5] It is left to the reader to work out the derivative explicitly.

We do not have to further specify $\mathbf{w}^*$ here. In fact, for a given $\mathbb{D}$ there could be many solutions of the form (3.23), but here it is sufficient to assume the existence of at least one. We will furthermore denote its squared norm as

$$Q^* \equiv \mathbf{w}^* \cdot \mathbf{w}^* = |\mathbf{w}^*|^2. \tag{3.24}$$

Note that any pair of vectors $\mathbf{w}, \mathbf{w}^* \in \mathbb{R}^N$ satisfies

$$0 \leq \frac{(\mathbf{w} \cdot \mathbf{w}^*)^2}{|\mathbf{w}^*|^2 \, |\mathbf{w}|^2} = \cos^2 \angle \{\mathbf{w}, \mathbf{w}^*\} \leq 1. \tag{3.25}$$

As discussed above, the algorithm yields - after t time steps - a weight vector of the form

$$\mathbf{w}(t) = \frac{1}{N} \sum_{\mu=1}^{P} x^\mu(t) \, \boldsymbol{\xi}^\mu S_T^\mu \quad \text{for} \quad \mathbf{w}(0) = 0. \tag{3.26}$$

In the Rosenblatt algorithm the quantity $x^\mu(t)$ is an integer that counts how often example μ has contributed a Hebbian term to the weights, cf. Sec. 3.3.3, The total number of non-zero updates is, therefore, given by

$$M(t) = \sum_{\mu=1}^{P} x^\mu(t). \tag{3.27}$$

Now let us consider the projection $R(t) = \mathbf{w}(t) \cdot \mathbf{w}^*$. Inserting (3.26) and exploiting the condition (3.23) we obtain the following lower bound:

$$R(t) = \frac{1}{N} \sum_{\mu=1}^{P} x^\mu(t) \, [\mathbf{w}^* \cdot \boldsymbol{\xi}^\mu S_T^\mu] = \frac{1}{N} \sum_{\mu=1}^{P} x^\mu(t) \underbrace{E^{\mu *}}_{\geq c} \geq \frac{1}{N} \, c \, M(t). \tag{3.28}$$

Similarly, we consider the squared norm $Q(t) = \mathbf{w}(t) \cdot \mathbf{w}(t)$ of the trained weight vector. At time step t with presentation of example $\nu(t)$ it changes as

$$\begin{aligned} Q(t+1) &= \left(\mathbf{w}(t) + \frac{1}{N} \Theta \left[c - E^{\nu(t)} \right] \boldsymbol{\xi}^{\nu(t)} \, S_T^{\nu(t)} \right)^2 \\ &= Q(t) + \frac{2}{N} \Theta \left[c - E^{\nu(t)} \right] E^{\nu(t)} + \frac{1}{N^2} \Theta^2 \left[c - E^{\nu(t)} \right] \left| \boldsymbol{\xi}^{\nu(t)} \right|^2 \end{aligned} \tag{3.29}$$

In any finite data set $\mathbb{D}$, one of the examples will have the largest norm. We can therefore always identify the quantity

$$\Gamma \equiv \frac{1}{N} \max_\mu \left\{ |\boldsymbol{\xi}^\mu|^2 \right\}_{\mu=1}^{P}, \tag{3.30}$$

where the scaling with dimension N is convenient in the following[6]. Next, we observe that $\Theta^2(x) = \Theta(x)$ for all x. Furthermore we note that

[6] We could also consider the simpler, less general case of normalized inputs $|\boldsymbol{\xi}^\mu|^2 = \Gamma N$.

$\Theta[c - E^{\nu(t)}] = 0$ and $E^{\nu(t)} \geq c$ in a zero learning step, while
$\Theta[c - E^{\nu(t)}] = 1$ and $E^{\nu(t)} < c$ in a non-zero learning step.

As a consequence, we can replace all $E^{\nu(t)}$ by c in Eq. (3.29) to obtain the upper bound

$$Q(t+1) \leq Q(t) + \frac{2}{N} c\, \Theta[c - E^{\nu(t)}] + \frac{1}{N} \Gamma\, \Theta[c - E^{\nu(t)}] \tag{3.31}$$

Here we exploit the fact that Q changes only in non-zero updates with $\Theta[\ldots] = 1$. Taking into account the initial value $Q(0) = 0$, we can conclude that

$$Q(t) \leq \frac{1}{N} (2\, c + \Gamma)\ M(t), \tag{3.32}$$

where $M(t)$ is the number of non-zero changes of Q. In summary, we have obtained the two bounds

$$R(t) \geq \frac{1}{N} c\, M(t) \quad \text{and} \quad Q(t) \leq \frac{1}{N} (2c + \Gamma)\, M(t), \tag{3.33}$$

respectively. Exploiting Eq. (3.25) we can write

$$1 \geq \frac{(\mathbf{w}(t) \cdot \mathbf{w}^*)^2}{(|\mathbf{w}(t)||\mathbf{w}^*|)^2} = \frac{R^2(t)}{Q^*\, Q(t)} \geq \frac{\frac{1}{N^2}\, c^2\, M^2(t)}{Q^*\, \frac{1}{N}(2c + \Gamma) M(t)} = \frac{c^2}{Q^*\, N\, (2c + \Gamma)}\, M(t). \tag{3.34}$$

We conclude that

$$M(t) \leq M^* = \frac{(2c + \Gamma)\, N\, Q^*}{c^2}, \tag{3.35}$$

where the right hand side involves only constants: N and Γ are obtained directly from the given data set. The constants c and Q^* characterize the assumed solution $\mathbf{w}^*$, which is - of course - unknown a priori. However, in any case, Eq. (3.35) implies that the number of non-zero learning steps remains finite, if a solution $\mathbf{w}^*$ exists for the given $\mathbb{D}$. In other words, after at most M^* non-zero learning steps, the perceptron classifies all examples correctly with $E^\mu \geq c > 0$. The number of required training epochs is also upper-bounded by M^*, because - as long as the algorithm does not stop - at least one non-zero step must occur in every epoch.

The dependence of M^* on the constant c deserves further attention: In the limit $c \to 0$, the upper bound appears to diverge ($M^* \to \infty$). However, if $\mathbb{D}$ is linearly separable, solutions $\mathbf{w}^*$ with $Q^* \propto c^2$ can be found for small values of c. This is due to the linear dependence of $E^{\mu *}$ on $|\mathbf{w}^*| = \sqrt{Q^*}$, which we already discussed in the context of Eq. (3.10). In the limit $c \to 0$ with $Q^* \propto c^2$ the upper bound becomes

$$\lim_{c \to 0} M^* \approx \Gamma N \frac{Q^*}{c^2} \quad \text{with} \quad \frac{Q^*}{c^2} = const. \tag{3.36}$$

Hence, we can express the PCT (3.22) without referring to a specific value of c.

3.3.6 A few remarks

The number of training steps

According to the PCT, the required number of training steps is *finite* for linearly separable data. However, their actual number depends on the detailed properties of $\mathbb{D}$ and can be very large, see [Roj96] for a discussion of the computational complexity of the Rosenblatt algorithm and further references.

More efficient alternatives to the simple Rosenblatt algorithm can be devised, for instance based on Linear Programming methods [Fle00, PAH19] as discussed in [Roj96].

Non-separable data

For the convergence proof, we had to assume the existence of a solution, i.e. linear separability. The Perceptron Convergence Theorem does not provide direct insight into the algorithm's performance if $\mathbb{D}$ is not separable. We will consider the problem of finding approximative solutions with minimum or low number of errors in non-separable data in Sec. 4.1.

Existence of a solution

In practice, it turns out difficult to decide whether a given $\mathbb{D}$ is linearly separable or not, cf. (Q2) in Sec. 3.2. If a solution is not found by the Rosenblatt algorithm after a number of steps, this could imply that it simply should be run for more steps or that, indeed, a solution does not exist.

A theorem borrowed from the theory of *duality* in optimization, see Appendix A.3.4 and [Fle00, PAH19], provides a surprisingly clear criterion for linear separablity, which is closely related to *Farkas' Lemma* [Fle00]. It is known as *Gordan's Theorem of the Alternative* in the literature [BB00, Man69]. In a notation familiar from the previous sections it reads:

GORDAN'S THEOREM OF THE ALTERNATIVE (3.37)

For a given matrix $\chi \in \mathbb{R}^{N\times P}$, exactly one of the following statements is true:

(1) a vector $\mathbf{w} \in \mathbb{R}^N$ exists with $[\chi^\top \mathbf{w}]^\mu > 0$ for all $\mu = 1, 2, \ldots, P$

or

(2) a non-zero vector $\vec{y} \in \mathbb{R}^P$ exists with $y^\mu \geq 0$ for all μ and $\chi\, \vec{y} = 0$.

If the matrix χ comprises the oriented input vectors of a given data set $\mathbb{D}$,

$$\chi = \left[\boldsymbol{\xi}^1 S_T^1,\ \boldsymbol{\xi}^2 S_T^2,\ \ldots\ \boldsymbol{\xi}^P S_T^P \right],$$

we can interpret $\mathbf{w}$ as the weight vector of a perceptron and statement (1) corresponds to (homogeneous) linear separability of the data set.

On the other hand, (2) states that a linear combination of the form

$$\sum_{\mu=1}^{P} y^\mu \, \boldsymbol{\xi}^\mu \, S_T^\mu \;\; = 0 \text{ with } \; \{y^\mu \geq 0\}_{\mu=1}^{P} \text{ and } \vec{y} \neq 0$$

exists. This goes beyond the condition of linearly dependent columns $\boldsymbol{\xi}^\mu S_T^\mu$ of χ, which would be relevant in the context of solving equations. The consideration of inequalities requires the existence of at least one non-zero vector of non-negative coefficients $\vec{y}$ resulting in a *zero* linear combination.

Observing that $\chi^\top \chi \, \vec{y} = 0 \Leftrightarrow \chi \vec{y} = 0$ we can re-formulate (2) also in terms of the null-space (the space of eigenvectors $\vec{y}$ with eigenvalue *zero*) of the symmetric and positive semi-definite matrix $C = [\chi^\top \chi]/N \in \mathbb{R}^{P \times P}$. We will encounter and make use of the matrix C frequently in later sections.

In practice, checking the validity of condition (2) in Gordan's Theorem may be quite involved: The computation of the null-space can be costly, already. Finding non-negative coefficients $\vec{y}$ for *zero* linear combinations is also non-trivial, in general.

Learning the unlearnable

In this context, several algorithms have been suggested that converge for both separable and non-separable data sets. As one example, D. Nabutovsky and E. Domany suggest such an iterative training procedure in [ND91]. For lin. sep. problems, a consistent perceptron vector is found. However, unlike the Rosenblatt algorithm, the algorithm terminates also in the presence of non-separable data and - in this case - indicates that no solution exists. In [ND91], references to other, similar concepts are provided and the suggested scheme is compared with techniques based on Linear Programming.

3.4 The capacity of a hyperplane

The existence of successful training algorithms for lin. sep. data is certainly of great value. The question remains how relevant linear separability is or, in other words, how serious the restriction to linearly separable functions is.

Linear separability depends on details of the individual, given data set. However, surprisingly general qualitative and quantitative results can be obtained which require only mild assumptions about the data.

3.4.1 The number of linearly separable dichotomies

In this section we address (Q3) from Sec. 3.2 and determine the number $C(P, N)$ of linearly separable binary class assignments or *dichotomies*[7] of P feature vectors in an N-dimensional input space. Quite surprisingly, this is possible under rather mild conditions on the input data.

[7] "There are two kinds of people: Those who couch everything in dichotomies and those who do not." — Unknown

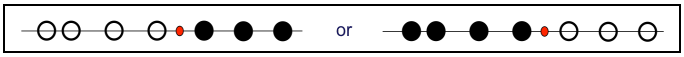


Figure 3.5: One-dimensional inputs ξ can be separated into two classes by the origin in $C(P, 1) = 2$ ways, independent of P.

The derivation for general P and N has been published several times in the literature [Win61, Cov65, MD89a] and was also reviewed in [HKP91]. Here we follow the presentation in [MD89a] to a large extent. Interestingly, the result is closely related to very early findings concerning high-dimensional geometry obtained by the Swiss mathematician Ludwig Schläfli in the 19th century already [Sch01].

More recently, very similar results have been re-discovered from a different point of view in the context of *Deep Networks*, see [Str19, PMB14, MPCB14]. They relate to more general theorems concerning *Face-Count Formulas for Partitions of Space by Hyperplanes*, which are due to T. Zaslavsky [Zas75]. Specifically, counting the number of *response regions* in layered networks with piecewise linear activation functions leads to very similar considerations and results [PMB14].

Some results for small N or P

We first address straightforward cases involving low dimensions N and/or small numbers P of feature vectors. The first, obvious result is that

$$C(1, N) = 2 \quad \text{for all} \quad N.$$

It corresponds to the fact that a single point can always be mapped to $S = \pm 1$ by a a hyperplane through the origin and setting the orientation corresponding to the target label. Here we exclude feature vectors $\boldsymbol{\xi} = 0$, which could always be avoided by an over-all translation of the data set.

The second observation

$$C(P, 1) = 2 \text{ for all } P,$$

reflects the insight that one-dimensional inputs $\xi \in \mathbb{R}$ can only be separated by the origin in two ways: By setting $S = \text{sign}[\xi]$ or by the inverted $S = -\text{sign}[\xi]$, see Fig. 3.5 for an illustration.

In $N = 2$ dimensions it is easy to see that for $P = 2$ typically all 2^P dichotomies can be realized, as displayed in Fig. 3.6 (panel a). However, we have to exclude a special case: If the two input vectors are collinear with the origin, it is impossible to separate them by a line representing a homogeneously linearly separable function.

For $P = 3$ we find that $C(3, 2) < 2^3$, see Fig. 3.6 (panels b,c) panel, even for generic data sets: In panel (b), of the eight possible dichotomies, the two cases with $S^1 = S^2 = S^3 = +1$ or -1 cannot be represented by a plane through the

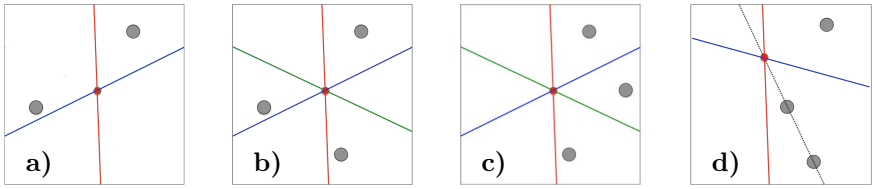


Figure 3.6: **Panel (a):** Two two-dimensional feature vectors (large filled circles) in general position, i.e. not collinear with the origin (small filled circle). Here, four linearly separable dichotomies exist: either both inputs are assigned to the same class $S^1 = S^2 \in \{-1, +1\}$ or they are separated with $S^1 = -S^2$. **Panels (b,c):** Three two-dimensional feature vectors in general position. Of the $2^3 = 8$ dichotomies only six are linearly separable, each one represented by one of the three planes with two possible orientations. In panel (b) the two dichotomies with $S^1 = S^2 = S^3$ cannot be realized, in panel (c) the rightmost point cannot be separated from the other two. **Panel (d):** As indicated by the dotted line, two of the three feature vectors are collinear with the origin. As a consequence only 4 different linearly separable functions can be found.

origin. Similarly, in panel (c) one of the data points cannot be separated from the other two, which also excludes two dichotomies. This implies that

$$C(3, 2) = 6.$$

Again, the result is only valid if no subset of two data points falls onto a line through the origin. In panel (d), a counterexample is displayed. Here, only four distinct dichotomies can be realized by a lin. sep. function, as it is not possible to separate the collinear points by a line through the origin. As a consequence they can only be assigned to the same class, as for instance by the two decision boundaries shown in the illustration.

Similar, explicit insights can still be achieved for three-dimensional data, which is left to the reader as an exercise. Again, one has to exclude specific configurations in which two or more feature vectors are aligned. Moreover, subsets of three data points should not fall into a two-dimensional plane that includes the origin.

General N and P

Our intuition – certainly that of the author – usually fails in higher dimensional spaces, i.e. for $N > 3$. However, it is indeed possible to obtain $C(P, N)$ for general N and P under rather mild assumptions. Similar to the exclusion of collinear input vectors in $N = 2$ and $N = 3$, we formulate the condition that the data set should be in *general position* [Cov65]:

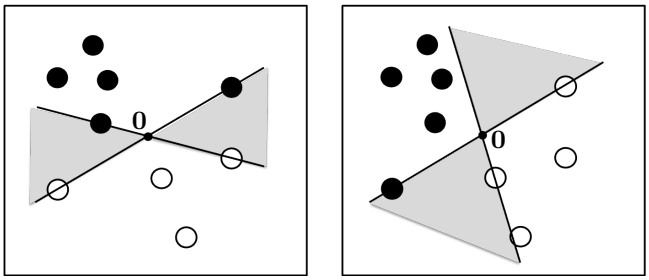


Figure 3.7: Two homogeneously linearly separable dichotomies $D_{N=2}^{P=9}$ of the same set of two-dimensional feature vectors. In both panels, all separating planes in the grey-shaded areas would realize the same assignment of labels.

GENERAL POSITION CONDITION (3.38)

A set of vectors $\mathbb{P} = \left\{\boldsymbol{\xi}^\mu \in \mathbb{R}^N\right\}_{\mu=1}^{P}$ is in general position, if every subset $\{\boldsymbol{\xi}^\rho\}_{\rho=1}^{K} \subset \mathbb{P}$ containing $K \leq N$ elements is linearly independent.

For $P > N$, obviously, subsets of more than N elements in N dimensions are always linearly dependent. In sloppy terms, the general position conditions requires that the vectors in $\mathbb{P}$ do not form *more hyperplanes than necessary*.

In a sense, the general position condition corresponds to the absence of degeneracies in the data. Real world data, however, is frequently prone to such degeneracies. Often, it is even assumed and exploited that nominally high-dimensional feature vectors fall into lower-dimensional manifolds due to correlations and interdependencies.

In the modelling and simulation of learning processes, e.g. when investigating training algorithm performances, one often resorts to randomized feature vectors of N independent components [HKP91]. The popular example of zero mean / unit variance Gaussian random numbers, for instance, leads to data sets that are in general position with probability one.

For the following argument we consider a fixed set of P input vectors $\boldsymbol{\xi} \in \mathbb{R}^N$ in general position. A dichotomy D_N^P of these feature vectors assigns a set of binary labels:

$$D_N^P : \boldsymbol{\xi}^\mu \in \mathbb{R}^N \;\rightarrow\; S^\mu \in \{-1, +1\} \quad \text{for } \mu = 1, 2, \ldots P.$$

The aim is to work out how many of the in total 2^P possible dichotomies correspond to linearly separable functions. Their number will be referred to as $C(P, N)$.

As an illustration, Fig. 3.7 displays a set of $P = 9$ feature vectors in $N = 2$ dimensions. The set has been labeled in two different ways, both of which are linearly separable, as shown in the left and right panel of the figure. All planes

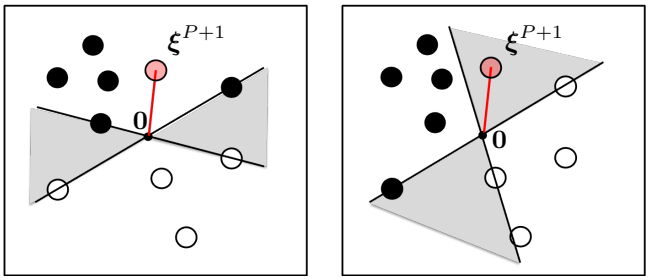


Figure 3.8: An additional feature vector $\boldsymbol{\xi}^{P+1}$ is added to the data set of Fig. 3.7. Its label is either *non-ambiguously* determined by the linearly separable function as in the left panel, or the label is *ambiguous*, $S^{P+1} = +1$ or -1, as shown in the right panel.

through the origin that fall into the shaded region would separate the two classes as represented by filled ($S = +1$) versus empty circles ($S = -1$).

Now we assume that an additional feature vector $\boldsymbol{\xi}^{P+1}$ is added to the data set, as displayed in Figure 3.8. We note that two essentially different situations can occur: In the left panel of the figure, all perceptrons that separate the already known P examples would assign $\boldsymbol{\xi}^{P+1}$ to the same class, i.e. $S^{P+1} = -1$ in the illustration. The dichotomy is said to be non-ambiguous with respect to the new data point.

On the contrary, in the right panel the response is ambiguous with respect to the additional feature vector: Some separating planes correspond to $S^{P+1} = -1$, others to $S^{P+1} = +1$. We will denote by[8]

$$\left.\begin{array}{ll} Z(P,N) & \text{the number of ambiguous} \\ E(P,N) & \text{the number of non-ambiguous} \end{array}\right\} \text{ dichotomies } D_N^P \text{ w.r.t. } S^{P+1}.$$

We note immediately that the total number of linearly separable dichotomies is

$$C(P,N) = Z(P,N) + E(P,N)$$

since each labelling is either ambiguous or non-ambiguous with respect to S^{P+1}.

Obviously, each non-ambiguous dichotomy D_N^P contributes exactly one lin. sep. labeling to $C(P+1,N)$. However, each of the $Z(P,N)$ ambiguous labellings contributes two lin. sep. dichotomies in the enlarged data set with $P+1$ feature vectors, as we have the choice between $S^{P+1} = +1$ and -1 while retaining separability. Hence we have

$$C(P+1,N) = E(P,N) + 2\,Z(P,N) = C(P,N) + Z(P,N). \tag{3.39}$$

Unfortunately this is not yet a useful recursion because $Z(P,N)$ is not known.

[8]Here the symbols E and Z are inspired by the German terms *eindeutig* and *zweideutig*.

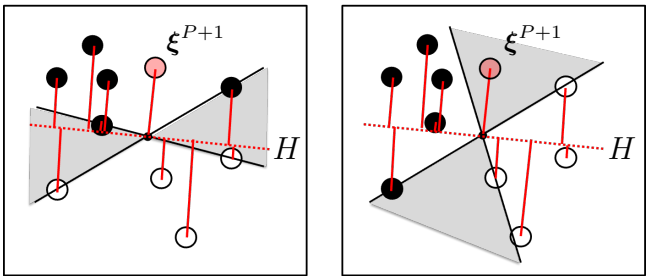


Figure 3.9: The projection of data into the auxiliary subspace $H \perp \boldsymbol{\xi}^{P+1}$ is either linearly separable in H (right panel, ambiguous case) or not separable as in the non-ambiguous case (left panel), respectively.

However the following consideration shows that $Z(P, N)$ can be related to the number of lin. sep. dichotomies in $N-1$ dimensions. In Fig. 3.9 we introduce the auxiliary $(N-1)$-dimensional subspace H, i.e. the hyperplane through the origin which is orthogonal to $\boldsymbol{\xi}^{P+1}$. In the left panel, H falls into the grey shaded area of separating hyperplanes, while in the right panel it does not.

Next, we project the P feature vectors into the auxiliary space H as shown in the figure. In situations exemplified by the left panel, the projections of the P feature vectors into the $(N-1)$-dim. auxiliary space are not linearly separable in H. In the right panel however, examples with $S = \pm 1$ fall into opposite half-spaces in H. In other words, their labels correspond to a linearly separable dichotomy of the P examples in $(N-1)$ dimensions.

On can show that there is indeed a one-to-one correspondence between the $Z(P, N)$ ambiguous lin. sep. functions and all linearly separable dichotomies in H. Assuming that there is *nothing special* about H, we conclude that $Z(P, N) = C(P, N-1)$. Hence we obtain the recursion

$$C(P+1, N) = C(P, N) + C(P, N-1). \tag{3.40}$$

The condition is, in fact, that the data points are in general position as defined above in (3.38). If, for instance the added data point would fall onto a line with another data point and the origin, clearly H could not be considered a generic subspace. For data in general position, however, we can assume that H has the same properties as any $(N-1)$-dim. subspace would.

The number of linearly separable dichotomies

It is straightforward to show that the following expression satisfies the recursion and matches the above given initial values $C(P, 1) = 2$ and $C(1, N) = 2$:

$$C(P, N) = \begin{cases} 2^P & \text{for} \quad P \leq N \\ 2\sum_{i=0}^{N-1} \binom{P-1}{i} & \text{for} \quad P > N, \end{cases} \tag{3.41}$$

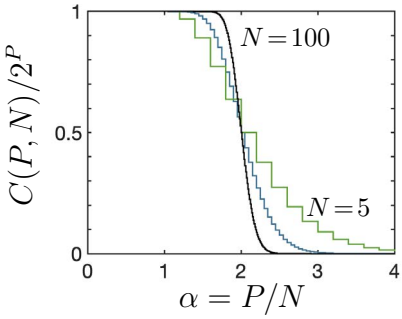


Figure 3.10: The fraction $P_{ls} = C(P,N)/2^P$ of linearly separable functions versus $\alpha = P/N$ for three different values of N $(5, 20, 100)$.

with the familiar bionomial coefficients $\binom{m}{k} = \frac{m!}{k!(m-k)!}$.

The expression in the second line of (3.41) would also reproduce 2^P for $P < N$, as $\binom{m}{k} = 0$ for $k > m$ and $2\sum_{k=0}^{m-1}\binom{m}{k} = 2^m$. The formal consideration of the two cases in (3.41) is only provided for the sake of clarity.

The proof can be done by induction and exploits that

$$\binom{P}{i} = \binom{P-1}{i-1} + \binom{P-1}{i} \text{ for arbitrary } i, P \text{ and } \binom{P}{i} = 0 \text{ for } i < 0.$$

Therefore, $C(P+1,N)$ according to Eq. (3.41) satisfies

$$2\sum_{i=0}^{N-1}\binom{P}{i} = 2\sum_{i=0}^{N-1}\binom{P-1}{i-1} + 2\sum_{i=0}^{N-1}\binom{P-1}{i} = \underbrace{2\sum_{i=0}^{N-2}\binom{P-1}{i}}_{C(P,N-1)} + \underbrace{2\sum_{i=0}^{N-1}\binom{P-1}{i}}_{C(P,N)}$$

which corresponds to the recursion relation (3.40).

Instead of the numbers $C(P,N)$ themselves, we can also consider the fraction of linearly separable dichotomies

$$P_{ls}(P,N) = \frac{C(P,N)}{2^P} = \begin{cases} 1 & \text{for } P \le N \\ 2^{1-P}\sum_{i=0}^{N-1}\binom{P-1}{i} & \text{for } P > N, \end{cases} \tag{3.42}$$

This allows to compare and represent the results graphically for different N as shown in Fig. 3.10. The fraction P_{ls} can be interpreted as the probability for a set of randomly assigned labels $\{S^\mu = \pm 1\}_{\mu=1}^P$ to be linearly separable.

3.4.2 Discussion of the result

For the above arguments and derivation it is essential to assume that the data set is in general position. Typically, violations of this condition will reduce the number of linearly separable dichotomies. In this sense, the results given in Eqs. (3.41) and (3.42) can be interpreted as an upper bound even in more realistic settings.

Figure 3.10 shows the fraction of linearly separable dichotomies $C(P,N)/2^P$ as a function of P/N. This scaling allows to conveniently display several curves for selected values of N in one graph.

The most striking and - at first sight - counterintuitive feature of the result (3.41) is that $C(P,N) = 2^P$ for $P \leq N$. It implies that all possible dichotomies are linearly separable, as long as the number of feature vectors does not exceed their dimension N. We have explicitly confirmed the statement for $N = 2$ in the discussion of Fig. 3.6 (panel a).

For $P > N$, the fraction of linearly separable dichotomies decreases and approaches its limiting value $P_{ls}(\alpha) \to 0$ for $\alpha \to \infty$. As exemplified in Fig. 3.10 for dimensions $N = 5, 20, 100$, the region with $P_{sl} \approx 1$ extends with increasing N and at the same time the decrease of P_{ls} with α becomes steeper. Moreover, all curves intersect in $\alpha = P/N = 2$. The behavior in the simultaneous limits

$$N \to \infty, \quad P \to \infty \ \text{ with finite } \ \alpha = P/N \tag{3.43}$$

is given as

$$P_{ls}^{\infty} = \begin{cases} 1 & \text{for} \quad \alpha \leq 2 \\ 0 & \text{for} \quad \alpha > 2. \end{cases}$$

Hence, in high dimensions, linearly separability is guaranteed (with probability one) up to $P \leq 2N$, which motivates to define the so-called

$$\text{storage capacity of the perceptron: } \alpha_c = 2. \tag{3.44}$$

In general, the storage capacities of more powerful network architectures are non-trivial to obtain, see for instance [EB01, WRB93].

For all N and P in the perceptron we have obtained the even stronger result that $P_{ls} = 1$ exactly for $\alpha \leq 1$, i.e. up to $P = N$. The latter is often referred to as the *Vapnik-Chervonenkis* (VC) dimension of the perceptron, see [HTF01] for a discussion and further references.

At first sight, the storage capacity of the perceptron or other student systems relates only to their ability to reproduce a given data set and implement the corresponding labels. However, as we will see in the following, storage capacity and VC-dimension play an important role with respect to the generalization ability and learning of a rule from example data.

3.4.3 Time for a pizza or some cake

It is amusing to note that the counting of lin. sep. dichotomies is closely related to a popular mathematical puzzle known as the *lazy caterer's problem*: The maximum number of pieces obtained by K straight cuts through a two-dimensional pizza (or a pancake, depending on cultural background) [Wet78, MD09] is given by

$$c(K,2) = \binom{K}{0} + \binom{K}{1} + \binom{K}{2} = \frac{2 + K + K^2}{2} = 1, 2, 4, 7, 11, 16 \ldots \tag{3.45}$$

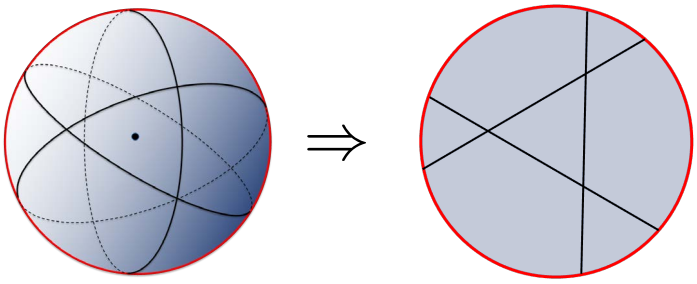


Figure 3.11: The Pizza Connnection: K planar cuts through the center ($\bullet$) of an N-dim. sphere (left panel) correspond to $K-1$ arbitrary straight cuts through the $(N-1)$-dim. surface of each *flattened* hemisphere (right panel).

The corresponding result for three-dimensional objects is known as the *cake number* [Wei99,Str19], i.e. the maximum number of pieces obtained by K planar cuts:

$$c(K,3) = \binom{K}{0} + \binom{K}{1} + \binom{K}{2} + \binom{K}{3} = \frac{6+5K+K^3}{6} = 1, 2, 4, 8, 15, 26 \ldots \tag{3.46}$$

In fact, one can show that the recursion relation

$$c(K+1,N) = c(K,N) + c(K,N-1)$$

holds in complete analogy to Eq. (3.40), albeit with different initial values. See also [Str19] for the proof of a similar result.

We can directly relate the $C(P,N)$ with the generalized cake numbers $c(K,N)$: The left panel of Figure 3.11 displays a sphere in $N=3$ dimension, which is cut by four *equatorial* planes through the center. They correspond to four feature vectors and divide the normalized weight space into regions representing 14 lin. sep. dichotomies. Note that one of the planes (marked by the red circle) is oriented such that the corresponding *upper hemisphere* is shown in top view. A two-dimensional simplified representation of the situation is shown in the right panel, which displays a schematic projection onto the selected equatorial plane. Apparently, there is a 1:1 correspondence of K planar cuts through the center of a spherical cake to $K-1$ unrestricted cuts through a two-dimensional pizza which represents one of the hemispheres. The same holds for the lower hemisphere in the left panel. Quite generally, this yields the simple relation

$$C(P,N) \;=\; 2\,c(P-1,N-1)$$

between the number of lin. sep. functions and the generalized cake numbers.

3.5 Learning a linearly separable rule

Obviously it is not the ultimate goal of perceptron training to reproduce the labels in a given data set, only. This could be done quite efficiently by simply storing $\mathbb{D}$ in memory and look it up when needed.

In general, it is the aim of machine learning to extract information from given data and formulate (parameterize) the acquired insight as a hypothesis, which can be applied to novel data not contained in $\mathbb{D}$ in the working phase.

Addressing (Q4) from Sec. 3.2, we assume that an unknown, linearly separable function or rule exists, which assigns any possible input vector $\boldsymbol{\xi} \in \mathbb{R}^N$ to the binary output $S_R(\boldsymbol{\xi}) = \pm 1$, where the subscript R stands for "rule".

Training of a perceptron from $\mathbb{D} = \{\boldsymbol{\xi}^\mu, S_T^\mu\}$ should infer some information about the unknown $S_R(\boldsymbol{\xi})$ as long as the training labels S_T^μ are correlated with the *correct* $S_R^\mu = S_R(\boldsymbol{\xi}^\mu)$. In the simplest and most clear-cut situation we have

$$S_T^\mu = S_R^\mu \quad \text{for all} \quad \mu = 1, 2, \ldots, P. \tag{3.47}$$

Hence, we assume that the set of training data $\mathbb{D} = \{\boldsymbol{\xi}^\mu, S_R^\mu\}_{\mu=1}^P$ comprises perfectly reliable examples for the application of the rule. It does, for instance, not contain mislabeled example data and is not corrupted by any form of noise in the input or output channel.

We will use the notation S_R^μ or $S_R(\boldsymbol{\xi}^\mu)$ for labels which are given by a rule, explicitly. Here, this indicates that the examples in $\mathbb{D}$ represent a linearly separable function, indeed. Note that more general data sets $\mathbb{D} = \{\boldsymbol{\xi}^\mu, S_T^\mu\}_{\mu=1}^P$ can also be linearly separable without direct correspondence of the S_T^μ to a lin. sep. rule: a small set of examples for a non-separable function or examples corrupted by noise can be very well linearly separable.

3.5.1 Student-teacher scenario

Even for data sets of reliable noise-free examples only, it is not a priori clear that a perceptron is able to reproduce the labels in $\mathbb{D}$ correctly, since the target might not be linearly separable.

To further simplify our considerations, we restrict ourselves to cases, in which the rule is indeed given by a function of the form

$$S_R(\boldsymbol{\xi}) = \operatorname{sign}(\mathbf{w}^* \cdot \boldsymbol{\xi}) \tag{3.48}$$

for a particular weight vector $\mathbf{w}^*$. In fact, all weight vectors $\lambda \mathbf{w}^*$ with $\lambda > 0$ would define the same rule and, therefore, we will assume $|\mathbf{w}^*| = 1$ implicitly, without loss of generality.

A perceptron with weights $\mathbf{w}^*$ is always correct[9]. It is therefore referred to as the teacher perceptron (or teacher, for short) and can be thought of as providing the example data from which to learn. Likewise, the trained perceptron with adaptive weights $\mathbf{w}$ will be termed the student.

[9] The characteristic trait of many teachers, at least in their self-perception.

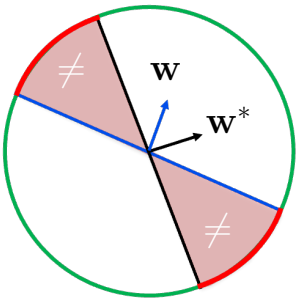


Figure 3.12: Generalization error of the perceptron in a student/teacher scenario. For N-dim. random input vectors generated according to an isotropic density, the probability of disagreement between student vector $\mathbf{w}$ and teacher $\mathbf{w}^*$ is proportional to the red shaded area, i.e. to the angle $\angle(\mathbf{w}, \mathbf{w}^*)$.

The choice of a specific student weight vector corresponds to a particular hypothesis, which is represented by the linearly separable function

$$S_{\mathbf{w}}(\boldsymbol{\xi}) \;=\; \text{sign}\,(\mathbf{w}\cdot\boldsymbol{\xi}) \quad \text{for all } \xi \in \mathbb{R}^N. \tag{3.49}$$

Student-teacher scenarios have been used extensively to model machine learning processes, aiming at a principled understanding of the relevant phenomena. They conveniently allow to control the complexity of the target rule vs. that of the trained system in model situations, thus enabling the systematic study of a variety of setups.

In the following sections we will consider idealized situations, in which student and teacher both represent linearly separable functions. Under this condition, a plausible guideline for training the student from a set $\mathbb{D} = \{\boldsymbol{\xi}^\mu, S_R^\mu\}$ is to achieve perfect agreement with the unknown teacher in terms of the given examples. However, the agreement should not only concern data in $\mathbb{D}$ as in the storage problem; it should extend or *generalize* to novel data, ideally.

Frequently, the so-called *generalization error* serves a measure of success of the learning process. In practical situations, it would correspond to the performance of the student with respect to novel data, for instance in a test set which was not used for training. In our idealized setup, we can revisit the basic geometrical interpretation of linearly separable functions. Figure 3.12 displays a student-teacher pair of weight vectors. The illustration is, obviously, two-dimensional. Note, however, that it can be interpreted as representing the two-dimensional subspace spanned by N-dimensional vectors $\mathbf{w}, \mathbf{w}^*$, more generally.

We assume that a test input $\boldsymbol{\xi}$ is generated according to an isotropic, unstructured density anywhere in $\mathbb{R}^N$. The corresponding generalization error ϵ_g, i.e. the probability for a disagreement

$$\text{sign}(\mathbf{w}\cdot\boldsymbol{\xi}) \neq \text{sign}(\mathbf{w}^*\cdot\boldsymbol{\xi})$$

between student and teacher is directly proportional to the area of the shaded segments, i.e. to the angle $\angle(\mathbf{w}, \mathbf{w}^*)$:

$$\epsilon_g \;=\; \frac{1}{\pi}\,\arccos\left(\frac{\mathbf{w}\cdot\mathbf{w}^*}{|\mathbf{w}||\mathbf{w}^*|}\right). \tag{3.50}$$

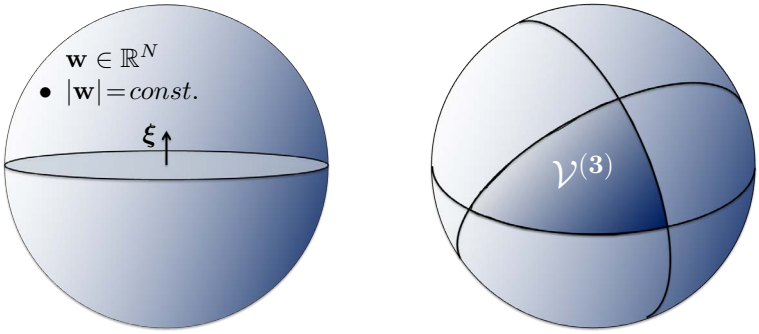


Figure 3.13: Dual geometrical interpretation of linear separability. Illustration in terms of labeled input vectors $\boldsymbol{\xi} \in \mathbb{R}^3$ and L_2-normalized weight vectors with $|\mathbf{w}| = const.$ on the surface of an N-dim. hypersphere.
Left: A single, labeled input $\boldsymbol{\xi}^1$ separates all weight vectors with $S^1_{\mathbf{w}} = +1$ from those with $S^1_{\mathbf{w}} = -1$. **Right:** A set of P labeled input vectors (here: $P = 3$) defines the (darker) region $\mathcal{V}^{(3)}$ of all weight vectors $\mathbf{w}$ with *correct response* $S_{\mathbf{w}}(\boldsymbol{\xi}^\mu) = S^\mu_R$ for $\mu = 1, 2, 3$. For clarity, the vectors $\boldsymbol{\xi}^\mu$ are not shown.

Orthogonal student-teacher pairs would result in $\epsilon_g = 1/2$, which corresponds to randomly guessing the output. Perfect agreement with $\epsilon_g = 0$ is achieved for $\mathbf{w} \parallel \mathbf{w}^*$. The latter statement holds true independent of the statistical properties of the test data, obviously.

3.5.2 Learning in version space

In the classical setup of supervised learning, the training data comprises the only available information about the rule. If the target rule is known to be linearly separable and for reliable noise-free example data, it appears natural to require that the hypothesis is perfectly consistent with $\mathbb{D}$.

But is this a promising training strategy? In other words, can we expect to infer meaningful information about the unknown $\mathbf{w}^*$ by "just" solving the perceptron storage problem with respect to $\mathbb{D}$?

In order to obtain an intuitive insight, we re-visit and extend the geometrical interpretation of linear separability. Following the so-called dual geometric interpretation of linear separability, see Fig. 3.13 (left panel), we can interpret weight vectors $\mathbf{w}$ as points in an N-dim. space. Note that every vector $\boldsymbol{\xi}$ defines a hyperplane through the origin of this space, which separates weight vectors $\mathbf{w} \in \mathbb{R}^N$ with positive $\mathbf{w} \cdot \boldsymbol{\xi}$ and $S_{\mathbf{w}}(\boldsymbol{\xi}) = +1$ from those with negative scalar product and $S_{\mathbf{w}}(\boldsymbol{\xi}) = -1$. Hence, given the correct target label $S_R(\boldsymbol{\xi})$, the plane orthogonal to $\boldsymbol{\xi}$ separates correct from wrong students in N-dimensional weight space.

Consequently, a set $\mathbb{D}$ of P labeled feature vectors defines a region or volume of vectors $\mathbf{w}$ which reproduce $S_{\mathbf{w}}(\boldsymbol{\xi}^\mu) = S_R(\boldsymbol{\xi}^\mu)$ for all $\mu = 1, 2, \ldots, P$, as

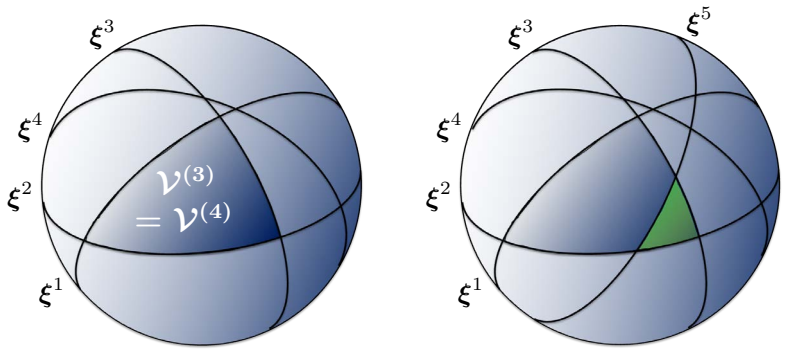


Figure 3.14: Illustration of perceptron learning in version space. **Left:** The hyperplane associated with the additional example $\boldsymbol{\xi}^4$ does not intersect the version space $\mathcal{V}^{(3)}$. Consequently, all weight vectors in $\mathcal{V}^{(3)}$ already classify $\boldsymbol{\xi}^4$ correctly and $\mathcal{V}^{(4)} = \mathcal{V}^{(3)}$. **Right:** The hyperplane associated with $\boldsymbol{\xi}^5$ cuts through $\mathcal{V}^{(4)}$ and, consequently, $\mathcal{V}^{(5)}$ corresponds to either the small *triangular* region (green) or the remaining dark area, depending on the actual target S_R^5.

illustrated in Fig. 3.13 (right panel).

The set of all perceptron weight vectors which are consistent with the P examples in $\mathbb{D}$, i.e. which give $S_{\mathbf{w}}^{\mu} = S_R^{\mu}$ for all $\mu = 1, 2, \ldots, P$, is termed version space and can be defined as

$$\mathcal{V} = \left\{ \mathbf{w} \in \mathbb{R}^N, \, \mathbf{w}^2 = 1 \;\middle|\; \mathrm{sign}(\mathbf{w} \cdot \boldsymbol{\xi}^{\mu}) = S_R^{\mu} \;\text{ for all }\; \{\boldsymbol{\xi}^{\mu}, S_R^{\mu}\} \in \mathbb{D} \right\}. \quad (3.51)$$

In the following, we will use the notation $\mathcal{V}^{(P)}$, if we want to refer to the number of examples in $\mathbb{D}$ explicitly.

In the context of the previous section, $C(P, N)$ in Eq. (3.41) can be interpreted as the number of different version spaces that can be constructed for a set of P input vectors in N-dimensional space by assigning linearly separable target labels $S^{\mu} = \pm 1$.

It is important to note that defining $\mathcal{V}$ as a set of normalized vectors with $\mathbf{w}^2 = 1$ is convenient but not essential. Obviously, the normalization is irrelevant with respect to the conditions $\mathrm{sign}(\mathbf{w} \cdot \boldsymbol{\xi}^{\mu}) = S_R^{\mu}$ and could be replaced by any other constant norm or even be omitted.

The version space is non-empty, $\mathcal{V} \neq \emptyset$, if and only if $\mathbb{D}$ is linearly separable: If a (normalized) teacher vector $\mathbf{w}^*$ defines the linearly separable rule represented by the examples in $\mathbb{D}$, then $\mathbf{w}^* \in \mathcal{V}$, necessarily. In words: at least the teacher vector itself must be inside version space. However, in absence of any information beyond the data set $\mathbb{D}$, the unknown $\mathbf{w}^*$ could be located anywhere in $\mathcal{V}$ with equal likelihood.

The term *learning in version space* refers to the idea of admitting only hypotheses which are perfectly consistent with the example data. Let us assume

that, given a set $\mathbb{D}$ of P reliable examples for a linearly separable rule, we have identified *some* vector $\mathbf{w} \in \mathcal{V}$. According to the Perceptron Convergence Theorem (3.22) this is always possible, e.g. by means of the Rosenblatt perceptron algorithm[10]. In our low-dimensional illustration, Fig. 3.13 (right panel), this means that we can always place a student vector $\mathbf{w}$ somewhere in the darker region representing $\mathcal{V}^{(3)}$ for $P = 3$ training examples.

Now, consider a fourth labeled example $\{\boldsymbol{\xi}^4, S_R^4\}$ as displayed in Fig. 3.14 (left panel). The hyperplane associated with $\boldsymbol{\xi}^4$ does not intersect the version space $\mathcal{V}^{(3)}$. Consequently all student vectors $\mathbf{w} \in \mathcal{V}^{(3)}$ fall into the same half-space with respect to the new example and yield the same perceptron output $\mathrm{sign}(\mathbf{w} \cdot \boldsymbol{\xi}^4)$.

This implies in turn that, if the extended data set $\{\boldsymbol{\xi}^\mu, S_R^\mu\}_{\mu=1}^4$ is still linearly separable, only one value of the target label S_R^4 is possible, which must be $S_R^4 = \mathrm{sign}(\mathbf{w} \cdot \boldsymbol{\xi}^4)$ for $\mathbf{w} \in \mathcal{V}^{(3)}$. Consequently we have that $\mathcal{V}^{(4)} = \mathcal{V}^{(3)}$; the version space with respect to the extended data set remains the same as for the previously known $P = 3$ examples. Hence, our strategy of learning in version space does not require modifying the hypothesis or selecting a new student vector $\mathbf{w}$ to parameterize it. In this sense, the input/output pair $\{\boldsymbol{\xi}^4, S_R^4\}$ is un-informative in the given setting.

The situation is different in the case illustrated in the right panel of Fig. 3.14. Here, the data set is amended by $\{\boldsymbol{\xi}^5, S_R^5\}$ with the plane orthogonal to $\boldsymbol{\xi}^5$ cutting through $\mathcal{V}^{(4)} = \mathcal{V}^{(3)}$. Elements of $\mathcal{V}^{(4)}$ on one side of the hyperplane correspond to the perceptron response $S_{\mathbf{w}}(\boldsymbol{\xi}^5) = +1$, while the others yield $S_{\mathbf{w}}(\boldsymbol{\xi}^5) = -1$.

Depending on the actual target S_R^5 of the additional example, the version space $\mathcal{V}(5)$ corresponds to either the green area or the remaining lighter region in the illustration. In any case, the extended data set $\{\boldsymbol{\xi}^\mu, S_R^\mu\}_{\mu=1}^5$ is linearly separable and the new version space $\mathcal{V}^{(5)}$ of consistent weight vectors is bound to be *smaller* than the previous $\mathcal{V}^{(4)} = \mathcal{V}^{(3)}$. The volume of consistent weight vectors $\mathbf{w}$ shrinks due to the information associated with the new example data.

As we add more examples to the data set, the corresponding version space can only remain the same or decrease in size. Indeed, one can show that $\mathcal{V}$ will shrink to a point with $P/N \to \infty$ under the rather mild condition of general position discussed in the previous section.

Together with the fact that the teacher $\mathbf{w}^* \in \mathcal{V}^{(P)}$ for any P, we can conclude that learning from version space will enforce $\mathbf{w} \to \mathbf{w}^*$ with increasing training set size. More precisely, we conclude that the angle $\angle(\mathbf{w}, \mathbf{w}^*) \to 0$ for unnormalized vectors $\mathbf{w}, \mathbf{w}^*$. Hence, we can expect that learning in version space yields hypotheses which agree with the unknown rule to a large extent, provided the data set contains many examples. In the sense of the above discussed generalization error (3.50), it will achieve perfect generalization $\epsilon_g \to 0$ for $P \to \infty$.

[10] ... with subsequent normalization in order to match the definition (3.51) precisely.

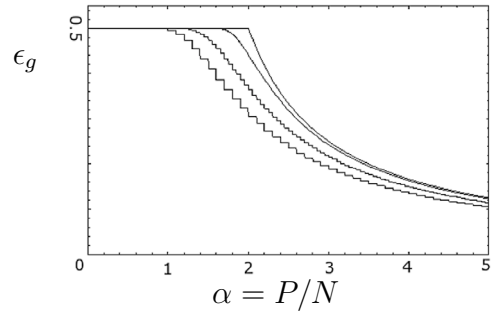


Figure 3.15: The generalization error ϵ_g vs. $\alpha = P/N$ as obtained from the number of ambiguous / non-ambiguous linearly separable functions, cf. Eq. 3.52. The curves correspond to $N = 5, 20, 100$ and to the limit $N \to \infty$, respectively (from left to right).

3.5.3 Generalization begins where storage ends

The above, elementary and illustrative considerations do not provide more concrete mathematical relations which, for instance, quantify the generalization error as a function of the training set size P.

However, we can re-visit and exploit the result of Section 3.4, where we counted the number of linearly separable functions of P feature vectors in N dimensions under quite general assumptions [Win61, Cov65, MD89a].

Assume that the training process has exploited P examples for a linearly separable rule and that we have identified a student weight vector in $\mathcal{V}^{(P)}$ which corresponds to one of the $C(P, N)$ linearly separable dichotomies of the data.

Implicitly, we assume that the true rule represents any of the $C(P, N)$ linearly separable functions with equal probability. Alternative approaches, e.g. from the statistical physics perspective, weight every dichotomy with the volume of the associated version space. We refrain from a detailed discussion of this subtlety and refer the reader to the more specialized literature [HKP91, EB01, WRB93].

Now assume that we present a novel input vector $\boldsymbol{\xi}^{test}$ with target label S^{test} to the student. In complete analogy to the considerations illustrated in Fig. 3.14, the version space could be ambiguous or non-ambiguous with respect to the perceptron output $S^{test}_{\mathbf{w}}$. In the latter case we know that any $\mathbf{w} \in \mathcal{V}^{(P)}$ will provide the correct response. If the version space is ambiguous with respect to $\boldsymbol{\xi}^{test}$, a fraction of the weight vectors in $\mathcal{V}^{(P)}$ will be correct while the remaining ones would predict an incorrect label. Therefore, in absence of more detailed knowledge, we expect that the response of a random $\mathbf{w} \in \mathcal{V}^{(P)}$ will be incorrect with probability 1/2.

As a consequence, the generalization error ϵ_g is proportional to the fraction of ambiguous dichotomies and we obtain

$$\epsilon_g = \frac{1}{2}\frac{Z(P,N)}{C(P,N)} = \frac{1}{2}\frac{C(P,N-1)}{C(P,N)} \tag{3.52}$$

with $C(P, N)$ from Eq. (3.41). The result is displayed in Fig. 3.15 as a function of the scaled number of examples $\alpha = P/N$. The curves represent input dimensions $N = 5, 20, 100$ and the limiting case $N \to \infty$, cf. Eq. (3.43).

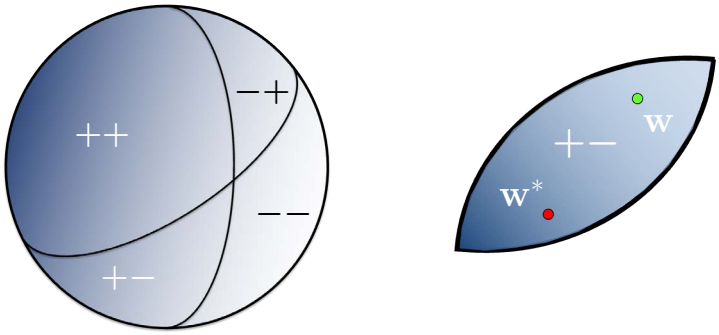


Figure 3.16: Linear separability for $P < N$.
Left panel: Version space corresponding to two feature vectors in $N = 3$ dimensions with all (four) possible combinations of labels ± 1. **Right panel:** Assume that, as an example, the region marked $+-$ corresponds to the set of correct (normalized) weight vectors. The version space is large enough to allow for significant disagreement between student $\mathbf{w}$ and teacher vector $\mathbf{w}^*$. Even $\epsilon_g = 1$ is possible for extreme settings with $\mathbf{w} = -\mathbf{w}^*$.

For arbitrary N and P we find that $\epsilon_g = 1/2$ as long as $P \leq N$. For larger N, the region of $\epsilon_g = 1/2$ extends, up to $\alpha = \alpha_c = 2$ in the limit (3.43). This finding indicates that for relatively small data sets, the perceptron output is no better than unbiased random guessing by, for instance, flipping a coin!

Hence, we learn from the above counting argument that storage without generalization is possible for small P/N. Figure 3.16 illustrates the situation in $N = 3$ dimensions. The version space for $P < N$ allows to place the student vector very far away from the (unknown) teacher vector in $\mathcal{V}^{(P)}$ without causing disagreement on the training data. Even total misalignment with $\mathbf{w} = -\mathbf{w}^*$ is possible in an extreme setting.

The result (3.52) indicates that for a randomly selected student inside the version space of a randomly selected lin. sep. dichotomy, storage without learning the rule can be observed up to the Vapnik-Chervonenkis-dimension $P = N$. For large $N \to \infty$ the region extends to the storage capacity, i.e. to $P = 2N$.

On the positive side, we also see that for data set sizes which exceed the storage capacity, it is inevitable for the perceptron to perform better than random. Learning in version space is guaranteed to yield non-trivial generalization as soon as the number of examples exceeds the storage capacity (or the VC-dimension for finite N, P).

These findings confirm the intuition that learning and generalization is not possible if the hypothesis space is too flexible. A very complex student system can implement large sets of example data without inferring useful information about the underlying rule.

Asymptotic behavior for $N \to \infty$

For large $N \to \infty$ with $P/N = \alpha$ one can show that the asymptotic form of Eq. (3.52) reads [Cov65]

$$\epsilon_g(\alpha) = \begin{cases} 1/2 & \text{for } \alpha \leq 2 \\ \frac{1}{2}\,\frac{1}{\alpha - 1} & \text{for } \alpha > 2. \end{cases} \tag{3.53}$$

As an alternative to the counting argument, methods borrowed from statistical physics have been applied to compute the typical version space volume in high dimensions ($N \to \infty$) and yield the same basic dependence, i.e.

$$\epsilon_g \propto \alpha^{-1} \quad \text{for } \alpha \to \infty$$

for various training algorithms in the presence of noise-free data, see for instance [EB01, WRB93, BSS94, Opp94].

However, the statistical physics based analysis and similar considerations also show that learning in version space typically performs better than random even for small training sets. It turns out beneficial to select specific students in version space. Even simple algorithms like the Rosenblatt scheme typically yield $\epsilon_g(\alpha) < 1/2$ already for small α. In the following sections we will consider, more specifically, the perceptron of optimal stability which achieves near optimal expected performance.

3.5.4 Optimal generalization

In the student-teacher setup discussed above, we only know that the teacher $\mathbf{w}^*$ is located somewhere in $\mathcal{V}$. In absence of additional knowledge it could be anywhere in the version space with equal probability.

Learning in version space places the student $\mathbf{w}$ also *somewhere* in $\mathcal{V}$. Its generalization error ϵ_g is, under very general circumstances, a decreasing function of the angle $\angle(\mathbf{w}, \mathbf{w}^*)$, which itself is a decreasing function of the Euclidean distance $|\mathbf{w} - \mathbf{w}^*|$ for normalized weight vectors $\mathbf{w}, \mathbf{w}^*$, see Fig. 3.12. As a consequence, the smallest expectation value of ϵ_g over all possible positions $\mathbf{w}^* \in \mathcal{V}$ would be achieved by placing the student vector in the center of mass $\mathbf{w}_{cm}$ of the version space

$$\mathbf{w}_{cm} = \int_{\mathcal{V}} \mathbf{w}\ d^N w. \tag{3.54}$$

By definition, it has the smallest average (angular) distance from all other points in the set. Note that the center of mass $\mathbf{w}_{cm}$ of the normalized vectors in $\mathcal{V}$ itself is not normalized, but realizes the same classification as $\mathbf{w}_{cm}/|\mathbf{w}_{cm}|$.

In principle, the definition (3.54) immediately suggests how to determine $\mathbf{w}_{cm}$ for a given lin. sep. data set $\mathbb{D}$: We would have to determine many, random elements $\mathbf{w}^{(i)} \in \mathcal{V}$ independently and compute the simple empirical estimate [Wat93]

$$\mathbf{w}_{cm}^{(est)} = \frac{1}{M} \sum_{i=1}^{M} \mathbf{w}^{(i)}. \tag{3.55}$$

In practice, sampling the version space with uniform density is a non-trivial task. The theoretical background and practical strategies for how to achieve the optimal generalization ability when learning a linearly separable rule are discussed in e.g. [Wat93, OH91, Ruj97].

3.6 The perceptron of optimal stability

Here we approach the question (Q5) of Sec. 3.2 of how to choose a *good* or near optimal weight vector in version space from a different perspective. We avoid the explicit computation of the center of mass of $\mathcal{V}$ and consider a well-defined problem of quadratic optimization instead.

3.6.1 The stability criterion

The so-called stability of the perceptron has been established as a meaningful optimality criterion. We first consider the stability of a particular example, which is defined as

$$\kappa^\mu \;=\; \frac{E^\mu}{|\mathbf{w}|} \;=\; \frac{\mathbf{w}\cdot\boldsymbol{\xi}^\mu\, S_T^\mu}{|\mathbf{w}|}. \tag{3.56}$$

Due to the linearity of E^μ in $\mathbf{w}$, cf. Eq. (3.8), the quantity κ^μ is invariant under a rescaling of the form $\mathbf{w} \to \lambda\mathbf{w}$ ($\lambda > 0$). In terms of the geometric interpretation of linear separability, the scalar product of $\boldsymbol{\xi}^\mu S_T^\mu$ and $\mathbf{w}/|\mathbf{w}|$ measures the distance of the input vector from the separating hyperplane, see Fig. 3.17. More precisely κ^μ is an oriented distance: For $\kappa^\mu > 0$, the input vector is classified correctly by the perceptron, while for $\kappa^\mu < 0$ we have $\mathrm{sign}(\mathbf{w}\cdot\boldsymbol{\xi}^\mu) = -S_T^\mu$ and the input is located on the *wrong side* of the plane.

The stability (its absolute value) quantifies how robust the perceptron response would be against small variations of $\boldsymbol{\xi}^\mu$. Examples with a large distance from the hyperplane will hardly be taken to the *opposite side* by noise in the input channel.

We define the stability of the perceptron as the smallest of all κ^μ in $\mathbb{D}$:

$$\kappa(\mathbf{w}) \;=\; \min\,\{\kappa^\mu\}_{\mu=1}^{P}\,. \tag{3.57}$$

Note that if the perceptron does not separate the classes correctly, $\kappa(\mathbf{w}) < 0$ corresponds to the negative κ^μ with the largest absolute value. Positive stability $\kappa(\mathbf{w}) > 0$ indicates that $\mathbf{w}$ is a solution of the PSP and separates the classes correctly in $\mathbb{D}$. In this case, $\kappa(\mathbf{w})$ corresponds to the smallest distance of any example from the decision boundary. It quantifies the size of the gap between the two classes or, in other words, the classification margin of the perceptron.

In a linear separable problem it appears natural to select the perceptron weights which maximize $\kappa(\mathbf{w})$. In principle, the concept of stability extends to negative κ^μ according to Eq. (3.56). Therefore, we can also define the perceptron of optimal stability without requiring linear separability of the data set $\mathbb{D} = \{\boldsymbol{\xi}^\mu, S_T^\mu\}$ with more general targets $S_T(\boldsymbol{\xi}^\mu) = \pm 1$.

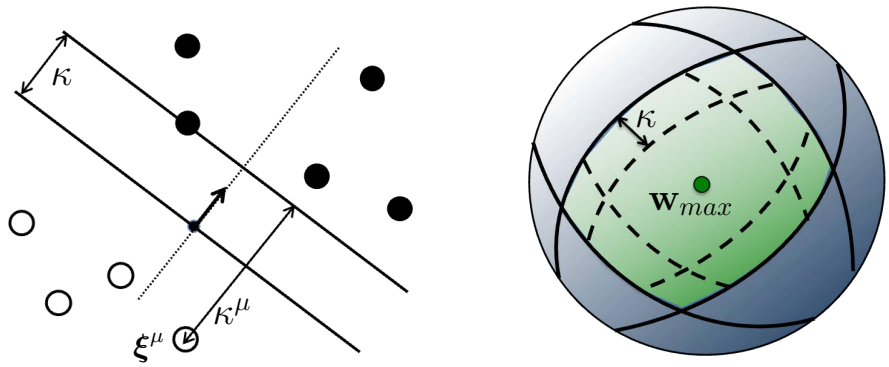


Figure 3.17: Stability of the perceptron. **Left:** The stability κ^μ, defined in Eq. (3.56), corresponds to the oriented distance of $\boldsymbol{\xi}^\mu$ from the plane orthogonal to $\mathbf{w}$. The stability of the perceptron $\kappa(\mathbf{w})$ is defined as the smallest κ^μ in the set of examples, i.e. $\kappa(\mathbf{w}) = \min_\mu\{\kappa^\mu\}$. Here, all inputs are classified correctly with $\kappa^\mu > 0$. **Right:** Restricting the student hypotheses to weight vectors with $\kappa(\mathbf{w}) > \kappa$ for a given data set, selects weight vectors $\mathbf{w}$ in the center region of the version space. The largest possible value of κ singles out the perceptron of optimal stability $\mathbf{w}_{max}$.

We define the perceptron of optimal stability (not very precisely termed the *optimal perceptron*, occasionally) as the target of the following problem:

PERCEPTRON OF OPTIMAL STABILITY (3.58)

For a given data set $\mathbb{D} = \{\boldsymbol{\xi}^\mu, S_T^\mu\}_{\mu=1}^P$, find the vector $\mathbf{w}_{max} \in \mathbb{R}^N$

with $\mathbf{w}_{max} = \underset{\mathbf{w}\in\mathbb{R}^N}{\text{argmax}}\ \kappa(\mathbf{w})$ for $\kappa(\mathbf{w}) = \min\left\{\kappa^\mu = \frac{\mathbf{w}\cdot\boldsymbol{\xi}^\mu S_T^\mu}{|\mathbf{w}|}\right\}_{\mu=1}^P$.

For linearly separable data, the search for $\mathbf{w}$ could be limited to the version space $\mathcal{V}$, formally. However, this restriction is non-trivial to realize [Ruj97] and would not constitute an advantage in practice. Moreover, for more general data sets of unknown separability, the version space might not even exist ($\mathcal{V} = \emptyset$) and $\kappa(\mathbf{w}_{max}) < 0$. We will discuss the usefulness of a corresponding solution $\mathbf{w}_{max}$ with negative stability $\kappa_{max} < 0$ later and focus on linearly separable problems in the following.

The perceptron of optimal stability $\mathbf{w}_{max}$ does not exactly coincide with $\mathbf{w}_{cm}$, cf. Sec. 3.5.4, in general. The "center" as defined by the "maximum possible distance from all boundaries" is identical with the center of mass only if $\mathcal{V}$ has a regular, symmetric shape. However, one can expect that $\mathbf{w}_{max}$ is in general quite close to the true center of mass and could serve as an approximative realization.

As a consequence, the perceptron of optimal stability should display favorable (near optimal) generalization behavior when trained from a given reliable, lin. sep. data set. In fact, the difference appears marginal from a practical perspective. For a theoretical comparison of optimal stability and optimal generalization in the perceptron see [Wat93].

3.6.2 The MinOver algorithm

An intuitive algorithm that can be shown to achieve optimal stability for a given lin. sep. data set $\mathbb{D}$ has been suggested in [KM87]. The so-called MinOver algorithm performs Hebbian updates for the currently least stable example in $\mathbb{D}$. We assume here *tabula rasa* initialization, i.e. $\mathbf{w}(0) = 0$, but more general initial states could be considered.

The prescription always aims at improving the least stable example. Note that for a given weight vector $\mathbf{w}(t)$, the minimal local potential coincides with minimum stability since $\kappa^\mu(t) = E^\mu(t)/|\mathbf{w}(t)|$.

The MinOver update is defined as follows:

MinOver algorithm

at discrete time step t with current $\mathbf{w}(t)$
- compute the local potentials $E^\mu(t) = \mathbf{w}(t) \cdot \boldsymbol{\xi}^\mu \, S_T^\mu$ for all examples in $\mathbb{D}$
- determine the index $\widehat{\mu}$ of the training example with *minimal overlap*, i.e. with the currently lowest local potential: $E^{\widehat{\mu}} = \min \{E^\mu(t)\}_{\mu=1}^P$
- update the weight vector according to

$$\mathbf{w}(t+1) \;=\; \mathbf{w}(t) + \frac{1}{N} \, \boldsymbol{\xi}^{\widehat{\mu}} \, S_T^{\widehat{\mu}} \tag{3.59}$$

[or, equivalently, increment the corresponding embedding strength

$$x^{\widehat{\mu}}(t+1) = x^{\widehat{\mu}}(t) + 1. \quad] \tag{3.60}$$

A few remarks:

- According to the original presentation of the algorithm [KM87], the Hebbian update is only performed if the currently smallest local potential also satisfies $E^{\nu(t)} \leq c$ for a given $c > 0$. This is reminiscent of *learning from mistakes* as in the Rosenblatt algorithm (3.3.3). However, as pointed out in [KM87], optimal stability is only achieved in the limit $c \to \infty$, which is equivalent to (3.59).

- In the above formulation (3.59), MinOver updates the weights (or embedding strengths) even if all examples in $\mathbb{D}$ are classified correctly already. As the algorithm keeps performing non-zero Hebbian updates, the temporal change of the weight vector $\mathbf{w}(t)$ itself does not constitute a reasonable

stopping criterion. Instead, one of the following quantities could be considered:

- the angular change $\angle\big(\mathbf{w}(t), \mathbf{w}(t+T)\big) = \frac{1}{\pi} \arccos\left(\frac{\mathbf{w}(t)\cdot\mathbf{w}(t+T)}{|\mathbf{w}(t)||\mathbf{w}(t+T)|}\right)$

  or the argument of the arccos, for simplicity.
- the total change of stabilities $\sum_{\mu=1}^{P}\Big[\kappa^\mu(t) - \kappa^\mu(t+T)\Big]^2$,

for example. For these and similar criteria, reasonably large numbers of training steps T should be performed, e.g. with $T \propto P$, in order to allow for noticeable differences. In both criteria, changes of only the norm $|\mathbf{w}(t)|$ are disregarded because they do not affect the classification or its stability.

- From the definition of the MinOver algorithm (3.59) we see that it can only yield non-negative, integer embedding strengths when the initialization is tabula rasa. This feature of $\mathbf{w}_{max}$ will be encountered again in the following section, together with several other properties of optimal stability.

- As proven in [KM87], the MinOver algorithm converges and yields the perceptron weights of optimal stability, if $\mathbb{D}$ is linearly separable. The proof of convergence is similar in spirit to that of the Perceptron Convergence Theorem (3.22). We refrain from reproducing it here. Instead, we show only that the perceptron of optimal stability can always be written in the form

$$\mathbf{w}_{max} = \frac{1}{N}\sum_{\mu=1}^{P} x^\mu_{max}\,\boldsymbol{\xi}^\mu\,S^\mu_T \quad \text{with embedding strengths} \quad \{x^\mu_{\max} \in \mathbb{R}\}_{\mu=1}^{P}\,. \tag{3.61}$$

The existence of embedding strengths x_{max} will be recovered *en passant* in the next section. However, it is instructive to prove the statement explicitly here. To this end, we consider two perceptron weight vectors: the first one is assumed to be given as a linear combination of the familiar form

$$\mathbf{w}_1 = \sum_{\mu=1}^{P} x^\mu_1 \boldsymbol{\xi}^\mu\,S^\mu_T \quad \text{with embedding strengths} \quad \{x^\mu_1\}_{\mu=1}^{P}\,, \tag{3.62}$$

while for the second weight vector we assume that

$$\mathbf{w}_2 = \mathbf{w}_1 + \boldsymbol{\delta} \quad \text{with} \quad |\boldsymbol{\delta}| > 0 \quad \text{and} \quad \boldsymbol{\delta}\cdot\boldsymbol{\xi}^\mu = 0 \quad \text{for all } \mu = 1, 2, \ldots, P. \tag{3.63}$$

Hence, $\mathbf{w}_2$ cannot be written in terms of embedding strengths as it contains contributions which are orthogonal to all input vectors in $\mathbb{D}$.

If we consider the local potentials with respect to $\mathbf{w}_2$, we observe that

$$E^\mu_2 = \mathbf{w}_2\cdot\boldsymbol{\xi}^\mu S^\mu_T = \mathbf{w}_1\cdot\boldsymbol{\xi}^\mu S^\mu_T + \underbrace{\boldsymbol{\delta}\cdot\boldsymbol{\xi}^\mu S^\mu_T}_{=0} = E^\mu_1. \tag{3.64}$$

On the other hand we have

$$|\mathbf{w}_2|^2 = |\mathbf{w}_1 + \delta|^2 = |\mathbf{w}_1|^2 + 2\underbrace{\mathbf{w}_1 \cdot \boldsymbol{\delta}}_{=0} + \underbrace{|\boldsymbol{\delta}|^2}_{>0} \quad \Rightarrow \quad |\mathbf{w}_2| > |\mathbf{w}_1|, \tag{3.65}$$

where the mixed term vanishes because $\mathbf{w}_1$ is a linear combination of the $\boldsymbol{\xi}^\mu \perp \boldsymbol{\delta}$. As a consequence, we observe that

$$\kappa_2^\mu = \frac{E_2^\mu}{|\mathbf{w}_2|} = \frac{E_1^\mu}{|\mathbf{w}_2|} < \frac{E_1^\mu}{|\mathbf{w}_1|} = \kappa_1^\mu \text{ for all } \mu, \text{ and therefore } \kappa_2 < \kappa_1. \tag{3.66}$$

We conclude that any contribution orthogonal to all $\boldsymbol{\xi}^\mu$ inevitably reduces the stability of a given weight vector. This implies that maximum stability is indeed achieved by weights of the form (3.61). The result also implies that the framework of iterative Hebbian learning is sufficient to find the solution.

Note that a non-zero $\boldsymbol{\delta}$ in Eq. (3.63) cannot exist for $P > N$, in general. Obviously, if $span\left(\{\boldsymbol{\xi}^\mu\}_{\mu=1}^P\right) = \mathbb{R}^N$, any N-dimensional vector including $\mathbf{w}_{max}$ can be written as a linear combination. In this case, Eq. (3.61) is trivially true.

Our simple consideration does not yield restrictions on the possible values that the embedding strengths can assume. However, the proven convergence of the MinOver algorithm implies that the perceptron of optimal stability can always be written in terms of non-negative $x_{max}^\mu \geq 0$. We will recover this result more formally in the next section.

3.7 Optimal stability by quadratic optimization

Here we will exploit the fact that optimal stability can be formulated as a problem of constrained quadratic optimization. Consequently, a wealth of theoretical results and techniques from optimization theory becomes available [Fle00, PAH19]. They provide deeper insight into the structure of the problem and allow for the identification of efficient training algorithms, as we will exemplify in terms of the so-called AdaTron algorithm [AB89,BAK91] Before deriving and discussing this training scheme we re-visit another closely related, prototypical training scheme from the early days of neural network models: B. Widrow and M.E. Hoff's *Adaptive Linear Neuron* or Adaline [WH60, WL90].

3.7.1 Optimal stability reformulated

For a given lin. sep. $\mathbb{D} = \{\boldsymbol{\xi}^\mu, S_T^\mu\}_{\mu=1}^P$, the perceptron of optimal stability corresponds to the solution of the following problem:

$$\underset{\mathbf{w} \in \mathbb{R}^N}{\text{maximize}} \quad \kappa(\mathbf{w}) \text{ where } \kappa(\mathbf{w}) = \min\left\{\kappa^\mu = \frac{E^\mu}{|\mathbf{w}|} = \frac{\mathbf{w}^\top \boldsymbol{\xi}^\mu S_T^\mu}{|\mathbf{w}|}\right\}_{\mu=1}^P \tag{3.67}$$

which is just a more compact version of (3.58).

Obviously, the stability κ can be made larger by increasing the minimal E^μ for constant norm $|\mathbf{w}|$. Analogously, κ increases with decreasing norm $|\mathbf{w}|$ if all local potentials obey the constraint $E^\mu \geq c > 0$. As discussed previously, the actual choice of the constant c is irrelevant because E^μ is linear in $\mathbf{w}$ and we can set $c = 1$ without loss of generality. This allows us to re-formulate the problem as follows:

$$\underset{\mathbf{w} \in \mathbb{R}^N}{\text{minimize}} \quad \frac{N}{2}\,\mathbf{w}^2 \quad \text{subject to inequality constraints} \quad \{E^\mu \geq 1\}_{\mu=1}^{P}\,. \tag{3.68}$$

Hence, we have rewritten the problem of maximal stability as the optimization of the quadratic cost function $N\mathbf{w}^2/2$ under linear inequality constraints of the form $E^\mu = \mathbf{w}^\top \boldsymbol{\xi}^\mu S_T^\mu \geq 1$. The solution $\mathbf{w}_{max}$ then displays the (optimal) stability $\kappa_{max} = 1 \Big/ |\mathbf{w}_{max}|$. Note that the pre-factor $N/2$ in (3.68) is irrelevant for the definition of the problem but is kept for convenience and consistency of notation.

3.7.2 The Adaptive Linear Neuron - Adaline

The problem of optimal stability in the formulation (3.68) involves a system of inequalities. Before we address its solution in the following sections, we resort to the more familiar case of constraints given by equations, i.e. we consider the simpler optimization problem

$$\underset{\mathbf{w} \in \mathbb{R}^N}{\text{minimize}} \quad \frac{N}{2}\,\mathbf{w}^2 \quad \text{subject to constraints} \quad \{E^\mu = 1\}_{\mu=1}^{P}\,. \tag{3.69}$$

This, in fact, has the form of a standard optimization problem with non-linear (here: quadratic) cost function and a set of (here: linear) constraints. A brief discussion of this type of problem is given in Appendix A.3.1.

Historically, the problem (3.69) relates to the the so-called *Adaptive Linear Neuron* or Adaline model which was introduced by B. Widrow and M.E. Hoff in 1960 [WH60]. Like the Rosenblatt Perceptron, it constitutes one of the earliest artificial neural network models and truly groundbreaking work in the area of machine learning. A series of very instructive and entertaining videos about B. Widrow's pioneering work is available at [Wid12].

Widrow realized Adaline systems in hardware as "resistors with memory" and introduced the term *Memistor* [Wid60].[11] The concept was also extended to layered Madaline (Many Adaline) networks consisting of several linear units which were combined in a majority vote for classification [WL90, Wid12].

For our purposes, the Adaline can be interpreted as a single layer perceptron which differs from Rosenblatt's model only in terms of the training procedure.

[11] Not to be confused with the more recent concept of *Memristor* elements [Chu71].

The Adaline framework essentially treats the problem of binary classification as a linear regression with subsequent thresholding of the continuous $\mathbf{w} \cdot \boldsymbol{\xi}^\mu$.

We will take a rather formal perspective based on the theory of Lagrange multipliers [Fle00, PAH19] as outlined in Appendix A.3.1. The P linear constraints of the form $E^\mu = \mathbf{w}^\top \boldsymbol{\xi}^\mu S_T^\mu = 1$ can be incorporated in the corresponding Lagrange function

$$\mathcal{L}\Big(\mathbf{w}, \{\lambda^\mu\}_{\mu=1}^P\Big) = \frac{N}{2}\mathbf{w}^2 \; - \; \sum_{\mu=1}^P \lambda^\mu \left(\mathbf{w}^\top \boldsymbol{\xi}^\mu S_T^\mu - 1\right). \tag{3.70}$$

With the gradient $\nabla_w = (\partial/\partial w_1, \partial/\partial w_2, ..., \partial/\partial w_N)^\top$ in weight space we obtain $\nabla_w \mathcal{L} = N\mathbf{w} – \sum_\mu \lambda^\mu \boldsymbol{\xi}^\mu S_T^\mu$. Furthermore, $\frac{\partial}{\partial \lambda^\nu}\Big[\sum_\mu \lambda^\mu (E^\mu – 1)\Big] = (E^\nu – 1)$. Hence, the first order stationarity conditions for a solution $\mathbf{w}^*, \{\lambda^{*\mu}\}$ of problem (3.69) read

$$\nabla_w \mathcal{L}|_* = 0 \quad \Rightarrow \quad \mathbf{w}^* = \frac{1}{N}\sum_{\mu=1}^P \lambda^{*\mu}\, \boldsymbol{\xi}^\mu S_T^\mu \tag{3.71}$$

$$\text{and} \quad \left.\frac{\partial \mathcal{L}}{\lambda^\mu}\right|_* = 0 \quad \Rightarrow \quad E^{*\mu} = \mathbf{w}^{*\top} \boldsymbol{\xi}^\mu S_T^\mu = 1 \;\; \text{for all } \mu \tag{3.72}$$

where the shorthand $(\ldots)|_*$ stands for the evaluation of $(\ldots)$ in $\mathbf{w} = \mathbf{w}^*$ and $\lambda^\mu = \lambda^{*\mu}$ for all μ. Note that sufficient (second oder) conditions for a solution of the problem are non-trivial to work out.

The second condition (3.72) merely reproduces the original equality constraints, which have to be satisfied by any candidate solution, obviously. The first, more interesting stationarity condition (3.71) implies that the formally introduced Lagrange parameters can be identified as the embedding strengths x^μ and can be renamed accordingly. Hence, we can eliminate the weights from $\mathcal{L}$ and obtain

$$\begin{aligned}\mathcal{L}\big(\{x^\mu\}_{\mu=1}^P\big) &= \frac{1}{2N}\sum_{\mu,\nu} x^\mu S_T^\mu \boldsymbol{\xi}^{\mu\top} \boldsymbol{\xi}^\nu S_T^\nu x^\nu - \sum_\mu x^\mu \left[\frac{1}{N}\sum_\nu x^\nu S_T^\nu \boldsymbol{\xi}^\nu\right]^\top \boldsymbol{\xi}^\mu S_T^\mu + \sum_\mu x^\mu \\ &= -\frac{1}{2N}\sum_{\nu,\mu=1}^P x^\mu S_T^\mu \boldsymbol{\xi}^{\mu\top}\, \boldsymbol{\xi}^\nu S_T^\nu x^\nu + \sum_{\mu=1}^P x^\mu. \end{aligned} \tag{3.73}$$

It turns out useful to resort to a compact notation which again exploits that the weights are of the form $\mathbf{w} = \frac{1}{N}\sum_\mu x^\mu \boldsymbol{\xi}^\mu S_T^\mu$. We introduce the symmetric correlation matrix

$$C = C^\top \in \mathbb{R}^{P\times P} \;\; \text{with elements} \;\; C^{\mu\nu} = \frac{1}{N} S_T^\mu S_T^\nu\, \boldsymbol{\xi}^{\mu\top} \boldsymbol{\xi}^\nu \tag{3.74}$$

and define the P-dimensional vectors $\vec{x} = (x^1, x^2, ...x^P)^\top$, $\vec{E} = (E^1, E^2, ...E^P)^\top$ as well as the formal $\vec{1} = (1, 1, ..., 1)^\top \in \mathbb{R}^P$, yielding, e.g., $\vec{x}^\top \vec{1} = \sum_\mu x^\mu$.

In addition, we use the notation $\vec{a} > \vec{b}$, which is popular in the optimization related literature and indicates that $a^\mu > b^\mu$ for all $\mu = 1, 2, ...P$. Analogous notations are employed for component-wise relations "$<$", "$\geq$" and "$\leq$".

In the convenient matrix-vector notation, we have furthermore

$$E^\nu = \frac{1}{N}\sum_{\mu=1}^{P} x^\mu \underbrace{S_T^\mu S_T^\nu \, \boldsymbol{\xi}^{\mu\top}\boldsymbol{\xi}^\nu}_{NC^{\mu\nu}} = [C\vec{x}]^\nu \quad \text{i.e } \vec{E} = C\vec{x} \tag{3.75}$$

$$\mathbf{w}^2 = \frac{1}{N^2}\sum_{\mu,\nu} x^\mu x^\nu \overbrace{S_T^\mu S_T^\nu \boldsymbol{\xi}^{\mu\top}\boldsymbol{\xi}^\nu} = \frac{1}{N}\vec{x}^\top C\,\vec{x} = \frac{1}{N}\vec{x}^\top\vec{E}. \tag{3.76}$$

In this notation, the Lagrange function (3.73) becomes $\mathcal{L}(\vec{x}) = -1/2\ \vec{x}^\top C\,\vec{x} + \vec{x}^\top\vec{1}$, which has to be maximized (!) with respect to $\vec{x}$, see the discussion of the Wolfe Dual in [Fle00, PAH19] and Appendix A.3.4. Consequently we can re-formulate (3.69) as the following unconstrained problem:

$$\underset{\vec{x}\,\in\,\mathbb{R}^P}{\text{maximize}} \quad f(\vec{x}) = -\frac{1}{2}\vec{x}^\top C\vec{x} + \vec{x}^\top\vec{1}. \tag{3.77}$$

Compared to (3.69), the cost function appears slightly more complicated. In turn, however, the constraints are eliminated. While completely equivalent with (3.69), the re-written problem is given in terms of the embedding strengths $\vec{x} \in I\!R^P$ only.

Parallel Adaline algorithm

In absence of constraints, it is straightforward to maximize $f(\vec{x})$ and we could employ a variety of methods, in practice. For instance, we can resort to simple gradient ascent, cf. Appendix A.4, with tabula rasa initialization $\vec{x}(0) = 0$. We can also identify an equivalent update in terms of weights with initial $\mathbf{w}(0) = 0$ by exploiting the relation $\mathbf{w}(t) = \frac{1}{N}\sum_{\mu=1}^{P} x^\mu(t)\,\boldsymbol{\xi}^\mu S_T^\mu$:

ADALINE algorithm, parallel updates

$$\vec{x}(t+1) = \vec{x}(t) + \eta\nabla_x f = \vec{x}(t) + \eta\left(\vec{1} - \vec{E}(t)\right) \tag{3.78}$$

$$\left[\text{ or } \quad \mathbf{w}(t+1) = \mathbf{w}(t) + \frac{\eta}{N}\sum_{\mu=1}^{P}\Big(1 - E^\mu(t)\Big)\boldsymbol{\xi}^\mu S_T^\mu \ \right], \tag{3.79}$$

where $E^\mu(t) = [C\vec{x}(t)]^\mu = \mathbf{w}(t)^\top\boldsymbol{\xi}^\mu S_T^\mu$. The learning rate η controls the magnitude of the update steps.

This version of the Adaline algorithm corresponds to standard, so-called batch gradient descent in the space of embedding strengths $\vec{x}$. Hence, η can be finite but has to be small enough enough to ensure convergence. The precise condition depends on the mathematical properties of the extremum, see Appendix A.4 for a general discussion.

Sequential Adaline algorithm

As an important alternative to the above batch or parallel update, sequential gradient-based methods can be devised, which present the example data repeatedly in, for instance, deterministic order and update a single embedding strength in each step. Intuitively, we would want to increase an embedding strength of any example with $E^\mu < 1$ and we expect the magnitude of the updated x^μ to increase with $(1 - E^\mu)$.

The sequential Adaline algorithm corresponds to *coordinate ascent* in terms of $\vec{x}$, a variant of gradient based optimization which is briefly discussed in Appendix A.5.1:

ADALINE algorithm, sequential updates (repeated presentation of $\mathbb{D}$)
– at time step t, present example $\mu(t) = 1, 2, 3, ..., P, 1, 2, 3, ...$
– update (only) the corresponding embedding strength

$$x^{\mu(t)}(t+1) \;=\; x^{\mu}(t) \,+\, \widetilde{\eta}\left(1 - E^{\mu(t)}(t)\right) \tag{3.80}$$

$$\left[\text{ or } \qquad \mathbf{w}(t+1) \;=\; \mathbf{w}(t) + \frac{\widetilde{\eta}}{N}\left(1 - E^{\mu(t)}(t)\right)\boldsymbol{\xi}^{\mu(t)} S_T^{\mu(t)} \right]. \tag{3.81}$$

Convergence of the Adaline

In the following we sketch the proof of convergence for the sequential Adaline algorithm (3.80) and obtain the corresponding condition on the learning rate $\widetilde{\eta}$. For the update of the current component x^μ we have

$$x^\mu \to x^\mu + \delta^\mu \quad \text{with} \;\; \delta^\mu \,=\, \widetilde{\eta}\,(1 - [C\vec{x}]^\mu) \,=\, \widetilde{\eta}\,(1 - E^\mu) \tag{3.82}$$

while all other embedding strengths remain unchanged: $\vec{\delta} = (0, \ldots, \delta^\mu, 0, \ldots 0)^\top$. Therefore, the change of the objective function under update (3.82) is

$$\begin{aligned} f(\vec{x} + \vec{\delta}) - f(\vec{x}) \;&=\; -\frac{1}{2}\,\vec{\delta}^{\top} C\vec{\delta} - \vec{\delta}^{\top} C\vec{x} + \vec{\delta}^{\top}\,\vec{1} \,=\, -\frac{1}{2}(\delta^\mu)^2\, C^{\mu\mu} \,+\, \delta^\mu\,(1 - E^\mu) \\ &=\; \widetilde{\eta}(1 - E^\mu)^2 \left[1 - \frac{1}{2}\,\widetilde{\eta}\, C^{\mu\mu}\right] \geq 0 \quad \text{for} \;\; \widetilde{\eta} < \frac{2}{C^{\mu\mu}}. \end{aligned} \tag{3.83}$$

Unless a solution $\vec{x}$ with $E^\mu = 1$ for all μ has been reached, the objective function will strictly increase in at least some of the individual steps of the sequential procedure as long as $\widetilde{\eta} < 2/C^{\mu\mu}$.

The diagonal elements of C are proportional to the norms of the corresponding feature vectors, $C^{\mu\mu} = (\boldsymbol{\xi}^\mu)^2/N$, which are of course available for any given data set. If we want to use a single, constant value of $\widetilde{\eta}$ and guarantee monotonic increase of f under the iteration, the learning rate has to satisfy

$$0 \,<\, \widetilde{\eta} \,<\, \frac{2}{\max\{C^{\mu\mu}\}_{\mu=1}^{P}}. \tag{3.84}$$

One can also show that the objective function is bounded from above if C is positive definite and $C\vec{x} = \vec{1}$ has a solution: while the linear term $\vec{1}^\top \vec{x}$ in f, Eq. (3.77), can grow in an unbounded way, the negative quadratic term will always dominate and limit the increase.

Hence, unless a solution with $C\vec{x} = \vec{E} = \vec{1}$ exists and has been found, the iteration performs at least one non-zero update in each sweep through the data set and f increases monotonically. Moreover, the cost function is bounded from above. Therefore, the coordinate-ascent-like version (3.80) of the Adaline converges and maximizes f under constraints $\vec{E} = \vec{1}$, if the learning rate satisfies (3.84).

Relation to linear regression and the SSE

Several interesting relations to other problems and algorithms are noteworthy: As pointed out in [DO87], the coordinate ascent (3.82) is closely related to the well-known Gauß-Seidel and Gauß-Jacobi methods [Fle00] for the iterative solution of the system of linear equations $C\vec{x} = \vec{1}$.

For non-singular C, the problem could be solved directly by $\vec{x} = C^{-1}\vec{1}$, which gives

$$x^\mu = [\, C^{-1}\vec{1}\,]^\mu = \sum_\nu [C^{-1}]^{\mu\nu}\, 1 \quad \text{and} \quad \mathbf{w} = \frac{1}{N} \sum_{\mu,\nu} [C^{-1}]^{\mu\nu} \boldsymbol{\xi}^\mu S_T^\mu \tag{3.85}$$

in weight space. If we incorporate the outputs $S_T^\mu = \pm 1$ in the modified input vectors $\widetilde{\boldsymbol{\xi}}^\mu = S_T^\mu \boldsymbol{\xi}^\mu$ the problem becomes equivalent to regression with the particular targets $y^\mu = (S_T^\mu)^2 = 1$ for all data points. Hence, the Adaline problem has the same mathematical structure as linear regression considered in Sec. 2.2.2 and Appendix A.2.2, which immediately links Eq. (3.85) to the pseudoinverse solutions (2.8) and (A.24), respectively.

It is interesting to note that although (3.79) is the weight space equivalent of (3.78), it cannot be interpreted as a gradient ascent along $\nabla_w f$ of the cost function (3.77). As outlined in Appendix A.4.3, even linear transformations of variables do not preserve the gradient property, in general.

However, (3.79, 3.81) can be seen as gradient descent in weight space with respect to a different, yet related cost function: the Sum of Squared Errors (SSE), cf. Eq. (2.5), for linear regression with target values $S_T^\mu = \pm 1$:

$$E^{SSE} = \frac{1}{2} \sum_{\mu=1}^{P} (1 - E^\mu)^2 = \frac{1}{2} \sum_{\mu=1}^{P} \left(S_T^\mu - \mathbf{w}^\top \boldsymbol{\xi}^\mu \right)^2 \tag{3.86}$$

$$\text{with } \nabla_w E^{SSE} = -\sum_{\mu=1}^{P} (1 - E^\mu) \boldsymbol{\xi}^\mu S_T^\mu. \tag{3.87}$$

Note that writing E^{SSE} in terms of embedding strengths would involve the quadratic form $\vec{x}^\top C^\top C \vec{x}$ which is not present in (3.77).

This consideration also indicates what the behavior of the Adaline will be if $\vec{E} = 1$ cannot be satisfied: the algorithm finds an approximate solution by minimizing the SSE.

The sequential algorithm (3.81) in weight space is equivalent to Widrow and Hoff's original LMS (Least Mean Square) method for the Adaline [WH60, Wid12]. The LMS, also referred to as the *delta-rule* or the *Widrow-Hoff algorithm* in the literature, can be seen as the most important ancestor of the prominent *Backpropagation of Error* for multilayered neural networks, cf. Sec. A.4 and Chapter 5. A review of the history and conceptual relations between these classical algorithms is given in [WL90].

3.7.3 The Adaptive Perceptron Algorithm - AdaTron

In Rosenblatt's perceptron algorithm an update is performed whenever the presented example is misclassified. All non-zero Hebbian learning steps are performed with the same magnitude, independent of the example's distance from the current decision plane. On the contrary, the Adaline learning rule is *adaptive* in the sense that the magnitude of the update depends on the actual deviation of E^μ from the target value 1. While this appears to make sense and is expected to speed up learning, it also results in negative Hebbian updates *("unlearning")* with $\eta(1 - E^\mu) < 0$ if the example is correctly classified with large $E^\mu > 1$, already. In fact, Adaline can yield negative embedding strengths $x^\mu < 0$ in order enforce $E^\mu = 1$ when the example otherwise could have $E^\mu > 1$.

This somewhat counter-intuitive feature of the Adaline is accounted for in the Adaptive Perceptron (AdaTron) algorithm [AB89, BAK91]. It retains the concept of adaptivity but takes into account the inequality constraints in problem (3.68), explicitly. In essence, it performs Adaline-like updates, but prohibits the embedding strengths from assuming negative values. As we will see, this relatively simple modification yields the perceptron of optimal stability for linearly separable data.

The method of Lagrange multipliers has been extended to the treatment of inequality constraints [Fle00, PAH19], see also Appendix A. Therefore, we can make use of several well-established results from optimization theory in the following. We return to the problem of optimal stability in the form (3.68), which we repeat here for the sake of clarity:

PERCEPTRON OF OPTIMAL STABILITY (weight space)

$$\underset{\mathbf{w}\,\in\,\mathbb{R}^N}{\text{minimize}} \quad \frac{N}{2}\,\mathbf{w}^2 \quad \text{subject to inequality constraints} \quad \{E^\mu \geq 1\}_{\mu=1}^{P}\,.$$

Formally, we have to consider the same Lagrange function as for equality constraints, already given in Eq. (3.70):

$$\mathcal{L}\left(\mathbf{w}, \{\lambda^\mu\}_{\mu=1}^{P}\right) = \frac{N}{2}\mathbf{w}^2 \; - \; \sum_{\mu=1}^{P} \lambda^\mu \left(\mathbf{w}^\top \boldsymbol{\xi}^\mu S_T^\mu - 1\right). \tag{3.88}$$

The Kuhn-Tucker Theorem of optimization theory, see [Fle00, PAH19], provides the first order necessary stationarity conditions for general, non-linear optimization problems with inequality and/or equality conditions. They are known as the so-called Kuhn-Tucker (KT) or Karush-Kuhn-Tucker conditions [Fle00, PAH19], see (A.26) in the Appendix. In our specific case, as worked out as an example in the Appendix, they read

KUHN-TUCKER CONDITIONS (optimal stability)

$$\mathbf{w}^* = \frac{1}{N}\sum_{\mu=1}^{P} \lambda^{*\mu}\, \boldsymbol{\xi}^\mu S_T^\mu \qquad \text{(embedding strengths } \lambda^\mu\text{)} \quad (3.89)$$

$$E^{*\mu} = \mathbf{w}^{*\top}\boldsymbol{\xi}^\nu S_T^\mu \geq 1 \;\text{ for all } \mu \qquad \text{(linear separability)} \quad (3.90)$$

$$\lambda^{*\mu} \geq 0 \;\; (\text{not all } \lambda^{*\mu} = 0) \qquad \text{(non-negative multipliers)} \quad (3.91)$$

$$\lambda^{*\mu}\,(E^{*\mu} - 1) = 0 \;\text{ for all } \mu \qquad \text{(complementarity).} \quad (3.92)$$

The first condition is the same as for equality constraints and follows directly from $\nabla_w \mathcal{L} = 0$. As in the case of the Adaline, it implies that the solution of the problem can be written in the familiar form (3.61). Moreover, the Lagrange multipliers λ^μ play the role of the embedding strengths and we can set $\lambda^\mu = x^\mu$ in the following considerations. The second condition (3.90) merely reflects the original constraint, i.e. linear separability.

Intuitively, the non-negativity of the multipliers, KT-condition (3.91), reflects the fact that an inequality constraint is only *active* on one side of the hyperplane defined by $E^\mu = 1$. As long as $E^\mu > 1$, the corresponding multiplier could be set to $\lambda^\mu = 0$ as the constraint is satisfied in the entire half-space. Since the solution can be written with $x^\mu = \lambda^\mu$ due to (3.89), we formally recover the insight from Sec. 3.6.2 which indicates that $\mathbf{w}_{max}$ can always be represented by non-negative embedding strengths.

After renaming the Lagrange multipliers λ^μ to x^μ, we refer to a solution $\vec{x}^*$ of the problem (3.68) as a KT-point. Using our convenient matrix-vector notation, the KT conditions of the simplified problem read

KUHN-TUCKER CONDITIONS (optimal stability, simplified)

$$\vec{E}^* = C\vec{x}^* \geq \vec{1} \qquad \text{(linear separability)} \quad (3.93)$$

$$\vec{x}^* \geq 0 \quad (\vec{x}^* \neq 0) \qquad \text{(non-negative embeddings)}^{12} \quad (3.94)$$

$$x^{*\mu}\,(E^{*\mu} - 1) = 0 \;\text{ for all } \mu \qquad \text{(complementarity).} \quad (3.95)$$

The most interesting condition is that of complementarity (3.92) and (3.95). It states that at optimal stability, any example with non-zero embedding will

[12] Obviously $\vec{x} = 0, \mathbf{w} = 0$ cannot be a solution.

have $E^{*\mu} = 1$ while input/output pairs with $E^{*\mu} > 1$ do not contribute to the linear combination (3.61). We will discuss this property in greater detail later. Complementarity also implies that $x^{*\mu} E^{*\mu} = x^{*\mu}$ for all μ and, therefore

$$\vec{x}^{*\top} C \vec{x}^* = \vec{x}^{*\top} \vec{E}^* = \sum_{\mu=1}^{P} x^{*\mu} E^{*\mu} = \sum_{\mu=1}^{P} x^{*\mu} = \vec{x}^{*\top} \vec{1}. \tag{3.96}$$

Now consider two potentially different KT-points $\vec{x}_1$ and $\vec{x}_2$, both satisfying the first order stationarity conditions. By definition, the matrix C is symmetric and positive semi-definite:

$$\vec{u}^{\top} C \, \vec{u} \propto \sum_{\mu,\nu} u^{\mu} S_T^{\mu} \boldsymbol{\xi}^{\mu} \cdot \boldsymbol{\xi}^{\nu} S_T^{\nu} u^{\nu} \; = \; \left(\sum_{\mu} u^{\mu} \boldsymbol{\xi}^{\mu} S_T^{\mu} \right)^2 \geq 0 \;\; \text{for all} \;\; \vec{u} \in \mathbb{R}^P.$$

Setting $\vec{u} = \vec{x}_1 - \vec{x}_2$ we obtain with (3.96):

$$\begin{aligned} 0 \;\; \leq \;\; (\vec{x}_1 - \vec{x}_2)^{\top} C (\vec{x}_1 - \vec{x}_2) \; &= \; \underbrace{\vec{x}_1^{\top} C \vec{x}_1}_{= \vec{x}_1^{\top} \vec{1}} + \underbrace{\vec{x}_2^{\top} C \vec{x}_2}_{= \vec{x}_2^{\top} \vec{1}} - \vec{x}_1^{\top} C \vec{x}_2 - \vec{x}_2^{\top} C \vec{x}_1 \\ &= \vec{x}_1^{\top} \left(\vec{1} - C \vec{x}_2 \right) + \vec{x}_2^{\top} \left(\vec{1} - C \vec{x}_1 \right). \end{aligned}$$

All components of the KT-points $\vec{x}_{1,2}$ are non-negative, while all components of vectors $(\vec{1} - C\vec{x}_{1,2}) \leq 0$ due to the linear separability condition. Therefore, we obtain $0 \leq (\vec{x}_1 - \vec{x}_2)^{\top} C (\vec{x}_1 - \vec{x}_2) \leq 0$ and hence:

$$\begin{aligned} \Rightarrow \; 0 \;\; &= \;\; (\vec{x}_1 - \vec{x}_2)^{\top} C (\vec{x}_1 - \vec{x}_2) \quad \propto \sum_{\mu,\nu} (x_1^{\mu} - x_2^{\mu}) S_T^{\mu} \boldsymbol{\xi}^{\mu} \cdot \boldsymbol{\xi}^{\nu} S_T^{\nu} (x_1^{\nu} - x_2^{\nu}) \\ &= \;\; \left[\sum_{\mu} (x_1^{\mu} - x_2^{\mu}) \, \boldsymbol{\xi}^{\mu} S_T^{\mu} \right]^2 \propto \left[\mathbf{w}_1 - \mathbf{w}_2 \right]^2 \quad \Rightarrow \mathbf{w}_1 = \mathbf{w}_2. \end{aligned}$$

We conclude that any two KT-points define the same weight vector, which corresponds to the perceptron of optimal stability. Even if $\vec{x}_1 \neq \vec{x}_2$, which is possible in spite of $C\vec{x}_1 = C\vec{x}_2$ for singular matrices C, the solution is unique in terms of the weights. Moreover, this finding implies that any local solution (KT-point) of the problem (3.68) is indeed a global solution. In contrast to many other cost function based learning algorithms, local minima do not play a role in optimal stability. This is a special case of a more general result which applies to all convex optimization problems [Fle00, PAH19].

The stationarity conditions also facilitate a re-formulation of the optimization problem. Similar to the simplification of the Adaline problem, it essentially amounts to the elimination of the weights and the interpretation of Lagrange multipliers as embedding strengths. It constitutes a special case of what is known as the Wolfe Dual in optimization theory, see [Fle00] for the general definition and proof of the corresponding Duality Theorem. A brief introduction is given in Appendix A.3.4. Duality is frequently employed to rewrite a given problem in terms of a modified cost function with simplified constraints.

As outlined in Appendix A.3.4 the following formulation is equivalent to problem (3.68):

PERCEPTRON OF OPTIMAL STABILITY (embedding strengths, dual problem)

$$\underset{\vec{x}}{\text{maximize}} \quad f(\vec{x}) \;=\; -\frac{1}{2}\vec{x}^{\top} C \vec{x} + \vec{x}^{\top}\vec{1} \quad \text{subject to} \quad \vec{x} \geq 0. \tag{3.97}$$

Hence, the resulting Wolfe Dual still comprises constraints, albeit simpler ones. In this simplified formulation of optimal stability, see also Appendix A.3.4, the goal is to maximize the objective function (3.77) under the constraint $\vec{x} \geq 0$ (with $\vec{x} \neq 0$). The search for the optimum is therefore restricted to the positive hyperoctant of $\mathbb{R}^P$.

These particularly simple conditions facilitate the design of a gradient-based active set algorithm, see Appendix A.3. Moreover, it is possible to combine this concept with coordinate ascent as discussed in Appendix A.5.1. The algorithm satisfies the constraint by simply *clipping* each x^{μ} to *zero* whenever the gradient step would lead into the excluded region of $x^{\mu} < 0$:

ADATRON algorithm, sequential updates (repeated presentation of $\mathbb{D}$)

– at time step t, present example $\mu(t) = 1, 2, 3, ..., P, 1, 2, 3, ...$
– update (only) the corresponding embedding strength:

$$x^{\mu(t)}(t{+}1) \;=\; \max\left\{ 0, \quad x^{\mu(t)}(t) \,+\, \widetilde{\eta}\left(1 - E^{\mu(t)}\right) \right\}, \tag{3.98}$$

where $E^{\nu} = [C\vec{x}]^{\nu}$. The name AdaTron has been coined for this Adaptive Perceptron algorithm [AB89, BAK91]. By comparison with the sequential Adaline algorithm (3.80) we observe that the change from equality constraints ($E^{\mu} = 1$) to inequalities ($E^{\mu} \geq 1$) leads to the restriction of embedding strengths to non-negative values. Otherwise, the update $+\widetilde{\eta}\,(1 - E^{\mu})$ is adaptive in the sense of the discussion of Adaline. Unlike the Rosenblatt algorithm, Adaline and AdaTron can decrease an embedding strength if E^{μ} is already large.

A parallel version of the algorithm can be formulated, which performs steps along the direction of $\nabla_x f$, but always ensures $\vec{x} > 0$ by limiting the step size where necessary. Thus, for sufficiently small values of $\widetilde{\eta}$, the cost function will also decrease monotonically [AB89, BAK91]. Here, we refrain from giving a more explicit mathematical formulation and discussion of the parallel updates.

Convergence of the sequential AdaTron algorithm

As we show in the following, the sequential AdaTron converges and yields optimal stability if $0 < \widetilde{\eta} < 2/C^{\mu\mu}$ for all μ, provided the data set $\mathbb{D}$ is linearly

separable [AB89, BAK91].[12] We consider the learning step $x^\mu \to x^\mu + \delta^\mu$ with

$$\delta^\mu = \begin{cases} \widetilde{\eta}\,(1-E^\mu) & \text{if } -x^\mu \le \widetilde{\eta}\,(1-E^\mu) \\ -x^\mu < 0 & \text{if } -x^\mu > \widetilde{\eta}\,(1-E^\mu) \end{cases} \tag{3.99}$$

and see that in both cases

$$|\delta^\mu| \le \widetilde{\eta}\,|(1-E^\mu)| \quad \Longrightarrow \frac{1-E^\mu}{\delta^\mu} \ge 1\Big/\widetilde{\eta}.$$

Hence, similar to Eq. (3.83) for the Adaline algorithm, we obtain

$$\begin{aligned} f(\vec{x}+\vec{\delta})-f(\vec{x}) &= -\frac{1}{2}\,\vec{\delta}^\top C\vec{\delta} - \vec{\delta}^\top C\vec{x} + \vec{\delta}^\top \vec{1} = -\frac{1}{2}(\delta^\mu)^2\, C^{\mu\mu} + \delta^\mu\,(1-E^\mu) \\ &= (\delta^\mu)^2 \left[\frac{1-E^\mu}{\delta^\mu} - \frac{C^{\mu\mu}}{2}\right] \ge (\delta^\mu)^2 \left[\frac{1}{\widetilde{\eta}} - \frac{C^{\mu\mu}}{2}\right]. \end{aligned} \tag{3.100}$$

The r.h.s. is obviously non-negative for $0 < \widetilde{\eta} \le 2/\,C^{\mu\mu}$. We recover the same condition (3.84) for the monotonic increase of f as in the sequential Adaline algorithm.

Furthermore, it can be shown that f is bounded from above in the hyperoctant $\mathbf{x} \ge 0$ if the data set is linearly separable. For general convex problems with linear constraints it has been proven that the Wolfe Dual is bounded if and only if the original problem has a solution [Fle00]. Here, this property is closely related to Farkas' Lemma and Gordan's Theorem of the Alternative (3.37), which we discussed briefly in Sec. 3.3.6. The latter implies for matrices $C \in \mathbb{R}^{P\times P}$ that

$$\left\{\vec{x}\,|\,C\,\vec{x} \ge \vec{1}\right\} = \emptyset \quad \Leftrightarrow \quad C\,\vec{v} = 0 \text{ for some } \vec{v} \ne 0 \text{ with } \{v^\mu \ge 0\}_{\mu=1}^{P}.$$

Note that if such a vector $\vec{v}$ exists, it is straightforward to show that f in Eq. (3.77) cannot be bounded in $\vec{x} > 0$: Setting $\vec{x} = \gamma\vec{v} > 0$, we obtain $\vec{x}^\top C\vec{x} = 0$ and

$$\lim_{\gamma\to\infty} f(\vec{x}) = \lim_{\gamma\to\infty} \vec{v}^\top\vec{1} \to \infty.$$

In summary, we have

a) For linearly separable $\mathbb{D}$, the cost function $f(\vec{x})$ is bounded from above in the hyperoctant $\vec{x} \ge 0$.

b) The function $f(\vec{x})$ increases monotonically under non-zero, sequential AdaTron updates with $0 < \widetilde{\eta} < 2/C^{\mu\mu}$.

c) By construction of the algorithm, any KT-point $\vec{x}^*$ of the problem, cf. Eq. (3.93–3.95), is a fixed point of the AdaTron algorithm and *vice versa.*

In combination, (a-c) imply that the algorithm converges to a KT-point which represents the (unique) perceptron of optimal stability.

[12] In [AB89, BAK91] normalized data with $C^{\mu\mu} = 1$ was considered, yielding the condition $0 < \widetilde{\eta} < 2$.

Embedding strengths only
In contrast to the Rosenblatt or Adaline algorithm, it is not straightforward to rewrite the AdaTron in terms of explicit updates in weight space. Obviously we can compute the new weight vector as $\mathbf{w}(t+1) = \sum_{\mu=1}^{P} x^\mu(t+1)\boldsymbol{\xi}^\mu S_T^\mu$ after each training step. However, a direct iteration of the weights $\mathbf{w}(t)$ cannot be provided without involving the $x^\mu(t)$, explicitly. This is due to the non-linear *clipping* of embeddings at *zero* which has no simple equivalent in weight space.

Non-separable data
The outcome of the AdaTron training (3.98) for data that is not linearly separable is not obvious. Modifications of the SSE (3.86) which take into account that for $E^\mu > 1$ the actual deviation is irrelevant have been considered in the literature, see the discussion in Sec. 4.1.

For these, corresponding gradient descent methods in weight space can be derived, which - for suitable choices of the cost function - resemble the AdaTron scheme. However, they are not identical and, in particular, additional constraints have to be imposed in order to achieve optimal stability for linearly separable data.

The concept of optimal stability has been generalized in the so-called *soft margin* classifier which tolerates misclassifications to a certain degree. A corresponding extension of the AdaTron algorithm is discussed in Sec. 4.1.2.

Efficient algorithms
The MinOver and AdaTron are presented here as prototypical approaches to solving the problem of optimal stability. While they are suitable for solving the problem in relatively small data sets, the computational load may become problematic when dealing with large numbers of examples and, consequently, a large number of variables x^μ.

A variety of algorithms have been devised aiming at computational efficiency and scalability, mainly in the context of Support Vector Machines, cf. Sec. 4.3. A prominent example is *Sequential Minimal Optimization (SMO)* approach [Pla98]. It is the basis of many implementations, see e.g. [svm99] for links and the SVM literature for further discussions and references [SS02, CST00, STC04, Her02].

3.7.4 Support vectors

The treatment of optimal stability as a constrained, convex optimization problem has provided useful insights, from which practical algorithms such as the AdaTron and other schemes emerged, see [Ruj93] for another example. The absence of local minima and the resulting availability of efficient optimization tools is one of the foundations of the Support Vector Machine and has contributed largely to its popularity [SS02, CST00, STC04, Her02, DFO20], see Sec. 4.3.

One of the most important observations with respect to optimal stability is a consequence of the complementarity condition (3.95). It implies that in a

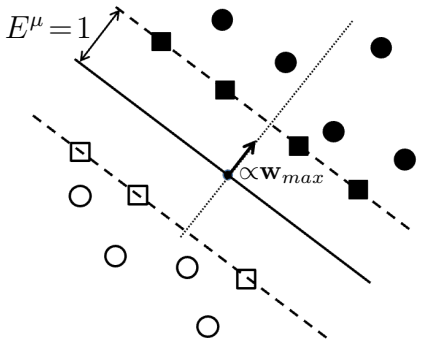


Figure 3.18: Support Vectors in the perceptron of optimal stability. The arrow represents the normalized weight vector $\mathbf{w}_{max}/|\mathbf{w}_{max}|$. Class membership is indicated by filled ($S_T = +1$) and open symbols ($S_T = -1$), respectively. Support vectors, marked as squares, fall into the two hyperplanes with $E^\mu = 1, i.e.\ \kappa^\mu = \kappa_{\max}$. All other examples (circles) display greater stability without being embedded explicitly.

KT-point we have

$$\text{either} \quad \left\{ \begin{array}{c} E^\mu = 1 \\ x^\mu \geq 0 \end{array} \right\} \quad \text{or} \quad \left\{ \begin{array}{c} E^\mu > 1 \\ x^\mu = 0 \end{array} \right\}. \tag{3.101}$$

In the geometrical interpretation of linear separability, the former set of feature vectors with $x^\mu \geq 0$ falls into one of the two hyperplanes with $\mathbf{w} \cdot \boldsymbol{\xi} = -1$ or $+1$, respectively. Both planes are parallel to the decision boundary and mark the maximum achievable gap between the two classes, see Figure 3.18. Only these examples, marked as squares in the figure, contribute to the linear combination (3.61) with non-zero embedding strength. They can be said to form the support of the weights and, therefore, are referred to as the set of support vectors. In a sense, they represent the *hard cases* in $\mathbb{D}$ which end up closest to the decision boundary. The remaining training data display larger distances from the separating hyperplane and do not explicitly contribute to the linear combination (3.61).

For the support vectors we observe that $\mathbf{w}_{max}$ is the weight vector that solves the system of linear equations $E^\mu = 1$ with minimal norm $|\mathbf{w}|$. All other examples appear to be stabilized "accidentally". Hence, if we were able to identify them beforehand in a given $\mathbb{D}$, we could solve the simpler problem (3.77) restricted to the support vectors by applying, e.g., the Adaline algorithm. Unfortunately, the set of support vectors is not known a priori and their determination is an integral part of the training process.

It is important to realize that, despite the special role of the support vectors, all examples in the data set are relevant. Even if some of them end up with *zero* embedding, the composition of the entire $\mathbb{D}$ implicitly determines the set of support vectors and, thus, the actual weight vector.

Remark: Complementarity in the MinOver algorithm
The MinOver algorithm (3.59,3.60) as presented in Sec. 3.6.2 yields, in general, non-zero integer $x^\mu > 0$ for all μ. However, the complementary condition (3.101) is satisfied in the sense that the embedding strengths of the support vectors increase in the training process, while those of the other examples remain

relatively small. In the limit of infinitely many update steps, properly rescaled x^μ satisfy condition (3.92).

3.8 Inhom. lin. sep. functions revisited

So far, following the arguments leading to Eq. (3.4) in Sec. 3.2 we have treated homogeneously and inhomogeneously linearly separable functions on the same footing. However, with respect to the stability criterion we have to take a subtlety into account: if we consider a perceptron with output

$$S(\boldsymbol{\xi}) = \text{sign}\left[\mathbf{w}\cdot\boldsymbol{\xi} - \theta\right] \text{ with adaptive quantities } \mathbf{w}\in\mathbb{R}^N \text{ and } \theta\in\mathbb{R}, \quad (3.102)$$

the proper definition of the stabilities is

$$\kappa^\mu_{inh} \;=\; (\mathbf{w}\cdot\boldsymbol{\xi}^\mu - \theta)\, S^\mu / \left|\mathbf{w}\right| \quad (3.103)$$

which corresponds to the oriented distance of feature vectors $\boldsymbol{\xi}^\mu$ from the separating hyperplane given by $(\mathbf{w}\cdot\boldsymbol{\xi}-\theta=0)$. As a consequence, the corresponding perceptron of optimal stability is given by the solution of the problem

$$\min_{\mathbf{w},\theta} \mathbf{w}^2/2 \quad \text{subject to} \quad \left[\mathbf{w}\cdot\boldsymbol{\xi}^\mu - \theta\right] S^\mu \geq 1 \quad \text{for all} \quad \mu = 1,2,\dots P. \quad (3.104)$$

Note that if for suitable θ, $\mathbf{w}^2$ can be minimized under the constraints, the choice of θ will influence the result, i.e. the achievable maximum stability. Hence, the optimization has to be done with respect to both $\mathbf{w}$ and θ.

If we naively exploit the equivalence of inhomogeneous separability in N dimensions with homogeneous separability in $N+1$ dimensions, c.f. Eq. (3.4), we define the modified weight vector $\widetilde{\mathbf{w}} = [\mathbf{w},\theta]$ and augmented feature vectors $\widetilde{\boldsymbol{\xi}} = [\boldsymbol{\xi},-1]$. The corresponding naive definition of the stabilities reads

$$\widetilde{\kappa}^\mu \;=\; \widetilde{\mathbf{w}}\cdot\widetilde{\boldsymbol{\xi}}^\mu S^\mu \Big/ \left|\widetilde{\mathbf{w}}\right|. \quad (3.105)$$

Here $\widetilde{\mathbf{w}}^2$ corresponds to $(\mathbf{w}^2+\theta^2)$ and the resulting optimal stability problem becomes

$$\min_{\widetilde{\mathbf{w}}} \widetilde{\mathbf{w}}^2/2 \quad \text{subject to} \quad \widetilde{\mathbf{w}}\cdot\widetilde{\boldsymbol{\xi}}^\mu S^\mu \geq 1 \quad \text{for all} \quad \mu = 1,2,\dots P, \quad (3.106)$$

where the constraints are equivalent to those in Eq. (3.104). However, treating θ as an additional weight in the spirit of (3.4) has altered the objective function in comparison with (3.104) as it includes a contribution θ^2 in the squared Euclidean norm $\widetilde{\mathbf{w}}^2$. Consequently the solution of (3.106) would differ from the (correct) optimal stability defined by Eq. (3.104).

For the sake of clarity, we have explicitly limited our discussion of optimal stability to the case of homogenous separation with $\theta = 0$. This also applies to the Support Vector Machine in Sec. 4.3. The corresponding modifications of the algorithms and other results to the modified definitions (3.103) and (3.104) for nonzero local threshold are conceptually straightforward, see e.g. [Str19, HTF01, DFO20].

3.9 Some remarks

In this chapter we have considered the simple perceptron as a prototypical machine learning system. It serves as a framework in which to obtain insights into the basic concepts of supervised learning. It also illustrates the importance of optimization techniques and the related theory in machine learning.

The restriction to linearly separable functions is, of course, significant. In the next chapter we consider a number of ways to deal with non-separable data sets and address classification problems beyond linear separability.

It is remarkable that the perceptron still ranks among the most frequently applied machine learning tools. This is due to the fact that the very successful Support Vector Machine is frequently used with a *linear kernel*, cf. Sec. 4.3. In this case, however, it reduces to a classifier that is equivalent to the simple perceptron of optimal stability or its extension to the soft margin classifier, cf. Sec. 4.1.2.

Chapter 4

Beyond linear separability

Non-linear means it's hard to solve.

— Arthur Mattuck

In the previous sections we studied *learning in version space* as a basic training strategy in terms of the simple perceptron classifier. We obtained important insights into the principles of learning a rule and obtained the concept of optimal stability.

As pointed out already, learning in version space only makes sense under a number of conditions, which – unfortunately – are rarely fulfilled in realistic settings. The key assumptions are

(a) The data set $\mathbb{D} = \{\boldsymbol{\xi}^\mu, S_T^\mu\}_{\mu=1}^P$ is perfectly reliable, labels are correct and noise–free.

(b) The unknown rule is linearly separable, or more generally: the student complexity perfectly matches the target task.

In practice, a variety of effects can impair the reliability of the training data: Some examples could be explicitly mislabeled by an unreliable expert to begin with. Class labels could be inverted due to some form of noise in the communication between student and teacher. Alternatively (or additionally), some form of noise or corruption may have distorted the input vectors $\boldsymbol{\xi}^\mu$ in $\mathbb{D}$, potentially rendering some of the labels S_T^μ inconsistent.

In fact, real world training scenarios and data sets will hardly ever meet the conditions (a) and/or (b). From the perspective of perceptron training, i.e. restricting the hypothesis space to linearly separable functions, the following situations are much more likely to occur:

i) The unknown target rule is linearly separable, but $\mathbb{D}$ contains mislabeled examples, for instance in the presence of noise. Depending on the number of examples P and the degree of the corruption the following may occur:

   i.1) In particular, small data sets $\mathbb{D}$ with few mislabeled samples can still be linearly separable and the labels S_T^μ can be reproduced by a perceptron student. While the non-empty version space $\mathcal{V}$ is consistent with $\mathbb{D}$, it is not perfectly representative of the target rule and the success of training will be impaired.

   i.2) The data set $\mathbb{D}$ is not linearly separable and, consequently, a version space of linearly separable functions does not exist: $\mathcal{V} = \emptyset$. In this case, learning in version space is not even defined for the perceptron. Hence, the data set still contains information (albeit noisy or corrupted) about the rule, but it is not obvious how it can be extracted efficiently.

ii) The target rule itself is not linearly separable and would require learning systems of greater complexity than the simple perceptron, ideally. Again, the consequences for perceptron training depend on the size of the available data set:

   ii.1) Small data sets may be linearly separable and one can expect or hope that a student $\mathbf{w} \in \mathcal{V}$ at least approximates the unknown rule to a certain extent, in this situation.

   ii.2) Larger data sets become non-separable and $\mathcal{V} = \emptyset$ should signal that the target is, in fact, more complex.

Of course, superpositions of cases i) and ii) can also occur: we could have to deal with a non-separable rule, represented by a data set which in addition is subject to some form of corruption.

As mentioned earlier, it is a difficult task in practice to determine whether a given data set $\mathbb{D}$ is linearly separable or not. Not finding a solution by use of the Rosenblatt algorithm, for instance, could simply indicate that a larger number of training steps is required. In [ND91], a perceptron-like algorithm is presented which either finds a solution or terminates and establishes non-separability [ND91], see also Sec. 3.3.6.

In the following, we discuss a number of basic ideas and strategies for coping with non-separable data sets:

- A simple perceptron could be trained to implement $\mathbb{D}$ approximately, in the sense that a large (possibly maximal) fraction of training labels is reproduced by the perceptron. In Sec. 4.1 we will present and discuss corresponding training schemes.
- More powerful, layered networks for classification can be constructed from perceptron-like units. We will consider the so-called committee machine and parity machine as example two-layer systems in Sec. 4.2. We show that the latter constitutes a *universal classifier*.

- Many perceptrons (or other simple classifiers) can be combined into an *ensemble* in order to take advantage of a *wisdom of the crowd* effect [OM99, Urb00]. So-called *Random Forests*, i.e. ensembles of decision trees [Bre01], constitute one of the most prominent examples in the literature, currently. We refrain from a detailed discussion of ensembles and refer to the literature [OM99, Urb00] for further references.
- A perceptron-like threshold operation of the form $S = \text{sign}(\ldots)$ with linear argument can be applied after a non-linear transformation of the feature vectors. Along these lines, the framework of the Support Vector Machines (SVM) was developed. It has been particularly successful and continues to do so. In Sec. 4.3 we will outline the concept and show that the SVM can be seen as a (highly non-trivial) conceptual extension of the simple perceptron.
- Perhaps the most frequently applied strategy is to treat the classification as a regression problem. A network of continuous units (including the output) can be trained by standard methods and, eventually, the output is thresholded. We have seen one example already in terms of the Adaline scheme, see Sec. 3.7. We return to this strategy in Chapter 5.
- Prototype-based systems [BHV16] offer another, very powerful framework for classification beyond linear separability. In Chapter 6 we will discuss the example of Learning Vector Quantization (LVQ), which implements – depending on the details – piecewise linear or piecewise quadratic decision boundaries. It is moreover a natural tool for multi-class classification.

4.1 Perceptron with errors

Assume that a given data set $\{\boldsymbol{\xi}^\mu, S_T^\mu\}$ with $\boldsymbol{\xi}^\mu \in \mathbb{R}^N$ and $S_T^\mu = \{-1, +1\}$ is not linearly separable due to one or several of the reasons discussed above.

In the following, we discuss extensions of perceptron training which tolerate misclassification to a certain degree. Intuitively, this corresponds to the assumption that the data set is nearly linearly separable or in other words: the unknown rule can be approximated by a linearly separable function.

4.1.1 Minimal number of errors

Aiming at a linearly separable approximation, one might want to minimize the number of misclassifications by choice of the weight vector $\mathbf{w} \in \mathbb{R}^N$ in a simple perceptron student. While at a glance the idea appears plausible, we should realize that the outcome will be very sensitive to individual, misclassified examples in $\mathbb{D}$.

Formally, we can consider the following problem:

MINIMAL NUMBER OF ERRORS (PERCEPTRON) (4.1)

For a given $\mathbb{D} = \{\boldsymbol{\xi}^\mu, S_T^\mu\}_{\mu=1}^P$ with $\boldsymbol{\xi}^\mu \in \mathbb{R}^N$ and $S_T^\mu \in \{-1, +1\}$,

$$\underset{\mathbf{w} \in \mathbb{R}^N}{\text{minimize}}\, H^{err}(\mathbf{w}) = \sum_{\mu=1}^{P} \epsilon(\mathbf{w}, \boldsymbol{\xi}^\mu, S_T^\mu) \text{ with } \epsilon = \begin{cases} 1 & \text{if } \operatorname{sign}(\mathbf{w} \cdot \boldsymbol{\xi}^\mu) = -S_T^\mu \\ 0 & \text{if } \operatorname{sign}(\mathbf{w} \cdot \boldsymbol{\xi}^\mu) = +S_T^\mu. \end{cases}$$

Note that this cost function does not differentiate between *nearly correct* examples with $E^\mu \approx 0$ close to the decision boundary and clear cases where the feature vector falls deep into the incorrect half-space.

The above defined problem proves very difficult. Note that the number of misclassified examples is obviously integer and can only display discontinuous changes with $\mathbf{w}$. On the other hand, $\nabla_w H^{err} = 0$ almost everywhere in feature space. As a consequence, gradient based methods cannot be employed for the minimization or approximate minimization of H^{err}.

A prescription that applies Hebbian update steps which can be seen as a modification of the Rosenblatt algorithm (3.15) has been introduced by S.I. Gallant [Gal90]. The so-called Pocket Algorithm relies on the stochastic presentation of simple examples in combination with the principle of learning from mistake for a weight vector $\mathbf{w}$. In addition to the randomized, yet otherwise conventional, Rosenblatt updates of the vector $\mathbf{w}$, the so far best (with respect to H^{err}) weight vector $\widehat{\mathbf{w}}$ is "put into the pocket". It is only replaced when the on-going training process yields a vector $\mathbf{w}$ with a lower number of errors.

POCKET ALGORITHM (4.2)

- initialize $\mathbf{w}(0) = 0$ and $\widehat{\mathbf{w}} = 0$ (tabula rasa), set $\widehat{H}(0) = P$
- at time step t, select a single feature vector $\boldsymbol{\xi}^\mu$ with class label S_T^μ randomly from the data set $\mathbb{D}$ with equal probability $1/P$
- compute the local potential $E^\mu(t) = \mathbf{w}(t) \cdot \boldsymbol{\xi}^\mu S_T^\mu$
- for $\mathbf{w}(t)$, perform an update according to (Rosenblatt algorithm)

$$\mathbf{w}(t+1) = \mathbf{w}(t) + \frac{1}{N}\, \Theta\left[-E^{\mu(t)}\right]\, \boldsymbol{\xi}^\mu\, S_T^\mu.$$

- compute $H(t+1) = H^{err}\big(\mathbf{w}(t+1)\big)$ acc. to Eq. (4.1)
- update the pocket vector $\widehat{\mathbf{w}}(\mathbf{t})$ and $\widehat{H}(t)$ according to

$$\widehat{\mathbf{w}}(t+1) = \begin{cases} \mathbf{w}(t+1) & \text{if } H(t+1) < \widehat{H}(t) \quad \text{(improvement)} \\ \widehat{\mathbf{w}}(t) & \text{if } H(t+1) \geq \widehat{H}(t) \quad (\widehat{\mathbf{w}} \text{ unchanged}) \end{cases}$$

$$\widehat{H}(t+1) = H^{err}\big(\widehat{\mathbf{w}}(t+1)\big).$$

Obviously, the number of errors $\widehat{H}(t)$ of the pocket vector $\widehat{\mathbf{w}}(t)$ can never increase under the updates (4.2). Moreover, one can show that the stochastic selection of the training sample guarantees that, in principle, $\widehat{\mathbf{w}}(t)$ approaches the minimum of H^{err} with probability *one* [Gal90, Roj96].

However, this quite weak *convergence in probability* does not allow to make statements about the expected number of updates required to achieve a solution of a particular quality. Certainly it is not possible to provide upper bounds as in the context of the Perceptron Convergence Theorem.

Several alternative, well-defined approaches have been considered which can be shown to converge in a more conventional sense, at the expense of having to accept sub-optimal H^{err}.

Along the lines discussed towards the end of Sec. 3.7.2, the gradient based minimization of several cost functions similar to Eq. (3.86) has been considered in the literature. For instance [GG91] discusses objective functions which correspond to

$$H^{[k]}(\mathbf{w}) = \sum_{\mu=1}^{P} (c - E^\mu)^k \, \Theta[c - E^\mu] \quad \text{with } E^\mu = \mathbf{w} \cdot \boldsymbol{\xi}^\mu S_T^\mu$$

in our notation. Note that the limiting case $c = 0, k \to 0$ would recover the non-differentiable H^{err}, Eq. (4.1). In [GG91] the above functions are, not quite precisely, referred to as the *perceptron cost function* for $k = 1$ and the *adatron cost function* for $k = 2$, respectively. We would like to point out that, despite the conceptual similarity, the case $c = 1, k = 2$ is not strictly equivalent to the AdaTron algorithm for non-separable data sets. Moreover, for lin. sep. data, any $\mathbf{w} \in \mathcal{V}$ gives $H^{[2]}(\mathbf{w}) = 0$. Hence, optimal stability would have to be enforced through an additional minimization of the norm $\mathbf{w}^2$.

4.1.2 Soft margin classifier

An explicit extension of the large margin concept has been suggested which allows for the controlled acceptance of misclassifications. For an introduction in the context of the SVM, see e.g. [SS02, DFO20] and references therein. The formalism presented there reduces to the *soft margin perceptron* in the case of a linear kernel function, see Sec. 4.3 for the relation.

A corresponding, extended optimization problem similar to (3.68) reads:

SOFT MARGIN PERCEPTRON (weight space)

$$\underset{\mathbf{w}, \vec{\beta}}{\text{minimize}} \quad \frac{N}{2} \mathbf{w}^2 + \gamma \sum_{\mu=1}^{P} \beta^\mu \qquad \text{subject to} \quad \{E^\mu \geq 1 - \beta^\mu\}_{\mu=1}^{P} \quad \text{and} \quad \{\beta^\mu \geq 0\}_{\mu=1}^{P} . \tag{4.3}$$

Here, we introduce so-called slack variables $\beta^\mu \in \mathbb{R}$ with $\begin{cases} \beta^\mu = 0 & \Leftrightarrow \quad E^\mu \geq 1 \\ \beta^\mu > 0 & \Leftrightarrow \quad E^\mu < 1. \end{cases}$

We recover the unmodified problem of optimal stability (3.68) with $\beta^\mu = 0$ for all $\mu = 1, 2, ..., P$, which also implies that all $E^\mu \geq 1$. On the contrary, non-zero $\beta^\mu > 0$ correspond to violations of the original constraints, i.e. examples with $E^\mu < 1$, which includes potential misclassifications ($E^\mu < 0$). In (4.3), the Lagrange multiplier $\gamma \in \mathbb{R}$ controls the extent to which non-zero β^μ are accepted.

In the by now familiar matrix-vector notation with $\vec{\beta} = (\beta^1, \beta^2, ..., \beta^P)^\top$ we can rewrite the problem in terms of embedding strengths:

SOFT MARGIN PERCEPTRON (embedding strengths)

$$\underset{\vec{x}, \vec{\beta}}{\text{minimize}} \quad \frac{1}{2}\, \vec{x}^\top C \vec{x} \; + \gamma \vec{\beta}^\top \vec{1} \qquad \text{subject to} \quad \vec{E} \geq \vec{1} - \vec{\beta} \qquad \text{and} \quad \vec{\beta} \geq 0. \tag{4.4}$$

Here, the derivation of the Wolfe Dual [Fle00] amounts to the elimination of the slack variables $\vec{\beta}$. Similar to the error-free case, cf. Sec. 3.7, we obtain a modified cost function with simpler constraints:

SOFT MARGIN PERCEPTRON (dual problem)

$$\underset{\vec{x}}{\text{maximize}} \quad -\frac{1}{2}\, \vec{x}^\top C \vec{x} \; + \; \vec{x}^\top \vec{1} \qquad \text{subject to} \quad \vec{0} \leq \vec{x} \leq \gamma \vec{1}. \tag{4.5}$$

In comparison to the dual problem (3.68) for the error-free case, the non-negative embedding strengths $\vec{x} \geq 0$ are now also bounded from above. The parameter γ limits the magnitudes of the x^μ.

In analogy to the derivation of the AdaTron algorithm (3.98) we can devise a similar, sequential projected-gradient descent algorithm:

ADATRON WITH ERRORS, sequential updates (repeated presentation of $\mathbb{D}$)

– at time step t, present the example $\mu = 1, 2, ..., P, 1, 2, ...$
– perform the update

$$\begin{aligned} \widetilde{x}^\mu(t+1) &= x^\mu(t) + \widehat{\eta}\, (1 - [C\vec{x}(t)]^\mu) && \text{gradient step} \\ \widehat{x}^\mu(t+1) &= \max\{0, \widetilde{x}^\mu(t+1)\} && \text{non-negative embeddings...} \\ x^\mu(t+1) &= \min\{\gamma, \widehat{x}^\mu(t+1)\} && \text{...with limited magnitude.} \end{aligned} \tag{4.6}$$

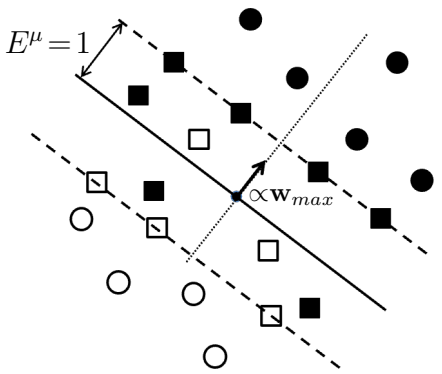


Figure 4.1: Support vectors in the soft margin perceptron.
The arrow represents the normalized weight vector $\mathbf{w}_{max}/|\mathbf{w}_{max}|$. Filled and open symbols correspond to the classes $S_T^\mu = \pm 1$, respectively. Support vectors, displayed as squares, fall onto the two hyperplanes with $E^\mu = 1$, into the region between the planes, or even deeper into the *incorrect half-space*. All other examples, marked by circles, display $E^\mu > 1$ without explicit embedding.

Compared to the original AdaTron for lin. sep. data, the only difference is the additional restriction of the search to the $\vec{x} \leq \gamma \vec{1}$. Obviously we recover the the original algorithm in the limit $\gamma \to \infty$.

As in the separable case, the algorithm follows the gradient of the cost function along $(\vec{1} - \vec{E})$, in principle. Hence, an individual embedding strength will increase if the corresponding example has a local potential $E^\mu < 1$, including misclassifications with $E^\mu < 0$. If some of the errors cannot be corrected because $\mathbb{D}$ is not separable, the corresponding x^μ would grow indefinitely. In the soft margin version (4.6), however, updates are clipped at $x^\mu = \gamma$ and the misclassification[1] of the corresponding example is tolerated. For fixed γ, the problem is well-defined and the AdaTron with errors (4.6) finds a solution efficiently. We refrain from further analysis and a formal proof.

It is important to realize that the precise nature of the solution depends strongly on the setting of the parameter γ: It controls the compromise between the goals of – on the one hand – minimizing the norm $\mathbf{w}^2$ (maximizing the margin) and – on the other hand – correcting misclassifications. More precisely, the emphasis is not explicitly on the number of errors, but on the violations of $E^\mu \geq 1$ and their severity.

If, for instance a mismatched (too small) value of γ is chosen, misclassifications will be accepted and favored, even in a linearly separable data set. In practice, a suitable value can be determined by means of a validation procedure which estimates the performance for different choices of γ.

In analogy to Sec. 3.7.4 the support vectors are characterized by $x^\mu > 0$, as before. However, only examples with $0 < x^\mu < \gamma$ will lie exactly in one of the planes with $E^\mu = 1$. Clipped embedding strengths $x^\mu = \gamma$ correspond to examples which fall into the region between the planes in Fig. 4.1 or even deeper into the *incorrect half-space*.

The soft margin concept for the toleration of misclassification is highly relevant in the context of the Support Vector Machine, see Sec. 4.3.

[1] the violation of $E^\mu \geq 1$, to be more precise

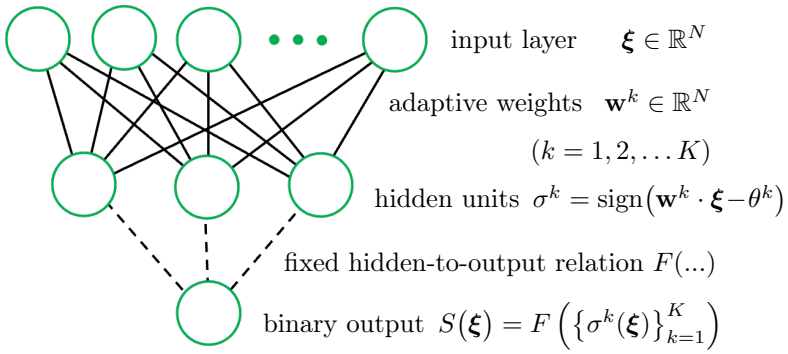


Figure 4.2: The architecture of a "machine" as introduced in Sec. 4.2. A number K of hidden units of the perceptron type are connected by adaptive weights with the N-dim. input layer, $K = 3$ in the illustration. The binary response $S(\boldsymbol{\xi})$ is determined by a pre-defined, fixed functional dependence $F\left(\sigma^1, \sigma^2, ...\sigma^K\right) = \pm 1$.

4.2 Layered networks of perceptron-like units

Perceptron-like units can be assembled in more powerful architectures as to overcome the restriction to linearly separable functions. We will highlight this in terms of a family of systems which are occasionally termed *machines* in the literature, see [MD89a, EB01, WRB93, AMB+18, MZ95, BO91, SH93] and references therein.

In Fig. 4.2 the architecture of a *machine* is illustrated. It comprises

- an input layer representing feature vectors $\boldsymbol{\xi} \in \mathbb{R}^N$
- a single layer of K perceptron-like hidden units
  $\sigma^k(\boldsymbol{\xi}) = \mathrm{sign}\left(\mathbf{w}^k \cdot \boldsymbol{\xi} - \theta^k\right)$
- a set of adaptive input-to-hidden weight vectors $\mathbf{w}^k \in \mathbb{R}^N$ and local thresholds[2] $\theta^k \in \mathbb{R}^N$
- a single, binary output, determined by a fixed function $F(\sigma^1, \sigma^2, ...\sigma^K)$

The function F ultimately determines the network's input/output relation. However, it is assumed to be pre-wired and cannot be adapted in the training process. Learning is restricted to the $\mathbf{w}^k$ connecting input and hidden layer.

[2]Thresholds could be replaced by a formal weight from a clamped input as outlined in (3.4).

4.2.1 Committee and parity machines

Two specific machines have attracted particular interest:

CM: The committee machine combines the hidden unit states σ^k in a majority vote [EB01, MD89a, WRB93, AMB+18]. This is realized by setting

$$F^{CM}\left(\{\sigma^k\}_{k=1}^K\right) = \text{sign}\left(\sum_{k=1}^K \sigma^k\right) \Rightarrow S^{CM}(\boldsymbol{\xi}) = \text{sign}\left(\sum_{k=1}^K \text{sign}\left[\mathbf{w}^k \cdot \boldsymbol{\xi} - \theta^k\right]\right) \tag{4.7}$$

which is only well-defined for odd values of K which avoids ties $\sum_k \sigma^k = 0$. The majority vote is reminiscent of an ensemble of independently trained perceptrons. The CM, however, is meant to be trained as a whole [SH93].

PM: In the parity machine the output is computed as a product over the hidden unit states σ^k [EB01, WRB93, BO91]:

$$F^{PM}\left(\{\sigma^k\}_{k=1}^K\right) = \prod_{k=1}^K \sigma^k \Rightarrow S^{PM}(\boldsymbol{\xi}) = \prod_{k=1}^K \text{sign}\left[\mathbf{w}^k \cdot \boldsymbol{\xi}\right] \tag{4.8}$$

which results in a well-defined binary output $S^{PM}(\boldsymbol{\xi}) = \pm 1$ for any K. Note that the product depends on whether the number of units with $\sigma^k = -1$ is odd or even. In this sense $F^{PM}(\ldots)$ is analogous to a parity operation.

The hidden-to-output relation of the PM, i.e. the parity operation, cannot be represented by a single perceptron unit.[3] We could realize the hidden-to-output function by a more complex multi-layered network. But since F is considered to be pre-wired and not adaptive, we do not have to specify a neural realization here. On the contrary, the committee machine can be interpreted as a conventional two-layered feed-forward neural network with an $N-K-1$ architecture as discussed in Sec. 1.3.2 and illustrated in Fig. 1.5 (right panel). However, compared to the general form of the output, Eq. (1.16), we have to use the activation function $g(z) = \text{sign}(z)$ throughout the net and fix all hidden-to-output to $v_k = 1$ in the CM.

Many theoretical results are available for CM and PM networks and more general *machines*. Among other things, their storage capacity and generalization ability have been addressed, see [MD89a, SH93, Opp94, PBG+94] for examples and [EB01, WRB93, AMB+18] for further references.

[3] The gist of Minsky and Papert's book [MP69] is often reduced to this single insight, which is a gross injustice to both, the book and the perceptron.

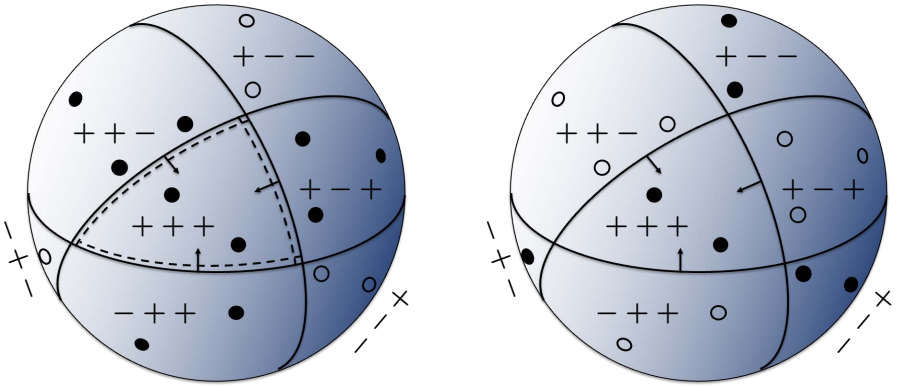


Figure 4.3: Output function of Committee machine and Parity machine, applied to identical sets of feature vectors $\boldsymbol{\xi}$ (filled and empty circles). The illustration corresponds to the surface of an N-dim. hypersphere. In both machines, $K = 3$ oriented hyperplanes tesselate the feature space. Arrows point to the respective half-space of $\sigma^k = +1$. The three hidden units states are marked by "+" and "-" signs in the corresponding regions. The networks' responses $S = \pm 1$ are marked by empty and filled circles. **Left panel:** The majority of σ^k determines the total response of the CM. Dashed lines mark *redundant* pieces of the hyperplanes which do not separate outputs $S = +1$ from $S = -1$. **Right panel:** In the PM, the total output is $S = \prod_k \sigma^k$. Every hyperplane separates total outputs $S = \pm 1$ locally, everywhere in feature space.

4.2.2 The parity machine: a universal classifier

Here, we focus on a particularly interesting result. We show that a PM with sufficiently many hidden units can implement any data set of the form $\mathbb{D} = \{\boldsymbol{\xi}^\mu, S_T^\mu\}_{\mu=1}^P$ with binary training labels.

In the following, we outline a constructive proof which is based on the application of a particular training strategy. So-called growth algorithms add units to a neural network with subsequent training until the desired performance is achieved and, for example, a given data set $\mathbb{D}$ has been implement. Mezard and Parisi coined the term tiling algorithm for a particular procedure [MN89], which adds neurons one by one. Several other similar growth schemes have been suggested, see [GM90, Fre90] for examples and references.

A particular tiling-like algorithm for the PM was introduced and analysed in [BO91]. It was not necessarily designed as a practical training prescription for realistic applications. Tiling-like training of the PM proceeds along the following lines:

TILING-LIKE LEARNING, (parity machine) (4.9)

(I) Initialization ($m = 1$):
Train the first unit with output $S_1(\boldsymbol{\xi}) = \sigma^1(\boldsymbol{\xi}) = \text{sign}[\mathbf{w}^1 \cdot \boldsymbol{\xi} - \theta_1]$ from the data set $\mathbb{D}_1 = \mathbb{D}$, aiming at a large number of correctly classified examples Q_1.[4]

(II) After training of m units:
Given the PM with m hidden units $\{\sigma^1, \sigma^2, ...\sigma^m\}$, re-order the indices μ of the examples such that the PM output is

$$S_m(\boldsymbol{\xi}^\mu) = \prod_{j=1}^{m} \sigma^j(\boldsymbol{\xi}^\mu) = \begin{cases} +S_T^\mu & \text{for } 1 \le \mu \le Q_m \\ -S_T^\mu & \text{for } Q_m < \mu \le P, \end{cases}$$

where Q_m is the number of correctly classified examples in $\mathbb{D}$.

Define the new training set $\mathbb{D}_{m+1} = \{\boldsymbol{\xi}^\mu, [S_m(\boldsymbol{\xi}^\mu)\, S_T^\mu]\}_{\mu=1}^{P}$

with labels $[S_m(\boldsymbol{\xi}^\mu) S_T^\mu] = \begin{cases} +1 & \text{for } 1 \le \mu \le Q_m \quad (S_m \text{ was correct}) \\ -1 & \text{for } Q_m < \mu \le P \quad (S_m \text{ was wrong }) \end{cases}$

(III) Training step:
add and train the next hidden unit $\sigma^{m+1}(\boldsymbol{\xi}) = \text{sign}\left[\mathbf{w}^{m+1} \cdot \boldsymbol{\xi} - \theta_{m+1}\right]$ as to achieve a low number of errors $(P - Q_{m+1})$ w.r.t. data set $\mathbb{D}_{m+1}$.

Note that if a solution with *zero error*, i.e. $Q_M = P$ is found in step (III) for the M-th hidden unit σ_M, the total output of the PM is

$$\prod_{m=1}^{M} \sigma_m(\boldsymbol{\xi}^\mu) = S_T^\mu \quad \text{for all examples in } \mathbb{D},$$

i.e. the data set is perfectly reproduced by the PM of M units.

It is surprisingly straightforward to show that the number of correctly classified feature vectors can be increased by at least one ($Q_{m+1} > Q_m$) when adding the $(m + 1)$-th unit in step (III) of the procedure (4.9).

To this end, we consider a set of normalized input vectors in the procedure (4.10). The normalization (4.11) could always be implemented in a preprocessing step. The second relation (4.12) is trivially satisfied: in every data set δ can be determined by computing all pair-wise scalar products.

[4]Any of the algorithms discussed in Sec. 4.1 could be used in this step. For the inhomogeneity, a clamped input as in Eq. (3.4) can be employed, for simplicity.

GRANDMOTHER NEURON (4.10)

Consider a set of feature vectors $\{\boldsymbol{\xi}^\mu\}_{\mu=1}^P$ with

$$|\boldsymbol{\xi}^\mu|^2 = \Gamma \quad \text{for all } \mu = 1, 2, ..., P \tag{4.11}$$

$$0 < \delta < \Gamma - \boldsymbol{\xi}^\mu \cdot \boldsymbol{\xi}^\nu \quad \text{for all } \mu, \nu \ (\mu \neq \nu). \tag{4.12}$$

Construct a perceptron weight vector and threshold as

$$\mathbf{w} = -\boldsymbol{\xi}^P \quad \text{and} \quad \theta = \delta - \Gamma. \tag{4.13}$$

It results in the inhomogeneously linearly separable classification

$$S_{\mathbf{w},\theta}(\boldsymbol{\xi}^\mu) = \operatorname{sign}\big[-\boldsymbol{\xi}^P \cdot \boldsymbol{\xi}^\mu - \delta + \Gamma\big] = \begin{cases} \operatorname{sign}\big[-\underbrace{\boldsymbol{\xi}^P \cdot \boldsymbol{\xi}^P}_{=\Gamma} + \Gamma - \delta\big] = -1 & \text{for } \mu = P \\ \operatorname{sign}\big[-\underbrace{\boldsymbol{\xi}^P \cdot \boldsymbol{\xi}^\mu + \Gamma}_{>\delta} - \delta\big] = +1 & \text{for } \mu \neq P. \end{cases} \tag{4.14}$$

The corresponding perceptron separates exactly one feature vector, $\boldsymbol{\xi}^P$, from all others in the set. The term *grandmother neuron* has been coined for this type of unit. It relates to the debatable concept that a single neuron in our brain is activated specifically whenever we see our grandmother.[5]

For the tiling-like learning (4.9), this implies that, by use of a grandmother unit, we can always separate $\boldsymbol{\xi}^P$ from all other examples in the training step (III). The hidden unit response for this input is $\sigma_{m+1}(\boldsymbol{\xi}^P) = -1$, which corrects the misclassification as the incorrect output $S_m(\boldsymbol{\xi}^P) = -S_T^P$ is multiplied with -1 yielding $S_{m+1}(\boldsymbol{\xi}^P) = S_T^P$. All other PM outputs are left unchanged.

Hence, at least one error can be corrected by adding a unit to the growing PM. With at most P units in the worst case, the number of errors is *zero* and all labels in $\mathbb{D}$ are reproduced correctly.

A few remarks

- The grandmother unit (4.14) serves at best as a minimal solution in the constructive proof - it is not suitable for practical purposes. The use of $\mathcal{O}(P)$ perceptron units for the labelling of P examples would be highly inefficient.

- Step (III) can be improved significantly as compared to the constructive solution by using efficient training algorithms such as the soft margin AdaTron, see Sec. 4.1.1.

[5] An idea which is possibly not quite as unrealistic as it may seem, see for instance [QRK+05] for a discussion of "Jennifer Aniston cells".

- Tiling-like learning imposes a strong ordering of the hidden units. Neurons added to the system later are supposed to correct only the (hopefully) very few misclassifications made by the first units. To some extent this contradicts the attractive concept of neural networks as fault-tolerant and robust distributed memories.
- The strength of the tiling concept is at the same time its major weakness: Unlimited complexity and storage capacity can be achieved by adding more and more units to the system, until error-free classification is achieved. This will lead to inferior generalization behavior as the system adapts to every little detail of the data. This suggests to apply a form of *early stopping*, which limits the maximum number of units in the PM according to validation performance.

We conclude that the parity machine is a **universal classifier** in the sense that a PM with sufficiently many hidden units can implement any two-class data $\mathbb{D}$:

UNIVERSAL CLASSIFIER (parity machine) (4.15)

For a given data set $\mathbb{D} = \{\boldsymbol{\xi}^\mu, S_T^\mu\}_{\mu=1}^P$ with binary labels $S_T^\mu \in \{-1, +1\}$ and normalized feature vectors with $|\boldsymbol{\xi}^\mu|^2 = const.$ for all $\mu = 1, 2, \dots P$,

weight vectors $\mathbf{w}^k \in \mathbb{R}^N$ and thresholds $\theta^k \in \mathbb{R}$ exist (and can be found) with

$$S^{PM}(\boldsymbol{\xi}^\mu) = \prod_{k=1}^{K} \text{sign}\left[\mathbf{w}^k \cdot \boldsymbol{\xi}^\mu - \theta^k\right] = S_T^\mu \quad \text{for all } \mu = 1, 2, \dots P.$$

Similar theorems have been derived for other "shallow" architectures with a single layer of hidden units and a single binary output.[6] These findings are certainly of fundamental importance. In contrast to the Perceptron Convergence Theorem, however, (4.15) and similar propositions are in general not associated with practical, efficient training algorithms.

The result parallels the findings of Cybenko and others [Cyb89, HTF01] that networks with a single, sufficiently large hidden layer of continuous non-linear units constitute universal function approximators, see Chapter 5 for details.

It is interesting to note that also extremely deep and narrow networks can be universal classifiers. As an example, stacks of single perceptron units with *shortcut connections* to the input layer have been investigated in [Roj17, Roj03].

4.2.3 The capacity of machines

Compared to the simple perceptron, the greater complexity of the CM or PM should lead to an increased storage capacity. In fact, following the work of

[6] Strictly speaking the PM does not fall into this class, as F^{PM} cannot be realized by a single perceptron-like unit.

Mitchison and Durbin [MD89a], we can extend the arguments presented in Sec. 3.4 to machines.

The number $C_K(P,N)$ of different dichotomies that a machine with K hidden units can realize for P feature vectors in N dimensions is obviously bounded from above as

$$C_K(P,N) \;\leq\; C(P,N)^K \tag{4.16}$$

with $C(P,N)$ from Eq. (3.41) for the perceptron. If we assume that the network can freely combine all lin. sep. functions realized by the perceptron units in the hidden layer, we would expect an equality in (4.16). In general, correlations between the hidden units and other redundancies or restrictions will reduce $C_K(P,N)$ as compared to the upper bound.

With the total number of possible dichotomies 2^P we obtain an upper bound for the probability of a random labelling to be separable with K hidden units:

$$P_K(P,N) = \min\left\{1,\; \frac{C(P,N)^K}{2^P}\right\} \tag{4.17}$$

where the minimum is applied to explicitly avoid $P_K(P,N) > 1$ resulting from the upper bound.

Further analytical treatment of Eq. (4.17) is involved, including the limit $N \to \infty$. However, we can obtain numerical estimates (upper bounds) of the storage capacity by identifying, for given K and N, the value of P for which[7]

$$\frac{C(P,N)^K}{2^P} = \frac{1}{2}. \tag{4.18}$$

This marks the characteristic point $\alpha_c(K) = P/N$ at which the probability $P_K(P,N)$ drops, which is in line with the observation that $C(2N,N)/2^{(2N)} = 1/2$ for the perceptron.

Example estimates obtained for $N = 1000$ and a few, small values of K are

$$\alpha_c(2) \approx 9.07 (\approx 2 \times 4.55),\; \alpha_c(6) \approx 40.53 (\approx 6 \times 6.76),\; \alpha_c(10) \approx 76.86 (\approx 10 \times 7.69)$$

which already shows that $\alpha_c(K)$ displays a superlinear dependence on K. In particular,

$$\alpha_c(K) > K \times 2 = K \times \alpha_c(1)$$

where the r.h.s. could be interpreted as a naive lower bound for combining K perceptrons without any synergy effect.

Figure 4.4 displays the numerical estimates for $N = 100$ and $K \leq 20$. One can show that for large values of K the capacity bound is given by [MD89a, Opp94, BO91]

$$\alpha_c(K) \leq\; K\, \ln K / \ln 2 \tag{4.19}$$

which is consistent with a faster than linear growth with the number of hidden units. The fact that the capacity grows rapidly with K agrees with the finding

[7] More precisely: the smallest integer P for which the upper bound exceeds 1/2.

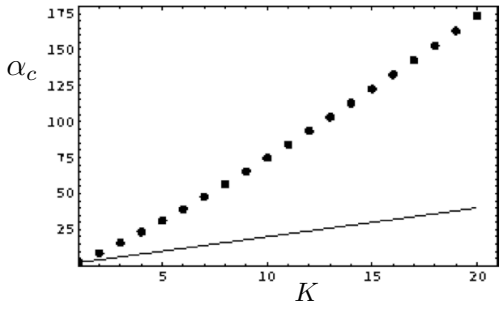


Figure 4.4: The storage capacity $\alpha_c(K)$ according to the estimate (4.18), obtained for $N = 100$. The solid line corresponds to the naive lower bound $2K$ for combining K perceptrons.

that a (parity) machine with a sufficiently large number of units can implement any given dichotomy.

In the context of Section 3.5.3, it also indicates that the number of hidden units should be carefully selected in order to avoid the realization of large storage capacity at the expense of poor generalization behavior.

The asymptotic $K \ln K$-dependence of the storage capacity for large K has been confirmed explicitly for the parity machine also by means of statistical physics based considerations [EB01, BSS94, WRB93, Opp94]. Interestingly, for the committee machine, the increase of α_c appears to be slightly weaker as $\alpha_c \propto K(\ln K)^{1/2}$ for $K \to \infty$ [Urb97].

This difference in the asymptotic growth of $\alpha_c(K)$ is consistent with our intuitive insight that the committee machine is subject to redundancies, while the parity machine makes full use of the separating hyperplanes in its hidden layer.

The storage capacities and/or VC dimensions of a variety of network types and architectures has been of great interest in the machine learning community. This is due to the role that the capacity plays with respect to the ultimate goal of learning, i.e. generalization. Examples from the statistical physics point of view can be found in [EB01, BSS94, WRB93, Opp94], see also references therein.

4.3 Support Vector Machines

The Support Vector Machine (SVM) constitutes one of the most successful frameworks in supervised learning for classification tasks. It combines the conceptual simplicity of the large margin linear classifier, a.k.a. the perceptron of optimal stability, with the power of general non-linear transformations to high-dimensional spaces. In particular, it employs the so-called *kernel trick* which makes it possible to realize the transformation implicitly.

For a detailed presentation of the SVM framework and further references see, for instance, [SS02, CST00, STC04, Her02, DFO20]. A comprehensive repository of materials, including a short history of the approach is provided at `www.svms.org` [svm99].

The concept of applying a kernel function in the context of pattern recognition dates back to at least 1964, see Aizerman et *al.* [ABR64]. Probably the

first practical version of Support Vector Machines, close to their current form, was introduced by Boser, Guyon and Vapnik in 1992 [BGV92] and relates to early algorithms developed by Vladimir Vapnik in the 1960s [VL63]. According to Isabelle Guyon [Guy16], the MinOver algorithm [KM87] triggered their interest in the concept of large margins which was then combined with the kernel approach. Eventually, the important and practically relevant extension to soft margin classification was introduced by Cortes and Vapnik in 1995 [CV95].

4.3.1 Non-linear transformation to higher dimension

The first important concept of the SVM framework exploits the fact that non-separable data sets can become linearly separable by means of a non-linear mapping of the form

$$\boldsymbol{\xi} \in \mathbb{R}^N \;\rightarrow\; \underline{\Psi}(\boldsymbol{\xi}) \in \mathbb{R}^M \quad \text{with components } \Psi_j(\boldsymbol{\xi}) \tag{4.20}$$

where M can be different from N, in general. A function of the form

$$S(\boldsymbol{\xi}) = \text{sign}\,[\underline{W} \cdot \underline{\Psi}(\boldsymbol{\xi})] \quad \text{with weights } \underline{W} \in \mathbb{R}^M \tag{4.21}$$

is by definition linearly separable in the space of transformed vectors $\underline{\Psi}$. So, while retaining the basic structure of the perceptron, formally, we will be able to realize functions beyond linearly separability by proper choice of the non-linear transformation $\boldsymbol{\xi} \rightarrow \underline{\Psi}$.

In fact, Rosenblatt already included this concept when introducing the Perceptron: the threshold function sign(...) is applied to the weighted sum of states in an *association layer*, cf. Fig. 3.1 showing the *Mark I Perceptron*. Its units are referred to as *masks* or *predicate units* in the literature [Ros58, HRM+60], see also [MP69]. In the hardware realization, for instance, 512 association units were connected to subsets of the 400 photosensor units and performed a threshold operation on an effectively randomized weighted sum of the incoming voltages, see [HRM+60] for details.

In the Support Vector Machine, the non-linear mapping is – in general – from an N-dimensional space to a higher-dimensional space with $M > N$ in order to achieve linear separability of the classes. As an illustration of the concept we discuss a simple example which was presented by Rainer Dietrich in [Die00]: Consider a set of two-dimensional feature vectors $\boldsymbol{\xi} = (\xi_1, \xi_1)^\top$, with two classes separated by a non-linear decision boundary as displayed in Fig. 4.5 (left panel). We apply the explicit transformation

$$\underline{\Psi}(\xi_1, \xi_2) \;=\; (\xi_1^2,\, \sqrt{2}\,\xi_1\,\xi_2,\, \xi_2)^\top \in \mathbb{R}^3$$

which is non-linear as it contains the square ξ_1^2 and the product $\xi_1\,\xi_2$. In the example, the plane orthogonal to the weight vector $\underline{W} = (1, 1, -1)^\top$ separates the classes perfectly in $M = 3$ dimensions, see the center and right panels of Fig. 4.5.

This is obviously only a toy example to illustrate the basic idea. In typical applications of the SVM the dimension N of the original feature space is already

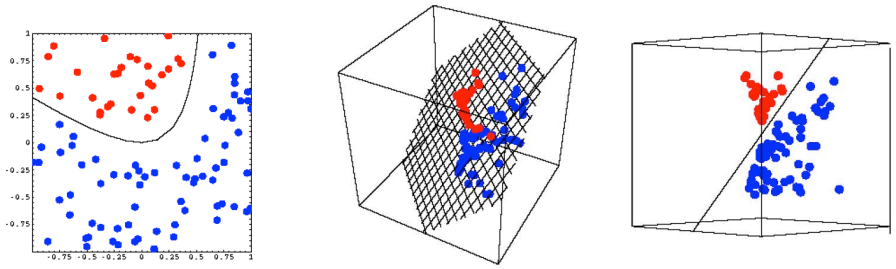


Figure 4.5: Illustration courtesy of Rainer Dietrich [Die00]. A two-dimensional data set with two classes that are not linearly separable (left panel) can become linearly separable after applying an appropriate non-linear transformation to a higher-dimensional space (center and right panel, two different viewpoints).

quite large and frequently M has to satisfy $M \gg N$ in order to achieve linear separability.

While the concept appears appealing, it is yet unclear how we should identify a suitable transformation $\boldsymbol{\xi} \to \underline{\Psi}$ for a given problem and data set. Before we return to this problem (and actually circumvent it elegantly), we discuss the actual training, i.e. the choice of a suitable weight vector $\underline{W}$ in the M-dimensional space.

4.3.2 Large margin classifier

Let us assume that for a given, non-separable data set $\mathbb{D}_N = \left\{\boldsymbol{\xi}^\mu \in \mathbb{R}^N, S_T^\mu\right\}_{\mu=1}^P$ we have found a suitable transformation such that

$$\mathbb{D}_M = \left\{\underline{\Psi}^\mu \in \mathbb{R}^M, S_T^\mu\right\}_{\mu=1}^P$$

is indeed linearly separable in M dimensions. Hence, we can apply conventional perceptron training in the M-dimensional space and, by means of the Perceptron Convergence Theorem (3.22), we are even guaranteed to find a solution.

However, in general, we do not have explicit control of (or reliable information about) how difficult the task will be. On the one hand, we would wish to use a powerful transformation to very high-dimensional $\underline{\Psi}$ in order to guarantee separability and make it easy to find a suitable $\underline{W}$. On the other hand, one could expect inferior generalization behavior in that case. Along the lines of the student-teacher scenarios discussed in Sec. 3.5.1, the corresponding version space of consistent hypotheses $\underline{W}$ might be unnecessarily large.

The SVM aims at resolving this dilemma by determining the solution of maximum stability $\underline{W}_{max}$. Hence, the potentially very large freedom in selecting a weight vector $\underline{W}$ in the high-dim. version space is efficiently restricted

and – following the arguments provided in Sec. 3.5.2 – we can expect good generalization ability.

The mathematical structure of the corresponding problem is fully analogous to the original (3.58). The M-dim. counterpart reads

PERCEPTRON OF OPTIMAL STABILITY (M-dim. feature space) (4.22)

For a given data set $\mathbb{D}_M = \{\underline{\Psi}^\mu, S_T^\mu\}_{\mu=1}^P$, find the vector $\underline{W}_{max} \in \mathbb{R}^M$

with $\underline{W}_{max} = \underset{\underline{W}}{\text{argmax}}\ \kappa(\underline{W})$ with $\kappa(\underline{W}) = \min\left\{\kappa^\mu = \frac{\underline{W}\cdot\underline{\Psi}^\mu S_T^\mu}{|\underline{W}|}\right\}_{\mu=1}^P$.

Obviously we can simply translate all results, concepts and algorithms from Sec. 3.6 to the transformed space.

So far we have assumed that the transformation $\boldsymbol{\xi} \to \underline{\Psi}$ exists and is explicitly known. Then we could for instance formulate and apply an M-dimensional version of the MinOver algorithm (3.59,3.60). Moreover, we can apply the optimization theoretical concepts and methods presented in Sec. 3.7 as exploited in the next sections. Among other aspects, this implies that the resulting classifier can be expressed in terms of support vectors, which ultimately motivates the use of the term Support Vector Machine.

4.3.3 The kernel trick

In analogy to the original stability problem, cf. Sec. 3.7, we can introduce the embedding strengths $\vec{X} = (X^1, X^2, ..., X^P)^\top \in \mathbb{R}^P$. With the shorthand $\underline{\Psi}^\mu = \underline{\Psi}(\boldsymbol{\xi}^\mu)$ we also define the correlation matrix

$$\Gamma \text{ with elements } \Gamma^{\mu\nu} = \tfrac{1}{M}\, S_T^\mu \underline{\Psi}^\mu \cdot \underline{\Psi}^\nu S_T^\nu \tag{4.23}$$

and analogous to Eqs. (3.76) we obtain

$$\underline{W} = \frac{1}{M}\sum_{\mu=1}^{P} X^\mu\, \underline{\Psi}^\mu\, S^\mu \quad \text{and } \underline{W}^2 = \frac{1}{M}\,\vec{X}^\top\,\Gamma\,\vec{X}. \tag{4.24}$$

Eventually, we can re-formulate the problem (4.22) as

PERCEPTRON OF OPTIMAL STABILITY (M-dim. feature space)

$$\underset{\vec{X}}{\text{minimize}}\ \ \frac{1}{2}\,\vec{X}^\top\Gamma\vec{X} \ \text{ subject to inequality constraints } \ \Gamma\vec{X} \geq \vec{1} \tag{4.25}$$

and proceed along the lines of Sec. 3.7 to derive, for instance, the corresponding AdaTron algorithm, see below.

The output of the M-dim. perceptron can be written as

$$S(\boldsymbol{\xi}) \;=\; \mathrm{sign}\Big[\underline{W}\cdot\underline{\Psi}(\boldsymbol{\xi})\Big] \;=\; \mathrm{sign}\left[\frac{1}{M}\sum_{\mu=1}^{P} X^{\mu}\,S^{\mu}\underline{\Psi}^{\mu}\cdot\underline{\Psi}(\boldsymbol{\xi})\right]. \tag{4.26}$$

We note that this involves the scalar products of the M-dimensional, transformed input vector with the transformed example training examples Ψ^{μ}. We define a so-called kernel function

$$K:\mathbb{R}^N\times\mathbb{R}^N\to\mathbb{R}\;\;\text{with}\;K(\boldsymbol{\xi},\widehat{\boldsymbol{\xi}}) \;=\; \frac{1}{M}\,\underline{\Psi}(\boldsymbol{\xi})\cdot\underline{\Psi}(\widehat{\boldsymbol{\xi}}) \;=\; \frac{1}{M}\sum_{j=1}^{M}\Psi_j(\boldsymbol{\xi})\,\Psi_j(\widehat{\boldsymbol{\xi}}) \tag{4.27}$$

which represents the scalar product in $\mathbb{R}^M$. We observe that

$$S(\boldsymbol{\xi}) \;=\; \mathrm{sign}\left[\sum_{\mu=1}^{P} X^{\mu}\,S^{\mu}K(\boldsymbol{\xi}^{\mu},\boldsymbol{\xi})\right] \tag{4.28}$$

does not involve the transformation $\Psi(\ldots)$ explicitly anymore. The kernel K is defined as a function of pairs of original feature vectors. Similarly, we have

$$E^{\mu} \;=\; \left[\Gamma\vec{X}\right]^{\mu} \;=\; S_T^{\mu}\sum_{\nu=1}^{P} S_T^{\nu}\,X^{\nu}\,K(\boldsymbol{\xi}^{\mu},\boldsymbol{\xi}^{\nu}). \tag{4.29}$$

One can also formulate the AdaTron algorithm for optimal stability in the M-dimensional space. The Kernel AdaTron was introduced and discussed in [FCC98] and has been applied in a variety of practical problems. In analogy to (3.98) it is given as

KERNEL ADATRON (sequential updates) (repeated presentation of $\mathbb{D}$)

– at time step t, present example $\mu = 1, 2, 3, ..., P, 1, 2, 3, ...$
– perform the update

$$X^{\mu}(t+1) \;=\; \max\left\{0,\; X^{\mu}(t)+\widehat{\eta}\left(1\;-\;S_T^{\mu}\sum_{\nu=1}^{P} S_T^{\nu}\,X^{\nu}\,K(\boldsymbol{\xi}^{\mu},\boldsymbol{\xi}^{\nu})\right)\right\}. \tag{4.30}$$

The training algorithm is also expressed in terms of the kernel and does not require explicit use of the transformation $\underline{\Psi}$, formally.
So far, the above insights suggest a strategy along the following lines:

a) For a given, non-separable $\mathbb{D}_N$, identify a suitable non-linear mapping $\boldsymbol{\xi}\to\underline{\Psi}$ from N to M dimensions that achieves linear separability of $\mathbb{D}_M$.

b) Compute the kernel function for all pairs of example inputs:
$K^{\mu\nu} = K(\boldsymbol{\xi}^{\mu},\boldsymbol{\xi}^{\nu}) = {}^1\!/_M\,\underline{\Psi^{\mu}}\cdot\Psi^{\nu}$.

c) Determine the embedding strengths $\vec{X}_{max}$ corresponding to optimal stability in the M-dim. weight space, for instance by use of the AdaTron (4.30).

d) Classify an arbitrary $\boldsymbol{\xi} \in \mathbb{R}^N$ according to

$$S(\boldsymbol{\xi}) \;=\; \text{sign}\left[\sum_{\mu=1}^{P} X^{\mu}_{max}\, S^{\mu} K(\boldsymbol{\xi}^{\mu}, \boldsymbol{\xi})\right]$$

In practice, of course, the problem is to find and implement a suitable transformation that yields separability in a given problem and data set. However, we observe that once step (a) is performed, the transformation $\boldsymbol{\xi} \to \underline{\Psi}$ is not explicitly used anymore. Even the weight vector $\underline{W}_{max}$ is not required explicitly: it is not directly updated in in the training (c), nor is it used for the classification in working phase (d). Instead, the representation (4.24) is used throughout.

Ultimately, this suggests to by-pass the explicit transformation in the first place and replace step (a) by

a') For a given, non-separable $\mathbb{D}_N$, identify a suitable kernel function $K(\boldsymbol{\xi}, \widehat{\boldsymbol{\xi}})$ and proceed from there as before.

This can only be mathematically sound if the selected kernel $K(\boldsymbol{\xi}, \widehat{\boldsymbol{\xi}})$ function represents some meaningful transformation, implicitly. It is obvious that for any transformation a kernel exists and we can work it out via the scalar products $\underline{\psi}(\boldsymbol{\xi}) \cdot \underline{\psi(\widehat{\boldsymbol{\xi}})}$. However, the reverse is less clear: given a particular kernel, can we guarantee that there is a *valid*, i.e. consistent, well-defined transformation? Fortunately, such statements can be made with respect to a large class of functions without having to work out the underlying $\boldsymbol{\xi} \to \underline{\Psi}$ explicitly.

Sufficient conditions for a kernel to be valid can be provided according to Mercer's Theorem [Mer09], see also [SS02,CST00,STC04,Her02]. Without going into the mathematical details and potential additional conditions, it can be summarized for our purposes as:

MERCER'S CONDITION (sufficient condition for validity of a kernel) (4.31)

A given kernel function K can be written as $K(\boldsymbol{\xi}, \widehat{\boldsymbol{\xi}}) = \dfrac{1}{M}\, \underline{\Psi}(\boldsymbol{\xi}) \cdot \underline{\Psi}(\widehat{\boldsymbol{\xi}})$,

with a transformation $\boldsymbol{\xi} \in \mathbb{R}^N \to \underline{\Psi} \in \mathbb{R}^M$ of the form (4.20), if

$$\iint g(\boldsymbol{\xi})\; K(\boldsymbol{\xi}, \widehat{\boldsymbol{\xi}})\; g(\widehat{\boldsymbol{\xi}})\;\, d^N\xi\; d^N\widehat{\xi} \;\geq 0$$

holds true for all square-integrable functions g with $\displaystyle\int\; g(\boldsymbol{\xi})^2 d^N\xi \;< \infty$.

Several families of kernel functions have been shown to satisfy Mercer's condition and are referred to as Mercer kernels, frequently. A few popular examples are discussed in the following.

Polynomial kernels

A polynomial kernel of degree q can be written as

$$K(\boldsymbol{\xi}^\mu, \boldsymbol{\xi}) = (1 + \boldsymbol{\xi}^\mu \cdot \boldsymbol{\xi})^q \quad \text{yielding} \quad S(\boldsymbol{\xi}) = \text{sign}\left[\sum_{\mu=1}^{P} X^\mu \, S_T^\mu \, (1 + \boldsymbol{\xi}^\mu \cdot \boldsymbol{\xi})^q\right] \tag{4.32}$$

as the input-output relation of the classifier.

As a special case, let us consider the simplest polynomial kernel:

Linear kernel ($q = 1$)

$$K(\boldsymbol{\xi}^\mu, \boldsymbol{\xi}) = (1 + \boldsymbol{\xi}^\mu \cdot \boldsymbol{\xi}) \quad \text{with } S(\boldsymbol{\xi}) = \text{sign}\left[\sum_{\mu=1}^{P} X^\mu \, S_T^\mu \, (1 + \boldsymbol{\xi}^\mu \cdot \boldsymbol{\xi})\right] \tag{4.33}$$

$$= \text{sign}\Bigg[\underbrace{\sum_{\mu=1}^{P} X^\mu \, S_T^\mu}_{\equiv M\,\Theta} + \underbrace{\sum_{\mu=1}^{P} X^\mu \, S_T^\mu \, \boldsymbol{\xi}^\mu}_{\equiv M\underline{W}} \cdot \boldsymbol{\xi}\Bigg].$$

In this case, we can provide an immediate, almost trivial interpretation of the kernel: it corresponds to the realization of a linearly separable function in the original feature space ($M = N$) with

$$\text{weights} \quad \underline{W} \mathrel{\widehat{=}} \mathbf{w} = \frac{1}{M} \sum_{\mu=1}^{P} X^\mu \, S_T^\mu \, \boldsymbol{\xi}^\mu \qquad \text{and off-set} \quad \Theta = \frac{1}{M} \sum_{\mu=1}^{P} X^\mu S_T^\mu.$$

The *SVM with linear kernel* is applied very frequently in practice. There is even an unfortunate trend to refer to it as *"the SVM"*. However – strictly speaking – the SVM is not a classifier but a framework for classification: it has to be specified by defining the kernel in use. In particular, the linear kernel reduces the SVM to the familiar perceptron of optimal stability (with local threshold Θ).

In order to take full advantage of the SVM concept, we have to employ more sophisticated kernels. The first non-trivial choice beyond linearity corresponds to $q = 2$:

Quadratic kernel ($q = 2$)

$$K(\boldsymbol{\xi}^\mu, \boldsymbol{\xi}) = (1 + \boldsymbol{\xi}^\mu \cdot \boldsymbol{\xi})^2 \; = \; 1 + 2\sum_{j=1}^{N} \xi_j^\mu \, [\xi_j] \; + \sum_{j,k=1}^{N} \xi_j^\mu \, \xi_k^\mu \; [\xi_j \, \xi_k]. \tag{4.34}$$

Hence, the output $S(\boldsymbol{\xi})$ in Eq. (4.32) with $q = 2$ corresponds to an inhomogeneously linearly separable function in terms of the N original features augmented by $N(N+1)/2$ products of the form $[\xi_j \xi_k]$ which includes the squares of features for $j = k$.

As intuitively expected, the use of the quadratic kernel represents the nonlinear mapping from N-dim. feature space to in total $M = N(N+3)/2$ transformed features (original, squares and mixed products). An explicit formulation is reminiscent of Quadratic Discriminant Analysis (QDA) [HTF01], albeit aiming at a different objective function in the training.

Similarly, for the general polynomial kernel (4.32) the separating class boundary becomes a general polynomial surface and the dimension M of the transformed feature space grows rapidly with its degree q.

Next, we consider a somewhat extreme, yet very popular choice:

Radial Basis Functions (RBF) kernel

$$K(\boldsymbol{\xi}^\mu, \boldsymbol{\xi}) \;=\; \exp\left[-\frac{1}{2\,\sigma^2}\left(\boldsymbol{\xi}^\mu - \boldsymbol{\xi}\right)^2\right] \tag{4.35}$$

which involves the squared Euclidean distance and a width parameter σ.

In an attempt to interpret the popular RBF Kernel along the lines of the discussion of polynomial kernels we could consider the Taylor series

$$\exp[x] = \sum_{k=1}^{\infty} \frac{1}{k!}\, x^k \;=\; 1 + x + \frac{1}{2}x^2 + \frac{1}{6}x^3 + \frac{1}{24}x^4 + \ldots$$

which shows that the dimension of the corresponding space would be $M \to \infty$ as all powers and products of the original features are involved.

The RBF kernel has become one of the most popular choices in the literature. The fact that an SVM with this extremely powerful kernel with, formally, $M \to \infty$ can generalize at all demonstrates the importance of the restriction to optimal stability (the large margin concept) which constitutes an efficient regularization of the classifier.

4.3.4 A few remarks

Selection of kernels and parameter setting

In practice, the choice of the actual kernel function can influence the performance of the corresponding classifier significantly. In addition, kernels may contain parameters which have to be tuned to suitable values by means of validation techniques. The RBF-kernel is just one example of kernels that feature a control parameter: the width σ in Eq. (4.35). The data-driven adaptation of kernel parameters as part of the training process has also been discussed in the literature, see [CCST99] for just one example.

Inhomogeneous separation of classes

In analogy to the perceptron of optimal stability, see Sec. 3.8, an offset from the origin of the high-dimensional feature space can be considered in the SVM. For the sake of simplicity we have restricted the discussion to the homogeneous case and refer the reader to the literature w.r.t. the conceptually straightforward extension to inhomogeneous separation, see e.g. [CST00, SS02].

Soft-margin SVM

In addition to the choice of the kernel and potential parameters thereof, one often resorts to a soft margin version of the SVM [CV95], see also [SS02, CST00, STC04, Her02, DFO20]. The considerations of Sec. 4.1.2 for the simple perceptron immediately carry over to the SVM formalism, once a kernel is defined.

The modified optimization problems (4.4) and (4.5) are easily generalized to the case of the Support Vector Machine by replacing the embedding strengths with $\vec{X}$ and the correlation matrix by Γ from Eq. (4.24) and (4.23), respectively.

Consequently, we can immediately derive suitable training algorithms for the soft margin SVM. For instance, the "AdaTron with errors" algorithm (4.6) carries over to the kernel-based formulation in a straightforward fashion.

The soft margin extension introduces an additional parameter in the training process: the parameter γ in (4.4) implicitly controls the tolerance of constraint violations (or even misclassifications). Like potential parameters of the kernel, it should be determined by means of a suitable validation procedure.

Overfitting

In the early days of the SVM, occasionally the claim was made that the strong intrinsic regularization related to the large margin idea would eliminate the risk of overfitting to a large extent, if not completely. However, practice shows that the use of too complex kernels or low tolerance towards misclassification can result in poor generalization.

Interestingly, the SVM offers a signal of overfitting which does not even require the explicit estimation of the generalization error: the number of support vectors n_s with nonzero embedding strengths $X^\mu > 0$. A relatively high fraction n_s/P indicates that the classifier may be overly specific to the given data set. The fact that only very few examples in the data set are stabilized by embedding the others suggests inferior classification performance with respect to novel data in the working phase. This indication can be exploited for the choice of a suitable kernel to begin with, for the tuning of its parameters and for the choice of control parameters in the optimization.

Efficient implementations

The remark at the end of Sec. 3.7.3 concerning computational efficiency and scalability carries over to the kernel-based SVM as well.

Efficient implementations, for instance based on the concept of *Sequential Minimal Optimization (SMO)* [Pla98] are available for a variety of platforms. As just one source of information, the reader is referred to a list of links provided at www.svms.org/software.html [svm99].

Chapter 5

Feed-forward networks for regression and classification

The fishermen in the north of Spain have been using Deep Networks for centuries. Their contribution should be recognized ...

— Javier Movellan

Layered Neural Networks have regained significant popularity due to their impressive successes in the context of *Deep Learning* applications such as image classification. The basic designs and training techniques have been established decades ago.

In the previous chapter we presented examples of layered networks constructed from perceptron-like threshold units. The by far most popular type of networks comprise continuous units with differentiable activation functions. In the following, we consider the use of such networks for regression and probabilistic classification and discuss their ability to approximate continuous functions. We present methods for their training by optimization techniques like gradient descent and variants thereof. Furthermore, we inspect specific example architectures and very briefly address the use of deep networks with many hidden layers and strategies for their training.

5.1 Feed-forward networks as non-linear function approximators

We first revisit the basic architecture and definition of strictly feed-forward, layered networks. We show that, in principle, suitable networks can approximate

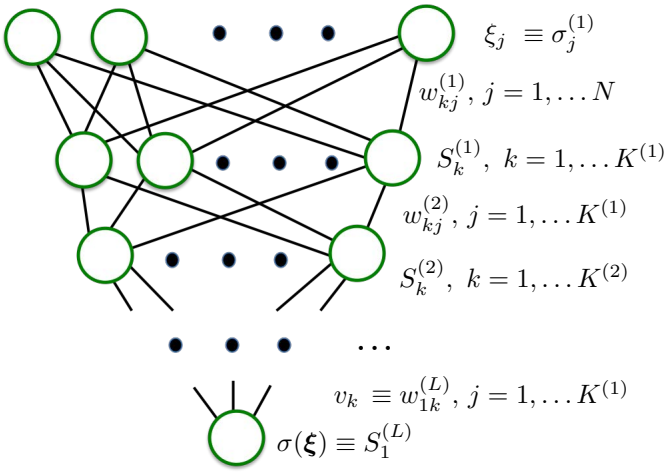


Figure 5.1: A feed-forward neural network with N input units, $L-1$ hidden layers and a single output unit.

any reasonable function from $\mathbb{R}^N$ to $\mathbb{R}$. This implies that layered networks can serve as tools for quite general problems of non-linear regression. In section 5.1.2 we show explicitly, that suitable layered networks can approximate any reasonable function to arbitrary precision. Cost function based training for the learning from examples is discussed in 5.2 with emphasis on gradient based methods.

5.1.1 Architecture and input-output relation

Figure 5.1 displays a multilayer neural network with a single output unit. The generalization to several output units is formally straightforward. The figure suggests a convergent architecture with the number of hidden units per layer decreasing towards the output. While in practice this is frequently the case, it is by no means required in the following.

The network in Fig. 5.1 is strictly feed-forward, i.e. the state of a particular hidden unit depends directly and only on the nodes in the previous layer. The resulting hidden unit activation is

$$S_k^{(M)} = g_k^{(M)} \left(\sum_{j=1}^{K^{(M-1)}} w_{kj}^{(M)} S_j^{(M-1)} \; - \; \theta_k^{(M)} \right) \tag{5.1}$$

where adaptive weights $w_{kj}^{(M)}$ connect the j-th unit in layer $(M-1)$ to the k-th unit of layer M and $\theta_k^{(M)}$ denotes an adaptive local threshold. Alternatively, we could introduce an additional, *clamped unit* $S_0^{(M)} \equiv -1$ in each layer and

represent the local threshold by a weight $w_{k0}^{(M-1)} \equiv \theta_k^{(M)}$. This would parallel our representation of inhomogeneously linear separable functions in Eq. (3.4).

We can include the input layer in the the notation of Eq. (5.1) by defining $S_j^{(0)} \equiv \xi_j$. Similarly, we can rename the single output as $S_1^{(L)} \equiv \sigma$ in the L-th layer with $K^{(L)} = 1$:

$$\begin{aligned} \sigma(\boldsymbol{\xi}) \equiv S_1^{(L)} &= g^{out}\left(\sum_{k=1}^{K^{(L-1)}} v_k\, S_k^{(L-1)} - \theta^{out}\right) \\ &= g_1^{(L)}\left(\sum_{k=1}^{K^{(L-1)}} w_{1k}^{(L)}\, S_k^{(L-1)} - \theta_1^{(L)}\right) \end{aligned} \tag{5.2}$$

with the alternative notations $v_k \equiv w_{1k}^{(L)}$ and $\theta^{out} \equiv \theta_1^{(L)}$.

The output according to Eq. (5.2) together with (5.1) can be interpreted as a function

$$\sigma : \mathbb{R}^N \to \mathbb{R}.$$

The precise form of the input-output relation is determined by the network architecture including its connectivity and the activation functions. Frequently we will assume that the same transfer function $g(\ldots)$ defines the activation of all hidden units, while the output might result from a specific activation $g^{out}(\ldots)$.

5.1.2 Universal approximators

Following the previous section we can use a feed-forward network with a single output to implement a function from $\mathbb{R}^N$ to $\mathbb{R}$. It is a quite common concept to realize or approximate functional dependencies by the superposition of specific basis functions. A most prominent example is the representation in terms of Fourier series which exploit properties of the trigonometric basis functions. Coefficients are chosen as to approximate a target function to a required precision. Frequently, they are fitted to a set of discrete points and the resulting series is used to interpolate or even extrapolate. Hence, the situation is reminiscent of more general non-linear regression.

Neural networks of the form discussed in the previous section can be seen as a particular framework for functional approximation: for a given architecture and connectivity, the function $\sigma : \mathbb{R}^N \to \mathbb{R}$ is parameterized by the choice of all adaptive quantities, i.e. weights and thresholds.

In the following we will see that the considered type of layered neural networks can approximate virtually any function[1] to arbitrary precision.

A network for piecewise constant approximation

Here we show by construction that any reasonable function $f : \mathbb{R}^N \to \mathbb{R}$ can be approximated by a layered neural network comprising sigmoidal and linear

[1] Mathematical subtleties are ignored here to some extent.

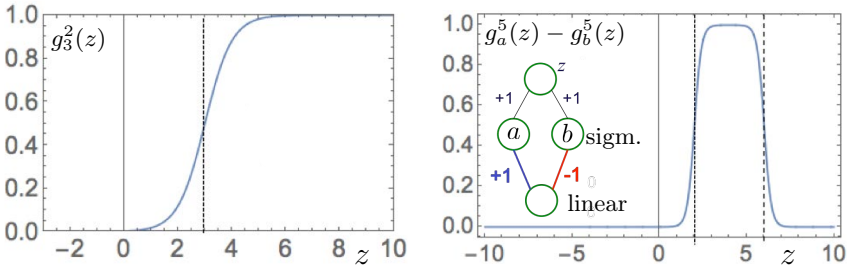


Figure 5.2: Left panel: A sigmoidal activation of the form (5.3) with $\gamma = 2, z_o = 3$ as an example. **Right panel:** The difference of two steep sigmoidals (here: $\gamma = 5, a = 2, b = 6$, respectively) singles out arguments $a \leq z \leq b$. The inset shows a graphical representation of the sigmoidals with equal γ connected to input z with thresholds a, b, respectively, and a linear unit computing the difference of their activations.

units. More precisely, we will consider functions which map inputs from a compact subset of $\mathbb{R}^N$ to a real valued output. We restrict the argument to feature vectors from the hypercube $\xi \in [0,1]^N$, which can always be generalized by transformations like rescaling and translation in input space.

Let us first consider sigmoidal neurons with, for instance, the activation

$$g^\gamma_{z_o}(z) = \frac{1}{1+\exp[-\gamma(z-z_o)]}, \tag{5.3}$$

with the argument $z \in \mathbb{R}$, threshold z_o and steepness parameter $\gamma > 0$, see Figure 5.2 (left panel) for an example. Two (steep) sigmoidal units can be combined in order to select a range of z-values, effectively:

$$G^\gamma_{[a,b]}(z) = g^\gamma_a(z) - g^\gamma_b(z) = \frac{1}{1+e^{-\gamma(z-a)}} - \frac{1}{1+e^{-\gamma(z-b)}} \approx \begin{cases} 1 & \text{if } a < z < b \\ 0 & \text{else.} \end{cases} \tag{5.4}$$

where the approximate identity becomes exact in the limit $\gamma \to \infty$.[2] This is illustrated in the right panel of Fig. 5.2.

For N-dim. inputs $\boldsymbol{\xi}$ we can realize the selection of a specific interval $[a_i, b_i]$ for each dimension $i = 1, 2, \ldots N$ separately, defining regions of interest (ROI)

$$R^{(j)} = [a_1^{(j)}, b_1^{(j)}] \times [a_2^{(j)}, b_2^{(j)}] \ldots \times [a_N^{(j)}, b_N^{(j)}], \quad j = 1, 2, \ldots M. \tag{5.5}$$

These can be constructed to cover the volume of all possible inputs $[0,1]^N$, which implies that M grows exponentially with N.

If we add up the corresponding N activations for one region of interest we have

$$\sum_{i=1}^N G^\gamma_{[a_i,b_i]}(\xi_i) \begin{cases} \approx N & \text{if } \boldsymbol{\xi} \in R \\ \leq N-1 & \text{if } \boldsymbol{\xi} \notin R. \end{cases}$$

[2] For the argument we can ignore the difference between open and closed intervals.

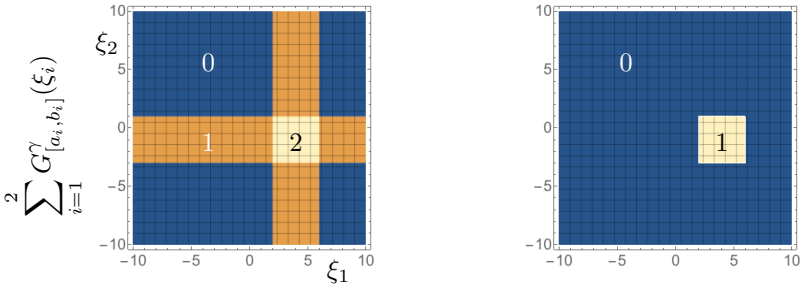


Figure 5.3: N threshold nodes (steep sigmoidal units) which select a specific interval per input dimension can be combined by adding up their activation. The **left panel** shows an illustration for $N = 2$ and $\xi_j \in [-10, 10]$. Only where all threshold units are activated, the sum reaches its maximum N. For $\boldsymbol{\xi} \notin R$, cf. Eq. 5.5, the total activation is $\sum_{i=1}^{2} G^{\gamma}_{[a_i,b_i]}(\xi_i) \leq N - 1$. Consequently, an additional threshold operation can be employed to single out a region of interest $R \subset \mathbb{R}^N$, as illustrated in the **right panel**.

Here we consider one particular ROI and have omitted the superscript (j) for simplicity. Another threshold unit, i.e. a steep sigmoidal with $z_o = N - 1/2$ can be applied to the sum to single out inputs $\boldsymbol{\xi} \in R$, see Fig. 5.3 for an illustration with $N = 2$.

Figure 5.4 displays the architecture of a layered net in which M such units correspond to the ROI of Eq. (5.5). Eventually, we select one representative value $v_j = f(\boldsymbol{\xi}^{(j)})$ of the target function in each ROI, for instance with $\boldsymbol{\xi}^{(j)}$ in the center of $R^{(j)}$. The v_j serve as weights for feeding the activations ϑ_j into a simple, linear output unit. Therefore, the resulting network response

$$\sum_{j=1}^{M} v_j \vartheta_j(\boldsymbol{\xi}) \;=\; v_k \quad \text{for} \quad \boldsymbol{\xi} \in R^{(k)} \tag{5.6}$$

amounts to a piecewise constant approximation of the target function in $[0, 1]^N$.

The constructive argument shows that the network in principle constitutes a universal approximator. However, a few remarks are in place:

- In order to achieve an accurate approximation of any non-trivial function, we would have to realize a rather fine-grained representation of input space. Assuming that we split $[0, 1]$ into, say, k equal size intervals in each of the N feature dimensions, the network of Fig. 5.4 would comprise $\mathcal{O}(Nk)$ units in the second and third layer, which appears reasonable. However, representing all possible combinations of N intervals in the fourth layer requires on the order of $\mathcal{O}\left(k^N\right)$ hidden units.

- The idea of *training* the feed-forward network by means of example data appears somewhat obscured. The simple-minded setting of the weights $v_j = f(\boldsymbol{\xi}^{(j)})$ in Eq. (5.6) could be interpreted as *learning* from one example per ROI. However, in view of the previous remark, the procedure

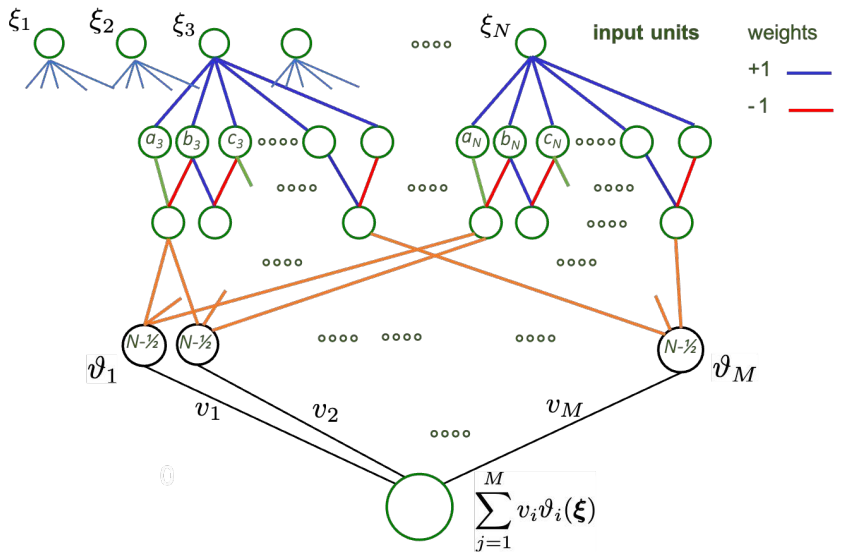


Figure 5.4: The constructed network for piecewise constant function approximation: Each input unit (top layer) is connected to a set of sigmoidal units in the second layer. Pairs of these connect to linear units in the third layer which select specific intervals $[a_i, b_i], [b_i, c_i], \ldots$ as illustrated in Fig. 5.2. Each unit marked as ϑ_j in the fourth layer performs a threshold operations on a particular sum of N activations in the third layer, corresponding to one selected interval in each feature dimension. An activation $\vartheta_j = 1$ indicates that $\boldsymbol{\xi} \in R^{(j)}$ and the resulting state of the linear output unit is given by $v_i = f(\boldsymbol{\xi}^{(i)})$ which corresponds to a representative $\boldsymbol{\xi}^{(i)} \in R^{(i)}$.

would require a number P of examples that grows exponentially with the dimension N.

- In this sense, the construction of ROI parallels the use of *grandmother neurons* when showing that the parity machine is a universal classifier in Section 4.2.

While the argument justifies and supports the use of feed-forward networks in principle, it does not provide insight into how to design and how to train such a system in practice.

Note that the number of layers required for universal approximation is limited. In the above construction scheme, four layers of processing units are sufficient. Obviously, *shallow* networks are sufficient to achieve this property. However, this does not imply that shallow architectures are necessarily suitable for all practical applications. In fact, the recent success of *deep* networks appears to suggest the contrary, in many cases.

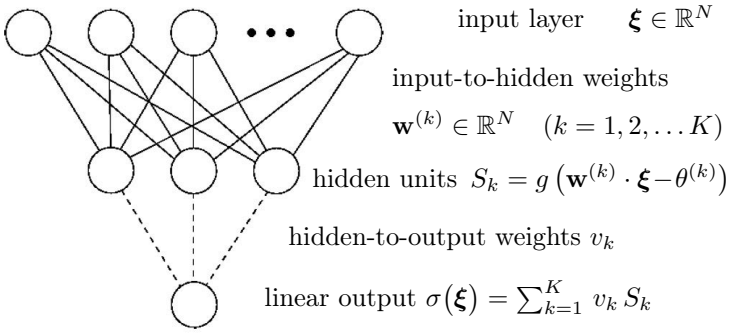


Figure 5.5: A so-called *Soft Committee Machine* realizes a functional approximation as considered in Cybenko's Theorem. A number K of hidden units with sigmoidal activation and adaptive thresholds $\theta^{(k)}$ are connected through adaptive weight vectors $\mathbf{w}^{(k)}$ with the N-dim. input layer, in the illustration $K = 3$. The linear response $\sigma(\boldsymbol{\xi})$ is determined as $\sigma = \sum_{k=1}^{K} v_k\, g\left(\mathbf{w}^{(k)} \cdot \boldsymbol{\xi} - \theta^{(k)}\right)$ with hidden-to-output weights $v_k \in \mathbb{R}$.

Variants of the Universal Approximation Theorem

In the literature various incarnations of the *Universal Approximation Theorem* have been presented, differing in the degree of generality and practical relevance. Some of them address particular classes of target functions, others focus on specific types of activation functions or network architectures, see [Hor89, Fri94, LLPS93, SM15, MM92, CMB00] for examples and reviews.

In the following we present an early and important formulation which is due to G. Cybenko [Cyb89]. The theorem states that a relatively simple network with a single hidden layer of sigmoidal units and a linear output is a universal function approximator:

CYBENKO'S THEOREM (5.7)

Consider inputs $\boldsymbol{\xi} \in [0,1]^N$ and a continuous, sigmoidal function

$$g : \mathbb{R} \to \mathbb{R} \;\text{ with }\; g(z) \to \begin{cases} 1 & \text{for } z \to +\infty \\ 0 & \text{for } z \to -\infty. \end{cases}$$

Let $\mathbf{w}^{(k)} \in \mathbb{R}^N, v_k \in \mathbb{R}, \theta^{(k)} \in \mathbb{R}$ for $k = 1, 2, \ldots K$.

Then, finite sums of the form

$$\sigma(\boldsymbol{\xi}) \;=\; \sum_{k=1}^{K} v_k\, g\left(\mathbf{w}^{(k)} \cdot \boldsymbol{\xi} - \theta^{(k)}\right)$$

are dense in the space of continuous functions $C([0,1])^N$.

This implies that for any continuous target function $\tau \in C([0,1]^N)$ and a given real number $\varepsilon > 0$, parameters

$$\left\{ \mathbf{w}^{(k)} \in \mathbb{R}^N, v_k \in \mathbb{R}, \theta^{(k)} \in \mathbb{R} \right\}_{k=1}^{K} \text{ exist with } \left| \sigma(\boldsymbol{\xi}) - \tau(\boldsymbol{\xi}) \right| < \varepsilon \text{ for all } \boldsymbol{\xi} \in [0,1]^N .$$

The parameters can be interpreted as the weights and thresholds of a network with a single hidden layer and linear output which is illustrated in Figure 5.5. The term *Soft Committee Machine* has been coined for this architecture, see [Saa99] and references therein. The name refers to the network's similarity with the (discrete output) committee machine for classification tasks which is discussed in Section 4.2.
In a sense, Cybenko's Theorem provides a stronger and more useful statement than the basic insight obtained by the construction in the previous section. However, the problem remains that a very large number K of hidden units might be required to exploit the approximation property in practice. Moreover, the theorem itself states only the existence of suitable parameters, it does not suggest how to find them.

The choice of appropriate network parameters based on example data is addressed in the following sections. We focus on training schemes which are based on the minimization of appropriate cost functions by means of gradient descent techniques.

5.2 Gradient based training of feed-forward nets

As we have seen, feed-forward neural networks can serve as universal function approximators. Hence it appears natural to employ them in the context of non-linear input/output relations which correspond to a real-valued target function $\sigma : \mathbb{R}^N \to \mathbb{R}$.

Extensions to multiple continuous outputs are obviously possible. Likewise, classification schemes could be realized by an additional binary threshold operation on the output σ or by appropriate *binning* in the case of multiple classes. Class memberships could also be represented by coding schemes applied to a number of output units as discussed in a forthcoming section.

Formally, we concatenate all M adaptive parameters of a feed-forward network in one vector $\underline{W} \in \mathbb{R}^M$. The convenient *flattened* notation facilitates a unified discussion of various network architectures in the following. In the Soft Committee Machine displayed in Fig. 5.5 as just one example, we have

$$\underline{W} = \left(w_1^{(1)}, \ldots\ldots, w_N^{(K)}, \theta^{(1)}, \ldots, \theta^{(K)}, v_1, \ldots, v_K \right) \in \mathbb{R}^M \quad \text{with} \quad M = KN+2K$$

and we can refer to any of the adaptive parameters as a component W_j.

For simplicity we focus on networks with a single, continuous output $\sigma(\boldsymbol{\xi})$ in the following. The goal of training is to implement or approximate a target function $\tau : \mathbb{R}^N \to \mathbb{R}$ by adapting a given network architecture to a given set of examples $\mathbb{D} = \{\boldsymbol{\xi}^\mu, \tau(\boldsymbol{\xi}^\mu)\}_{\mu=1}^{P}$.

To this end, we define an error measure which is suitable for the comparison of the network output $\sigma(\boldsymbol{\xi})$ with the target function $\tau(\boldsymbol{\xi})$ for a given input vector. A popular and intuitive choice is the simple quadratic deviation

$$e(\sigma, \tau) \;=\; \frac{1}{2}\,(\sigma - \tau)^2\,. \tag{5.8}$$

Here and in the following, shorthands $\sigma = \sigma(\boldsymbol{\xi}), \sigma^\mu = \sigma(\boldsymbol{\xi}^\mu), \tau = \tau(\boldsymbol{\xi})$ and $\tau^\mu = \tau(\boldsymbol{\xi}^\mu)$ refer to a generic input $\boldsymbol{\xi}$ or a particular example input $\boldsymbol{\xi}^\mu \in \mathbb{D}$, respectively.

While many alternative measures can be considered, see the discussion in Section 5.3, the quadratic error (5.8) remains very popular and is particularly intuitive. It treats deviations $\sigma > \tau$ and $\sigma < \tau$ in a symmetric way and yields $e = 0$ only for perfect agreement with the target.

Given a set of examples $\mathbb{D}$ we can define a corresponding training set specific cost function[3]:

$$E(\underline{W}) \;=\; \frac{1}{P}\sum_{\mu=1}^{P} e^\mu \quad \text{with} \;\; e^\mu = e(\sigma^\mu, \tau^\mu). \tag{5.9}$$

In the context of regression, $E(\underline{W})$ plays the role of the *training error* and quantifies the network performance with respect to $\mathbb{D}$. As in classification, the expectation is that the trained network represents a hypothesis that can be applied successfully to novel data in the working phase.

A plethora of numerical optimization methods could be used to optimize the cost function in practice. Most of these employ local gradient information or higher order derivatives of $E(\underline{W})$ in order to iteratively find a (local) minimum of the cost function. If possible, derivatives can be computed analytically or are estimated numerically. A few prominent examples are the so-called Newton and quasi-Newton methods, conjugate gradient descent, Levenberg-Marquardt algorithm or line search methods. Here we point the reader to the literature, e.g. [Fle00, PAH19, Str19, DFO20, SNW11, Bis95a, HKP91] where an overview can be obtained and further references are provided.

Relatively simple gradient descent techniques have been particularly popular in the context of machine learning for decades, as a very early example we have already discussed the Adaline algorithm [WH60] in Sec. 3.7.2.

From an optimization theoretical point of view, simple gradient descent is certainly inferior to state-of-the art methods. However, it remains an important, very popular tool in many machine learning frameworks, including powerful multi-layered architectures in Deep Learning. This is due to the conceptual simplicity and *hands-on* character of the descent and the fact that the mathematical structure of feed-forward networks appears particularly suitable for the computation of the required gradients.

Moreover, although the training is generically formulated as an optimization, the actual minimization of E serves only as a *proxy* for the ultimate goal, which

[3]Other terms frequently used in this context are: objective function, loss function, or *energy*.

is the successful application of the trained system to novel data. Hence, the precision to which a minimum is determined can play a minor role and the potential existence of many (suboptimal) local minima of E is not as problematic as one might expect.

5.2.1 Computing the gradient: Backpropagation of Error

A key property of feed-forward layered neural networks with differentiable activation functions is that the network output itself is a differentiable function of the inputs. Likewise, the output and the error measure $e(\sigma, \tau)$ are differentiable with respect to the adaptive parameters in the network for any given input $\boldsymbol{\xi}$:

$$\frac{\partial e(\sigma, \tau)}{\partial W_j} = (\sigma - \tau) \, \frac{\partial \sigma}{\partial W_j}.$$

Consequently, also the data set specific cost function $E = \sum_{\mu=1}^{P} e(\sigma^\mu, \tau^\mu)$ is a differentiable function of all components of $\underline{W}$.

For strictly feed-forward architectures as shown in Figure 5.1 we can obtain derivatives of σ with respect to any network parameter W_j recursively by applying the chain rule [Bis95a, Bis06, HKP91]. The mathematical structure facilitates a very efficient calculation of the gradient: Weights and thresholds serve as coefficients when computing the output for a given input in a feed-forward network. The actual output is compared with the target and the *error* is said to *propagate back* (towards the input) when computing the derivatives in a layer-wise fashion, which involves the very same coefficients again. Gradients for an example of a shallow network architecture are worked out explicitly in Appendix A.6.

The term *Backpropagation of Error* (*Backpropagation* or *Backprop* for short) was originally used for the efficient implementation of the gradient only [RM86]. Later it became synonymous with the the entire gradient based training of multi-layered feed-forward networks and is nowadays mostly used in this sense.

The ambiguity of the term partly complicates the on-going debates about who invented Backpropagation or coined the term Deep Learning.[4] Here we refrain from taking part in these discussions of questionable usefulness. For some original articles and reviews of the history of Backpropagation, see e.g. [Wer74, LBH18, CR95, Ama93, WL90, HKP91, GBC16].

5.2.2 Batch gradient descent

In principle, the minimization of $E(\underline{W})$ could be done by any suitable method of non-linear optimization. In the context of layered neural networks, relatively simple gradient based techniques continue to play a very important role. Here we focus on the use of standard gradient descent. Gradient based techniques are also discussed in Appendix A.4.

[4] For an example thread initiated by J. Schmidhuber in the *connectionists* mailing-list see http://mailman.srv.cs.cmu.edu/pipermail/connectionists/2021-December/037086.html

In analogy to Eq. (A.39) in the Appendix, the basic form of the updates is given as

BATCH GRADIENT DESCENT (basic form)
at discrete time step t perform an update step of the form

$$\underline{W}(t+1) = \underline{W}(t) - \eta \; \nabla_W E|_{\underline{W}=\underline{W}(t)} \tag{5.10}$$

with the learning rate η and cost function E of the form (5.9).

At each time step, the gradient with respect to all adaptive quantities $\underline{W}$ is computed as a sum over all examples in $\mathbb{D}$:

$$\nabla_W E \; = \; \frac{1}{P} \sum_{\mu=1}^{P} \nabla_W e^\mu \; = \; \frac{1}{P} \sum_{\mu=1}^{P} \Big(\sigma(\boldsymbol{\xi}^\mu) - \tau(\boldsymbol{\xi}^\mu) \Big) \nabla_W \sigma(\boldsymbol{\xi}^\mu), \tag{5.11}$$

where the r.h.s. is given for the quadratic error (5.8) but can be worked out for alternative cost functions as well.

The terms *batch* or *offline* gradient descent refer to the fact that the entire set $\mathbb{D}$ of example data is used in every update. Careful changes of $\underline{W}$ in the direction of $-\nabla_W E$ decrease the value of the cost function in each individual step and, consequently, the descent approaches some local minimum $\underline{W}^*$ of the cost function. If several local minima [5] exist, the actual stationary $\underline{W}^*$ depends on the initialization $\underline{W}(t=0)$ of the system.

The specific form of $\nabla_W \sigma$ has to be worked out by means of the chain rule in layered networks as explained in the previous section. It depends obviously on the network architecture, the activation functions and the set of all adaptive quantities in the system. In Appendix A.6 a specific example is given for a Soft Committee Machine, c.f. Section 5.1.2.

The general discussion of gradient descent in the appendix shows that its convergence near a local minimum $\underline{W}^*$ of E is governed by the symmetric, positive definite Hessian of second derivatives

$$H^* = H(\underline{W}^*) \in \mathbb{R}^{M \times M} \text{ with elements } H^*_{ij} = H^*_{ji} = \left. \frac{\partial^2 E(\underline{W})}{\partial W_i \partial W_j} \right|_{\underline{W}=\underline{W}^*} . \tag{5.12}$$

In a local minimum, all eigenvalues $\{\rho_i\}_{i=1}^M$ of H^* are positive and can be sorted by magnitude:

$$0 < \rho_1 \leq \rho_2 \leq \; \ldots \; \leq \rho_{max}.$$

We show in App. A.4.2 that with a given, constant learning rate $\eta > 0$ the following qualitative behavior can be expected near a local optimum $\underline{W}^*$:

[5] The term refers to local properties of the function and possibly includes global minima.

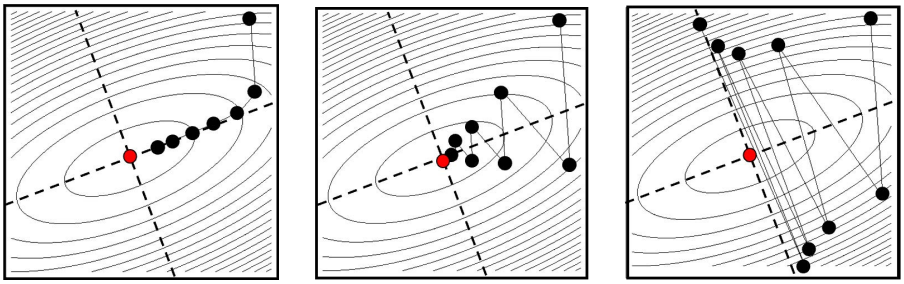


Figure 5.6: Illustration of the behavior of gradient descent near a local minimum $\underline{W}^*$ (marked by the red dot in the center). The contour lines represent the quadratic approximation of $E(\underline{W})$ in the vicinity of a local minimum, the black symbols correspond to the iterates $\underline{W}(t)$ in Eq. (5.10). For small step size η **(panel a)** the iteration converges smoothly into the minimum, intermediate step sizes **(panel b)** result in convergent yet oscillatory behavior. Failure to converge is observed for too large step sizes **(panel c)**.

(a) $\boldsymbol{\eta \leq 1/\rho_{max}}$

For small, finite learning rates, the iteration approaches the local minimum smoothly and converges to $\lim_{t\to\infty} \underline{W}(t) = \underline{W}^*$, see Fig. 5.6 (a). Obviously, with very small rates $\eta \approx 0$, the approach can become unnecessarily slow.

(b) $\boldsymbol{1/\rho_{max} < \eta \leq 2/\rho_{max}}$

In this regime, convergence is still achieved, but in at least one of the eigendirections of H^* an oscillatory behavior is observed. As illustrated in panel (b) of Fig. 5.6, the alternating behavior occurs in eigendirections with large curvature, which correspond to *narrow troughs* in the *landscape* $E(\underline{W})$, while the approach towards $\underline{W}^*$ is smooth along directions of small ρ_i in which E resembles a *shallow basin*.

(c) $\boldsymbol{\eta > 2/\rho_{max}}$

Too large learning rates result in divergent behavior of the iterations as displayed in Fig. 5.6 (c). Depending on η in relation to the individual ρ_i, the distance from the minimum can increase in one, several, or all eigendirections of H^*.

This insight is valid for any local minimum of E. However, two important points should be noted. Firstly, the analysis presented in App. A.4.2 is only valid close to a local minimum, where the Taylor expansion up to second order, Eq. (A.41), is a good approximation of E. Secondly, different minima can display very different properties in terms of the Hessian and its eigenvalues. In any case, minima are not known in advance, which would render the training unnecessary. Since the cost function E can have many local minima, the outcome

of the training process may depend strongly on the initialization of the system. Therefore, the practical relevance of the mathematical analysis is limited.

In practice, a relatively large η could be used in the initial phase of training, assuming that the system is far away from any local minimum. Schemes have been suggested in the literature, which monitor the iterations and adjust the learning rate, for instance whenever a *zigzagging* behavior is observed. An intuitive example of a heuristic step size adaptation in batch gradient descent is presented in [PBB11].

The most important insight of this section is that in batch gradient descent, non-zero finite learning rates can be used to reach a local minimum. This is in contrast to the *stochastic gradient descent* strategy discussed in the next section. There, the learning rate has to be reduced to *zero* in the course of training in order to enforce convergence of the network configuration.

5.2.3 Stochastic gradient descent

The cost function $E(\underline{W})$, Eq. (5.9), is given as a sum over examples in $\mathbb{D}$:

$$E(\underline{W}) = \frac{1}{P}\sum_{\mu=1}^{P} e^{\mu}(\underline{W}) \qquad \text{with } \nabla_W E = \frac{1}{P}\sum_{\mu=1}^{P} \nabla_W\, e^{\mu}, \tag{5.13}$$

where e^{μ} quantifies the contribution of an individual example to the total costs. Virtually all machine learning objectives mentioned and discussed in this text can be written in such a form, with the actual function $e^{\mu}(\underline{W})$ and the precise definition of $\mathbb{D}$ depending on the details, of course. This includes the log-likelihood in maximum-likelihood problems, the SSE (2.5) in regression, the quantization error in unsupervised Vector Quantization [HKP91,BHV16] or the objective function of the Generalized LVQ scheme introduced in Chapter 6.

We note that costs of the form (5.13) can be interpreted as an empirical average of e^{μ} over the data set, corresponding to randomly drawing examples from $\mathbb{D}$ with equal probability $1/P$. Accordingly, the gradient of E as in Eq. (5.13) can also be seen as a data set average of the single example terms $\nabla_W e^{\mu}$.

As a consequence, we can approximate the gradient of E by computing a restricted empirical mean over a random subset of $\mathbb{D}$. As an extreme case, we can inspect single, randomly selected examples:

Stochastic gradient descent (SGD)

at discrete time step t

- select a single example $\{\boldsymbol{\xi}^{\mu(t)}, \tau^{\mu(t)}\}$ randomply with equal probability
- perform an update step

$$\underline{W}(t+1) = \underline{W}(t) + \Delta\underline{W}(t) = \underline{W}(t) - \widehat{\eta}(t)\ \nabla_W e^{\mu(t)}\Big|_{\underline{W}=\underline{W}(t)}. \tag{5.14}$$

with the learning rate $\widehat{\eta}(t)$ and error terms e^{μ} as given in Eq. (5.13).

The learning rate is denoted as $\widehat{\eta}(t)$ in order to indicate a possible explicit time-dependence and to distinguish it from η in batch gradient descent, Eq. (5.10). Clearly, the computational costs per update are lower than in the batch procedure which involves the sum of P gradient terms in each step.

The update of the form (5.14) is referred to as *online* or *stochastic gradient descent*, in contrast to offline batch algorithms. It can be seen as a special case of *stochastic approximation*, see [RM51, Bis95a, HTF01] and Appendix A.5.3.

The stochastic approximation of the gradient introduces noise in the training process. As a consequence, $E(\underline{W})$ may increase in individual update steps. The intuitive motivation for stochastic descent is that this noise helps the system to explore the search space more efficiently and to overcome *barriers* (e.g. saddle points of E) which separate different local minima of the cost function.

The index $\mu(t)$ of the presented example at time t in Eq. (5.14) is drawn randomly from the set $\{1, 2, \ldots, P\}$ with equal probability $1/P$. In general, the resulting individual $\Delta\underline{W}(t)$ will deviate from the direction of steepest descent $-\nabla_W E$. However, on average over the random selection of an example from $\mathbb{D}$ the update (5.14) follows the negative gradient of the total costs E. Therefore, in a local minimum $\underline{W}^*$

$$\overline{\Delta\underline{W}(t)\big|_*} = -\frac{1}{P}\sum_{\mu=1}^{P}\widehat{\eta}(t)\,\nabla_W e^\mu\big|_* = -\widehat{\eta}(t)\nabla_W E\big|_* = 0$$

where $\overline{(\ldots)}$ denotes the average over the selection of $\mu(t)$. The notation $(\ldots)|_*$ indicates that a term is evaluated in $\underline{W}(t) = \underline{W}^*$.

Hence, the average update becomes *zero* in the local minimum. However, individual updates remain non-zero, in general. This can be seen by considering the averaged squared norm of $\Delta\underline{W}$:

$$\overline{|\Delta\underline{W}|^2}\Big|_* = \frac{1}{P}\sum_{\mu=1}^{P}\widehat{\eta}^2(t)\Big[\,\nabla_W e^\mu\,|_*\,\Big]^2 \begin{cases} = 0 & \text{only if } \nabla_W e^\mu\big|_* = 0 \text{ for all } \mu \\ > 0 & \text{else.} \end{cases}$$

In general, not all of the gradient contributions will vanish in $\underline{W}^*$. One exception would be a minimum of E in which all individual e^μ are minimized. For the quadratic costs with $e^\mu = (\sigma(\boldsymbol{\xi}^\mu) - \tau^\mu)^2/2$ this would correspond to a global minimum $E(\underline{W}^*) = 0$, i.e. a perfectly solvable case where $\sigma(\boldsymbol{\xi}^\mu) = \tau^\mu$ for all μ.

The generic behavior for constant learning rate $\widehat{\eta} > 0$ is illustrated schematically in the left panel of Fig. 5.7. After reaching the vicinity of a local minimum, the iteration follows a seemingly irregular trajectory corresponding to the random sequence of individual gradient terms with $|\,\nabla_W e^\mu|_*\,|^2 > 0$.

We can enforce convergence near a local minimum in the sense of

$$\lim_{t\to\infty}\underline{W}(t) = \underline{W}^* \quad \text{and} \quad \lim_{t\to\infty}\Delta\underline{W}(t) = 0 \tag{5.15}$$

by employing an explicitly time-dependent learning rate $\widehat{\eta}(t)$ which decreases appropriately with the number of descent steps. Conditions for suitable learning

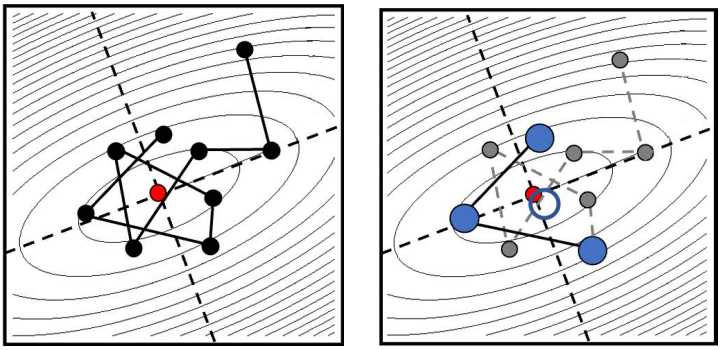


Figure 5.7: Schematic illustration of the behavior of stochastic gradient descent near a local minimum $\underline{W}^*$ as marked by the (red) dot in the center. The contour lines represent the quadratic approximation of $E(\underline{W})$ in the vicinity of $\underline{W}^*$. **Left panel:** Black symbols correspond to the iterates $\underline{W}(t)$ in Eq. (5.14). With a constant learning rate $\widehat{\eta} > 0$, the descent overshoots the point of stationarity in an oscillatory way. **Right panel:** The average of the most recent (here: 3) positions $\underline{W}(t)$ of the stochastic descent (large filled circles) results in a favorable estimate (large empty circle) of the local minimum.

rate schedules $\widehat{\eta}(t) \to 0$ can be found already in the seminal paper by Robbins and Monro [RM51] which introduced the concept of stochastic approximation in 1951, originally in the context of finding *zeros* of a function. Further references and more detailed discussions can be found in several textbooks, e.g. in [Bis95a, HTF01].

Robbins and Monro showed that schedules which satisfy $\lim_{t\to\infty} \widehat{\eta}(t) = 0$ with

$$\textbf{(I)} \lim_{T\to\infty} \sum_{t=0}^{T} \widehat{\eta}(t)^2 < \infty \quad \text{and} \quad \textbf{(II)} \lim_{T\to\infty} \sum_{t=0}^{T} \widehat{\eta}(t) \to \infty \tag{5.16}$$

facilitate convergence. Intuitively, the first condition (I) states that $\widehat{\eta}(t)$ has to decrease *fast enough* in order to achieve a truly stationary configuration with $\lim_{t\to\infty} \Delta\underline{W}(t) = 0$. However, enforcing $\widehat{\eta}(t) \to 0$ too rapidly would result in trivial stationarity at arbitrary positions in $\underline{W}$-space. Therefore, condition (II) implies that the decrease is *slow enough* so that the entire search space can be explored efficiently.

Simple schedules which reduce the learning rate asymptotically like $\widehat{\eta}(t) \propto 1/t$ for large t satisfy both conditions in (5.16). This relates to the well-known results that $\sum_{n=1}^{\infty} n^{-2} = \pi^2/6$ while $\sum_{n=1}^{\infty} n^{-1} \to \infty$. Just one possible (popular) realization of such a decrease is of the form

$$\widehat{\eta}(t) = \frac{a}{b+t}$$

with constant parameters $a, b > 0$.

Various schedules which satisfy the conditions (I) and (II) of Eq. (5.16) can be considered, including power laws $\widehat{\eta}(t) \propto t^{-\beta}$, logarithmic schedules like $\widehat{\eta}(t) \propto 1/(t \ln t)$, or other explicitly time-dependent schemes, see for instance [DM92] and references therein.

Stochastic gradient descent is arguably the most popular basic scheme for the training of neural networks, including systems with many layers in the context of *Deep Learning*.

5.2.4 Practical aspects and modifications

Numerous alternative approaches or modifications of the gradient based schemes have been suggested and are of great practical relevance, in particular in the context of *Deep Learning*. Here, only a few can be mentioned and explained briefly. Note that some of these concepts can also be useful in batch gradient descent.

SGD-training in epochs: In practice, we do not have to actually draw a random example from $\mathbb{D}$ independently at each time step. Most frequently, updates are organized in epochs, e.g. by generating a random permutation of $\{1, 2, \ldots P\}$ and presenting the entire $\mathbb{D}$ in this order before moving on to the next epoch with a novel randomized order of examples.

Mini-batch training: Instead of performing the stochastic approximation with respect to a single example, a random subset of $\mathbb{D}$ can be employed replacing the full gradient $\nabla_W E$ by partial sums in each training step. In a sense, this strategy retains the advantages of SGD, i.e. lower computational costs and the introduction of noise, but yields more reliable estimates of the gradient. The size of the mini-batches constitutes a hyperparameter which can be tuned in practice to achieve good performance and efficiency.

Averaged SGD: While training with constant learning rate $\widehat{\eta}$ will not result in a converging behavior $\underline{W}(t) \to \underline{W}^*$, one can expect the iterates $\underline{W}(t)$ to approach the vicinity of a local minimum and to assume more or less random positions centered around $\underline{W}^*$. This can be exploited by considering a (potentially moving) average of the form

$$\underline{W}_{av}(t) = \tfrac{1}{k} \sum_{j=0}^{k-1} \underline{W}(t-j)$$

which takes into account the last k iterations of SGD. In the simplest setting with $k = t$, the average is performed over all update steps up to t. As illustrated in Fig. 5.7 (right panel) the averaged $\underline{W}_{av}$ is expected to be closer to the local minimum than the individual $\underline{W}(t)$, once the training has reached the vicinity of the optimum. Averaging SGD for faster and smoother convergence was suggested and studied in [PJ92, Rup88], originally.

Local learning rates: For both, batch and stochastic gradient descent, it has been suggested to use local learning rates for different layers, nodes, or even individual weights in the network. As an early, simple rule of thumb, Plaut et *al.* [PNH86] suggest to use local learning rates inversely proportional to the *fan-in* of the given neuron, i.e. the number of units it receives input from, see the discussion in [HKP91]. More sophisticated methods make use of the local properties of the cost function in terms of second derivatives as motivated by Newton's method [Fle00, HKP91]. Frequently, only the diagonal elements of the local Hessian are used to compute an individual learning rate for the update of $\underline{W}_j$ which is, for instance, inversely proportional to $\frac{\partial^2 E}{(\partial W_i)^2}$, see [BL89] or [HKP91] for further references.

Note that gradient based algorithms with local or even individual learning rates do not follow the steepest descent in E anymore. However, they still realize a descent procedure, see the discussion in the Appendix A.4.

Momentum: Already in [RHW86] a modification of simple gradient descent was suggested, in which the update contains a *memory term* representing information about recently performed updates. In its simplest form the update is a linear combination of the (stochastic) gradient term and the previous update:

$$\Delta\underline{W}(t) = -\widehat{\eta}\,\nabla_W e^{\mu}(\underline{W}(t)) + \alpha\Delta\underline{W}(t-1) \quad \text{with } \alpha > 0. \tag{5.17}$$

The decay factor α controls the influence of previous update steps on the current change of $\underline{W}$. Obviously, momentum could also be incorporated in batch gradient descent.

The idea is to overcome *flat* regions where $\nabla_W E \approx 0$, by *keeping the momentum* of previous *downhill* moves. Furthermore, momentum should milden oscillatory behavior of the updates when E displays anisotropic curvatures.

Adaptive learning rate schedules: The design and optimization of learning rate schedules and schemes for the automated adaptation of $\widehat{\eta}$ in SGD or η in batch algorithms plays a key role for efficient training prescriptions. Frequently, the learning rate adaptation is combined with the concept of momentum. Prominent examples are algorithms termed *AdaGrad*, *RMSProp*, *Adam*, or variance-based SGD *(vSGD)*. For a first introduction and further references, Section 8.5 in [GBC16] can serve as a starting point. A systematic comparison of several popular schemes can be found in [LBV17], there in the context of gradient-based LVQ training.

5.3 Objective functions

So far we have discussed the training of layered networks based on the quadratic deviation (5.8) which, arguably, constitutes the most prominent cost function in the context of regression. However, insights into the actual target problem,

heuristic assumptions or concrete statistical models of the observations may motivate the use of alternative objective functions in the training process. Moreover, the use of differentiable neural networks for classification tasks motivates the use of cost functions which are designed for this particular purpose.

In the following we briefly discuss very few important examples of cost functions for regression and classification based on layered neural networks with differentiable activation functions.

5.3.1 Cost functions for regression

The very intuitive quadratic deviation or SSE cost function appears plausible and suitable for a variety of regression problems. A variety of alternative objective functions have been suggested in the literature that can be optimized by gradient descent or similar procedures.

Heuristic cost functions: Numerous heuristic schemes have been suggested which adjust an instantaneous objective function while training proceeds. The goal could be to *smooth out* the costs initially by levelling out details, which could help to avoid regions that contain unfavorable local minima. Gradually, more and more detail are re-introduced and, eventually, the genuine objective is optimized [HKP91].

As just one example, Makram-Ebeid et *et al.* suggest in [MSV89] the modified quadratic costs

$$E(\underline{W}) = \sum_{\mu=1}^{P} \begin{cases} \gamma\,(\sigma^\mu - \tau^\mu)^2 & \text{if } \sigma^\mu\,\tau^\mu > 0 \\ (\sigma^\mu - \tau^\mu)^2 & \text{if } \sigma^\mu\,\tau^\mu \leq 0 \end{cases} \tag{5.18}$$

with the parameter $\gamma \in [0, 1]$ increasing during the training process, e.g. according to an explicit time-dependence $\gamma(t)$. For $\gamma = 0$, deviations $(\sigma^\mu - \tau^\mu)^2$ do not contribute to the costs if the output has the correct sign. Thus, training will focus on achieving agreement in terms of $\text{sign}(\sigma^\mu) = \text{sign}(\tau^\mu)$, initially. As $\gamma \to 1$, the system is fine-tuned to achieve $\sigma^\mu \approx \tau^\mu$ for all μ, eventually.

Minkowski-r errors: In the following we focus on a class of cost functions that can be derived from a noise model which is assumed to describe the statistical properties of the data at hand.

The popular quadratic deviation can be explicitly motivated by assuming Gaussian distributed training labels. These could result from, e.g., additive Gaussian noise corrupting the true target values in the available data, as in Eq. (2.12). In Section 2.2.2 we have seen in the specific example of linear regression that a corresponding Maximum Likelihood approach leads immediately to the SSE-criterion (2.5).

Starting from different assumptions about the statistical properties leads to specific choices of the cost function. Assume, for instance, that the training

labels in a regression problem deviate from the true target function by independent noise terms η^μ with

$$P(\eta^\mu) \propto e^{-\beta\,|\eta^\mu|^r} \quad \text{with } r > 0, \tag{5.19}$$

which is normalized to $\int P(\eta^\mu) d\eta^\mu = 1$. Obviously we recover a Gaussian density with a β-dependent variance for $r = 2$. The optimization of the corresponding Maximum Likelihood criterion introduces a cost function of the form

$$E = \frac{1}{P} \sum_{\mu=1}^{P} |\sigma^\mu(\underline{W}) - \tau^\mu|^r \tag{5.20}$$

where terms independent of the network parameters $\underline{W}$ have been omitted. This objective function is referred to as the Minkowski-r error in the literature [HB87], see also [Bis95a].

We note again that for $r = 2$ the familiar MSE criterion is recovered, the special case of $r = 1$ corresponds to the so-called *Manhattan distance* or *city block metric* [6] $|\sigma - \tau|$. Intuitively, costs with $r < 2$ will be less sensitive to *outliers*, i.e. to examples with very large $|\sigma - \tau|$, than the conventional SSE.

5.3.2 Cost functions for classification

Heuristically, we can apply networks with differentiable activation functions and output also for classification, retaining regression type cost functions like the simple quadratic deviation or (5.18) in the training. In the simple case of two classes we could perform one additional threshold operation on a single output of the trained network to obtain a *crisp* binary classifier. Similar ideas can be applied to multi-class problems.

A more systematic approach realizes network responses that can be interpreted as a probabilistic assignment of the input vector to one of the classes. Here we follow to a large extent the presentation in [Bis95a]. First, we restrict ourselves to the case of two classes, here represented by target values $\tau^\mu \in \{0, 1\}$, which correspond to crisp training labels in the simplest case. Moreover, we assume that also the output of the network satisfies $0 \leq \sigma(\boldsymbol{\xi}^\mu) \leq 1$, as for instance realized by a proper sigmoidal output activation.

After training, we want to interpret $\sigma(\boldsymbol{\xi})$ as the class-membership probability

$$p(\tau = 1|\boldsymbol{\xi}) = \sigma(\boldsymbol{\xi}) \quad \text{and} \quad p(\tau = 0|\boldsymbol{\xi}) = 1 - \sigma(\boldsymbol{\xi}).$$

This can be written conveniently in the compact form

$$p(\tau|\boldsymbol{\xi}) = \sigma(\boldsymbol{\xi})^\tau \;\; [1 - \sigma(\boldsymbol{\xi})]^{1-\tau}. \tag{5.21}$$

If we consider this as our model for the occurrence of a label τ in the data set and we furthermore assume that the examples are generated independently, the

[6] Here, we ignore the subtle difficulty that $|x|^r$ it is not differentiable in $x = 0$ for $r \leq 1$.

likelihood of generating a given set of labels $\{\tau^\mu\}_{\mu=1}^P$ with the network reads

$$\prod_{\mu=1}^{P} (\sigma^\mu)^{\tau^\mu} \ [1-\sigma^\mu]^{1-\tau^\mu} \quad \text{with the shorthand} \ \ \sigma^\mu = \sigma(\boldsymbol{\xi}^\mu).$$

Maximizing this likelihood by choice of the network parameters $\underline{W}$ is equivalent to minimizing the negative log-likelihood

$$E(\underline{W}) = -\sum_{\mu=1}^{P} \Big(\tau^\mu \ln \sigma^\mu \ + \ (1-\tau^\mu) \ln(1-\sigma^\mu) \Big). \tag{5.22}$$

In contrast to the SSE, we omit the irrelevant pre-factor $1/P$ here for simplicity.

The cost function quantifies the similarity of the outputs σ^μ which we interpret as probabilities with the targets τ^μ in terms of their *cross entropy* [Hop87, BW88, SLF88]. Note that it is bounded from below by the entropy

$$E_o = -\sum_{\mu=1}^{P} \Big(\tau^\mu \ln \tau^\mu \ + \ (1-\tau^\mu) \ln(1-\tau^\mu) \Big)$$

which can only be achieved if all $\sigma^\mu = \tau^\mu$. Since the bound E_o does not depend on $\underline{W}$ we can subtract it from the cross entropy and consider the equivalent objective function

$$E(\underline{W}) - E_o = -\sum_{\mu=1}^{P} \Big(\tau^\mu \ln \frac{\sigma^\mu}{\tau^\mu} + \ (1-\tau^\mu) \ln \frac{1-\sigma^\mu}{1-\tau^\mu} \Big) \geq 0 \equiv D_{KL}(\tau||\sigma). \tag{5.23}$$

This is the so-called *relative entropy* or *Kullback-Leibler divergence* $D_{KL}(\tau||\sigma)$ between the specific probability distributions σ and τ, see for instance [Bis95a] for a discussion in the machine learning context. [7]

The cross entropy $E(\underline{W})$, Eq. (5.22), or equivalently the Kullback-Leibler divergence D_{KL} constitute differentiable objective function of the adaptive parameters $\underline{W}$ and can be minimized by gradient-descent based or other optimization methods for a given data set $\mathbb{D} = \{\boldsymbol{\xi}^\mu, \tau^\mu\}_{\mu=1}^P$. This way we achieve a network with outputs $\sigma(\boldsymbol{\xi})$ that can be interpreted as a class membership probability.

Multi-class problems: The formalism can be extended to multiclass problems with targets $\tau_k^\mu \in [0,1], k \in \{1,2,\ldots,C\}$ which satisfy $\sum_{k=1}^C \tau_k^\mu = 1$. In the simple case of crisp training labels we have that for each example exactly one $\tau_j^\mu = 1$ which indicates that $\boldsymbol{\xi}^\mu$ is assigned to class j in the training data.

Obviously we have to consider a network architecture and activations which can represent C assignment probabilities $\sigma_k \in [0,1]$ with $\sum_{k=1}^C \sigma_k = 1$. This can be achieved, for instance, with a layer of C output units with a so-called *soft-max* or *normalized exponential* activation, see also the following Sec. 5.4.

[7] Note that the KL-divergence is in general non-symmetric: $D_{KL}(\tau||\sigma) \neq D_{KL}(\sigma||\tau)$.

Now the equivalent of cost function (5.23) becomes

$$D_{KL}(\tau||\sigma) = -\sum_{\mu=1}^{P}\sum_{k=1}^{C} \tau_k^\mu \ln\left(\frac{\sigma_k^\mu}{\tau_k^\mu}\right), \tag{5.24}$$

which reduces to (5.23) in the binary case with $C = 2$ where $\sigma_1 = 1 - \sigma_2$ and $\tau_1 = 1 - \tau_2$.

Given a network structure with C outputs σ_k as described above, we can determine the adaptive parameters of the system by minimization of the above differentiable cost function. All gradient-based or more sophisticated techniques discussed here can be applied.

5.4 Activation functions

When designing a neural network for a given task, the key step is the choice of the network architecture and size. The choice of activation functions is equally important, as it should reflect properties of the problem and the data. Obviously, the output unit (or units) should realize the appropriate range of possible responses in a regression problem. In classification, properly defined outputs should encode the crisp or probabilistic class assignments. The choice of activations in intermediate, hidden layers influences the complexity of the network and can be crucial for the success of training, e.g. by gradient descent or other techniques.

So far we have mainly considered threshold or sigmoidal activation functions, with the notable exception of simple linear units when constructing a universal function approximator in 5.1.2. A large variety of activation functions have been suggested and investigated in the literature, see e.g. [HKP91, Bis95a, GBC16]. In this section, only a few important and/or popular choices are presented.

In the following we refrain from including gain parameters, local thresholds or similar parameters in the description, the corresponding extensions are straightforward. Similarly the range of activations can be trivially shifted and scaled: for example, a sigmoidal function $h(x)$ with $0 \leq h(x) \leq 1$ can be transformed as $g(x) = 2h(x) - 1$ with $-1 \leq g(x) \leq 1$. It is understood that all functions given as $g(x)$ in the following can be modified like $a\, g(\gamma x) + b$ if needed.

5.4.1 Sigmoidal and related functions

In Chapter 1 we motivated the use of sigmoidal activation functions as a rough approximation of biological neuron responses in a *firing rate* picture. A number of functions satisfy the conditions (1.2) or (1.4), see Fig. 5.8 for a few prominent examples. The left panel displays erf[x] (chain line, green), tanh[x] (dashed, blue), and the logistic function $1/(1 + \exp[-x])$. The latter was shifted and scaled to realize the range $g(x) \in [-1, 1]$. In the right panel, the Heaviside step function of the McCulloch Pitts neuron and a piecewise linear activation, which resembles a sigmoidal function, are shown.

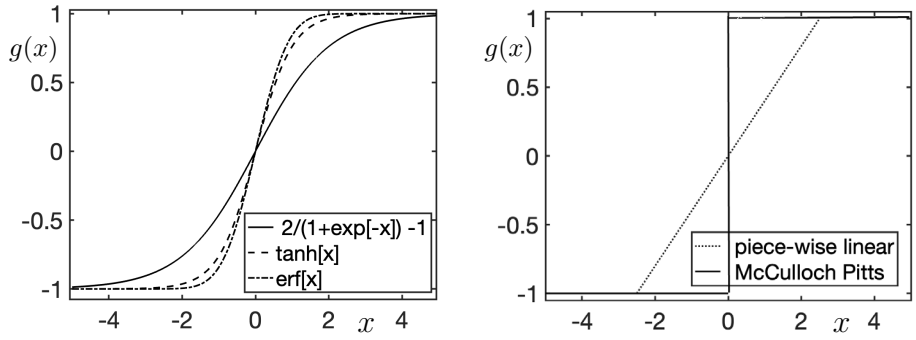


Figure 5.8: Sigmoidal and related activation functions. **Left panel:** three differential sigmoidal functions. **Right panel:** The limiting case of the McCulloch Pitts activation (Heaviside step function) and a piecewise linear function.

5.4.2 One-sided and unbounded activation functions

The simplest of all activation functions, the trivial identity $g(x) = x$ is unbounded. In the context of biologically inspired firing rate models this does not make sense, as there are no limits to the frequency of spike generation. However, in artificial networks, linear neurons are very often employed and indeed useful for specific tasks, e.g. as trainable output units attached to an otherwise more complex network for regression. Examples will be presented in Sec. 5.5.

Recitfied Linear Unit and variations thereof: The so-called *Rectified Linear Unit* or ReLU activation [NH10], see Fig. 5.9 (left panel), has gained significant popularity in the context of Deep Networks [8]. In its original form it corresponds to

$$g(x) = \max\{0, x\} = \begin{cases} 0 & \text{if} \quad x < 0 \\ x & \text{if} \quad x \geq 0 \end{cases} \quad \text{with } g'(x) = \begin{cases} 0 & \text{if} \quad x < 0 \\ 1 & \text{if} \quad x > 0. \end{cases} \tag{5.25}$$

We ignore here the subtlety that the ReLU function is not continuously differentiable in $x = 0$.

Note that the function (5.25) has been known and used for long in various mathematical, technical and engineering fields, ranging from signal processing and filtering to finance mathematics. Depending on the context, it is know as the *ramp function*, *hockey stick*, or *hinge function*.

In the literature, several advantages are associated with the ReLU activation when compared to, for instance, sigmoidal activations [GBC16]:

- Obviously, the ReLU is computationally cheap, and so is its derivative.

[8] As stated in [GBC16] : "In modern neural networks, the default recommendation is to use the rectified linear unit ...".

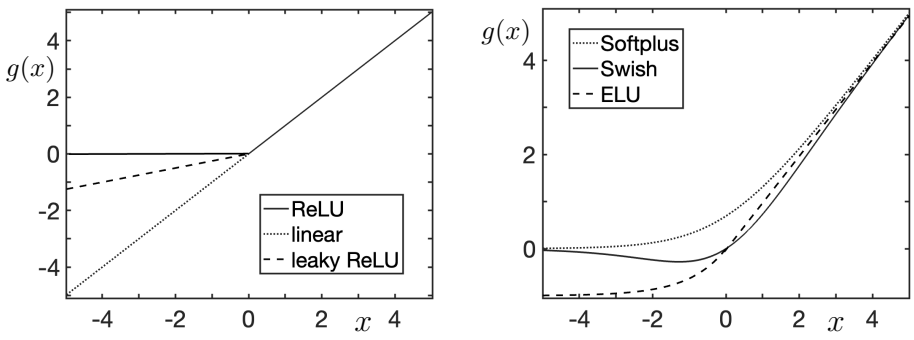


Figure 5.9: Unbounded and one-sided activation functions. **Left panel:** Simple linear activation $g(x) = x$ (dotted), Rectified Linear Unit ReLU (solid), Eq. (5.25), and leaky ReLU, Eq. (5.26), with $a = 0.25$ (dashed). **Right panel:** Exponential linear unit ELU (dashed, black), cf. Eq. (5.28), Swish (solid, green), Eq. (5.29), and Softplus (dotted, blue), Eq. (5.27).

- The ReLU is one-sided, i.e. $g(x) = 0$ for $x < 0$. As a consequence, a considerable fraction of units will display *zero* activity in a given network, typically. In this sense, a network of rectified linear units realizes sparse activity which is considered advantageous in many cases.
- When computing derivatives via the chain rule in a network of many layers, the multiplication of many derivatives $|g'| < 1$ causes the so-called problem of *vanishing gradients*. Supposedly, the problem is absent in ReLU networks where $g'(x) = 1$ at least for $x > 0$.
- Several empirical comparisons of networks with ReLU and other activations have been published, in which ReLU networks display favorable training behavior and performance. Recently, theoretical studies of model situations seem to support these claims [OSB20].

In the so-called *leaky ReLU* (LReLU) activation [HZRS15], the constant *zero* for $x < 0$ is replaced by a linear dependence, usually with a slope $0 < a < 1$, see also the left panel of Fig. 5.9. Hence, it reads

$$g(x) = \begin{cases} a\,x & \text{if} \quad x < 0 \\ x & \text{if} \quad x > 0 \end{cases} \quad \text{with} \quad g'(x) = \begin{cases} a & \text{if} \quad x < 0 \\ 1 & \text{if} \quad x \geq 0. \end{cases} \tag{5.26}$$

Differentiable, unbounded activations: Several differentiable functions which maintain or approximate the linear behavior $g(x) = x$ for large positive arguments $x > 0$ have been suggested in the literature. Fig. 5.9 (right panel) displays three examples:

- the so-called *Softplus* function [GBB11]

$$g(x) = \ln(1 + \exp[x]) \tag{5.27}$$

- the *Exponential Linear Unit* (ELU) [CUH16] with

$$g(x) = \begin{cases} \exp[x] - 1 & \text{for } x < 0 \\ x & \text{for } x \geq 0, \end{cases} \tag{5.28}$$

- the *Swish* function [EYG92] with

$$g(x) = \frac{x}{1 + \exp[-x]}. \tag{5.29}$$

Interestingly, the Swish activation is even non-monotonic and displays a minimum in a negative value of the argument. Several empirical studies seem to show favorable convergence behavior of gradient descent based training and improved performance of Swish-networks compared to other activation functions [EYG92, VRV+20].

5.4.3 Exponential and normalized activations

Frequently, units within a particular layer are coupled, e.g. through a normalization, which deviates from the activation by local synaptic interaction.

Softmax function: The most prominent example is the representation of assignment probabilities in an output layer of C units $\{\sigma_k\}_{k=1}^C$. As outlined in Sec. 5.3.2, the activations should obey $0 \leq \sigma_k \leq 1$ individually. In addition, $\sum_{k=1}^C \sigma_k = 1$ is required to justify their interpretation as probabilities.

An obvious and popular choice is to set $\sigma_k = g_\beta\left(\{x_k\}_{k=1}^C\right)$ with the C *pre-activations* x_k and the so-called *soft-max* or *normalized exponential* activation:

$$g_\beta\left(\{x_k\}_{k=1}^C\right) = \frac{\exp[\beta x_k]}{\sum_{j=1}^C \exp[\beta x_j]}. \tag{5.30}$$

Note that the required normalization couples the units σ_k and their states cannot be interpreted as independently activated by synaptic interaction: each unit depends on all *pre-activations* in the layer. For $\beta \to 0$ all activations will be equal ($\sigma_k = 1/C$), while for $\beta \to \infty$ the unit with maximum x_k is singled out with $\sigma_k = 1$.

Radial Basis Functions (RBF): Another popular class of activations also deviates from the familiar concept of synaptic interactions. The RBF activation of a given unit σ with input from neurons $\{s_j\}_{j=1}^L$ which are concatenated in the vector $\vec{s} \in \mathbb{R}^L$ is computed as

$$\sigma = g\left(||\vec{s} - \vec{c}||\right) \quad \text{with} \quad \vec{c} \in \mathbb{R}^L. \tag{5.31}$$

The activation depends on the Euclidean distance of the activation vector $\vec{s}$ from the adaptive *center* vectors $\vec{c}$. The term Radial Basis Function refers to the fact that σ is isotropically centered around $\vec{c}$.

A most prominent example is the Gaussian RBF, which is frequently referred to as the RBF:

$$\sigma = \exp\left[-\beta\,(\vec{s}-\vec{c})^2\right] \quad \text{with parameter } \beta > 0. \tag{5.32}$$

Frequently, normalized Gaussian RBF are considered in a layer of hidden or output units $\sigma_k (k = 1, 2, \ldots K)$:

$$\sigma_k(\vec{s}) = \frac{\exp\left[-\beta(\vec{s}-\vec{c}_k)^2\right]}{\sum_{j=1}^{K}\exp\left[-\beta(\vec{s}-\vec{c}_j)^2\right]}. \tag{5.33}$$

In the limit $\beta \to \infty$, the normalized Gaussian RBF singles out the unit with smallest $(\vec{s}-\vec{c}_k)^2$, i.e. with the closest center vector for a given $\vec{s}$.

A popular network architecture for regression comprises a potentially high-dimensional input layer, a single hidden layer with K units (5.33), and a linear output unit with adjustable weights. It is described briefly in Sec. 5.5.

5.4.4 Remark: universal function approximation

In Sec. 5.1.2 we constructed a piecewise constant function approximator using sigmoidal and linear units. It is interesting to note that it is often straightforward to extend these considerations to other activation functions. Note for instance, that the combination of two ReLU units, equipped with suitable local thresholds and gain parameters, can replace a piecewise linear activation of the type displayed in Fig. 5.8 (right panel):

$$\max\left\{0, \frac{x-a}{b-a}\right\} - \max\left\{0, \frac{x-b}{b-a}\right\} = \begin{cases} 0 & \text{for } x < a \\ \frac{x-a}{b-a} & \text{for } a \le x < b \\ 1 & \text{for } x \ge b. \end{cases}$$

Using the resulting piecewise linear activation, we can implement the selection of ROI, cf. Sec. 5.1.2, in analogy to the sigmoidal activations assumed there. Hence, networks of ReLU and/or piecewise linear units also constitute universal approximators. Similar arguments can be provided for large families of activation functions.

Similarly, units with normalized RBF activation, Eq. (5.32), can be readily used to define ROI in input space and facilitate universal function approximation when combined with piecewise constant representations as in Sec. 5.1.2.

5.5 Specific architectures

In this section we consider a selection of network architectures which play a role in practical applications and can be handled with the algorithmic approaches that we have studied so far. In the next section, specific shallow networks are introduced. In Sec. 5.5.2 we briefly discuss the design and training of multilayered *deep* neural networks.

5.5.1 Popular shallow networks

So far, we have developed training prescriptions in terms of shallow, feed-forward architectures with only one or very few hidden layers. In particular we have seen that a single hidden layer is sufficient to provide universal function approximations. An important example that we already discussed is the parity machine for classification, with hidden and output units of the McCulloch Pitts type[9].

A *soft* version of the committee machine, cf. Sec. 4.2, with sigmoidal activation can be shown to be provide universal function approximation in the context of regression, see Sec. 5.1.2. Analogous proofs exist for similar architectures with alternative hidden activations.

Radial Basis Function networks

Radial basis functions (RBF) as activation functions have been addressed in Sec. 5.4 already. Frequently, $N-K-1$ architectures with K RBF hidden units and linear output units are referred to as *RBF Networks* [BL88, MD89b].

In the popular case of Gaussian RBF and a single, linear output with bias w_o we have the input-output relation

$$\sigma(\boldsymbol{\xi}) \;=\; \sum_{j=1}^{M} w_j \phi_j\left(\boldsymbol{\xi}\right) + w_o \quad \text{with} \;\; \phi_j\left(\boldsymbol{\xi}\right) = \exp\left[-\frac{\left(\boldsymbol{\xi}-\mathbf{c}_j\right)^2}{2\sigma_j^2}\right]. \tag{5.34}$$

This corresponds to Eq. (5.32) with unit-specific parameters $\beta_j = 1/(2\sigma_j^2)$. Each unit is equipped with an adaptive vector $\mathbf{c_i} \in \mathbb{R}^N$ which defines the center of the receptive field. The response of the unit to a given input $\boldsymbol{\xi} \in \mathbb{R}$ depends on its Euclidean distance from the center vector. Here, we also include an adaptive local bias $w_o \in \mathbb{R}$ in the activation. Instead, we could introduce an additional hidden unit with constant activation $\phi_o(\boldsymbol{\xi}) = 1$ for all $\boldsymbol{\xi}$, similar to the *clamped input* employed in Eqs. (2.4) and (3.4).

RBF networks of this form are universal approximators, see the general discussion in [Bis95a] and specifically, [GP90]. Hence, we can employ networks of the type (5.34) for general regression tasks.

The complexity of the RBF network can be increased by allowing for adaptive (inverse) covariance matrices in the Gaussian activations:

$$\phi_j\left(\boldsymbol{\xi}\right) = \exp\left[-\left(\boldsymbol{\xi}-\mathbf{c}_j\right)^\top \Sigma_j^{-1}\;\left(\boldsymbol{\xi}-\mathbf{c}_j\right)\right]. \tag{5.35}$$

Nominally, this introduces $N(N+1)/2$ additional adaptive parameters per symmetric matrix $\Sigma_j \in \mathbb{R}^{N\times N}$. In turn, fewer units might be required to achieve the same accuracy and performance as a larger network with hidden unit activations of the form (5.34). Modifications can be considered, such as the restriction to diagonal Σ_j or the pooling of covariances with a single adaptive $\Sigma = \Sigma_1 = \ldots = \Sigma_M$.

[9] The parity machine is strictly speaking not an $(N-K-1)$ architecture, see Sec. 4.2.

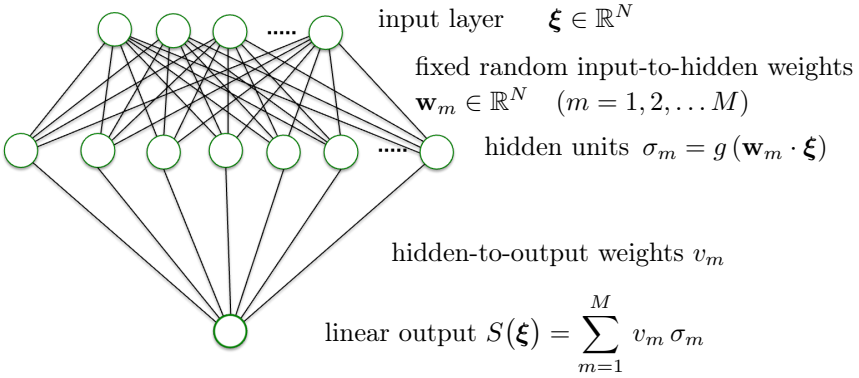


Figure 5.10: Illustration of an Extreme Learning Machine (ELM). The N-dimensional input is connected to a hidden layer with $M \gg N$ units by fixed (non-adaptive) random weights. In the example, the single output unit is linear, e.g. for the purpose of regression. Extensions to a threshold unit or a full layer of outputs are straightforward.

The RBF architecture could be used for classification tasks by attaching a single or multiple output classifier to the hidden layer $\{\phi_j(\boldsymbol{\xi})\}_{j=1}^{M}$. A more natural approach is to normalize the M activations in the hidden layer as in (5.33) and interpret them as probabilistic class assignments:

$$\widehat{\phi}_j(\boldsymbol{\xi}) = \frac{\phi_j(\boldsymbol{\xi})}{\sum_{k=1}^{M} \phi_k(\boldsymbol{\xi})} \quad \text{with} \quad \phi_j \quad \text{from} \quad (5.34) \text{ or } (5.35). \tag{5.36}$$

These non-local activations satisfy $\widehat{\phi}_j \in [0, 1]$ and $\sum_j \widehat{\phi}_j = 1$ and hence we can train the system according to a classification specific cost function like (5.23) for binary problems and (5.24) in a multi-class setting.

Remark: RBF systems for classification display a striking similarity with prototype-based classifiers. In an LVQ system as presented in Chapter 6, the prototypes correspond to the center vectors $\mathbf{c}_j$ and the softmax scheme of the classifier would be replaced by a *crisp* Nearest Prototype classification (NPC). Similarly, the matrix Σ_j^{-1} in (5.35) is the equivalent of a prototype-specific, local relevance matrix Λ_j in Eq. (6.13), see Sec. 6.2.2.

Extreme Learning Machines

Frank Rosenblatt already suggested randomized connections from an input layer to a so-called *association layer*, which then was classified by threshold units in the *Mark I* realization of the perceptron [HRM+60], see Section 3, Fig. 3.1. More recently, random projections have been become popular as a technique to achieve sparse, low-dimensional representations of high-dimensional data sets, see for instance [BM01, LHC06].

A specific feed-forward architecture, termed the *Extreme Learning Machine* (ELM), was introduced in 2004 by Huang et *al.* [HZS06]. It is schematically

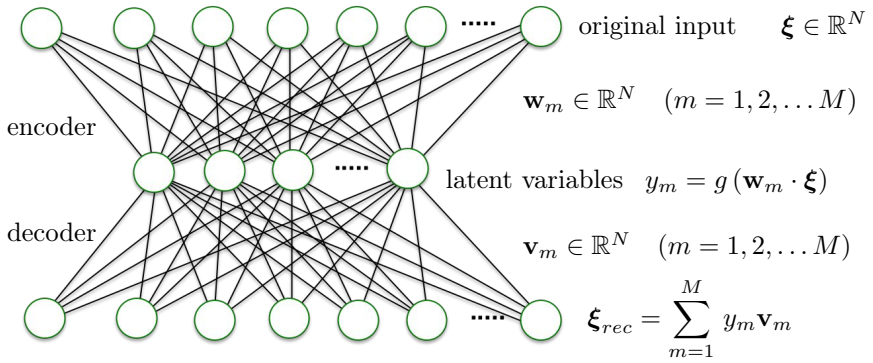


Figure 5.11: An example of a shallow auto-encoder network: The N-dim. inputs are represented in a hidden layer with $M < N$ units. Here, N (linear) output units represent the target reconstruction $\boldsymbol{\xi}_{rec} \in \mathbb{R}^N$, the extension to non-linear output units is obviously possible.

represented in Fig. 5.10. The random mapping of inputs to a high-dimensional hidden layer makes it, for instance, possible to separate classes or perform regression tasks with a single linear (threshold) unit which would not be sufficient to realize the target in terms of the original data. This is similar in spirit to the basic idea of the Support Vector Machine, cf. 4.3. The relation of ELM and SVM was first discussed in [FV10].

Shallow autoencoders

A particular feed-forward type of networks can be used to find a low-dimensional representation of high-dimensional feature vectors $\{\boldsymbol{\xi}^\mu\}_{\mu=1}^P$. To this end, we can employ a so-called auto-encoder as illustrated in Figure 5.11. The *encoder* represents N-dim. input vectors $\boldsymbol{\xi}$ in a single hidden layer of $M < N$ units. The output $\boldsymbol{\xi}_{rec}$ is again N-dimensional and the goal is to minimize the *reconstruction error* in the *decoder*:

$$E_{rec} = \frac{1}{2} \sum_{\mu=1}^{P} \left(\boldsymbol{\xi}_{rec}^\mu - \boldsymbol{\xi}^\mu\right)^2 . \tag{5.37}$$

Training amounts to the (e.g. gradient based) minimization of E_{rec} with respect to the network weights $\mathbf{w}_m, \mathbf{v}_m \in \mathbb{R}^N$. Obviously, the mathematical structure is the same as in function approximation, with the special target to approximate the identity function $\mathbb{R}^N \to \mathbb{R}^N$. The resulting M-dimensional *latent variables* $\{y_m\}_{m=1}^P$ serve as the low-dimensional representations of high-dimensional data.

In Fig. 5.11 the output units are assumed to be linear, the possible generalization to non-linear reconstructions is straightforward. One special case is particularly interesting: for linear activations g in the hidden layer and linear reconstruction $\boldsymbol{\xi}_{rec}$, the minimization of the reconstruction error (5.37) is analogous

to the well known Principal Component Analysis, e.g. [Bis95a, HKP91, HTF01]. More precisely, the weight vectors $\mathbf{w}_m$, which minimize E_{rec}, span the same sub-space as the M leading principal components of the data set.

Using sigmoidal or other non-trivial hidden and/or output activations in the auto-encoder network, generalizes the concept of PCA to non-linear low-dimensional representations and reconstructions. In the next section we will also briefly mention *deep* auto-encoders, where several hidden layers represent and process the data internally [GBC16].

Obviously we can exploit the latent variables of an auto-encoder immediately for the purpose of visualizing complex, high-dim. data if $M = 2$ or 3. Moreover, after having trained the auto-encoder to realize a faithful internal representation, one could attach a feed-forward classifier or regression network to the hidden layer and apply supervised learning to realize a target function, e.g. of the type $\mathbb{R}^M \to \mathbb{R}$.

5.5.2 Deep and convolutional neural networks

At a glance, the term *Deep Learning* refers to the use of feed-forward neural networks with many hidden layers. While this over-simplified definition ignores several aspects of Deep Learning, e.g. the potential use of feedback and recurrent systems, we will focus here on deep feed-forward architectures. As phrased by Goodfellow, Bengio and Courville [GBC16]:

> *[There is no] consensus about how much depth a model requires to qualify as 'deep'. However, deep learning can safely be regarded as the study of models that either involve a greater amount of composition of learned functions or learned concepts than traditional machine learning does.*

The enormous success of *Deep Learning* and its popularity after, say, 2010, can be attributed to a number of developments, including the following:

- The availability of large amounts of data, e.g. from image data bases or large collections of commercial data in e-commerce
- The pre-training of deep networks or sub-networks from *unspecific* data bases, followed by task-specific fine tuning (transfer learning)
- The ever-increasing computational power, made available through super-computers or local solutions (e.g. GPU)
- The refinement of (mostly) gradient-based training techniques, e.g. w.r.t. to the automatic adaptation of learning rates or efficient regularization techniques such as *drop out*
- The exploitation and combination of concepts that had been developed earlier for shallow networks, e.g. the ideas of *weight-sharing* or *momentum*.

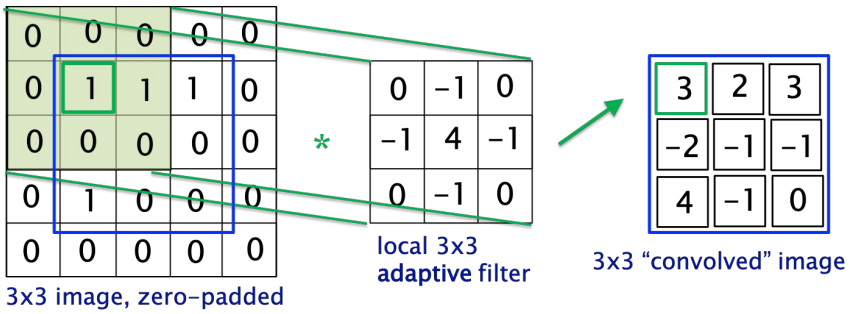


Figure 5.12: A 3×3 'convolutional' filter kernel is applied to a 3×3 image, here zero-padded. The 3×3 kernel with weights denoted in the illustration is centered on every pixel of the image to obtain the pixel values in the 3×3 convolved image. Note that the operation does not reduce the dimension of the data.

- The use of activation functions that (supposedly) improve the efficiency and performance of networks in training and working phase, for example the *Rectified Linear Unit* (ReLU), cf. Sec. 5.4.2

- The consideration of particular architectures, e.g. *Convolutional Neural Networks* (CNN), designed for the analysis of specific types of data, such as images, language, time series or other data with a low-dimensional spatial, temporal or functional structure.

Quite a few of these concepts have been known well before the rise of Deep Learning. Ultimately their combination made the great success of Deep Networks possible, see e.g. [Sch15, LBH18].

Convolutional and pooling layers

All convolutional neural networks (CNN) share some characteristic design features which facilitate the processing of structured data. For instance, in images or time series data, we expect localized information: Time series like the €/$ exchange rate will display short time correlations that decrease over time. Likewise, pixels in a photographic image are expected to be similar in intensity and color if they belong to the same object, while far away pixels may be totally unrelated. In the following we will discuss CNN in the context of images, the transfer to other structured data is straightforward.

The localization is accounted for by connecting units in a first layer to limited neighborhoods or patches of the input data, see Fig. 5.12 for an illustration. By choice of the weights in such a *filter kernel*, nodes can implement a particular local operation or *convolution*. [10] The weights can be adapted in the training

[10] The term is used somewhat loosely in the context of Deep Learning.

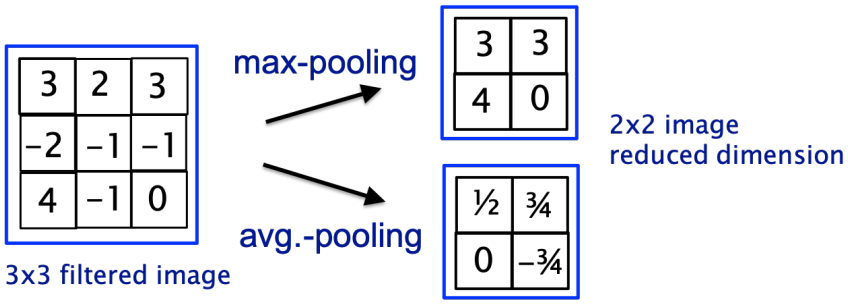


Figure 5.13: The 3×3 convolved image from Fig. 5.12 is reduced to 2×2 pixels by applying a *max-pooling* (upper) or *average pooling* (lower) to all 2×2 patches in the 3×3 filtered image.

process, e.g. by using gradient based Backpropagation of Error for the entire network. A number K of different filters is applied to all patches of the same size in the input data. As each filter applies the same operation, the number of adaptive weights depends only on K and on the dimension and type of the kernels, while it is independent of the dimension of the input. This basic concept of *weight-sharing* was already present in very early network models, see below for Fukushima's Neocognitron as an example.

The first convolutional layer in a CNN therefore represents the input data in terms of many versions of the image obtained through a potentially large set of adaptive filters. These may include but are not limited to (approximations of) intuitive operations like the detection of edges or other local patterns.

Most frequently, after convolution, a *pooling* operation is applied in a subsequent layer. Pooling reduces the dimensionality by combining, usually small, patches of nodes into a *super-pixel*. Popular examples replace plaquettes of 2×2 or 3×3 nodes by the average activation (average pooling) or by the maximum activity in the patch, see Fig. 5.13. Typically, the pooling nodes are *hard wired*, i.e. not trainable, although one could for instance consider adaptive weighted averages for pooling.

After the first convolution and pooling, the input image is represented by a number of filtered and dimension reduced versions. Frequently, convolutional and pooling layers are stacked in alternating fashion, yielding increasingly abstract representations of decreasing dimension. Ultimately, one or several trainable *dense layers*, fully connected as in conventional feed-forward architectures, are employed to represent the target classification or regression.

A large number of network similar architectures have been developed and are made available in the public domain, see https://modelzoo.co for an example repository. Many of these networks have been pre-trained on more or less generic data sets and can be fine-tuned by the user for the specific task at hand.

Despite the simplifications through weight-sharing and similar regularization techniques, Deep Networks often comprise a huge number of adaptive weights and consequently can be very *data hungry*. In fact, it appears often surpris-

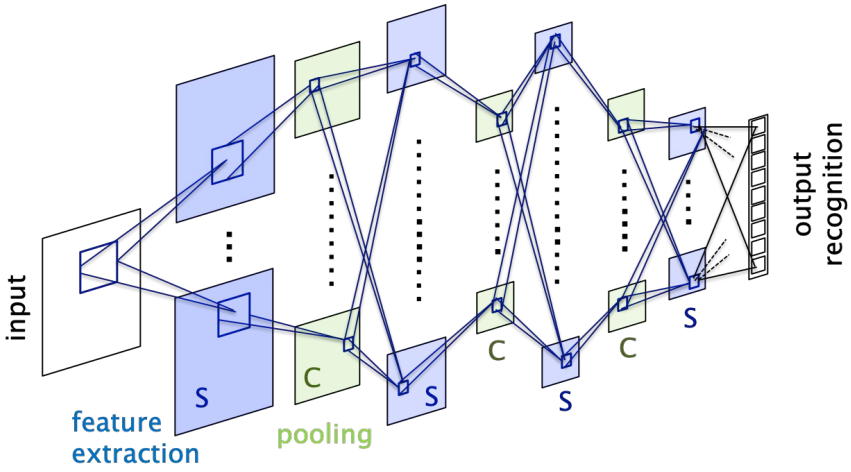


Figure 5.14: An early deep architecture (schematic) known as the *Neocognitron*, first introduced by K. Fukushima in 1980 (1979 in Japanese) [Fuk80]. Besides input and output (recognition), several layers of so-called S-cells and C-cells are stacked see the text for details. Illustration redrawn after [Fuk19] with kind permission from the author.

ing that heavily over-parameterized networks can be trained successfully at all and suffer less drastically from over-fitting effects than one might expect on theoretical grounds.

Early examples: Neocognitron and LeNet

Inspired by an early model of human vision of Hubel and Wiesel [HW59], Kunihiko Fukushima introduced the so-called Neocognitron network architecture as early as 1979 (in Japanese) and 1980 [Fuk80,Fuk88], see preceding S-also [Fuk19] for a more recent presentation and discussion of different versions of the basic architecture.

The Neocognitron already comprises many elements of *modern* Convolutional Neural Networks. As illustrated in Figure 5.14, the network consists of an input and output layer, with stacked hidden layers of alternating types. Units are typically connected to patches of nodes in the preceding layer. Feature extraction layers (shaded blue in Fig. 5.14) employ local filters to these patches. Their units correspond to the *simple neurons* or *S-cells* suggested by Hubel and Wiesel, which are activated, for instance, by characteristic patterns like straight lines of a particular orientation. Nodes in a subsequent layer of *complex* or *C-cells* perform pooling operations in the sense that they are activated independent of the precise location of the stimulation within their receptive field in the preceding S-layer. Thanks to the averaging or *pooling* C-cells, the Neocognitron is, to a certain degree, insensitive to shifts and distortions of input patterns.

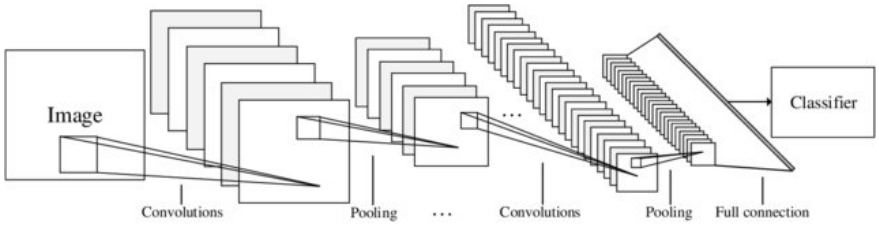


Figure 5.15: A deep architecture (schematic) known as *LeNet*, specifically LeNet-5, introduced by LeCun et al. in [LBD+89]. Image available under license CC BY3.0 at https://www.researchgate.net/publication/319905492_Image_retrieval_method_based_on_metric_learning_for_convolutional_neural_network/figures.

The sequence of S and C layers represents the input in decreasing detail and, ultimately, the network response (e.g. a classification) is provided in the output layer. In contrast to more recent CNN architectures, Fukushima did not train the Neocognitron *end-to-end* by gradient descent techniques. Instead, the filters realized by S-cells were either pre-wired or adapted by means of unsupervised learning techniques.

The Neocognitron has been studied and used in the context of brain-inspired pattern recognition, including handwritten digit recognition and similar tasks. It constitutes a groundbreaking work that inspired many, if not all modern Deep Networks for visual pattern recognition and similar tasks.

Another groundbreaking architecture, known as *LeNet* is due to LeCun and collaborators [LBD+89]. It is an early example of a Convolutional Neural Network (CNN) for pattern recognition and image processing and can be considered the starting point for this popular type of architecture. The structure is similar to the above discussed Neocognitron. Alternating *convolutional* and *pooling* layers process an input with increasing abstraction towards a fully connected output layer. The LeNet can be trained from end-to-end by gradient based Backpropagation. It was initially introduced for the task of handwritten digit recognition (ZIP-code reading).

Groups of nodes perform the same task on different patches of the input. Consequently, many nodes can share the same weight values which reduces the effective number of adaptive quantities drastically. LeNet constitutes a very early example of *weight-sharing*, which to date plays an important role in the training of deep networks, see also Sec. 7.2.5. Further improvements of the network performance were achieved by developing and applying a specific *pruning* technique named *Optimal Brain Damage* [LDS90], which we introduce in Sec. 7.2.4.

Appraisal and critique of deep learning

The success of Deep Learning, mainly in the context of image processing, has triggered a lot of excitement in, both, academia and the general public. This includes exaggerated claims with respect to *general intelligence* or applications

in critical areas like clinical medicine.

Recently, several scholars have expressed criticism of Deep Learning and the *hype* surrounding it, with [Mar18,Pre21,Zad19,AN20] being just a few examples. In a sense, the situation is highly reminiscent of the strong expectations and the later disappointments in previous waves of machine learning popularity.

In the biased opinion of the author of this text, several deplorable trends can be observed in academia and in the general public:

- Deep Learning is often presented as fundamentally new and totally different from so-called conventional, supposedly *old school* machine learning, ignoring decades of research that facilitated the recent developments.
- The mis-identification of machine learning with image analysis only, in particular in the non-scientific media. While image classification, scene analysis, face recognition etc. are certainly relevant and particularly appealing problems, machine learning should be seen from a much broader perspective.
- A wide-spread tendency to use overly complex DNN to tackle even relatively simple, specific applications without investing a thoughtful analysis and often without a proper critical evaluation of the performance or comparison with baseline techniques.
- The exclusive use of ready-made programming environments while having a limited understanding of the basic underlying principles.[11] Frequently such systems offer only limited user control or the options are not exploited properly.
- A lack of theoretical insight into Deep Learning, or - much worse - a lack of interest into a better understanding of the relevant phenomena.

The last points are particularly unfortunate in view of the many interesting challenges and open questions posed by Deep Learning, some of which are summarized in [Sej20].

Despite these and other points of criticism, Deep Learning will play an important role in forthcoming years. It will certainly facilitate the exploration of new and exciting application areas. At the same time Deep Learning will continue to provide highly interesting theoretical challenges that deserve significant attention.

[11] Even worse, this is often combined with displaying code in illegible small but colorful fonts on a black background.

Chapter 6

Distance-based classifiers

One can state, without exaggeration, that the observation of and the search for similarities and differences are the basis of all human knowledge.

— Alfred Nobel

The use of distances or dissimilarities for the comparison of observations with a set of labeled reference data points provides a simple yet powerful tool for classification. In particular, the use of *prototypes* or *exemplars*, derived from a given data set, is the basis for a very successful family of machine learning approaches. Prototype-based classifiers are appealing for a number of reasons. The extraction of information from previously observed data in terms of typical representatives, the prototypes, is particularly transparent and intuitive, in contrast to many, more *black-box* like systems. The same is true for the working phase, in which novel data are compared with the prototypes by use of a suitable (dis-)similarity or distance measure.

Prototype systems are frequently employed for the *unsupervised* analysis of complex data sets, aiming at the detection of underlying structures, such as clusters or hierarchical relations, see also Chapter 8 and, for instance, [HTF01, Bis95a, DHS00]. Competitive Vector Quantization, the well-known K-means algorithm or Self-Organizing Maps are prominent examples for the use of prototypes in the context of unsupervised learning [Koh97, HTF01, DHS00].

In the following the emphasis is on supervised learning in prototype-based systems. In particular, we focus on the framework of Learning Vector Quantization (LVQ) for classification. Besides the most basic concepts and training prescriptions we present extensions of the framework to unconventional distances and to the use of adaptive measures in so-called relevance learning schemes [BHV16].

The aim of this chapter is far from giving a complete review of the ongoing fundamental and application oriented research in the context of prototype-based

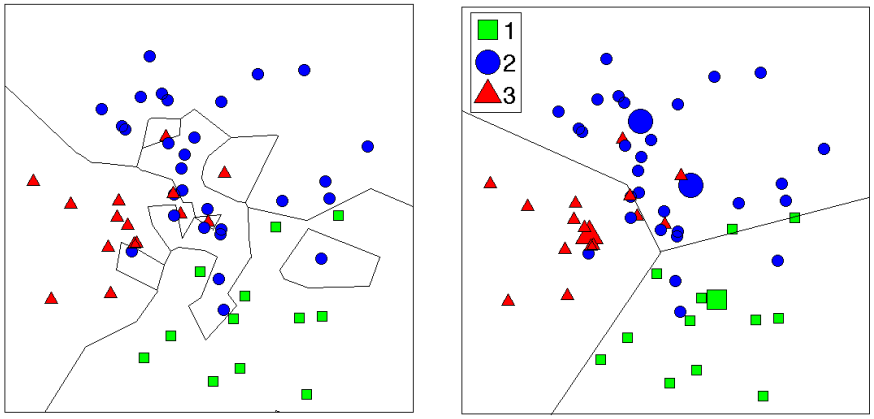


Figure 6.1: Left panel: Illustration of the Nearest Neighbor (NN) Classifier for an artificial data set containing three different classes. **Right panel:** A corresponding NPC scheme for the same data. Prototypes are represented by larger symbols. Both schemes are based on Euclidean distance and yield piecewise linear decision boundaries.

learning. It provides, at best, first insights into supervised schemes and can serve as a starting point for the interested reader.

The emphasis will be on Teuvo Kohonen's Learning Vector Quantization and its extensions [Koh97]. Examples for training prescriptions are given and the use of unconventional distance measures is discussed. As an important conceptual extension of LVQ, Relevance Learning is introduced, with Matrix Relevance LVQ serving as an example.

In a sense, the philosophies behind LVQ and the SVM are diametrically opposed to each other: while support vectors represent the *difficult cases* in the data set, which are closest to the decision boundary, cf. Sec. 4.3, LVQ represents the classes by - supposedly - typical exemplars relatively far from the class borders.

Note that LVQ systems could be formulated and interpreted as layered neural networks with specific, distance-based activations and a crisp output reflecting the Winner-Takes-All principle. In fact, after years of denying the relation in the literature, it has become popular again to point out the conceptual vicinity to neural networks. Recent publications also discuss the embedding of LVQ modules in deep learning approaches [VMC16, VBVS17].

6.1 Prototype-based classifiers

Among the many frameworks developed for supervised machine learning, prototype-based systems are particularly intuitive, flexible, and easy to implement. Although we restrict the discussion to classification problems, many of the concepts carry over to regression or, to a certain extent, also to unsupervised learning, see [BHV16].

Several prototype-based classifiers have been considered in the literature. Some of them can be derived from well-known unsupervised schemes like the Self-Organizing-Map or the Neural Gas [Koh97, RMS92, HSV05], which can be extended in terms of a posterior labelling of the prototypes. Here, the focus is on the so-called Learning Vector Quantization (LVQ), a framework which was originally suggested by Teuvo Kohonen [Koh97]. As a starting point for the discussion, we briefly revisit the well-known k-Nearest-Neighbor (kNN) approach to classification, see [DHS00, CH67].

6.1.1 Nearest Neighbor and Nearest Prototype Classifiers

Nearest Neighbor classifiers [DHS00, CH67] constitute one of the simplest and most popular classification schemes. In this classical approach, a number of labeled feature vectors is stored in a reference set:

$$\mathbb{D} = \{\boldsymbol{\xi}^\mu, y^\mu = y(\boldsymbol{\xi}^\mu)\}_{\mu=1}^{P}$$

with $\boldsymbol{\xi}^\mu \in \mathbb{R}^N$. In contrast to the discussion of the perceptron and similar systems, here we do not have to restrict the presentation to binary labels. We therefore denote the (possibly multi-class) labels by $y^\mu \in \{1, 2, \dots C\}$ where C is the number of classes.

An arbitrary novel feature vector or *query* $\boldsymbol{\xi} \in \mathbb{R}^N$ can be classified according to its (dis-) similarities to the samples stored in the reference data. To this end, its distance from all reference vectors $\boldsymbol{\xi}^\mu \in \mathbb{D}$ has to be computed. Most frequently, the simple (squared) Euclidean distance is used in this context: $d(\boldsymbol{\xi}, \boldsymbol{\xi}^\mu) = (\boldsymbol{\xi} - \boldsymbol{\xi}^\mu)^2$. The query $\boldsymbol{\xi}$ is then assigned to the class of its Nearest Neighbor exemplar in $\mathbb{D}$. In the more general kNN classifier, the assignment is determined by means of a voting scheme that considers the k closest reference vectors [CH67].

The NN or kNN classifier is obviously very easy to implement as it does not even require a *training phase*. Nevertheless one can show that the kNN approach bears the potential to realize Bayes optimal performance if the number k of neighbors is chosen carefully [HTF01, DHS00, CH67]. Consequently, the method serves, to date, as an important baseline algorithm and is frequently used as a benchmark to compare performances with.

Fig. 6.1 (left panel) illustrates the NN classifier and displays how the system implements piecewise linear class borders. Several difficulties are evident already in this simple illustration. Class borders can be overly complex, for instance if single data points in the set have been classified incorrectly. Furthermore, the

fact that every data point contributes with equal weight can lead to overfitting effects because the classifier over-rates the importance of individual examples. As a consequence, it might not perform well when presented with novel, unseen data.

Straightforward implementations of kNN compute and sort the distances of $\boldsymbol{\xi}$ from all available examples in $\mathbb{D}$. While methods for efficient sorting can reduce the computational costs to a certain degree, the problem persists and is definitely relevant for very large data sets.

Both drawbacks could be attenuated by reducing the number of reference data in an intelligent way while keeping the most relevant properties of the data. Indeed, the selection of a suitable subset of reference vectors by *thinning out* $\mathbb{D}$ was already suggested in [Har68]. An alternative, essentially *bottom-to-top* approach is considered in the following sections.

6.1.2 Learning Vector Quantization

This successful and particularly intuitive approach to classification was introduced and put forward by Teuvo Kohonen [Koh97,RMS92,BHV16,Koh95,SK99, NE14]. The basic idea is to replace the potentially large set of labeled example data by relatively few, representative prototype vectors.

LVQ was originally motivated as a simplifying approximation of a Bayes classifier under the assumption that the underlying density of data corresponds to a superposition of Gaussians [Koh97]. LVQ replaces the actual density estimation by a simple and robust method of supervised Vector Quantization. Each of the C classes is to be represented by (at least) one representative. Formally, we consider the set of prototype vectors

$$\left\{ \mathbf{w}^j, c^j \right\}_{j=1}^{M} \text{ with } \mathbf{w}^j \in \mathbb{R}^N \text{ and } c^j \in \{1, 2, \dots C\}. \tag{6.1}$$

Here, the prototype labels $c^j = c(\mathbf{w}^j)$ indicate which class the corresponding prototype is supposed to represent. The so-called Nearest Prototype classifier (NPC) assigns an arbitrary, e.g. novel, feature vector $\boldsymbol{\xi}$ to the class $c^* = c(\mathbf{w}^*)$ of the closest prototype

$$\mathbf{w}^*(\boldsymbol{\xi}) \text{ with } d\Big(\mathbf{w}^*(\boldsymbol{\xi}), \boldsymbol{\xi}\Big) = \min \Big\{ d(\mathbf{w}^j, \boldsymbol{\xi}) \Big\}_{j=1}^{M} \tag{6.2}$$

where ties can be broken arbitrarily.

In the following, the closest prototype $\mathbf{w}^*(\boldsymbol{\xi})$ of a given input vector will be referred to as the *winner*. For brevity we will frequently omit the argument of $\mathbf{w}^*(\boldsymbol{\xi})$ and use the shorthand $\mathbf{w}^*$ when it is obvious which input vector it refers to.

Figure 6.1 (right panel) illustrates the NPC concept: class borders corresponding to relatively few prototypes are smoother than the corresponding NN decision boundaries shown in the left panel. Consequently, an NPC classifier

can be expected to be more robust and less prone to overfitting effects.

The performance of LVQ systems has proven to be competitive in a variety of practical classification problems [Neu02]. In addition, their flexibility and interpretability constitute important advantages of prototype-based classifiers since prototypes are obtained and can be interpreted within the space of observed data, directly. This feature facilitates the discussion with domain experts and stands in contrast to many other, less transparent machine learning frameworks.

An LVQ system for nearest prototype classification can be interpreted as a neural network with a single hidden layer. The prototypes correspond to hidden units with a distance-based activation which feed into a *winner-takes-all* output unit. This appealing analogy is elaborated in, for example, [RKSV20]. However, it is not essential for the following.

6.1.3 LVQ training algorithms

So far we have not addressed the question of where and how to place the prototypes for a given data set. A variety of LVQ training algorithms have been suggested in the literature [Koh97, SK99, Koh90, NE14, BGH07, Gho21, Wit10].

The first, original scheme suggested by Kohonen [Koh97] is known as LVQ1. Essentially, it already includes all aspects of the many modifications that were suggested later. The algorithm can be summarized in terms of the following steps:

LVQ1 algorithm, random sequential presentation of data

- at time step t, select a single feature vector $\boldsymbol{\xi}^\mu$ with class label y^μ randomly from the data set $\mathbb{D}$ with uniform probability $1/P$.
- identify the *winning prototype*, i.e. the currently closest prototype $\mathbf{w}^*_\mu = \mathbf{w}^*(\boldsymbol{\xi}^\mu)$ given by $d(\mathbf{w}^*_\mu, \boldsymbol{\xi}^\mu) = \min\left\{d(\mathbf{w}^j, \boldsymbol{\xi}^\mu)\right\}_{j=1}^{M}$ (6.3) with class label $c^*_\mu = c(\mathbf{w}^*_\mu)$.
- perform a *Winner-Takes-All* (WTA) update: (6.4)

$$\mathbf{w}^*_\mu(t{+}1) = \mathbf{w}^*_\mu(t) + \eta_w\, \Psi(c^*_\mu, y^\mu)\left(\boldsymbol{\xi}^\mu - \mathbf{w}^*_\mu\right) \text{ with } \Psi(c,y) = \begin{cases} +1 \text{ if } c = y \\ -1 \text{ else.} \end{cases}$$

The magnitude of the update is controlled by the learning rate η_w. The actual update step (6.4) moves the winning prototype even closer to the presented feature vector if $\mathbf{w}^*_\mu$ and the example carry the same label as indicated by

$\Psi(c^*_\mu, y^\mu) = +1$. On the contrary, $\mathbf{w}^*_\mu$ is moved farther away from $\boldsymbol{\xi}^\mu$ if the winning prototype represents a class different from y^μ, i.e. $\Psi(c^*_\mu, y^\mu) = -1$.

A popular initialization strategy is to place prototypes in the class-conditional mean vectors in the data set, i.e.

$$\mathbf{w}^j(0) = \sum_{\mu=1}^{P} \delta[y^\mu, c^j]\ \boldsymbol{\xi}^\mu \Big/ \sum_{\mu=1}^{P} \delta[y^\mu, c^j]$$

with the Kronecker-delta $\delta[i, j]$. If several prototypes are employed per class, independent random variations could be added in order to avoid coinciding prototypes, initially. More sophisticated initialization procedures can be realized, for instance by applying a K-means procedure in each class separately.

After repeated presentations of the entire training set, the prototypes should represent their respective class by assuming *class-typical* positions in feature space, ideally.

Numerous modifications of this basic LVQ scheme have been considered in the literature, see for instance [Koh90, NE14, BGH07, Gho21, Wit10] and references therein. In particular, several approaches based on differentiable cost-functions have been suggested. They allow for training in terms of gradient descent or other optimization schemes. Note that LVQ1 and many other heuristic schemes cannot be interpreted as descent algorithms in a straightforward fashion.

One particular cost function based algorithm is the so-called Robust Soft LVQ (RSLVQ) which has been motivated in the context of statistical modelling [SO03]. The popular Generalized LVQ (GLVQ) [SY95, SY98] is guided by an objective function that relates to the concept of large margin classification [CGBNT03]:

$$E^{GLVQ} = \sum_{\mu=1}^{P} \Phi(e^\mu) \quad \text{with} \quad e^\mu = \frac{d(\mathbf{w}^J_\mu, \boldsymbol{\xi}^\mu) - d(\mathbf{w}^K_\mu, \boldsymbol{\xi}^\mu)}{d(\mathbf{w}^J_\mu, \boldsymbol{\xi}^\mu) + d(\mathbf{w}^K_\mu, \boldsymbol{\xi}^\mu)}. \tag{6.5}$$

Here, the vector $\mathbf{w}^J_\mu$ denotes the closest of all prototypes which carry the same label as the example $\boldsymbol{\xi}^\mu$, i.e. $c^J_\mu = y^\mu$. Similarly, $\mathbf{w}^K_\mu$ denotes the closest prototype with a label different from y^μ. For short, we will frequently refer to these vectors as the *correct winner* $\mathbf{w}^J_\mu$ and the *incorrect winner* $\mathbf{w}^K_\mu$, respectively.

The cost function (6.5) in general comprises a non-linear and monotonically increasing function $\Phi(e)$. A particularly simple choice is the identity $\Phi(e) = e$, while the authors of [SY95, SY98] suggest the use of a sigmoidal $\Phi(e) = 1/[1 + \exp(-\gamma\, e)]$, where $\gamma > 0$ controls its steepness.

Negative values $e^\mu < 0$ indicate that the corresponding training example is correctly classified in the NPC scheme, since then $d(\mathbf{w}^J_\mu, \boldsymbol{\xi}^\mu) < d(\mathbf{w}^K_\mu, \boldsymbol{\xi}^\mu)$.[1] For large values of the steepness γ the costs approximate the number of misclassified

[1] Note that the argument of Φ obeys $-1 \leq e^\mu \leq 1$.

training data, while for small γ the minimization of E^{GLVQ} corresponds to maximizing the margin-like quantities e^μ.

A popular and conceptually simple strategy to optimize E^{GLVQ} is *stochastic gradient descent* in which single examples are presented in randomized order [RM51, Bot91, FP96]. In contrast to LVQ1, two prototypes are updated in each step of the GLVQ procedure:

GENERALIZED LVQ (GLVQ), stochastic gradient descent

- at time step t, select a single feature vector $\boldsymbol{\xi}^\mu$ with class label y^μ randomly from the data set $\mathbb{D}$ with uniform probability $1/P$.
- identify the *correct* and *incorrect winners*, i.e. the prototypes

$$\begin{aligned} &\mathbf{w}_\mu^J \quad \text{with} \quad d(\mathbf{w}_\mu^J, \boldsymbol{\xi}^\mu) = \min \left\{ \; d(\mathbf{w}^j, \boldsymbol{\xi}^\mu) \;\middle|\; c_\mu^j = y^\mu \right\}_{j=1}^M \\ &\mathbf{w}_\mu^K \quad \text{with} \quad d(\mathbf{w}_\mu^K, \boldsymbol{\xi}^\mu) = \min \left\{ \; d(\mathbf{w}^j, \boldsymbol{\xi}^\mu) \;\middle|\; c_\mu^j \neq y^\mu \right\}_{j=1}^M \end{aligned} \tag{6.6}$$

  with class labels $c_\mu^J = y^\mu$ and $c_\mu^K \neq y^\mu$, respectively.
- update both winning prototypes according to:

$$\begin{aligned} \mathbf{w}_\mu^J(t+1) &= \mathbf{w}_\mu^J(t) - \eta_w \, \frac{\partial \Phi(e^\mu)}{\partial \mathbf{w}_\mu^J} \\ \mathbf{w}_\mu^K(t+1) &= \mathbf{w}_\mu^K(t) - \eta_w \, \frac{\partial \Phi(e^\mu)}{\partial \mathbf{w}_\mu^K}, \end{aligned} \tag{6.7}$$

  where the gradients are evaluated in $\mathbf{w}_\mu^L(t)$ for $L = J, K$.

For the full form of the gradient terms we refer the reader to [SY95, SY98]. Note that – if Euclidean distance is used – the chain rule implies that the updates are along the gradients

$$\frac{\partial d(\mathbf{w}_\mu^L, \boldsymbol{\xi}^\mu)}{\partial \mathbf{w}_\mu^L} \propto \left(\mathbf{w}_\mu^L - \boldsymbol{\xi}^\mu \right) \quad \text{for} \quad L = J, K. \tag{6.8}$$

Moreover, the signs of the pre-factors in (6.7) are given by $\Psi(c^L, y^\mu) = \pm 1$ as in (6.1.3). In essence, GLVQ performs updates which move the correct (incorrect) prototype towards (away) from the feature vector, respectively. Hence, the basic concept of the intuitive LVQ1 is preserved in GLVQ.

In both LVQ1 and GLVQ, very often a decreasing learning rate η_w is used to ensure convergence of the prototype positions [RM51]. Alternatively, schemes for automated learning rate adaptation or more sophisticated optimization methods can be applied, see e.g. [SNW11], which we will not discuss here.

6.2 Distance measures and relevance learning

So far, the discussion focussed on Euclidean distance as a standard measure for the comparison of data points and prototypes. This choice appears natural and it is arguably the most popular one. One has to be aware, however, that other choices may be more suitable for real world data. Depending on the application at hand, unconventional measures might outperform Euclidean distance by far. Hence, the selection of a specific distance constitutes a key step in the design of prototype-based models. In turn, the possibility to choose a distance based on prior information and insight into the problem contributes to the flexibility of the approach.

6.2.1 LVQ beyond Euclidean distance

As discussed above, training prescriptions based on Euclidean metrics generically yield prototype displacements along the vector $(\boldsymbol{\xi}^\mu - \mathbf{w})$ as in Eq. (6.8). Replacing the Euclidean distance by a more general, differentiable measure $\delta(\boldsymbol{\xi}^\mu, \mathbf{w})$, allows for the analogous derivation of LVQ training schemes. This is conveniently done in cost function based schemes like GLVQ, cf. Eq. (6.5), but it is also possible for the more heuristic LVQ1, which will serve as an example here. As a generalization of Eq. (6.4) we obtain the analogous WTA update from example μ at time t:

$$\mathbf{w}^*_\mu(t+1) = \mathbf{w}^*_\mu(t) \; - \; \eta_w \; \Psi(c^*_\mu, y^\mu) \; \tfrac{1}{2} \; \frac{\partial \delta\left(\mathbf{w}^*_\mu, \boldsymbol{\xi}^\mu\right)}{\partial \mathbf{w}^*_\mu}. \tag{6.9}$$

Obviously, the winner $\mathbf{w}^*_\mu$ has to be determined by use of the same measure δ, for the sake of consistency.

Along these lines, LVQ update rules can be derived for quite general dissimilarities, provided the distance δ is differentiable with respect to the prototype positions. Note that the formalism does not require metric properties of δ. As a minimal condition, non-negativity $\delta(\mathbf{w}, \boldsymbol{\xi}) \geq 0$ should be satisfied for $\mathbf{w} \neq \boldsymbol{\xi}$ and $\delta(\boldsymbol{\xi}, \boldsymbol{\xi}) = 0$.

Note that cost function based approaches can also employ non-differentiable measures if one resorts to alternative optimization strategies which do not require the use of gradients [SNW11]. Alternatively, differentiable approximations of non-differentiable δ can be used, see [HV05] for a discussion thereof.

In the following, we mention just a few prominent alternatives to the standard Euclidean metrics that have been used in the context of LVQ classifiers. We refer to, e.g., [BHV16, BHV14, HV05] for more detailed discussions and further references.

Statistical properties of a given data set can be taken into account explicitly by employing the well-known *Mahalanobis distance* [Mah36]. This classical measure is a popular tool in the analysis of data sets. Duda et *al.* present a detailed discussion and several application examples [DHS00].

Standard *Minkowski distances* satisfy metric properties for values of $p \geq 1$

in

$$d_p(\boldsymbol{\xi}, \widehat{\boldsymbol{\xi}}) = \left[\sum_{j=1}^{N} \left|\xi_j - \widehat{\xi}_j\right|^p\right]^{1/p} \quad \text{for} \quad \boldsymbol{\xi}, \widehat{\boldsymbol{\xi}} \in \mathbb{R}^N, \tag{6.10}$$

which includes Euclidean distance as a special case for $p = 2$. Larger (smaller) values of p put emphasis on the components ξ_j and $\widehat{\xi}_j$ with larger (smaller) deviations $|\xi_j - \widehat{\xi}_j|$, respectively. For instance, in the limit $p \to \infty$ we have

$$d_\infty(\boldsymbol{\xi}, \widehat{\boldsymbol{\xi}}) = \max_{j=1,\ldots N} \left|\xi_j - \widehat{\xi}_j\right|.$$

Setting $p \neq 2$ has been shown to improve performance in several practical applications, see [BBL07, GW10] for specific examples.

The squared Euclidean distance can be rewritten in terms of scalar products:

$$d(\mathbf{w}, \boldsymbol{\xi})^2 = (\mathbf{w} \cdot \mathbf{w} - 2\mathbf{w} \cdot \boldsymbol{\xi} + \boldsymbol{\xi} \cdot \boldsymbol{\xi}). \tag{6.11}$$

So-called *kernelized distances* [Sch01] replace all inner products in (6.11) by a kernel function κ:

$$d_\kappa(\mathbf{w}, \boldsymbol{\xi})^2 = \kappa(\mathbf{w}, \mathbf{w}) - 2\kappa(\mathbf{w}, \boldsymbol{\xi}) + \kappa(\boldsymbol{\xi}, \boldsymbol{\xi}). \tag{6.12}$$

As in the SVM formalism, the function κ can be associated with a non-linear transformation from $\mathbb{R}^N$ to a potentially higher-dimensional feature space. In SVM training one takes advantage of the fact that data can become linearly separable due to the transformation, as discussed in Sec. 4.3. Similarly, kernel distances can be employed in the context of LVQ in order to achieve better classification performance, see [VKNR12] for a particular application.

As a last example, *statistical divergences* can be used to quantify the dissimilarity of densities or histogram data. For instance, image data is frequently characterized by color or other histograms. Similarly, text can be represented by frequency counts in a *bag of words* approach. In the corresponding classification problems, the task would be to discriminate between class-characteristic histograms. Euclidean distance is frequently insensitive to the relevant discriminative properties of histograms. Hence, the classification performance can benefit from using specific measures, such as statistical divergences. The well-known Kullback-Leibler divergence is just one example of many measures that have been suggested in the literature. For further references and an example application in the context of LVQ see [MSS+11, Mwe14]. There, it is also demonstrated that even non-symmetric divergences can be employed properly in the context of LVQ, as long as the measures are used in a consistent way.

6.2.2 Adaptive distances in relevance learning

In the previous subsection, a few alternative distance measures have been discussed. In practice, a particular one could be selected based on prior insights or according to an empirical comparison in a validation procedure.

The elegant framework of relevance learning allows for a significant conceptual extension of distance-based classification. It is particularly suitable for prototype systems and was was introduced and put forward in the context of LVQ in [HV02, SBH09, Sch10, Bun11, SBS+10, BSH+12], for instance.

Relevance learning has proven useful in a variety of applications, including biomedical problems and image processing tasks, see for instance [Bie17].

In this very elegant approach, only the parametric form of the distance measure is fixed in advance. Its parameters are considered adaptive quantities which can be adjusted or optimized in the data-driven training phase. The basic idea is very versatile and can be employed in a variety of learning tasks. We present here only one particularly clear-cut and successful example in the context of supervised learning: the so-called Matrix Relevance LVQ for classification [SBH09].

Similar to several other schemes (e.g. [WS09, BAP+12, BCLC15]), Matrix Relevance LVQ employs a generalized quadratic distance of the form

$$\delta_\Lambda(\mathbf{w}, \boldsymbol{\xi}) \;=\; (\mathbf{w}-\boldsymbol{\xi})^\top \;\Lambda\; (\mathbf{w}-\boldsymbol{\xi}) \;=\; \sum_{i,j=1}^{N} (w_i - \xi_i)\, \Lambda_{ij}\, (w_j - \xi_j). \qquad (6.13)$$

Heuristically, diagonal entries of Λ quantify the importance of single feature dimensions in the distance and can also account for potentially different magnitudes of the features. Pairs of features are weighted by off-diagonal elements, which reflect the interplay of the different dimensions. Note that for $\Lambda = I_N/N$, Eq. (6.13) recovers the simple squared Euclidean distance.

In order to fulfill the minimal requirement of non-negativity, $\delta_\Lambda \geq 0$, a convenient re-parameterization is introduced in terms of an auxiliary, unrestricted matrix $\Omega \in \mathbb{R}^{N\times N}$:

$$\Lambda = \Omega^\top \Omega, \quad \text{i.e.} \quad \delta_\Lambda(\mathbf{w}, \boldsymbol{\xi}) \;=\; \left[\Omega\; (\mathbf{w}-\boldsymbol{\xi})\right]^2. \qquad (6.14)$$

Hence, δ_Λ can be interpreted as the conventional squared Euclidean distance but after a linear transformation of feature space. Note that Eqs. (6.13, 6.14) define only a *pseudo-metric* in $\mathbb{R}^N$ since Λ can be singular with $\text{rank}(\Lambda) < N$ implying that $\delta_\Lambda(\mathbf{w}, \boldsymbol{\xi}) = 0$ is possible even if $\mathbf{w} \neq \boldsymbol{\xi}$.

Obviously, we could employ a fixed distance of the form (6.13) in GLVQ or LVQ1 as outlined in the previous sections. The key idea of relevance learning, however, is to consider the elements of the relevance matrix $\Lambda \in \mathbb{R}^{N\times N}$ as adaptive quantities which can be optimized in the data-driven training process.

Numerous simplifications or extensions of the basic idea have been suggested in the literature. The restriction to diagonal matrices Λ corresponds to the original formulation of Relevance LVQ in [HV02], which assigns a single, non-negative weighting factor to each dimension in feature space. Rectangular $(M \times N)$-matrices matrices Ω with $M < N$ can be used to parameterize a low-rank relevance matrix [BSH+12]. The corresponding low-dimensional intrinsic representation of data facilitates, for instance, the class-discriminative visualization of complex data [BSH+12]. The flexibility of the LVQ classifier is enhanced

significantly when local distances are used, i.e. when separate relevance matrices are employed per class or even per prototype [SBH09, BSH+12].

Here we restrict the discussion to the simplest case of a single, $N \times N$ matrix Ω corresponding to a global distance measure. The heuristic extension of the LVQ1 prescription by means of relevance matrices is briefly discussed in [BHV16] and its convergence behavior is analysed in [BHS+16].

Gradient based updates for the simultaneous adaptation of prototypes and relevance matrix can be derived from a suitable cost function. We observe that

$$\frac{\partial \delta_\Lambda(\mathbf{w}, \boldsymbol{\xi})}{\partial \mathbf{w}} = \Omega^\top \Omega \, (\boldsymbol{\xi} - \mathbf{w}) \quad \text{and} \quad \frac{\partial \delta_\Lambda(\mathbf{w}, \boldsymbol{\xi})}{\partial \Omega} = \Omega \, (\mathbf{w} - \boldsymbol{\xi}) \, (\mathbf{w} - \boldsymbol{\xi})^\top . \quad (6.15)$$

The full forms of the gradients with respect to the terms e^μ in the GLVQ cost function are presented in [SBH09], for instance. They yield the so-called Generalized Matrix Relevance LVQ (GMLVQ) scheme, which can be formulated as a stochastic gradient descent procedure:

GENERALIZED MATRIX LVQ (GMLVQ), STOCHASTIC GRADIENT DESCENT

- at time step t, select a single feature vector $\boldsymbol{\xi}^\mu$ with class label y^μ randomly from the data set $\mathbb{D}$ with uniform probability $1/P$.
- with respect to the distance δ_Λ (6.13) with $\Lambda = \Omega(t)^\top \Omega(t)$, identify the *correct* and *incorrect winners*, i.e. the prototypes

$$\mathbf{w}_\mu^J \quad \text{with} \quad \delta_\Lambda(\mathbf{w}_\mu^J, \boldsymbol{\xi}^\mu) = \min \left\{ \delta_\Lambda(\mathbf{w}^j, \boldsymbol{\xi}^\mu) \;\middle|\; c_\mu^j = y^\mu \right\}_{j=1}^M$$

$$\mathbf{w}_\mu^K \quad \text{with} \quad \delta_\Lambda(\mathbf{w}_\mu^K, \boldsymbol{\xi}^\mu) = \min \left\{ \delta_\Lambda(\mathbf{w}^j, \boldsymbol{\xi}^\mu) \;\middle|\; c_\mu^j \neq y^\mu \right\}_{j=1}^M \quad (6.16)$$

  with class labels $c_\mu^J = y^\mu$ and $c_\mu^K \neq y^\mu$, respectively.

- update both winning prototypes and the matrix Ω according to:

$$\mathbf{w}_\mu^J(t+1) = \mathbf{w}_\mu^J(t) - \eta_w \frac{\partial \Phi(e^\mu)}{\partial \mathbf{w}_\mu^J}$$

$$\mathbf{w}_\mu^K(t+1) = \mathbf{w}_\mu^K(t) - \eta_w \frac{\partial \Phi(e^\mu)}{\partial \mathbf{w}_\mu^K},$$

$$\Omega(t+1) \quad = \quad \Omega(t) \; - \; \eta_\Omega \frac{\partial \Phi(e^\mu)}{\partial \Omega}. \quad (6.17)$$

  where the gradients are evaluated in $\Omega(t)$ and $\mathbf{w}_\mu^L(t)$ for $L = J, K$.

In both GMLVQ and Matrix LVQ1, the relevance matrix is updated in order to decrease or increase $\delta_\Lambda(\mathbf{w}_\mu^L, \boldsymbol{\xi}^\mu)$ for the winning prototype(s), depending on the class labels in the, by now, familiar way.

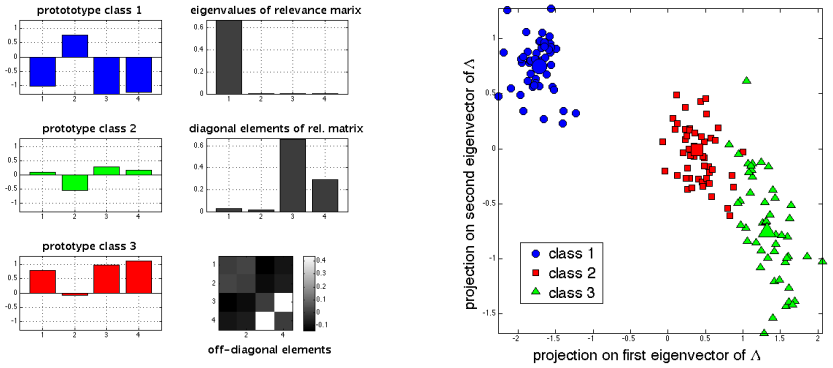


Figure 6.2: Visualization of the Generalized Matrix Relevance LVQ system as obtained from the z-score transformed Iris flower data set, see Sec. 6.2.2 for details.
Left panel: Class prototypes are shown as bar plots with respect to the four feature space components in the left column. The right column shows the eigenvalue spectrum of Λ, the diagonal elements of Λ, and the off-diagonal elements in a gray-scale representation (top to bottom).
Right panel: Projection of the $P = 150$ feature vectors onto the two leading eigenvectors of the relevance matrix Λ.

Frequently, the learning rate of the matrix updates is chosen to be relatively small, $\eta_\Omega \ll \eta_w$, in the stochastic gradient descent procedure. This follows the intuition that the prototypes should be enabled to follow changes in the distance measure. The relative scaling can be different in batch gradient realizations of GMLVQ as for instance in [VWB21].

The matrix Ω can be initialized as the N-dim. identity or in terms of independent random elements. In order to avoid numerical difficulties, a normalization of the form $\sum_i \Lambda_{ii} = \sum_{i,j} \Omega_{i,j}^2 = 1$ is frequently imposed [SBH09].

In the following we illustrate Matrix Relevance LVQ in terms of a classical benchmark data set. In the famous Iris flower data set [Fis36], four numerical features are used to characterize 150 samples from three different classes which correspond to particular species of Iris flowers. We obtained the data set as provided at [Lic13] and used one prototype per class and a global relevance matrix $\Lambda \in \mathbb{R}^{4\times4}$. For the training, we employed the freely available *beginner's toolbox for GMLVQ* with default parameter setting [VWB21]. An additional z-score transformation was applied, resulting in re-scaled features with zero mean and unit variance in the data set, see Sec. 8.1.1. This allows for an immediate interpretation of the relevances, without having to take into account the potentially different magnitudes of the features.

Figure 6.2 visualizes the obtained classifier. The resulting LVQ system achieves almost perfect, error-free classification of the training data. It also

displays very good generalization behavior with respect to validation or test set performance not presented here.

In the left panel, the prototypes after training and the resulting relevance matrix and its eigenvalues are displayed. As discussed above, the diagonal elements Λ_{ii} can be interpreted as the relevance of features i in the classification. Apparently, features 3 and 4 are the dominant ones in the Iris classification problem. The off-diagonal elements represent the contribution of pairs of different features. Here, also the interplay of features 3 and 4 appears to be important.

In more realistic and challenging data sets, Relevance Matrix LVQ can provide valuable insights into the problem. GMLVQ has been exploited to identify the most relevant or irrelevant features, e.g. in the context of medical diagnosis problems, see [Bie17] for a variety of applications. A recent application in the context of galaxy classification based on astronomical catalogue data is presented in [NWB18, NWB$^+$19].

In an N-dimensional feature space, the GMLVQ relevance matrix introduces $\mathcal{O}(N^2)$ additional adaptive quantities. As a consequence, one might expect strong overfitting effects due to the large number of free model parameters. However, as observed empirically and analysed theoretically, the relevance matrix displays a strong tendency to become singular and displays very low $\mathrm{rank}(\Lambda) = \mathcal{O}(1) \ll N$ after training [BHS$^+$16]. This effect can be interpreted as an implicit, intrinsic mechanism of regularization, which limits the complexity of the distance measure, effectively.

In addition, the low rank relevance matrix allows for the discriminative visualization of the data by projecting feature vectors (and prototypes) onto its leading eigenvectors. As an illustrative example, Fig. 6.2 displays the Iris flower data set.

6.3 Concluding remarks

Prototype-based models continue to play a highly significant role in putting forward advanced machine learning techniques. We encourage the reader to explore recent developments in the literature. Challenging problems, such as the analysis of functional data, non-vectorial data or relational data, to name only very few, are currently being addressed, see [BHV16, NE14] for further references. At the same time, exciting application areas are being explored in a large variety of domains.

Most recently, prototype-based systems are also re-considered in the context of Deep Learning [GBC16, Sch15, LBH18, Hue19]. The combination of multilayer network architectures with prototype- and distance-based modules appears very promising and is the subject of on-going research, see, for instance, [SHRV18, VMC16] and references therein.

Chapter 7

Model evaluation and regularization

Accuracy is not enough.
— Paulo Lisboa

In supervised learning the aim is to infer relevant information from given data, to parameterize it in terms of a model, and to apply it to novel data successfully. It is obviously essential to know or at least have some estimate of the performance that can be expected in the working phase.

In this chapter we discuss several aspects related to the evaluation and validation of supervised learning. In Sec. 7.1, we take a rather general perspective on overfitting and underfitting effects without necessarily addressing a particular classifier or regression framework. We present methods for controlling the complexity of neural networks in Sec. 7.2. Cross-validation and related methods are in the focus of Sec. 7.3. Specific quality measures beyond the simple *overall accuracy* for the evaluation of classifiers and regression systems are presented in 7.4.3. Finally, we address the importance of interpretable models in machine learning in Sec. 7.5.

7.1 Bias and variance, over- and underfitting

Different sources of error can influence the performance of supervised learning systems. Here we decompose the expected prediction error into two main contributions, see e.g. [HTF01, Bis06]: the so-called *bias* corresponds to systematic deviations of trained models from the true target, while the term *variance* refers to variations of the model performance when trained from different realizations

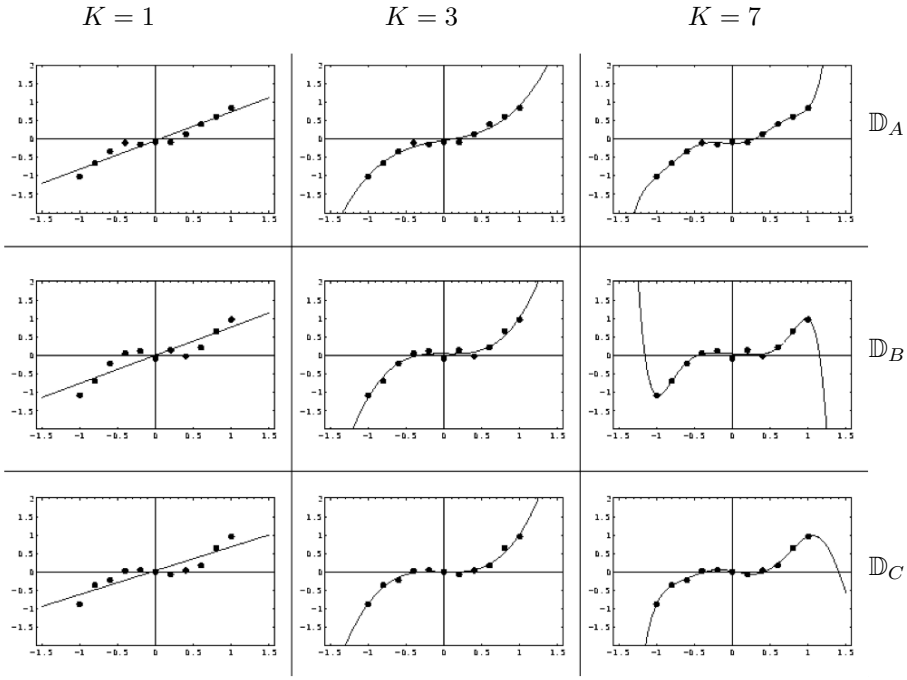


Figure 7.1: Illustration of the bias-variance dilemma in regression. In each row of graphs, a particular set of 10 points $\{x_i, y_i\}_{i=1}^{10}$ is approximated by least square linear regression ($K = 1$), by a cubic fit ($K = 3$), and by fitting a polynomial of degree seven ($K = 7$). Rows correspond to three randomized, independently generated data sets $\mathbb{D}_{A,B,C}$.

of the training data[1].

Frequently, a so-called *irreproducible error* is considered as a third, independent contribution [HTF01]. It could, for instance, stem from intrinsic noise in the test data which cannot be predicted even with perfect knowledge of the target rule. As the irreproducible error is beyond our control anyway, we refrain from including it in the discussion.

7.1.1 Decomposition of the error

For the purpose of illustration, we consider a simple one-dimensional regression problem. The obtained insights, however, carry over to much more complex systems. In our example, least squares fits are based on data sets of the form $\mathbb{D} = \{x^\mu, y^\mu\}_{\mu=1}^P$. They contain real-valued arguments $x^\mu \in \mathbb{R}$, e.g. equidistant values in $[-1, 1]$, and their corresponding target labels $y^\mu \in \mathbb{R}$. The data sets

[1]Note that the terms bias and variance are used in many different scientific contexts with area specific meanings.

represent a function $f(x)$ which is of course unknown to the learning system. We assume that the training labels are *noisy* versions of the true targets:

$$y^\mu = f(x^\mu) + r^\mu \quad \text{with} \quad \langle r^\mu \rangle = 0 \quad \text{and} \quad \langle r^\mu \, r^\nu \rangle = \rho^2 \delta_{\mu\nu} \tag{7.1}$$

with the Kronecker-delta $\delta_{\mu\nu}$. Hence, the deviation of the training labels from the underlying target function is given by uncorrelated, *zero mean* random quantities r^μ in each data point. Further details are irrelevant for the argument, but we could, for instance, consider independent Gaussian random numbers with variance ρ^2, i.e. $r^\mu \sim \mathcal{N}(0, \rho)$. We perform polynomial fits of the form

$$f_K(x) = \sum_{j=0}^{K} a_j x^j \quad \text{with coefficients } a_j \in \mathbb{R} \tag{7.2}$$

for powers x^j with maximum degree K. Given a data set $\mathbb{D} = \{x^\mu, y^\mu\}_{\mu=1}^P$, the a_j can be determined by minimizing the familiar quadratic deviation

$$E^{SSE} = \frac{1}{2} \sum_{\mu=1}^{P} \Big(f_H(x^\mu) - y^\mu \Big)^2. \tag{7.3}$$

Fig. 7.1 displays three randomized data sets $\mathbb{D}_{A,B,C}$ with $P = 11$ equidistant x^μ and noisy y^μ representing the underlying target function $f(x) = x^3$. For each of the three slightly different data sets, polynomial least square fits were performed with $K = 1$ (linear), $K = 3$ (cubic) and with degree $K = 7$. Hence, the same data sets were analysed by using models of different complexity.

In order to obtain some insight into the interplay of model complexity and expected performance, we consider the *thought experiment* of performing the same training/fitting processes for a very large number of slightly different data sets of the same size, which all represent the target.

We denote by $\langle \ldots \rangle_{\mathbb{D}}$ an average over many randomized realizations of the data set or – more formally – over the probability density of the training data in $\mathbb{D}$. In this sense, the expected total quadratic deviation of a hypothesis function f_H from the true target f in an arbitrary point $x \in \mathbb{R}$ is given by

$$\Big\langle \big[f_H(x) - f(x) \big]^2 \Big\rangle_{\mathbb{D}}, \tag{7.4}$$

where the randomness of $\mathbb{D}$ is reflected in the outcome f_H of the training. We could also consider the integrated deviation over a range of x-values, which would relate the SSE to the familiar generalization error. However, the following argument would proceed in complete analogy due to the linearity of, both, integration and averaging. Performing the square in Eq. (7.4) we obtain

$$\big\langle f_H^2(x) \big\rangle_{\mathbb{D}} - 2 \, \langle f_H(x) \rangle_{\mathbb{D}} \, f(x) + f^2(x). \tag{7.5}$$

Note that the true target $f(x)$ obviously does not depend on the data and can be left out from the averages.

For the sake of brevity, we omit the argument $x \in \mathbb{R}$ of the functions f_H and f in the following. Including redundant terms (*) which add up to *zero* we can rewrite (7.5) as

$$\underbrace{\langle f_H \rangle_{\mathbb{D}}^2}_{*} - 2\,\langle f_H(x) \rangle_{\mathbb{D}}\; f \;+\; f^2 \;+\; \langle f_H^2 \rangle_{\mathbb{D}} \underbrace{-2\,\langle f_H \rangle_{\mathbb{D}}^2 + \langle f_H \rangle_{\mathbb{D}}^2}_{*} \tag{7.6}$$

and obtain a decomposition of the expected quadratic deviation in x:

$$\left\langle \left[f_H - f \right]^2 \right\rangle_{\mathbb{D}} \;=\; \underbrace{\left(f - \langle f_H \rangle_{\mathbb{D}} \right)^2}_{\text{bias}^2} + \underbrace{\left\langle \left(f_H - \langle f_H \rangle_{\mathbb{D}} \right)^2 \right\rangle_{\mathbb{D}}}_{\text{variance}} . \tag{7.7}$$

The equality with (7.6) is straightforward to show by expanding the squares and exploiting that $\langle f_H \, \langle f_H \rangle_{\mathbb{D}} \rangle_{\mathbb{D}} = \langle f_H \rangle_{\mathbb{D}}^2 = \left\langle \langle f_H \rangle_{\mathbb{D}}^2 \right\rangle_{\mathbb{D}}$.

Hence we can identify two contributions to the total expected error:

- **Bias (squared):** $\left(f - \langle f_H \rangle_{\mathbb{D}} \right)^2$

  The bias term quantifies the deviation of the mean prediction from the true target, where the average is over many randomized data sets and corresponding training processes. A small bias indicates that there is very little systematic deviation of the hypotheses from the unknown target rule.

- **Variance:** $\left\langle \left(f_H - \langle f_H \rangle_{\mathbb{D}} \right)^2 \right\rangle_{\mathbb{D}}$

  The variance measures how much the individual predictions, obtained after training on a given $\mathbb{D}$, typically differ from the mean prediction. The observation of a small variance implies that the outcome of the learning is robust with respect to details of the training data.

Similar considerations apply to more general learning problems, including classification schemes [Dom00].

7.1.2 The bias-variance dilemma

Ideally, we would like to achieve low variance and low bias at the same time, i.e. a robust and faithful approximation of the target rule. Both goals are clearly legitimate, but very often they constitute conflicting aims in practice, as further illustrated in the following.

This is often referred to as the bias-variance dilemma or trade-off [HTF01, Bis95a, Bis06, Dom00]. It is closely related to the problem of overfitting in unnecessarily complex systems and its counterpart, the so-called underfitting in simplistic models.

The concept of bias and variance is illustrated in Fig. 7.2 (left panel). In the illustration, the fits of two different models are displayed in the space of adaptive quantities, e.g. weights in a neural network, while bias and variance are defined in terms of the prediction error. However, we can assume that the deviation in weight space from the target is in general associated with the error.

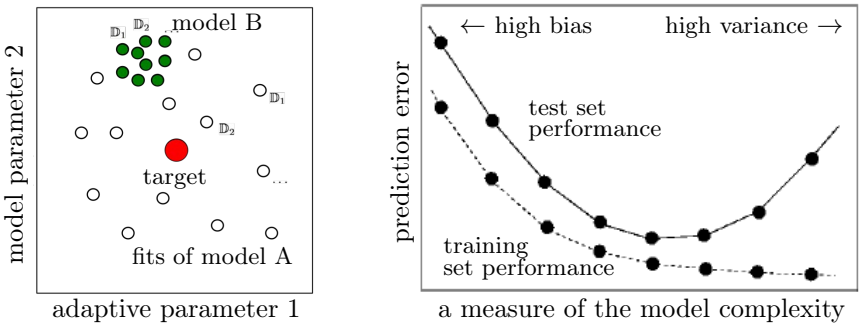


Figure 7.2: Left panel: Illustration of the bias-variance Dilemma. The true target is represented by the filled circle in the center. Fits obtained from different data sets $\mathbb{D}_i$ in model A (open circles) show low bias and large variance, while model B (filled circles) displays large systematic bias with smaller variance. **Right panel:** Schematic illustration of underfitting and overfitting (after [HTF01]): expected error with respect to a test set (generalization error) and training set performance as a function of the model complexity, e.g. K in the polynomial fits of Fig. 7.1.

Overfitting: low bias – high variance

In terms of our polynomial regression example we can achieve low bias by employing powerful models with large degree K. In the extreme case of $K = P$, for instance, we can generate models which perfectly reproduce the data points, $f_K(x^\mu) = y^\mu$ for all μ, in each individual training process.

Because the training labels themselves are assumed to be unbiased with $\langle y^\mu \rangle_{\mathbb{D}} = f(x^\mu)$, cf. Eq. (7.1), the averaged fit result will also be in exact agreement with the target in the arguments x^μ. In fact, since the objective function (7.3) of the training treats positive and negative deviations symmetrically as well, there is no reason to expect systematic deviations of the fits with $f_K(x) > f(x)$ or $f_K(x) < f(x)$ for all fits in some arbitrary value of x.

However, using a very flexible model with large K will result in fits which are very specific to the individual data set. As can be seen in Fig. 7.1, (right column), already for a moderate degree of $K = 7$, very different models emerge from the individual training processes.

For the sample points themselves, the variance of the nearly perfect fit would be essentially determined by the statistical variance of the training labels ρ^2 in Eq. (7.1). However, when interpolating or even extrapolating to $x \notin \{x^\mu\}_{\mu=1}^P$, the different fits will vary a lot in their prediction $f_K(x)$. Consider, for instance, the extrapolation to $x = \pm 1.1$ in Fig. 7.1 as a pronounced example of the effect.

Underfitting: low variance – high bias

If emphasis is put on the robustness of the model, i.e. low variance, we would prefer simple models with low degree K in (7.2). This should prevent the fits

from being overly specific to the individual data sets. As illustrated in the left column of Fig. 7.1, we achieve nearly identical linear models from the different data sets. However, a price is paid for the robustness: systematic deviations occur in each training procedure. We observe, for instance, that the linear fits (left column) obtained from $\mathbb{D}_{A,B,C}$ are virtually identical. However, they display quite large deviations from the sample points – which represent a non-linear function, after all. These deviations are systematic in the sense that they are reproduced qualitatively in each data set. For instance, for the first sample point $x^1 = -1$ we can see that always $f_{K=1}(x) > y^1$. As a consequence, interpolation and extrapolation will also be subject to systematic errors.

Matched model complexity

In our example, fits of degree $K = 3$ seem to constitute an ideal compromise. In the sample points, they achieve small deviations with no systematic tendency and, consequently, have relatively low bias. At the same time the fits $f_{K=3}(x)$ appear also robust against variations of the data set, corresponding to a relatively small variance.

This is of course not surprising, since polynomials of degree $K = 3$ perfectly match the complexity of the underlying, true target function. It is important to realize that this kind of information is rarely available in practical situations.

In fact, in absence of knowledge about the complexity of the target rule, it is one of the key challenges in supervised learning to select an appropriate model that achieves a good compromise with respect to bias and variance. In the following sections we will consider a variety of ways to control the complexity of a learning system with emphasis on feed-forward neural networks.

The trade-off

The above considerations suggest that there is a trade-off between the goals of small variance and bias [HTF01, NMB+18, Dom00]. Indeed, in many machine learning scenarios one observes such a trade-off which is illustrated in Fig. 7.2 (right panel). It shows schematically the possible dependence of the prediction performance in the training set and the generalization error (test set performance) as a function of the model complexity.

In our simple example, we could use the polynomial degree K as a measure of the latter. It could be also interpreted as, for instance, the degree of a polynomial kernel in the SVM, the number of hidden units in a two-layer neural network, or the number of prototypes per class in an LVQ system. Similarly, the x-axis could correspond to a continuous parameter that controls the flexibility of the training algorithm, e.g. a weight decay parameter or the training time itself [HKP91, Bis95a, Bis06].

Generically, we expect the training error to be lower than the generalization error for any model. After all, the actual optimization process is based on the available training examples.

Simplistic models that cannot cope with the complexity of the task display, both, poor training set and d due to large systematic bias. Increasing the

model's flexibility will reduce the bias and, consequently, training and test set error decrease with K in Fig. 7.2 (right panel). However, overly training set specific models display overfitting: while the training error typically decreases further with increasing K, the test set error displays a U-shaped dependence which reflects the increase of the model variance.

It is important to realize that the extent to which the actual behavior follows the scenario in a practical situation depends on the detailed properties of the data and the problem at hand. While one should be aware of the possible implications of the bias-variance dilemma, the plausibility of the above discussed trade-off must not be over-interpreted as a mathematical proof. Note that the decomposition (7.7) itself does not imply the existence of a trade-off, strictly speaking.

As argued and demonstrated in e.g. [Sch93] and [NMB$^+$18], a given practical problem does not necessarily display the U-shaped dependence of the generalization error shown in Fig. 7.2 (right panel). There is also no general guarantee that measures which reduce the variance in complex models will really improve the performance of the system [Sch93]. In many practical problems, however, the assumed bias-variance trade-off can indeed be controlled to a certain degree and may serve as a guiding principle for the model selection process.

According to the above considerations, a reliable estimate of the expected generalization ability would be highly desirable in any given supervised learning scenario. It would be very useful to be able to compare and evaluate the use of different approaches, e.g. SVM and LVQ, in a given practical problem. Similarly, the expected performance should guide the selection of model parameters like the number of hidden units in a neural network. The aim is to select the most suitable student complexity in a situation as sketched in Fig. 7.2. The same applies to selecting a training procedure and suitable parameter values, e.g. the learning rate.

In Sec. 7.3 we present the basic idea of how to obtain estimates of the generalization performance by means of *cross-validation* and related schemes.

7.1.3 Beyond the classical bias-variance trade-off (?)

The *unreasonable effectiveness of Deep Learning in Artificial Intelligence* [Sej20] seems to raise some doubts about the validity of the bias-variance trade-off. Deep Learning systems are frequently heavily over-parameterized with very large numbers of layers, units and weights. For instance, the currently very popular Large Language Models can comprise billions of adaptive parameters, see Table 2.1 in the preprint version of [BMR$^+$20]. According to the reasoning of the previous sections, one would expect serious overfitting effects in such extremely powerful systems. In practice, however, over-parameterized Deep Learning systems are trained and applied with great success.

In this context, a publication by Belkin et *al.* [BHMM19] has attracted a lot of attention. The authors discuss the so-called *double descent* or *peaking phenomenon* which is illustrated in Fig. 7.3. Beyond the classical under- and overfitting scenario of Fig. 7.2, the test error frequently undergoes a second

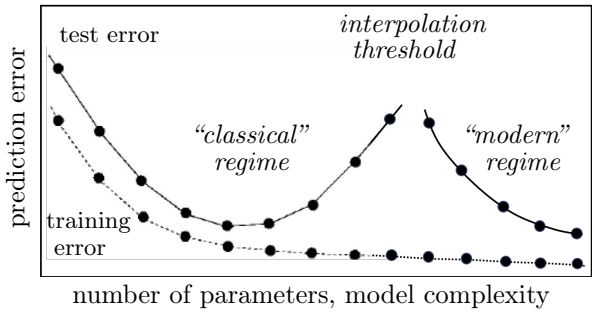


Figure 7.3: Illustration of the double descent phenomenon, after [BHMM19].

descent as a function of the number of adaptive parameters. This descent occurs in the over-parameterized regime, while the test error displays a peak at the so-called *interpolation threshold*. The illustration 7.3 refers to what is sometimes called *model-wise* double descent: for a given problem or data set, models of increasing complexity are considered. Similar peaking effects can be observed in models of fixed complexity with varying data set size in the so-called *sample-wise* double descent [You21, LVM$^+$20, Vie23].

Double descent is the subject of ongoing discussions and apparently has led several researchers and practicioners to the somewhat hasty conclusion that *the bias-variance trade-off generalization is wrong* [You21]. Moreover, it is often assumed that double descent is a relatively novel phenomenon that was discovered specifically in Deep Learning, motivating the terms *classical* and *modern regime* in Fig. 7.3. However, as already mentioned in [BHMM19], double descent occurs also in much simpler settings, including shallow networks and elementary regression systems. In [LVM$^+$20], the authors present a *brief prehistory of double descent*, pointing out that it had been observed already in basic learning problems like linear regression or perceptron training, e.g. [OKKN90].

Plausible explanations for the occurrence of double descent have been provided by several authors. It is important to note that, depending on the details of problem and method, it is incorrect to naively identify the number of parameters with the capacity or complexity of the model, as we suggestively did in Fig. 7.3. As one example, Daniela Witten presents an insightful discussion in terms of fitting cubic splines to a number of data points in [Wit20] (a twitter thread). There, the interpolation threshold corresponds to the situation in which the number of parameters exactly matches the number of data points. This is analogous to other regression or interpolation schemes, such as the poynomial fits discussed in Sec. 7.1. Right at the interpolation threshold, there is only one possible solution for a given data set and the resulting model is very sensitive to small variations or noise. Above the interpolation threshold, many fits are possible. The specific selection of the solution with the minimum norm of coefficients restricts the flexibility of the system drastically, resulting in the observed peaking and double descent. On the contrary, fitting under other types of reg-

ularization, cf. Sec. 7.2, would not necessarily display the peaking and double descent [Wit20]. The influence of implicit and explicit regularization on the emergence of double descent is also discussed in the context of ordinary least squares regression in [KLS20].

In general, explicit regularization as well as details of the training prescription can play an important role in determining the effective flexibility of a system. The important conclusion is that, if the model complexity is taken into account correctly, the bias-variance trade-off is still valid.

7.2 Controlling the network complexity

So far we have discussed the control of the student complexity in terms of the actual model design, i.e. by choosing the degree of a polynomial fit, the number of prototypes in LVQ, or the size of a hidden layer in a neural network. These choices are made prior to the actual training and can be evaluated by comparing different settings after training.

In a variety of approaches the (effective) complexity of a learning system is controlled by imposing constraints on the training process in a given architecture. Appropriate restrictions can prevent the training algorithm from fully exploring the space of adaptive quantities. We discuss two basic methods: the so-called *early stopping* strategy in Sec. 7.2.1 and the concept of *weight decay* in 7.2.2, respectively. The latter is an important example for *regularization* by introducing a penalty term into the objective function that guides the training. Here, we will use the term *regularization* more generally for all methods of implicit or explicit complexity control.

Constructive algorithms which incorporate the addition of units or layers into the training process are discussed in Sec. 7.2.3. In Sec. 7.2.4 we present so-called *pruning* procedures that remove *unnecessary* weights or units during or after training. Eventually, two techniques that are particular relevant in the context of Deep Learning, *weight-sharing* and *Dropout*, are presented in Sec. 7.2.5 and 7.2.6, respectively.

In practice, all these methods require or benefit from reliable estimates of the performance with respect to the prediction on novel data. For now we assume that such estimates are available, for instance by computing suitable error measures on a large representative test set. Practical methods like the well-known n-fold cross-validation will be discussed in Sec. 7.3.

7.2.1 Early stopping

A conceptually very simple idea is to end the training process before the system becomes overly specific to the data set. For example, if we *manually* stop gradient descent updates after a suitable number t_{max} of epochs, the resulting weight configuration may display a relatively low value of the objective function, yet without representing one particular local minimum too faithfully. Fig. 7.4 (left

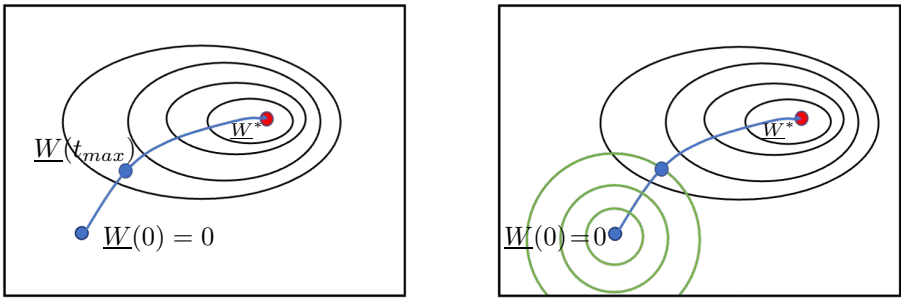


Figure 7.4: Schematic illustration of early stopping and weight decay. Ellipses correspond to contour lines of the objective function. The blue solid lines represent the unrestricted hypothetical updates by, for instance, gradient descent.
Left panel: after t_{max} epochs, the training is stopped, which hinders the weight configuration from reaching the local minimum $\underline{W}^*$ (red dot).
Right panel: weights are initialized *tabula rasa* and restricted to small norms. Circles correspond to lines of equal norm $|\underline{W}|$.

panel) shows an illustration of the effect of early stopping on the optimization process.

Being one of the most intuitive concepts of regularization, early stopping has been discussed very early in the context of neural networks, see for example [BC91, SL92]. A thorough discussion and further references can be found in text books as well, e.g. in [Bis06, GBC16].

The early stopping parameter t_{max} plays a role that is comparable to the degree K in the example of polynomial fits, cf.7.1. Note that t_{max} and other parameters that control the training process are often referred to as *hyper-parameters* in order to distinguish them from the actual adaptive quantities, e.g. weights and thresholds in a neural network.

In order to set discrete parameter like the number of hidden units in the network, we have to train different systems separately and compare their training and test set retrospectively. In early stopping we can monitor the system on the fly and stop as soon as overtraining effects set in. The proper choice of t_{max} based on heuristic criteria and the use of *cross-validation*, cf. Sec. 7.3.1 has been addressed in the literature, see e.g. [Pre97, Pre98, STW11] for examples and [GBC16, Bis06] for general discussions.

7.2.2 Weight decay and related concepts

We have encountered weight decay as a regularization technique already in the discussion of simple linear regression in Sec. 2.2.2. There, a penalty term was added to the objective function that prevents the L_2-norm of the weight vector from growing arbitrarily large. Depending on the perspective, this can be motivated heuristically in a pragmatic machine learning approach or on the

basis of assuming prior knowledge in the context of statistical learning theory [HTF01, HKP91, Bis95a, Bis06, DHS00]. In the linear regression problem, weight decay facilitates the construction or computation of a meaningful solution of the regression problem.

Here we extend the concept to the implementation of non-linear functions in non-linear networks. Modifications of weight decay, e.g. by considering general Minkowski-norms or heuristically motivated penalty terms are discussed at the end of this subsection.

Weight decay in non-linear neural networks

The concept of weight decay generalizes to a variety of classification and regression problems, including the training of non-linear layered neural networks, see [Hin86, KSV88] for early works. The influence of weight decay has also been investigated in model scenarios from the statistical physics of learning perspective, see [Bös96, ABS99, SR98] for examples.

Compared to the case of linear regression, the formulation of weight decay based on statistical learning theory, is less obvious for non-linear neural networks. However, the heuristic interpretation remains valid: restricting the magnitude of weights prevents the system from exploring the search space exhaustively and thus limits the effective complexity of the network. Figure 7.4 (right panel) illustrates the effect of limiting the Euclidean L_2-norm of the weights $\underline{W}$.

The effect of weight decay can be motivated in terms of a single non-linear unit. Assume that the activation of the unit is given by the non-linear function

$$g(x) \in \mathbb{R} \quad \text{with} \quad x = \mathbf{w} \cdot \boldsymbol{\xi}, \quad \mathbf{w}, \boldsymbol{\xi} \in \mathbb{R}^N. \tag{7.8}$$

For weight vectors with small norm $|\mathbf{w}| \approx 0$ we have $x \approx 0$ and a Taylor expansion implies that the activation is approximately

$$g(x) \approx g(0) + g'(0)\, x + \frac{1}{2} g''\, x^2 + \ldots \tag{7.9}$$

Thus, the activation is effectively linearized. By the same argument, the output of a layered network of in principle non-linear units, will become nearly linear if the magnitude of the weights are very small.

If we represent the set of all weights in a network by $\underline{W}$ and consider gradient descent with respect a cost function $E(\underline{W})$, we can limit the norm of $\underline{W}$ heuristically by reducing the magnitude of weights in or after each update step:

$$\underline{W}(t+1) = \underline{W}(t)\Big[1 - \gamma\Big] - \eta\ \nabla_W E|_{W(t)} \tag{7.10}$$

with the (small) weight decay parameter $\gamma > 0$. The update can also be interpreted as gradient descent with respect to a modified objective function:

$$\widehat{E} = E + \frac{1}{2}\,\frac{\gamma}{\eta}\,|\underline{W}|^2 \quad \text{with} \quad \nabla_W \widehat{E} \,=\, \nabla_W E \,+\, \frac{\gamma}{\eta}\,\underline{W} \tag{7.11}$$

which also leads to (7.10).[2]

As an alternative to the use of a penalty term, sometimes a constraint of the type $|\underline{W}|^2 \leq c$ with constant $c > 0$ is imposed. This can be done explicitly by projecting back onto the sphere in weight space $|\underline{W}|^2 = c$ whenever the constraint is violated by the updates. This so-called *max-norm* regularization is considered in [SHK+14] in combination with Dropout, see Sec. 7.2.6. Weight decay as given by Eqs. (7.10, 7.11) can be interpreted as a *soft* implementation with Lagrange parameter γ/η.

Variants of weight decay

Following an argument presented in e.g. [HKP91] and [NH92b], the penalty term $\propto \sum_j W_j^2$ in Eq. (7.11) favors several non-zero weights of similar magnitude over a combination of zero weights with a few larger W_j. This can be seen by comparing the penalty of a pair of weights $\{w/2, w/2\}$ to that of $\{w, 0\}$:

$$\left(\frac{w}{2}\right)^2 + \left(\frac{w}{2}\right)^2 < w^2 + 0^2.$$

In the context of sparse classifiers or regression systems the aim is a system with a significant fraction of zero or very small weights, which could be removed. This can be achieved by employing modified penalty terms and update rules, for instance the one discussed in [HKP91]:

$$\widehat{E} = E + \frac{1}{2}\frac{\gamma}{\eta}\sum_j \frac{W_j^2}{1+W_j^2} \quad \text{with} \quad \frac{\partial \widehat{E}}{W_k} = \frac{\gamma}{\eta}\frac{W_k}{(1+W_k^2)^2}. \tag{7.12}$$

Compared to (7.11) this leads to a W_k-dependent decay term in the gradient descent updates which favors the further decrease of small weights. Consequently, the modified weight decay, together with a removal of weights with $|W_j| \approx 0$ after training, can serve as a method for *pruning* the neural network. Further methods for the removal of *unnecessary* weights in a trained network are briefly presented in Sec. 7.2.4.

A family of systematic variations of weight decay can be based on L_p-regularization, where the penalty is given by the more general term

$$||\underline{W}||_p = \Big(\sum_j W_j^p\Big)^{1/p}. \tag{7.13}$$

Eq. (7.13) constitutes a proper norm only for $p \geq 1$. Formally, extensions to $0 < p < 1$ are possible but involve mathematical subtleties. The familiar Euclidean norm is recovered for $p = 2$. Another case of particular interest is $p = 1$, corresponding to the so-called Manhattan norm. In linear regression, a (non-differentiable) penalty term proportional to $\sum_j |W_j|$ appears in the Lagrangian form of the so-called *Least Absolut Shrinkage And Selection Operator*

[2] Here, the scaling of γ with η merely guarantees formal equivalence with Eq. (7.10).

(LASSO), see [HTF01] for a thorough discussion and comparison with L_2-based *Ridge Regression*. Similar to the above discussed heuristic penalty (7.12), L_1 regularization can also be used to enforce some weights to become exactly *zero*. It thus also relates to *feature selection* and *pruning*, cf. Sec. 7.2.4.

7.2.3 Constructive algorithms

In the context of classification we have discussed constructive algorithms such as the so-called tiling algorithms, cf. Sec. 4.2.2. In these schemes, units or layers are added to the network until the given, labeled data set can be implemented. Similar ideas have been applied in layered networks for regression. Reviews of suggested methods and corresponding references can be found in [KY97, SC10]. A key issue of all constructive algorithms is the need to avoid overfitting due to the addition of too many units. Suitable stopping criteria are discussed in [KY97].

A popular constructive algorithm for regression is the so-called Cascade-Correlation algorithm suggested by Fahlmann and Lebiere in 1997 [FL90]. Very similar to the tiling-like algorithm (4.9), the original Cascade-Correlation scheme adds hidden units one at a time. However, in contrast to the construction in (4.9), the added node receives input from all previous hidden units. The new unit is trained by maximizing the correlation of its output with the residual error achieved so far, see [FL90] for details. The specific algorithm can lead to an architecture with many single unit hidden layers, i.e. a *deep* and *narrow* network as opposed to the *shallow* and *wide* architectures considered in Sec. 4.2.2 or 5.1.2.

7.2.4 Pruning

The concept of *pruning* or *trimming* a neural network is diametrically opposed to that of constructive algorithms. The idea of pruning is to first train a relatively complex system which is capable of realizing the desired task to a satisfactory extent with respect to the given training data. In order to avoid or reduce overfitting, the network is then simplified by removing weights and/or nodes from the system without detoriating its performance too much. The dilution of trained networks has been considered very early, see [HKP91] for references.

Pruning is usually done after training, and the system may be retrained thereafter. Likewise, it can be realized in intermediate steps of the training procedure. Pruning is considered an important ingredient of neural network training also in recent applications of machine learning. However, as Hugo Tessier puts it on https://towardsdatascience.com [Tes21]:
"Unfortunately, the dozens, if not hundreds of papers published each year are revealing the hidden complexity of a supposedly straightforward idea."

In the literature, many recently suggested pruning procedures appear to be closely related to early works like [LDS90, HSW93], see [Ree93] for a survey. According to reviews like [BGFG20], many authors fail to relate and compare their work properly to early publications in the area. Consequently, we restrict

ourself to the discussion of some early works that represent the basic ideas and inspired later, more specific schemes. We also limit the discussion to strategies for the removal of weights rather than entire units or layers. The latter is frequently referred to as *structural pruning*, see for instance [AK13] for a review and references.

Modified weight decay procedures can be employed for the selection of *unimportant* weights as outlined in Sec. 7.2.2. In the following we present two classical methods which are not based on weight decay, but remove weights explicitly according to their importance for the minimization of the cost function that guides the training process.

Optimal Brain Damage (OBD) and Optimal Brain Surgeon(OBS)

Two classical pruning algorithms with slightly macabre names have been suggested in the 1990s already: LeCun, Denker and Solla's *Optimal Brain Damage* (OBD) [LDS90] and the *Optimal Brain Surgeon* (OBS) by Hassibi, Stork and Wolff [HSW93]. Both schemes are based on a ranking of individual weights W_j according to their *saliency*, i.e. the sensitivity of the system with respect to their removal (setting $W_j = 0$).

Assume a network has been trained and the weight configuration is sufficiently close to a local minimum $\underline{W}^*$ of the cost function $E(\underline{W})$ with $E(\underline{W}^*) = E^*$. A Taylor expansion for $\underline{W} = \underline{W}^* + \underline{U}$ yields

$$E(\underline{W}) \approx E^* + \frac{1}{2}\sum_{i,j} U_i\, H^*_{ij}\, U_j \;=\; E^* + \frac{1}{2}\sum_i H^*_{ii}\, U_i^2 + \frac{1}{2}\sum_{i,j(i\neq j)} U_i\, H^*_{ij}\, U_j. \quad (7.14)$$

Here, the linear term vanishes in the minimum and $H^*_{ij} = \left.\frac{\partial^2 E}{\partial W_i \partial W_j}\right|_*$ is an element of the Hesse matrix computed in $\underline{W}^*$. Moreover, we have simply separated diagonal and off-diagonal contributions of the quadratic term.

In [LDS90], the authors suggest to avoid the computation of the full, potentially very high-dimensional Hesse matrix and focus on the diagonal terms. Assuming that H^* is approximately diagonal, i.e. dominated by the H^*_{jj}, Eq. (7.14) reduces to

$$E(\underline{W}) \approx E^* + \frac{1}{2}\sum_j H^*_{jj} U_j^2. \quad (7.15)$$

An efficient computation of the diagonal elements of H^* is also outlined in [LDS90]. The so-called saliencies

$$s_k = H^*_{kk} W_k^2 \quad (7.16)$$

can be used as a guideline for the selection of weights that could be removed from the system without increasing E significantly. The OBD procedure can be summarized as follows (after [LDS90]):

OPTIMAL BRAIN DAMAGE (OBD) (7.17)

1. Train a network until a local minimum $\underline{W}^*$ of E is reached or sufficiently well approximated
2. Compute the diagonal second derivatives H^*_{kk}
3. Compute the saliencies $s_{kk} = H^*_{kk} W_k^2$
4. Sort the weights by saliency and set some low-saliency weights to *zero*
5. Potentially retrain the remaining weights
6. Go to step 1.

The alternative scheme of Optimal Brain Surgery (OBS) as suggested in [HSW93] follows a similar line of thought. However, there the saliencies are defined as

$$\widehat{s}_k = \frac{1}{2} \frac{W_k^2}{[H^{*-1}]_{kk}}. \tag{7.18}$$

Compared to Eq. (7.18), the factor H^*_{kk} is replaced by $1/[H^{*-1}]_{kk}$ with H^{*-1} denoting the inverse of the Hessian H^*. Note that for diagonal matrices H^* the definitions (7.16) and (7.18) are identical. Further differences between OBD and OBS concern details of the procedure, see [HSW93].

Numerous modifications and extensions of OBD and OBS have been proposed in the literature. For instance, the authors of [PHL96] suggest a scheme which computes the saliencies with respect to the estimated generalization error rather than based on the cost function or training error. Hence, their pruning procedures, termed γOBS and γOBD, are more closely related to the actual goal of training.

7.2.5 Weight-sharing

An elementary and intuitive method to reduce the flexibility of a neural network is to consider subsets of weights that assume the same value. This so-called *weight-sharing* usually relies on insights into the problem and data. Assume that we want to apply a set of (adaptive) filters to patches of a given input image. It appears natural to train and use only one shared set of weights per type of filter. This reduces the effective number of weights drastically and consequently simplifies the training a lot. The strategy is ubiquitous in the context of, for instance, Convolutional Neural Networks for image classification.

For defining the sets of shared weights in advance, prior knowledge is required which might not always be available. Nowlan and Hinton suggested a soft version of weight-sharing in [NH92b, NH92a]. They propose a penalty term

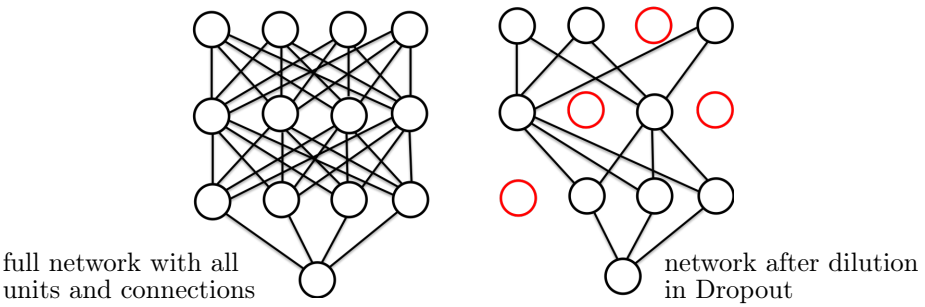


Figure 7.5: Illustration of regularization by Dropout (redrawn after [SHK+14]). **Left panel:** the network with all nodes and weights. **Right panel:** four randomly selected units (red circles) and their incoming and outgoing weights are temporarily removed.

that enforces the distribution of weights to follow a mixture of Gaussians, the parameters of which are also subject to updates in the training process. Eventually, weights can be grouped according to their membership to the contributing Gaussians. Within these clusters, weights can display very similar values but are not necessarily identical.

Further discussions of *hard or soft* weight-sharing and references can be found in e.g. [UMW17]. More recently, the authors of [OLLB20] conclude that weight-sharing "is a pragmatic optimization approach" but "it is not a necessity in computer vision applications." They also argue that approximate weight-sharing emerges in a self-organized way in unrestricted CNN, for example when trained from images that display translational invariance.

7.2.6 Dropout

We have already discussed methods from the *dilution* of networks by removing weights from a given network, either by weight decay and related methods or by applying pruning techniques after training, see [HKP91] as well as sections 7.2.2 and 7.2.4.

In the so-called Dropout regularization [HSK+12, SHK+14, GBC16], a random dilution is applied in the training process. More concretely: in every individual update step, e.g. in stochastic gradient descent, individual input and hidden units are excluded from the network. Only the remaining subnetwork is trained as usual. DropConnect has been suggested as a variation of the basic idea that excludes randomly selected weights instead of units [WZZ+13].

In Dropout, at each update step a randomly determined subnetwork of limited complexity is considered. Therefore, Dropout reduces the flexibility of the system and restricts its adaptation to the details of the training data. The removal of nodes occurs independently with probability $(1-p)$, the hyperparameter p determines the fraction of units present in the subnetwork. According

to [SHK+14, GBC16], a value of $p = 1/2$ is typically used for hidden units, while p is close to 1 for input units (e.g. $p = 0.8$). Note that an individual Dropout dilution could by chance remove an entire layer or some other way cut all connections from input to output. Such configurations have to be excluded explicitly, but are very unlikely to occur in large networks.

In the working phase and for testing purposes, the full network is used. Updating with Dropout will yield larger individual weights than conventional training. This is compensated for by multiplying all weights that were included in the Dropout by a factor p in the full network.

Dropout can be interpreted as to simulate the training of an ensemble of simpler systems (the subnetworks). In the working phase, the complete network yields an estimate of the corresponding ensemble average. Moreover, due to this analogy, the potential usefulness of Dropout goes beyond the purpose of regularization: in the working phase it can be used for uncertainty estimation, see [GG16].

While Dropout and DropConnect were introduced and are mostly used in the context of deep neural networks, the concept can be transferred to other machine learning systems, see [RKSV20] for the consideration of Dropout in Learning Vector Quantization.

7.3 Cross-validation and related methods

In supervised learning, the availability of well-defined performance measures allows us to formulate the training process as the optimization of a suitable objective function. However, one has to be aware that this does not necessarily reflect the ultimate goal of the learning. The cost function can only be defined with respect to the training set, while the generic goal of machine learning is to apply the inferred hypothesis to novel, unseen data. Objective functions serve, at best, as proxies for the actual aim of training.

As a consequence, the strict minimization of the training error, for example, can be even counter-productive as it may lead to overtraining or overfitting effects. From a more positive perspective, this also implies that we should not take optimization too seriously in the machine learning context. For instance, the existence of local minima in gradient descent based learning frequently turns out much less problematic than expected. In fact, the very success of simple-minded techniques like stochastic gradient descent relates to the fact that strict minimization of the cost function is usually not the primary goal of machine learning and could be even harmful. In this sense, the use of SGD can be interpreted as an implicit regularization which helps to avoid overfitting. Similarly, several of the explicit regularization techniques discussed in the previous sections hinder the strict minimization of the cost function in order to achieve better generalization behavior.

On the downside, it becomes necessary to acquire reliable information about the expected performance on novel data, if we do not want to face unpleasant surprises in the working phase. Clearly, the training itself and the performance

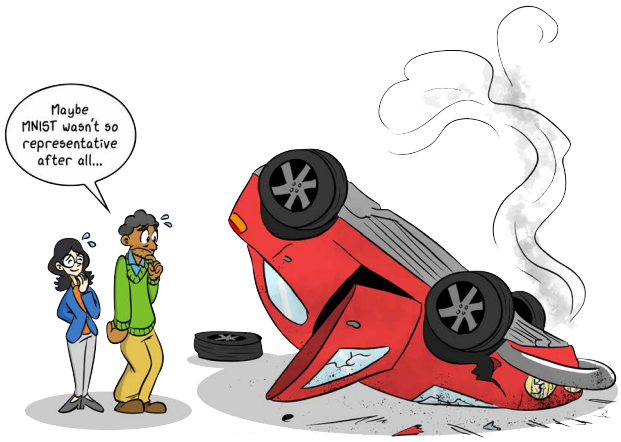


Figure 7.6: The requirement that the training data should be representative of the actual task at hand seems obvious, but is not always met in practice. © Jonathan van Engelenhoven, see https://www.instagram.com/banjoofjustice for more of his work. Cartoon reproduced from [Vie23] with kind permission of the artist.
Remark: MNIST (see e.g. https://paperswithcode.com/dataset/mnist) is a popular *benchmark* database of handwritten digits.

on the actual training data does not provide us with such insights.

Validation procedures can be employed which allow us to at least estimate the expected generalization performance [HTF01, Bis06]. The key idea is rather obvious: split the available data into a training set and a disjoint test set of examples. The former is then used for the adaptation of the model, which is eventually applied to the test set. This way, we can simulate working phase behavior while using only the available data.

Obviously, the data is assumed to be representative for the task at hand, see Fig. 7.6 for a tounge-in-cheek illustration. This crucial assumption was already discussed in Sec. 2.1.2 in the context of the generic workflow of supervised learning.

7.3.1 n-fold cross-validation and related schemes

The simple idea of splitting the available data randomly into one training and one test set has several problems:

- If only relatively few examples are available, as it is very often the case in practical problems,[3] we cannot afford to disregard the information con-

[3]Image classification tasks have become one of the few prominent exceptions due to the availability of very large databases.

tained in a subset of these in the training.

- The composition of the subsets could be *lucky* or *unlucky* in the sense that the test set might contain only very *difficult* or very *easy* cases. As a consequence, the test set performance might be overly pessimistic or optimistic, respectively.
- Performing a single test only cannot give valid insight into the robustness of the system with respect to details of the training set, i.e. the variance in the sense of Sec. 7.1.1.

All these issues are addressed in a very popular standard approach known as n-fold cross-validation, see e.g. [HTF01, Bis06, Ras18]. The idea is to split the available data randomly into a number n of disjoint subsets of (nearly) equal size, train on $n-1$ of the subsets and use the remaining subset for the validation:

n-FOLD CROSS VALIDATION (7.19)

- Generate subsets of $\mathbb{D}$ of (approximately) equal size:
  $\mathbb{D} = \cup_{i=1}^{n} \mathbb{D}_i$ with $\cap(\mathbb{D}_i, \mathbb{D}_j) = \emptyset$ for $i \neq j$ and cardinalities $|\mathbb{D}_i| = P/n$.
- For $i = 1 : n$
  - Train the system on $\mathbb{D}_i^{train} = \mathbb{D} \setminus \mathbb{D}_i$
  - Determine a suitable performance measure on $\mathbb{D}_i$
- Compute average and variance of the performance measures over the n training processes.

For each split we train the classifier or regression system on $(1 - 1/n)\, P$ examples and evaluate its performance with respect to the P/n left out samples. Eventually, we have obtained n systems trained on slightly different data sets with n estimates of the performance, for instance in terms of the accuracies of a classifier or the MSE in regression problems.

While the n validation sets are disjoint, we have to be aware that the training sets strongly overlap. The obtained estimates of the generalization error and, even more so, of the training error are definitely not statistically independent. Hence, the mean or variance obtained over the n-fold training process should not be over-interpreted. Nevertheless, the procedure will provide us with some insight into the expected generalization performance and the robustness of the system with respect to small changes in the training set.

Obviously, the parameter n in n-fold cross validation will influence the workload and the quality of the results:

- For large n, many training processes have to performed, each one based on a large fraction of the available data. In turn, the individual validation sets will be relatively small.

- For small values of n we obtain fewer, but more reliable individual estimates from larger validation sets. At the same time, the computational workload is reduced in comparison with the use of larger n. However, averages are performed over few individual results, only. Moreover, each training process make use of a relatively small subset of the data and cannot take full advantage of the available information.

In a practical situation, the choice of n will depend primarily on the number P of available examples to begin with. For a more detailed discussion and corresponding references see, for instance, [HTF01, Ras18]. In the literature, a canonical value of $n = 10$ has become a standard choice, apparently.

Variants

Many variations of the basic idea of cross-validation have been considered in the literature [Ras18]. The split into n disjoint subsets of data may also suffer from lucky/unlucky set composition. Therefore, one often resorts to a few repetitions of the n-fold scheme, performed with randomized splits and an additional average over the realizations.

In principle one could aim at realizing all possible splits of the data into $(P-p)$ training and p validation examples. Their number $\binom{P}{p}$ grows rapidly with P and the computational costs can become unrealistically high. Alternatively, in *repeated random sub-sampling validation*, also known as *Monte Carlo cross validation*, one generates a number of independently generated random splits into $P - p$ and p examples and computes averages and variances accordingly. This is closely related to so-called *bootstrap* methods [HTF01], which differ from cross-validation by resampling subsets of data with replacement. We refrain from a thorough comparison and discussion of the advantages and disadvantages of these variations of cross-validation. We refer the reader to, for example, [Efr83, ER97, Bur89, BHT23] and to textbooks like [HTF01, Bis95a, Bis06].

Leave-one-out cross-validation

As a popular extreme case, one often resorts to the so-called Leave-One-Out validation [HTF01, Bis95a, Bis06, Ras18], in particular for very small data sets. It follows the idea of cross-validation, but selects just one example as the smallest possible validation set in each training run. Hence, we set $n = P$ and run P training processes to obtain an average of the performance measure of interest.

It is important to note that the Leave-One-Out estimate can be unreliable. It even bears the risk of systematically yielding misleading results: In very small data sets, leaving out one sample from a specific class can lead to a bias in the training set towards the other class(es), which may result in overly pessimistic estimates of the generalization performance [Ras18, APW+09]. A modification that mildens the effect is known by the self-explanatory name *Leave-one-out from each class*, generating validation sets that represent all classes [APW+09] with equal weight.

7.3.2 Model and parameter selection

The above discussed validation schemes can be employed in the context of model selection and, similarly, for the setting of parameters or hyper-parameters [Ras18]. We can use, for instance, n-fold cross-validation to compare the expected performance of different classifiers or regression systems. We can also employ it to determine the size of a hidden layer in a feed-forward neural network, to set algorithm parameters like the learning rate in gradient descent, or to select a particular kernel in an SVM, to name just a few examples. In the illustration of Fig. 7.2, for instance, we would select the model complexity that corresponds to the minimum of the U-shaped generalization error curve.

Similarly, (hyper-)parameters of the regularization schemes discussed in Sec. 7.2 can be tuned according to the performance of the system w.r.t. the validation sets. However, one has to be aware of the risk to over-interpret or even mis-use the results of cross-validation. As an example, consider gradient based training of a network with one hidden layer on a given data set $\mathbb{D}$. Assume that, on the basis of n-fold cross-validation, we conclude that systems with, say, $K = 3$ hidden units yield the best generalization ability.[4]

Is it justified to expect the observed, averaged performance for $K = 3$ when applying the system to novel data? The problem is that we have used all of $\mathbb{D}$ to determine the supposedly best parameter setting. This constitutes a data-driven learning process and could be subject to overfitting effects in itself: The supposedly best choice of K may be very specific to $\mathbb{D}$ and could fail in the working phase.

In order to obtain a more reliable estimate of the expected performance we would have to perform an extended validation procedure. We could split $\mathbb{D}$ into training set $\mathbb{D}^{train}$ and $\mathbb{D}^{test}$ once, then apply n-fold cross-validation on $\mathbb{D}^{train}$ in order to determine a suitable value of K. Eventually, a system with the supposedly best setting can be re-trained on $\mathbb{D}^{train}$ and validated on $\mathbb{D}^{test}$ to obtain a more realistic estimate. Now, of course, we face the problem of lucky/unlucky set compositions again, which suggests to perform a full loop along the lines of n-fold cross-validation (or one of the discussed variants).

Strictly speaking, this has to be done separately for every independent parameter or hyperparameter in an additional *layer* of validation. Obviously, practical limitations apply, in particular when only small data sets are available.

7.4 Performance measures for regression and classification

Despite the conceptual clarity of supervised learning, even the choice of an appropriate measure of success can constitute a non-trivial issue in practice.

[4] For the difficulty to even define "the best" see the next sections.

7.4.1 Measures for regression

In regression, a differentiable objective or loss function typically guides the training process. It appears natural to consider the same function also for the evaluation of a system in validation or working phase. Most frequently, the familiar mean squared error (MSE) is used in both contexts.

Depending on the application domain, alternative measures different from the loss function can be employed for the evaluation of regression models. Two popular measures that are essentially different from the MSE are:

- Mean Absolute Error [WM05]

$$MAE = \frac{1}{Q} \sum_{\mu=1}^{Q} |\sigma^\mu - \tau^\mu| \tag{7.20}$$

  which weights deviations from the target differently than the MSE in a set of Q test or validation samples. The MAE also satisfies $0 \leq MAE \leq \infty$ where lower values correspond to better quality of the regression.

- Coefficient of Determination (CoD) [DS98]. [5]

$$CoD = 1 - \frac{\sum_\mu (\sigma^\mu - \tau^\mu)^2}{\sum_\mu (\tau^\mu - \langle\tau\rangle^\mu)^2} \tag{7.21}$$

  where $\langle\tau^\mu\rangle$ is the mean target value in the data set. The CoD compares the mean squared deviation of the predictions from the targets, scaled by the variance of the target values in the data set. The measure satisfies $-1 \leq CoD < 1$, a value of $CoD = 0$ indicates that all predictions are $\sigma^\mu = \langle\tau^\mu\rangle$, $CoD = 1$ corresponds to perfect regression with all $\sigma^\mu = \tau^\mu$.

A variety of further evaluation criteria, e.g. based on distance metrics or correlations, is available in the literature. A typology of measures is provided in [Bot19].

7.4.2 Measures for classification

The actual goal of machine learning for classification problems is to achieve a model that assigns data to the correct class with high probability in the working phase. Very often, the actual objective functions used in training provide at best a *proxy* of this ultimate goal. Loss functions for probabilistic classifiers based on cross-entropy like (5.23) and (5.24) could be used for training and evaluation.

Most frequently, the evaluation of *crisp* classifiers is guided by criteria that directly relate to the accuracy with respect to the validation or test data. An overview of a variety of performance measures for classification is given in [SL09].

Assume we are comparing the performances of two different classifiers, A and B, which have been trained to perform a given binary classification. By means

[5] In the literature, the CoD is most frequently termed R^2. The author refuses to denote a quantity that can become negative by a square.

of cross-validation we can obtain the estimates for the generalization ability in terms of the overall error as, say,

$$\epsilon_g^A = 0.05 \;\text{ and }\; \epsilon_g^B = 0.30.$$

Apparently, we could conclude that A is the better classifier and should be used in the working phase.

A closer look into the available data $\mathbb{D}$, however, might reveal that it consists of 95% class 1 samples, while only 5% of the data represent class 2. We furthermore might find that classifier 1 trivially assigns all feature vectors to class 1, resulting in 95% accuracy in $\mathbb{D}$. On the other hand, model B might have *learned* from the data and provides 70% correct responses in both classes of the data set.

Clearly, this insight might make us reconsider our previous evaluation of classifier A as the better one. If we are just after good overall accuracy and have reason to believe that the true prevalence of class 1 data is also about 95% in the real world, we can - of course - settle for the trivial model. If our main goal is to detect and identify the relatively rare occurrences of class 2, classifier B is obviously to be preferred.

This somewhat extreme example illustrates two major questions that arise in the practical approach to classification problems:

- How can we cope with strongly biased data sets when evaluating the performance of a classifier?
- Can we evaluate classifiers beyond their *overall accuracies* in order to obtain better insight into the performance?

Here we do not address the question of how to take class bias into account in the training process. Some strategies for the training from imbalanced data sets will be discussed in Section 8.7.

7.4.3 Receiver Operating Characteristics

For two-class problems, both of the above mentioned questions can be addressed in the framework of the so-called Receiver Operating Characteristics (ROC) [HTF01, Bis95a, Bis06, DHS00, Faw06]. The concept and terminology goes back to signal processing tasks originally, but has become popular in the machine learning community.

Most classifiers we have discussed obtain a binary assignment by applying a threshold operation to a so-called discriminative function g. In terms of the simple perceptron, for instance, we assign an input $\boldsymbol{\xi} \in \mathbb{R}^N$ to class $S = \pm 1$ according to

$$S = \text{sign}\left[g(\boldsymbol{\xi})\right] \;\text{ with }\; g(\boldsymbol{\xi}) = \sum_{j=1}^{N} w_j \xi_j, \tag{7.22}$$

as discussed in Chapter 3 in great detail. Having trained the perceptron as to implement the homogeneous lin. sep. function (7.22), we can introduce a

threshold Θ after training and consider the modified classification

$$S_\Theta = \text{sign}\left[g(\boldsymbol{\xi}) - \Theta\right]. \tag{7.23}$$

While this is formally identical with the consideration of an inhomogeneously lin. sep. function, see Sec. 3.3, here the perspective is different: We assume the threshold is introduced and varied manually after training. Furthermore, the concept could be applied to any discriminatory function for binary classification.

Quite generally, for a large family of classifiers it is possible to realize and control a class bias by tuning Θ in Eq. 7.23. For very large negative $\Theta \to -\infty$, all inputs will be assigned to class $S_\Theta = -1$, while large positive $\Theta \to +\infty$ result in $S_\Theta = +1$ exclusively.

In a similar way, probabilistic models can be used for *crisp* classification by thresholding the class membership probability which serves as the discriminative function g in Eq. 7.23. Employing a probability threshold $0 < \Theta < 1$ different from $1/2$ imposes a bias towards one of the two classes.

Very often it is important to distinguish the two class-specific errors that can occur: If a feature vector which is truly from class -1 is misclassified as $S = +1$, we account this as a *false positive* or *false alarm* type of error. The terminology reflects the idea that class $+1$ is to be detected, for instance in a medical test which discriminates diseased (positive test result) from healthy control patients (negative outcome). Analogously, the term *false negative error* is used when the classifier misses to detect a truly positive case.[6] Similarly, the complementary *true positive* or *true negative* rates correspond to the class-wise accuracies in the two-class problem.

The introduction of a controlled bias can be achieved in other classification frameworks as well and is, by no means, limited to linear classifiers. For instance, we can modify the Nearest Prototype Classification (NPC) in LVQ, Eq. (6.2). Identifying $\mathbf{w}^*_{(-1)}$, the closest one among all prototypes representing class -1 and the closest class-$(+1)$-prototype $\mathbf{w}^*_{(+1)}$, we can assign an arbitrary feature vector $\boldsymbol{\xi}$ to class $+1$ if

$$d\Big(\mathbf{w}^*_{(+1)}, \boldsymbol{\xi}\Big) < d\Big(\mathbf{w}^*_{(-1)}, \boldsymbol{\xi}\Big) - \Theta \tag{7.24}$$

and to class -1 else, thus introducing a margin Θ in the comparison of distances. Similarly, we could consider the output unit activation in a multilayered feed-forward neural network as the discriminative function and perform a biased thresholding along the same lines in order to obtain a crisp class assignment.

For a given value of the threshold Θ we can obtain, e.g. from a validation or test set, the observed absolute number of false positive classifications FP, false negatives FN, true positives TP and true negatives TN. The corresponding rates are defined as

$$fpr = \frac{FP}{FP+TN},\ \ tpr = \frac{TP}{FN+TP},\ \ fnr = \frac{FN}{FN+TP},\ \ tnr = \frac{TN}{FP+TN}. \tag{7.25}$$

[6] In the literature, other terms like *type I/II* errors are used frequently, but these are avoided here for the sake of clarity.

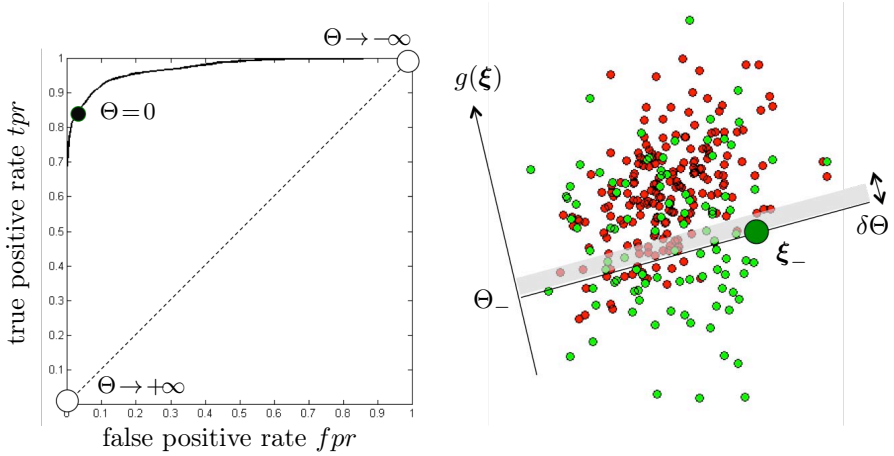


Figure 7.7: Left panel: Schematic illustration of Receiver Operating Characteristics. The extreme working points with $\Theta \to \pm\infty$ are marked by empty circles. A filled circle corresponds to an unbiased classifier with $\Theta = 0$, while the dashed line represents random, biased guesses. **Right panel:** Illustration of a two-class data set with discriminative function $g(\boldsymbol{\xi})$. Feature vectors from the negative (positive) class are displayed as green (light) and red (dark) filled circles, respectively. A randomly selected negative example $\boldsymbol{\xi}_-$ is marked by the large filled circle and corresponds to $g(\boldsymbol{\xi}_-) = \Theta_-$. The variation of the threshold by $\delta\Theta$ is referred to in the arguments employed in Sec. 7.4.4 to obtain the statistical interpretation of the AUC.

Different names are used for the same quantities in the literature, depending on the actual context and discipline. In medicine, for instance, the term *sensitivity* ($SENS$) is frequently used for the tpr, while *specificity* ($SPEC$) refers to the tnr. An overview of the many quantities that can be derived from the four basic quantities TP, TN, FP, FN and rates (7.25) can be found in [Wik22], which also provides further relevant references.
The quantities in Eq. (7.25) are not independent: Obviously, they satisfy

$$tpr + fnr = 1 \quad \text{and} \quad tnr + fpr = 1.$$

Consequently, two of the four rates can be selected to fully characterize the classification $S_\Theta(\boldsymbol{\xi})$.

In the framework of Receiver Operating Characteristics (ROC) one determines $tpr(\Theta)$ and $fpr(\Theta)$ for a meaningful range of thresholds Θ and displays the true positive rate as a function of the false positive rate by eliminating the threshold parameter[7].

Figure 7.7 (left panel) displays an example ROC curve for illustration. The lower left corner, as marked by an empty circle, would correspond to the extreme

[7] For efficient implementation ideas see [DHS00, Faw06].

setting $\Theta \to \infty$ with all inputs assigned to the negative class. Obviously, the false positive rate is *zero* for this setting, the classifier does not give any *false alarms*. On the other hand, no positive cases are detected and $tpr = 0$, as well. The upper right corner in $tpr = fpr = 1$, also marked by an open circle, corresponds to $\Theta \to -\infty$ in (7.23) or (7.24): The classifier simply assigns every feature vector to the positive class, thus maximizing the true positive rate at the expense of having $fpr = 1$. An ideal, error-free classifier would obtain $fpr = 0, tpr = 1$ in the upper left corner of the ROC graph.

The performance of an unmodified classifier with $\Theta = 0$ is marked by a filled circle in the illustration 7.7 (left panel). It could correspond, for instance, to the NPC in LVQ or the homogeneous, unbiased perceptron, Eq. (7.22). By selecting a particular threshold $-\infty < \Theta < +\infty$, the user can realize any combination of $\{tpr, tpr\}$ that is available along the ROC curve. This way, the domain expert can adjust the actual classifier according to the specific needs in the problem at hand. In medical diagnosis systems, for instance, high sensitivity (tpr) might be more important than specificity ($1 - fpr$) or vice versa.

To a certain extent, we can also compensate for the effects of unbalanced training data: In the illustrative example shown in Fig. 7.7 (left panel), the classifier with $\Theta = 0$ realizes very low fpr, which might be a consequence of an over-representation of negative cases in the data set $\mathbb{D}$. An objective function which is related to the number of mis-classification will favor classifiers with small fpr over those with higher tpr. In retrospect, this can be compensated for by biasing the classifier towards the detection of positive cases and move the working point closer to the upper left corner in the ROC.

The hypothetical, best possible ROC is obviously given by the step function including the ideal working point $fpr = 0, tpr = 1$, i.e. the complete square in Fig. 7.7. On the other hand, a completely random guess with biased probability $tpr = fpr$ for assignments to class $+1$ would correspond to the diagonal, i.e. the dashed line in the left panel of the illustration.

7.4.4 The area under the ROC curve

When evaluating different classifiers (or frameworks, rather) one often resorts to the comparison of the area under the ROC curve, the so-called $AUROC$ or less precisely AUC [Faw06]. Intuitively the AUC with $0 \leq AUC \leq 1$ provides information about the degree to which the ROC deviates from the diagonal with $AUC = 1/2$. Clearly, an $AUC > 1/2$ indicates better–than–random classification and the AUC is often used as a single numerical quality measure for the evaluation of classifiers. In principle, the precise shape of the ROC should be taken into account as well, as individual ROC can differ significantly from the idealized shape displayed in Fig. 7.7.

The AUC with respect to novel data can be estimated, for instance, in the course of cross-validation along the lines of Sec. 7.3. It provides better insight into the performance of the trained system than a single specific working point. Therefore, it serves as the basis for model selection or the setting of parameters.

Moreover, the AUC can be associated with a well-defined statistical interpretation. Fig. 7.7 (right panel) illustrates a two-class data set which can be classified according to a discriminative function which, in the illustration, is assumed to increase monotonically along the $g(\boldsymbol{\xi})$-axis. Note that here it is convenient, but not necessary, to argue in terms of a linear classifier like the perceptron, in which the weight vector $\mathbf{w}$ defines the discriminative direction.

In the illustration, a particular, e.g. randomly selected, negative example $\boldsymbol{\xi}_-$ is marked by a filled circle with the value $g(\boldsymbol{\xi}_-)$ of the discriminative function. In other words, in a classifier with $\Theta = g(\boldsymbol{\xi}_-)$ the considered example would be located precisely at the decision boundary.

Now assume that we select a random example $\boldsymbol{\xi}_+$ from the positive class, i.e. one of the feature vectors marked as red circles in the illustration. The probability for such an example to satisfy $g(\boldsymbol{\xi}_+) > \Theta_- = g(\boldsymbol{\xi}_-)$ is given precisely by $tpr(\Theta_-)$, which is the fraction of positive examples located on the *correct* side of the decision boundary defined by $g(\boldsymbol{\xi}) = \Theta_-$.

On the other hand, the local density of negative examples is given by the derivative $^{dfpr}/_{d\Theta}$ in Θ_-: Shifting the threshold by $\delta\Theta$, as marked by the gray shaded area, will result in correcting the output for $\delta fpr = (^{dfpr}/_{d\Theta})\,\delta\Theta$ many samples from the negative class.

In summary, this implies that for a pair of feature vectors comprising one randomly selected $\boldsymbol{\xi}_-$ from the negative class and one randomly selected $\boldsymbol{\xi}_+$ from the positive class, the probability that $g(\boldsymbol{\xi}_+) > g(\boldsymbol{\xi}_-)$ is given by the integral

$$\int_{-\infty}^{+\infty} tpr(\vartheta)\,\frac{dfpr}{d\vartheta}\,d\vartheta \;=\; \int_0^1 tpr\,dfpr \;=\; AUC.$$

Hence, the AUC quantifies the probability with which a randomly selected pair $\{\boldsymbol{\xi}_-, \boldsymbol{\xi}_+\}$ is ordered according to class membership in terms of the discriminative function $g(\ldots)$. This corresponds to the probability that a threshold value Θ exists, at which the classifier would separate such a pair of inputs correctly.

This intuitive interpretation of the AUC in the ROC-analysis also makes it possible to perform the training of a classifier in such a way that the expected AUC is maximized, for details see [HR04, VKHB16].

7.4.5 Alternative measures for two-class problems

A variety of evaluation criteria for binary classification schemes have been suggested in the literature. The Precision-Recall (PR) formalism [DG06] can be considered as an alternative to the ROC. The PR evaluation is also based on the four quantities TP, TN, FP, and FN. Precision ($Prec$) and Recall (Rec) are defined as

$$Prec = \frac{TP}{TP + FP} \quad Rec = \frac{TP}{TP + FN} = tpr. \tag{7.26}$$

Similar to the ROC, a PR curve displaying $Prec$ vs. Rec can be generated by varying a bias parameter. Also, the area under the PR curve is often provided as a single quality parameter. However, unlike the $AUROC$ it lacks an

appealing statistical interpretation. For a discussion of supposed disadvantages or advantages of the PR formalism over the ROC see [DG06] and references therein.

Like other quantities, $Prec$ and Rec can also be computed at a single, specific working point of the classifier. Various application domain specific measures have been defined, see e.g. [Wik22,KEP17,KHV14] for overviews and references. In fact, the large number of related quality measures can be a source of considerable confusion. Here we present only two popular examples:

While the overall accuracy for a test set of in total n_{tot} samples is computed as $(TP + TN)/n_{tot}$, the so-called balanced accuracy corresponds to an equal weight average over classes:

$$BAC = \frac{tpr + tnr}{2} = \frac{1}{2}\left(\frac{TP}{TP + FN} + \frac{TN}{TN + FP}\right). \tag{7.27}$$

It is supposedly more suitable for class imbalance in training and validation sets. Here, TP, FP and FN are the absolute numbers of true positives, false positives and false negatives observed in the data set. Similar claims have been made for the popular F_1-measure, which is given by the harmonic mean of $Prec$ and Rec:

$$F_1 = \frac{TP}{TP + (FP + FN)/2} = 2\,\frac{Prec \cdot Rec}{Prec + Rec} \tag{7.28}$$

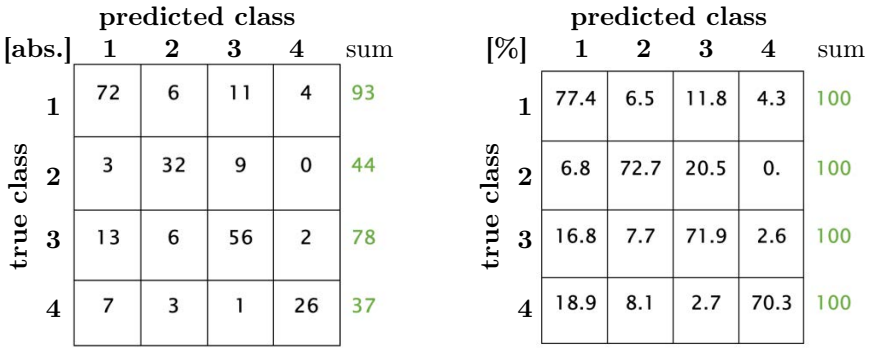
7.4.6 Multi-class problems

The evaluation of multi-class systems is more subtle. Several single valued measures can be computed, see below, but a multi-dimensional generalization of the ROC formalism or the PR scheme is far from obvious.

Confusion matrix

Most commonly, the so-called confusion matrix is provided in order to summarize the class-specific performance of a multi-class system with respect to a given data set. Illustration 7.8 displays the confusion matrix of a hypothetical 4-class classification problem. In the left panel, each element corresponds to the number of feature vectors which belong to class i and are assigned to class j by the classifier. In the right panel of Fig. 7.8, percentages (rounded) are displayed. Here, diagonal elements correspond to the class-wise accuracies. Off-diagonal elements provide insight into which classes are relatively easy or difficult to separate.

Note that although the confusion matrix provides detailed information about class-wise performances, it still corresponds to a single working point of the classifier. A multi-dimensional ROC- or PR-like analysis for multi-class problems is non-trivial [Faw06, HT01]. However, many of the above mentioned single quantities for a given working point can be extended to multi-class problems in a straightforward fashion, see [KEP17]. An obvious example is the multi-class BAC which weights every class equally by $1/C$ in a C-class problem.



[abs.]	predicted class 1	2	3	4	sum
true class 1	72	6	11	4	93
2	3	32	9	0	44
3	13	6	56	2	78
4	7	3	1	26	37

[%]	predicted class 1	2	3	4	sum
true class 1	77.4	6.5	11.8	4.3	100
2	6.8	72.7	20.5	0.	100
3	16.8	7.7	71.9	2.6	100
4	18.9	8.1	2.7	70.3	100

Figure 7.8: Confusion matrix of a hypothetical imbalanced 4-class problem. **Left panel:** matrix elements correspond to the absolute number of samples from class i which are assigned to class j. **Right panel:** the corresponding percentages. Diagonal entries represent the class-wise accuracies.

7.4.7 Averages of class-wise quality measures

A set of class-wise quantities can be derived from the confusion matrix or by considering sub-classification schemes of the type *class i vs. all others*. This way, all measures suitable for two-class problems, like $AUROC$ or F_1, can be considered in C-class problems as well.

Averages of class-wise quantities can be computed in different ways, e.g. by weighting each class equally or by taking the number of examples per class into account. In the following, this subtlety is illustrated in terms of Precision ($Prec$) and Recall (Rec) as defined in Eq. (7.26) for two-class problems and the F_1-measure (7.28).

From the confusion matrix $c(i,j)$ in terms of sample counts of a C-class problem we can determine the class-wise quantities

$$TP_i = c(i,i),\ FN_i = \sum_{j,(\neq i)} c(i,j),\ FP_i = \sum_{j(\neq i)} c(j,i) \quad \text{for} \quad i = 1,2\ldots C. \tag{7.29}$$

For the matrix displayed in Fig. 7.8 (left panel) we would obtain $TP_1 = 72$, $FN_1 = 6 + 11 + 4 = 21$, and $FP_1 = 3 + 13 + 7 = 23$ with respect to the first class.

Macro-averages

Based on Eq. (7.29) we can also compute the class-wise Precision and Recall values in analogy with Eq. (7.26):

$$Prec_i = \frac{TP_i}{TP_i + FP_i}, \quad Rec_i = \frac{TP_i}{TP_i + FN_i}. \tag{7.30}$$

The so-called *macro-averages*

$$Prec^{mac} = \frac{1}{C}\sum_{i=1}^{C} Prec_i \quad \text{and} \quad Rec^{mac} = \frac{1}{C}\sum_{i=1}^{C} Rec_i. \tag{7.31}$$

are obtained with equal weight assigned to the C classes. Alternatively, a *weighted macro-average* of the form

$$Prec^{w-mac} = \frac{1}{n_{tot}}\sum_{i=1}^{C} n_i\, Prec_i \quad \text{and} \quad Rec^{w-mac} = \frac{1}{n_{tot}}\sum_{i=1}^{C} n_i Rec_i \tag{7.32}$$

is frequently considered. Here n_i denotes the number of samples in class i and the total number is $n_{tot} = \sum_{i=1}^{C} n_i$.

Now we have at least two options for defining a macro-F_1 measure:

a) as an arithmetic mean of the class-wise F_1^i given by

$$F_1^i = 2\frac{Prec_i\, Rec_i}{Prec_i + Rec_i}, \qquad F_1^{mac-a} = \frac{1}{C}\sum_{i=1}^{C} F_1^i \tag{7.33}$$

b) or as a harmonic mean of macro-Recall and macro-Precision, i.e.

$$F_1^{mac-b} = 2\,\frac{Prec^{mac}\, Rec^{mac}}{Prec^{mac} + Rec^{mac}} \tag{7.34}$$

Unfortunately, both measures appear in the literature under the same name, often without clarifying which version was used. A recently published note [OB21] compares F_1^{mac-a} and F_1^{mac-b}. The authors show that the two quantities can differ significantly with $F_1^{mac-b} \geq F_1^{mac-a}$. They demonstrate that F_1^{mac-b} can yield overly large scores in class-imbalanced problems, while F_1^{mac-a} appears to be more robust in that respect. The authors conclude that "at the very least, researchers should indicate which formula they are using."

Micro-averages

In contrast to the above discussed macro-averages, *micro-averaging* considers all data points without taking class-specifics into account. It is obtained by considering the averages

$$\widehat{TP} = \frac{1}{C}\sum_{i=1}^{C} TP_i, \quad \widehat{TN} = \frac{1}{C}\sum_{i=1}^{C} TN_i, \quad \widehat{FP} = \frac{1}{C}\sum_{i=1}^{C} FP_i, \quad \widehat{FN} = \frac{1}{C}\sum_{i=1}^{C} FN_i \tag{7.35}$$

and computing

$$Prec^{mic} = \frac{\widehat{TP}}{\widehat{TP} + \widehat{FP}} \quad \text{and} \quad Rec^{mic} = \frac{\widehat{TP}}{\widehat{TP} + \widehat{FN}}. \tag{7.36}$$

We note that

$$(\widehat{TP} + \widehat{FP}) = \frac{1}{C}\sum_{i=1}^{C}(TP_i + FP_i) = \frac{1}{C}\sum_{i=1}^{C}(TP_i + FN_i) = \widehat{TP} + \widehat{FN} = \frac{n_{tot}}{C},$$

as both sums add up all elements of the confusion matrix. As a consequence

$$Prec^{mic} = Rec^{mic} = \frac{1}{n_{tot}}\sum_{i=1}^{C} TP_i, = F_1^{mic}. \tag{7.37}$$

Since $Prec^{mic} = Rec^{mic}$, their harmonic mean F_1^{mic} also equals the overall accuracy.

In general, macro-averages put the same emphasis on each class which makes sense if the aim is an approximately class-independent performance. The micro-average targets the overall quality, while the weighted macro-average constitutes a compromise between the micro- and macro-approach.

Which averaging procedure should be used to evaluate a multi-class system depends on the actual target and the preference of the user. The class-composition in the training data set compared to the one that is expected in the working phase plays an important role. Assume, for instance, that a particular class is represented by very few samples in the training set, but can be expected to be more frequent in the real world data. Then, micro-averaging bears the risk of disregarding the class in the evaluation with potential poor performance in the working phase as a consequence.

7.5 Interpretable systems

The above discussed quality measures and validation procedures focus on the performance of a trained system in terms of classification or regression accuracies. This is also true for the considered more sophisticated measures beyond overall or class-wise accuracies. Accuracy appears to be a natural evaluation criterion, if the goal is to, say, distinguish cats from dogs in images or perhaps discriminate diseased patients from healthy controls in a diagnosis problem.

However, machine learning systems should be evaluated and compared to each other also according to complementary criteria. Potentially, some of these cannot even be expressed in terms of simple quantitative measures. As an illustration, we discuss an entertaining and frequently quoted example [Nik17]. It illustrates and summarizes an important issue in machine learning along the lines of the opening quote of this chapter: “Accuracy is not enough.”

The story is that a classifier was trained to distinguish dogs from wolves based on a labeled set of still images. It seemed to work perfectly in training and validation, but failed completely on novel photos in the *working phase*. Eventually, a check of the database showed that all wolves had been photographed in the snow, while dogs were shown on green grass. The classifier had “learned” to distinguish the image backgrounds, not the actual animals.

As usual with stories like that, it is told in many versions, see [gwe09] for an interesting account of similar examples of supposedly mislead classifiers. Apparently, the origin of the *wolves vs. dogs* problem is [RSG16], a publication in which the authors purposefully trained a classifier on the misleading data. The aim was to illustrate the problem and to test a method for explain the inner workings of the classifier.

However, the moral of the story is definitely relevant: unnoticed biases in data sets can result in seemingly good or even excellent performances. The effect is frequently much more subtle and more difficult to detect than in the *wolves vs. dogs* problem. As a particularly important example, medical data sets are frequently prone to selection biasses that facilitate a seemingly successful discrimination of diseased from healthy subjects. For example, the age or gender distribution in the classes could be different, while being essentially unrelated with the actual target diagnosis. Even the more frequent occurrence of missing values in one of the groups could be exploited by the machine learning system, resulting in seemingly good yet useless performance. It is the nature of machine learning systems that they are excellent *artefact detectors*.

Hence, the evaluation and comparison of supervised learning models in terms of accuracy only (or similar performance oriented criteria) can be misleading and even dangerous. Responsible use of machine learning techniques requires at least a certain degree of insight into what is the basis of the system's response. Which features, for instance, appear most relevant in a classification scheme? In the example of *wolves vs. dogs*: is it the properties of the animals or the color of the background that the assignment relies upon?

In this sense, machine learning systems should be transparent and interpretable. At the very least, an effort should be made to understand how a given classifier or regression system works. Intuitive prototype-based systems and relevance learning constitute just two examples of approaches that can be useful in this context. A qualitative or quantitative study of the importance of given features can be performed in a variety of learning frameworks.

The motivation for favoring *white-box* approaches is not limited to the detection of potential biases. Interpretable models also facilitate the discussion with the domain expert and increase the user acceptance for machine learning based support systems. Consequently, the topic of improved interpretability has attracted considerable interest within the machine learning community and continuous to do so. Partly, these efforts aim at closing the gap (if any) between the goals of statistical modelling and machine learning discussed in Sec. 2.2.

For recent reviews and research articles we refer the reader to several special sessions which have been organized at the European Symposium on Neural Networks (ESANN) [VMGL12, BL13, BBVZ17]. The overview articles and session contributions should provide a useful starting point for further explorations of the topic. A comprehensive discussion of explainability and interpretability with many references can also be found in [Gho21].

Chapter 8

Preprocessing and unsupervised learning

I will let the data speak for itself when it cleans itself.

— Unknown

In most of these lecture notes we have implicitly assumed that feature vectors and labels are provided *ready-to-use* for training. In practical situations, this is rarely the case. In general, real world data sets have to be thoroughly checked for inconsistencies, missing values or other issues. A list of useful rules for initial data analysis is provided in [BCS+22].

Quite often, *outliers* are removed from the data, e.g. single data points that appear implausible because they are very different from the bulk of the data or display unrealistic values of individual features. Such a *cleaning* of the data has to be performed with utmost care and in a controlled, reproducible way. The criteria and goals of data set curation and cleaning are usually very specific to the application domain. Therefore, it should rely on insights from the domain experts and it should never be guided by the idea of making the data consistent with a given research goal or hypothesis. For instance, removing suspected outliers may seriously affect the overall quality of the data set and introduce unwanted biases.[1]

Here, we focus on broadly applicable and popular preprocessing steps. The design of a classifier or regression system usually begins with the choice of how to represent the data to the system. This can amount to the extraction of engineered features, e.g. derived from images. But also seemingly simple data sets of directly observed numerical feature vectors often require careful preprocessing.

[1] (or in the worst case even *wanted* biases that support the desired result)

Depending on the type of data at hand, a multitude of preprocessing steps can be considered. Some operations have little or no effect on the subsequent analysis. Others may seem natural but are far from trivial and can even be harmful. Here we will highlight only a few key techniques and selected popular methods:

- Frequently, some form of **normalization** or coordinate **transformation** is performed which then facilitates the application of a particular machine learning framework. This can account for mismatched scaling or skewed distributions of feature values, for instance.
- **Dimensionality reduction** plays a key role in learning problems, where the (nominal) dimension of feature vectors is relatively high compared to their *intrinsic dimension* and/or to the number of example data. The identification of specifically two- or three-dimensional representations plays an important role in the exploration and in the human-understandable **visualization** of data sets or the trained systems. **Feature selection** can be interpreted as a specific form of dimensionality reduction, aiming at a the identification of a subset of available features that is suitable and sufficient for the given task.
- Unsupervised **density estimation**, **Vector Quantization** and **clustering** can provide important insights into the structure of the data. For example, the detection of pronounced clusters of data in a preprocessing step can help to design a specific classifier or regression system.
- Often, the **imputation** of missing values in incomplete data sets is necessary, in particular when data is scarce and incomplete feature vectors cannot be simply discarded.
- **Under-** and **oversampling** techniques are applied when data sets are imbalanced with respect to the classes in a supervised learning problem. Similarly, **data augmentation** aims at enriching the training data in order to achieve better robustness against noise or to impose certain invariances in the training process, e.g. by generating rotated or shifted training images in object recognition.

The above goals and methods are obviously interrelated. For example, dimensionality reduction by Principal Component Analysis obviously constitutes also a coordinate transformation. Similarly, density estimation can be used for the imputation of missing values.

8.1 Normalization and transformations

Probably the most frequently used preprocessing steps concern some form of normalization or transformation of the feature vectors prior to the actual machine learning analysis. Clearly, such operations should never be applied blindly without evaluating their effect on the subsequent analysis.

8.1.1 Coordinate-wise transformations

Frequently, coordinate- or feature-wise transformations are applied in order to harmonize the range or the statistical properties of the features.

Centering and z-score transformation

Given a set of observations or measurements resulting in N-dim. features vectors $I\!P = \left\{\widetilde{\boldsymbol{\xi}}^{\mu} \in \mathbb{R}^N\right\}_{\mu=1}^{P}$, it can be convenient and/or useful to subtract the mean observed in the data set. As a result, one obtains centered feature vectors

$$\boldsymbol{\xi}^{\mu} = \widetilde{\boldsymbol{\xi}}^{\mu} - \widetilde{\mathbf{m}} \;\text{ where }\; \widetilde{\mathbf{m}} = \frac{1}{P}\sum_{\mu=1}^{P}\widetilde{\boldsymbol{\xi}}^{\mu} \quad \text{with} \;\; \frac{1}{P}\sum_{\mu=1}^{P}\boldsymbol{\xi}^{\mu} = 0. \tag{8.1}$$

Here, we subtract the empirical mean $\widetilde{\mathbf{m}}$ as observed in the data set. In (rare) cases where the *true mean* of the corresponding probability density is known, it could be used to replace the empirical one.

In the N-dimensional Euclidean space, the centering (8.1) corresponds to a simple translation of all data points, which has no effect on, for instance, pair-wise Euclidean distances or angles defined by triplets of data points.

A slightly more involved transformation is frequently applied in order to obtain *zero mean, unit variance* features:

$$z_j^{\mu} \;=\; \frac{\widetilde{\xi}_j^{\mu} - \widetilde{m}_j}{\widetilde{\sigma}_j} \quad \text{with} \;\; \widetilde{\sigma}_j^2 = \frac{1}{P}\sum_{\mu=1}^{P}\left(\widetilde{\xi}_j^{\mu} - \widetilde{m}_j\right)^2 \tag{8.2}$$

Here, $\widetilde{m}_j$ is the empirical mean of component $\widetilde{\xi}_j^{\mu}$ in $I\!P$ as defined in Eq. (8.1) and $\widetilde{\sigma}_j$ is the corresponding standard deviation. As a result, the so-called *standard scores* or z-scores z_j^{μ} display zero mean and unit variance in the given data set. Hence, z_j^{μ} quantifies by how many standard deviations a feature value differs from the empirical mean in the data set. Positive (negative) z_j^{μ} correspond to above (below) average values, respectively.

The z-score transformation or *standardization* is used to account for features that scale differently, which can be detrimental in a supervised or unsupervised analysis of the data. The transformation renders the representation independent of linear rescaling of individual features and choice of units of measure (as in *miles* or *centimeters* for the length of an object). On a univariate level, considering only single features, the effect of the monotonic transformation is not critical. However, one should be aware that it can alter the relations between features in a non-trivial way. As one example, a z-score transformation can affect the outcome of a Vector Quantization scheme, see the discussion of Fig. 8.6 in Sec. 8.4.3.

A variety of similar linear transformations can be motivated for specific problems and data sets. For instance, we could employ the feature *median* and *interquartile range* (IQR) for a scaling analogous to (8.2) in order to achieve zero median and unit IQR data.

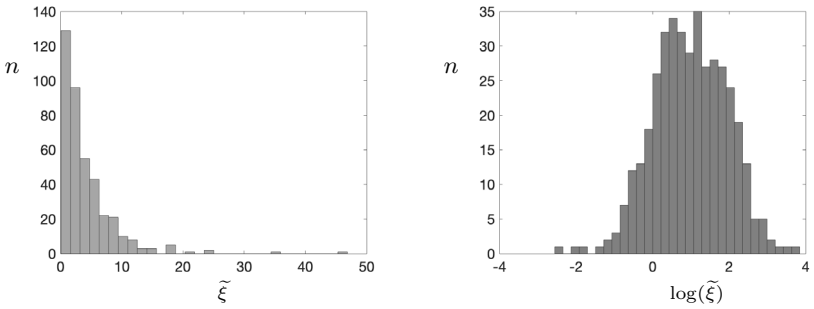


Figure 8.1: Left panel: skewed histogram of a of a feature $\widetilde{\xi}$ that displays many small and relatively few large values in a given data set (illustration). **Right panel:** histogram of the log-transformed feature $\log(\widetilde{\xi})$.

Min-Max feature scaling

Sometimes it is desirable to transform features to a fixed range of values, e.g. reflecting the specific requirements of the machine learning system in use. With $min_j = \min\left\{\widetilde{\xi}_j^\mu\right\}_{\mu=1}^P$ and $max_j = \max\left\{\widetilde{\xi}_j^\mu\right\}_{\mu=1}^P$ we can achieve that

$$\xi_j = \frac{\widetilde{\xi}_j - min_j}{max_j - min_j} \in [0,1]. \tag{8.3}$$

Note that the use of minimum and maximum values can be very sensitive to the presence or absence of extreme values in a data set.

This or similar transformations are occasionally also referred to as *normalization* in the literature. In order to avoid confusion, we will use the term only in the context of the actual L_p-normalization of vectors as described in the following subsection.

Non-linear feature transformations

The effect of linear feature-wise transformations can be highly non-trivial, for instance in the context of classification or unsupervised clustering, cf. Sec. 8.4. Obviously, the impact of non-linear transformations can be even greater. A large variety of such transformations can be considered, having specific goals in mind and taking into account domain knowledge about the properties of the data set at hand.

As just one particular example we discuss briefly the popular log-transform. Often, features of the considered data display a skewed distribution of values. The illustration in Figure 8.1 (left panel) shows the histogram of a non-negative feature which frequently assumes small values and only rarely larger ones. In such cases, a feature-wise logarithmic transformation of the form

$$\xi_j = \log(\widetilde{\xi}_j) \quad \text{or} \quad \xi_j = \log(\widetilde{\xi}_j + \epsilon) \quad \text{with} \quad \epsilon > 0 \tag{8.4}$$

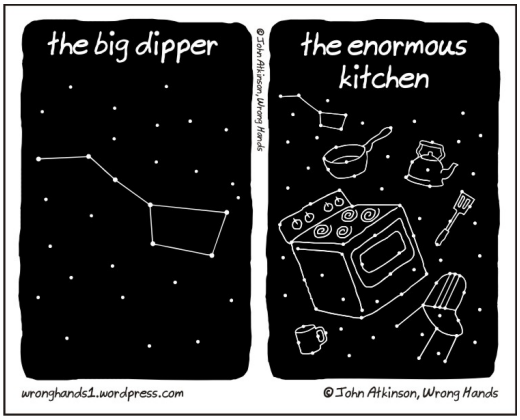


Figure 8.2: Astrology as an artefact of implicit normalization by projecting stars onto the surface of a sphere. The right panel can also be viewed as an example of (mental) overfitting.
Reproduced with kind permission from John Atkinson for non-commercial use, see https://wronghands1.com/ for more of his brilliant cartoons.

reshapes the histogram to appear – loosely speaking – *more Gaussian.*[2]

The transformed data is exactly Gaussian only if the original data follows a *log-normal* distribution of the form

$$P(\widetilde{\xi}_j) \propto \exp\left[-\frac{(\log\widetilde{\xi}_j - \widetilde{\mu})^2}{2\widetilde{\sigma}^2}\right]. \tag{8.5}$$

While this is rarely exactly the case in real world data, skewed distributions similar to the histogram in Fig. 8.1 (left) are not uncommon in practice, e.g. in the context of medical data relating to bio-markers, see [Bie17] for an example.

In any case, which transformations make sense in a particular problem depends on available domain knowledge and insights into the data structure.

8.1.2 Normalization

One of the most frequently applied type of transformation is the normalization of observed N-dimensional feature vectors. Based on so-called L_p-norms

$$||\widetilde{\boldsymbol{\xi}}||_p = \left[\sum_{j=1}^{N} |\,\widetilde{\xi}_j\,|^p\right]^{1/p} \quad \text{we can consider vectors} \quad \boldsymbol{\xi} = \widetilde{\boldsymbol{\xi}}\Big/||\widetilde{\boldsymbol{\xi}}||_p, \tag{8.6}$$

which then all display L_p-norm *one*.

For $p = 2$, corresponding to the familiar Euclidean norm, we obtain vectors of the same *length* in feature space. In other words, all data points are projected onto the unit sphere in $\mathbb{R}^N$. Using Euclidean distance after L_2- normalization is equivalent to measuring distances in terms of angles between the original feature vectors. The L_2-normalization appears to be a natural thing to do, for instance in the context of the Perceptron, see Chapter 3.

In more general contexts, however, the effects of normalization can be non-trivial and even lead to artefacts and mis-interpretations of the data. Astrology

[2]The choice $\epsilon = 1$ is often used for non-negative features as it maps $\widetilde{\xi}_j = 0$ to $\xi_j = 0$.

is a superb example: unrelated stars appear to form meaningful clusters when they are implicitly projected onto a sphere, see Fig. 8.2 for a tongue-in-cheek illustration.

As another example, L_1-normalization using the so-called *Manhattan* norm yields transformed feature vectors with $\sum_j |\xi_j| = 1$. For non-negative features e.g. representing amounts of chemical components in a sample, the normalized data can be interpreted as *concentrations*. Similarly, *event counts* would be transformed to normalized *frequencies* or probabilities.

Normalization as a preprocessing step can be helpful and greatly beneficial in practical problems. In any case, it should be applied based on available domain knowledge, and its effects on the subsequent analysis should be carefully evaluated.

8.2 Dimensionality reduction

The low-dimensional representation of high-dimensional data plays an important role in the context of data analysis and machine learning, see e.g. [Bis06, HTF01, LV07, MPH09, BBH12] for a variety of methods and concepts. Several interconnected, overlapping motivations for dimension reduction can be identified:

- **Preprocessing**
  If high-dimensional data are presented to a machine learning system, the number of adaptive quantities will be (at least) of the same order. This may hinder successful training, in particular if the number of available examples is limited. Dimension reduction can help to overcome this difficulty.

- **Exploration**
  Low-dimensional representations help to explore a given data set prior to, say, supervised learning. Linear or non-linear projections can reveal structures in the data, e.g. clusters or subspaces in which the feature vectors are located. Such insights can help to design appropriate systems in the subsequent analysis by taking specific properties of the data into account.

- **Visualization**
  Closely related to the previous point, two- or three-dimensional representations provide visualizations of the data set which facilitate interaction with the user or domain expert. Before further analysis, visualization can give useful insight into the structure of the problem. In retrospect, e.g. after the training of a classifier, visualization helps to evaluate its performance and provides information about regions where classes overlap or individual misclassified data points are located [SHH20].

It is important to realize that the *intrinsic dimensionality* of feature vectors $\boldsymbol{\xi} \in \mathbb{R}^N$ does not necessarily coincide with the nominal dimension N. For

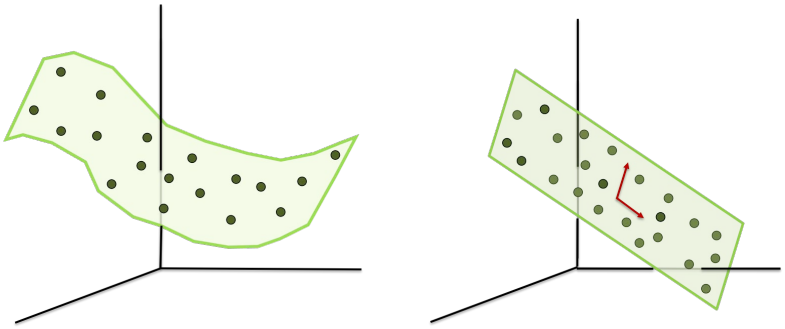


Figure 8.3: Left panel: schematic illustration of three-dimensional data points falling into a two-dimensional manifold. **Right panel:** the special case of a linear subspace or hyperplane that contains the data points.

instance, a set of vectors $\{\boldsymbol{\xi}^\mu\}_{\mu=1}^P$ could fall into (or close to) a low-dimensional manifold $\mathcal{M} \subset \mathbb{R}^N$ as illustrated in Fig. 8.3. A particular simple example is a linear subspace, i.e. a hyperplane which contains the linear dependent feature vectors, an illustration is shown in the right panel of Fig. 8.3.

Note that the popular term *curse of dimensionality* [Bel57] is avoided here, because (a) it is not obvious that a nominally high dimension is necessarily detrimental for the performance of machine learning methods (see e.g. [HKK$^+$10, GT18]) and (b) superstition brings bad luck.

We can distinguish two essentially different concepts for the low-dimensional representation of high-dim. data: in one family of approaches, each original data point is represented by an individual counter-part in a low-dim. latent space without requiring a parameterized mapping between the spaces. The positions of the representatives are directly determined by means of optimizing a suitable cost function which is based on the aim of (approximately) preserving neighborhood relations, pair-wise distances, or the overall topology of the original data points. These approaches do not require an explicit linear or non-linear functional mapping from the high- to the low-dimensional space. As an important example, we present *Multi-dimensional Scaling* (MDS) below and briefly mention a few other popular methods.

In the second major framework, an explicit mapping is determined which provides linear or non-linear *projections* of the original data into the latent space. The projection is optimized according to a specific criterion which is evaluated w.r.t. a given data set. While these methods are less flexible due to the pre-defined form of the actual mapping, they offer the possibility to project *out-of-sample* data after training. The most popular example of the basic concept is the well-known Principal Component Analysis (PCA) which is discussed in Sec. 8.3 also from a neural network perspective.

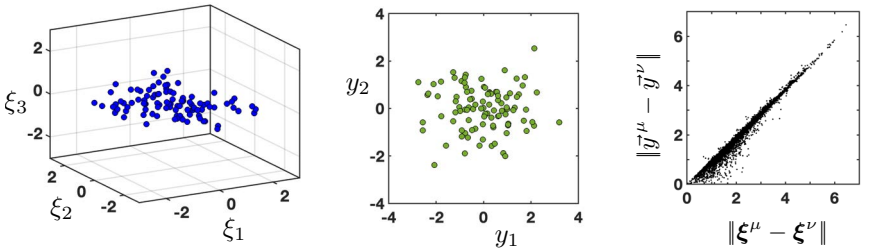


Figure 8.4: Left panel: three-dimensional data points $\boldsymbol{\xi}^\mu \in \mathbb{R}^3$ which display a relatively low variance in component ξ_3 . **Center:** the two-dim. representations $\vec{y}^{\,\mu} \in \mathbb{R}^2$ as obtained by metric MDS. **Right panel:** scatter plot of pair-wise Euclidean distances in $N = 3$ dimensions vs. distances in $M = 2$ after MDS.

8.2.1 Low-dimensional embedding

Consider a given set of N-dim. feature vectors $I\!P = \left\{\boldsymbol{\xi}^\mu \in \mathbb{R}^N\right\}_{\mu=1}^{P}$ with a distance measure $d_N(\boldsymbol{\xi}, \boldsymbol{\xi'})$ that quantifies the pair-wise dissimilarity of data points in $I\!P$. As a straightforward example we can think of the Euclidean distance in N dimensions, but the consideration of any meaningful dissimilarity is possible. We refer to the pair-wise distances as $d_N^{\mu\nu} = d_N(\boldsymbol{\xi}^\mu, \boldsymbol{\xi}^\nu)$ in the following.

A number of methods aim at finding an embedding of the data points in a low-dimensional Euclidean vector space that preserves relations between the individual feature vectors in the original space, as much as possible. The goal could be to approximately reproduce the pair-wise distances themselves, their rank structure, or associated probability densities.

8.2.2 Multi-dimensional Scaling

The goal of metric Multi-dimensional Scaling (MDS) [3] is to find representatives $\vec{y}^{\,\mu} \in \mathbb{R}^M$ with $M < N$, which preserve pair-wise distances and their relations as far as possible. To this end we consider coordinates $y_j^\mu \in \mathbb{R}$ ($j = 1, \ldots M$ and $\mu = 1, \ldots P$) in the target space. There, we evaluate the dissimilarities of the corresponding vectors $\{\vec{y}^{\,\mu}\}_{\mu=1}^{P}$ by means of a metric $d_M(\vec{y}, \vec{y}')$. Again, this measure could be based on any reasonable vector norm, with Euclidean metric being the classical and by far most popular choice. Note that, in general, the measures d_N and d_M need not be of the same type. An example cost function

[3] So-called *classical* MDS is also known as *Principal Coordinates Analysis* and is not discussed here [Mea92].

to optimize in MDS is the quadratic deviation

$$E\left(\{\vec{y}^{\,\mu}\}_{\mu=1}^{P}\right) = \sum_{\substack{\mu,\nu=1 \\ \mu<\nu}}^{P} \left[d_N^{\mu\nu} - d_M(\vec{y}^{\,\mu}, \vec{y}^{\,\nu})\right]^2 . \tag{8.7}$$

All coordinates y_j^μ are considered degrees of freedom that can be obtained by minimization of E.

A (local) minimum of (8.7) corresponds to an arrangement of P points in M dimensions which reflects the pair-wise distances $d_N^{\mu\nu}$ as well as possible. For $M < N$ it is in general not possible to obtain a perfect solution with $E = 0$. Obviously, we can also not expect to find a unique minimum of E, the actual outcome of MDS will depend on the initial configuration of $\vec{y}^{\,\mu}$ and on the potential randomness in the training process. Figure 8.4 illustrates MDS in terms of a simple example based on Euclidean distance in both spaces: three-dimensional feature vectors $\boldsymbol{\xi}^\mu$ (left panel) display a relatively small variance in component ξ_3. The corresponding two-dimensional representations $\vec{y}^{\,\mu}$, obtained by minimizing the quadratic deviation (8.7), is displayed in the center panel. The scatter plot in the right panel shows that very similar pair-wise distances have been achieved in $M = 2$ dimensions. Quantitative measures, e.g. a correlation coefficient, could be obtained in order to evaluate the quality of the MDS result. Note that the quality of the embedding is invariant under, e.g., simultaneous translations or rotations of the $\vec{y}^{\,\mu}$.

The term MDS is frequently meant to imply the use of Euclidean distances in $\mathbb{R}^N$ and $\mathbb{R}^M$, and minimizing the quadratic deviation (8.7). Various modifications of the basic idea have been suggested and applied in practice. Obviously, a variety of distance measures d_N and d_M could be considered. Moreover, specific cost functions can be chosen which put, for instance, different emphasis on small or large distances. A good overview of MDS related methods can be obtained from [LV07] and [MPH09]. In particular [BBH12] discusses several choices in a unified framework, including the so-called *Isomap* [TdSL00], *Sammon mapping* [Sam69], *Local Linear Embedding* (LLE) [RS00], and *Laplacian Eigenmaps* [BN03].

8.2.3 Neighborhood Embedding

Another popular family of methods is often used for the visual inspection of complex data sets. In the original *Stochastic Neighborhood Embedding* (SNE) as introduced by Hinton and Roweis [HR03], the data is characterized in terms of Gaussian probabilities in the original and in the embedding space. These are replaced by long-tailed student-t distributions in the popular *t-distributed SNE* (t-SNE) that was later suggested by van der Maaten and Hinton [MH08].

Both, SNE and t-SNE, aim at the minimization of the Kullback-Leibler divergence as a measure of (dis-)similarity between the assumed densities in the original and the embedding space. The optimization can be done using gradient-based methods. Recently, an alternative known as *Uniform manifold*

approximation and projection (UMAP) has become popular [MHM20]. Its key feature is the assumption that data is distributed in (or close to) a particular manifold along which distances are computed.

Embedding methods are very popular, mainly in the context of data exploration and visualization. For more detailed presentations of neighborhood embedding, we refer the reader to the original literature and reviews like [LV07, BBH12,BBK20]. One of the main drawbacks of these methods is that, generally speaking, the embedding coordinates $\vec{y}$ are not directly interpretable and their connection to the original feature space is often unclear. In this sense, embedding methods are not very appealing as preprocessing steps for further supervised learning (regression or classification). Moreover, out-of-sample extension, i.e. the post-hoc embedding of data points that were not in $I\!P$ is non-trivial. One option is to add them to the set and then recompute all $\vec{y}^{\mu}$ on the basis of the extended data, which is obviously very costly. Alternatively, one can try to obtain an explicit mapping function Ψ that approximately realizes $\Psi : \boldsymbol{\xi}^{\mu} \to \vec{y}^{\mu}$ and can be applied to novel data points, see for instance [EHT20] for a Deep Learning based approach.

8.2.4 Feature selection

A rather direct way of reducing the dimensionality of input data is to select a subset of features, neglecting all others. It appears to be a natural idea to simply try all different combinations of features and select the best possible subset. In practice, however, the number of subsets with k features selected from N dimensions is $\binom{N}{k}$ and grows very rapidly with k and N.

An overview of basic feature selection methods can be found in [GE03,CS14, JBB15]. Three main strategies for the selection of feature subsets have been considered in the literature:

- **Filter methods:**
  So-called filter methods aim at the identification of useful features without actually training a classification or regression model. In unsupervised approaches, features are selected or rejected based on their statistical properties, independent of the actual target problem. For instance, correlations between different features could be exploited in order to discard redundant features in a given data set.

  Supervised filtering takes into account the label information in the data. For instance, in the context of regression, individual features can be evaluated and selected in terms of their correlations with the target. Similarly, in classification problems, mutual information or cross entropy between features and the target can serve as a selection criterion. Most frequently, these criteria are applied in a univariate fashion, feature by feature. This bears the risk of missing the relevance of combinations of features which would be revealed in an appropriate multivariate analysis.

- **Wrapper methods:**
  In the wrapper approach, the idea is to consider different combinations of

features in terms of the performance of trained regressors or classifiers. For each candidate set of features, training and potentially validation schemes have to be performed, which can result in considerable computational costs. An advantage of the wrapper approach over univariate filter methods is that, in principle, favorable combinations of features can be found.

Obviously, there is no straightforward way to find the *optimal* set of features for a given task. One can start with the full set of available features and remove single ones from the set, in every step selecting the one that has the least impact on the performance after training. Alternatively, one could add features to the set, following a greedy strategy to improve the quality of the classification or regression in every step.

- **Embedded methods:**
  In embedded methods the selection of a subset of features is an integral part of training a specific type of model, i.e. classifier or regression system. A popular example would be the training of decision trees in a Random Forest [Bre01], in which a tree selects a particular feature in every branching step. Hence, the actual feature set is compiled while the system is trained. While this can be more efficient than the wrapper approach, it is - in a sense - limited to an essentially univariate evaluation of features.

8.3 PCA and related projection methods

A variety of methods derives an explicit mapping of the form

$$\Psi : \mathbb{R}^N \to \mathbb{R}^M \quad \text{with} \quad \vec{y} = \Psi(\boldsymbol{\xi}) \in \mathbb{R}^M \tag{8.8}$$

directly from the data set $I\!P$. The mapping Ψ is parameterized and obtained in a data driven process. Hence, unlike MDS or other embedding methods, the mapping can be applied to out-of-sample data $\boldsymbol{\xi} \notin \mathbb{P}$ immediately. This is particularly important for methods of supervised learning, where we can derive Ψ from the training data and apply the same mapping to novel data in the working phase.

In principle, we can obtain a meaningful projection by parameterizing the function Ψ in a suitable way and then optimizing its parameters according to an appropriate cost function. All criteria discussed in the context of embedding in the previous section could be employed for the identification of an explicit mapping as well.

We discuss here only methods which employ linear projections of the data. Most methods, including Principal and Independent Component Analysis can be extended in various ways. Non-linear versions can be formulated, for instance, in terms of shallow or deep neural autoencoders as discussed in Sec. 5.5.1. Kernelized versions also exist, see for instance [SSM98, LG19] and references therein.

8.3.1 Principal Component Analysis

Due to their simplicity and intuitive nature, linear mappings are of particular interest and practical relevance. For example, Principal Component Analysis (PCA) is one of the most important and frequently used explicit mappings in the context of data analysis, unsupervised learning, low-dimensional representation and visualization. Primarily, PCA is employed to determine low-dimensional representations. In addition it can be used to identify supposedly non-informative contributions in a given data set. The basic idea is to disregard linear combinations of features that hardly vary over the observed data.

Like many other standard and well-established methods, PCA can be motivated and derived from a variety of perspectives. Here we follow a mostly heuristic line of thoughts and point out the relation to the theoretical backgrounds in passing.

For convenience we will assume throughout the following that the data set used for the identification of the mapping Ψ is centered, i.e. $\frac{1}{P}\sum_{\mu=1}^{P} \boldsymbol{\xi}^{\mu} = 0$. If this has been achieved by a centering of the form (8.1), the same transformation using the empirical mean in $I\!P$ has to be performed on novel input vectors, before Ψ can be applied.

The direction $\mathbf{w} = \mathbf{u}_1 \in \mathbb{R}^N$ along which the centered data displays the largest variance is of particular interest. We can formulate the search as an optimization problem with respect to the empirical variance along $\mathbf{w}$:

$$E_{var} = \frac{1}{P}\sum_{\mu=1}^{P} \left(y^{\mu}\right)^2 \quad \text{with the projections} \quad y^{\mu} = \mathbf{w}\cdot\boldsymbol{\xi}^{\mu}. \tag{8.9}$$

It is plausible to assume that the corresponding solution of largest variance, is the direction in which most of the information about $\boldsymbol{\xi}$ is contained. In fact, this is rigorously true under the assumption that all features follow a normal distribution. If we define the empirical covariance matrix [4]

$$\mathcal{C} \in \mathbb{R}^{N\times N} \quad \text{with elements} \quad \mathcal{C}_{ij} = \frac{1}{P}\sum_{\mu=1}^{P} \xi_i^{\mu}\,\xi_j^{\mu}, \tag{8.10}$$

we can rewrite the objective function, Eq. (8.9), as a quadratic form

$$\left[E_{var} = \sum_{j,k=1}^{N} w_j\, w_k\, \frac{1}{P}\sum_{\mu=1}^{P} \xi_j^{\mu}\xi_k^{\mu}\right] = \mathbf{w}^{\top}\,\mathcal{C}\,\mathbf{w}. \tag{8.11}$$

The matrix $\mathcal{C}$ is positive semi-definite by definition, the quadratic form is non-negative but otherwise unbounded. We can assume that $\mathcal{C}$ has ordered eigenvalues

$$\lambda_1 \geq \lambda_2 \geq \ldots \geq \lambda_N \geq 0. \tag{8.12}$$

[4] not to be confused with the complementary $P\times P$ matrix C with elements $C_{\mu\nu} \propto \sum_j \xi_j^{\mu}\xi_j^{\nu}$ defined in (3.74) in the context of classification.

Restricting the search to normalized $\mathbf{w}$ with $|\mathbf{w}| = 1$, it is straightforward to show that the maximum of E_{var} is achieved if $\mathbf{w}$ is the eigenvalue of $\mathcal{C}$ with the largest eigenvalue λ_1:

$$\mathbf{w} = \mathbf{u}_1 \quad \text{with} \quad \mathcal{C}\mathbf{u}_1 = \lambda_1 \mathbf{u}_1.$$

In case of degenerate leading eigenvalues, $\lambda_1 = \lambda_2 = \ldots = \lambda_m$, we have to consider the corresponding m-dimensional eigenspace. Note that the variance associated with the normalized $\mathbf{u}_1$ is given directly by its eigenvalue:

$$\frac{1}{P}\sum_{\mu=1}^{P}(\mathbf{u}_1 \cdot \boldsymbol{\xi}^\mu)^2 = \mathbf{u}_1^\top \mathcal{C}\mathbf{u}_1 = \lambda_1(\mathbf{u}_1)^2 = \lambda_1. \tag{8.13}$$

Once we have determined $\mathbf{u}_1$ as the direction of largest variation, we proceed by identifying the direction $\mathbf{u}_2$ which displays the largest variance in the space orthogonal to $\mathbf{u}_1$. In general we can determine the ordered sequence of directions $\mathbf{u}_m$ in which the data displays the largest variance with $\mathbf{u}_m \cdot \mathbf{u}_k$ for all $k = 1, 2 \ldots m-1$. Hence, we identify the M leading Principal Components of $I\!P$ with the leading eigenvectors of $\mathcal{C}$

$$\{\mathbf{w}_k\}_{k=1}^{M} \quad \text{with projections} \quad y_k^\mu = \mathbf{w}_k \cdot \boldsymbol{\xi}^\mu \quad \text{and variance} \quad \frac{1}{P}\sum_{\mu=1}^{P}(y_k^\mu)^2 = \lambda_k. \tag{8.14}$$

In the simple illustration shown in Fig. 8.3 (right panel), the vectors marked in red correspond to the two leading eigenvectors or principal components. They can serve as coordinate axes defining the two-dimensional subspace of largest variation in the data set.

The M-dim. vectors $\vec{y}^\mu = (y_1^\mu, y_2^\mu, \ldots y_M^\mu)^\top$ serve as M-dimensional representations of the data set. The associated linear subspace displays the largest variances in the data set. Under appropriate assumptions of Gaussianity, this implies that the vectors $\vec{y}^\mu$ have the maximum possible information content about the high-dimensional $\boldsymbol{\xi}^\mu$.

Equivalently, one can show that, for fixed dimensionality M, PCA can be interpreted as a linear autoencoder, cf. Sec. 5.5.1, realizing a dimensionality reducing a linear mapping and back-transformation:

$$\vec{y}_k^\mu = \mathbf{u}_k \cdot \boldsymbol{\xi}^\mu, \qquad \boldsymbol{\xi}_{est}^\mu = \sum_{k=1}^{M} y_k^\mu \, \mathbf{u}_k, \tag{8.15}$$

realizes the smallest quadratic reconstruction error (5.37). Introducing the matrix $U = [\mathbf{u}_1, \mathbf{u}_2, \ldots \mathbf{u}_M]^\top \in \mathbb{R}^{M \times N}$ we can write (8.15) in the compact form

$$\vec{y}^\mu = U\,\boldsymbol{\xi}^\mu, \quad \boldsymbol{\xi}_{est}^\mu = U^\top \vec{y}^\mu. \tag{8.16}$$

Generically, we will apply PCA to high-dimensional data sets, in particular if the number of examples is lower than the nominal dimension, i.e. $P < N$.

Obviously the subspace of non-zero variability in the data set is at most P-dimensional in this case, as reflected by the rank of $\mathcal{C}$. Hence, the maximum number of meaningful principal components is $M = P$. Typically, fewer components are used: under the assumption that the leading eigenvectors carry most information, the mapping is truncated at relatively small M, neglecting variations along minor eigendirections as *noise*.

Powerful tools exist that can be used to determine the leading eigenvectors of the symmetric, semi-definite matrix $\mathcal{C}$. Often, numerical methods for the more general *singular value decomposition* (SVD) are preferred [Str19, DFO20]. SVD reduces to eigenvalue decomposition for diagonalizable matrices, but it is claimed to be numerically more stable.

Whitening

PCA can also be used for a so-called *Whitening* transformation. The transformed coordinates

$$\widehat{y}_k^\mu \;=\; \frac{\mathbf{u}_k \cdot \boldsymbol{\xi}^\mu}{\sqrt{\lambda_k}} \tag{8.17}$$

display zero mean and unit variance for all k. Hence, on the level of *second order statistics*, the data appears totally isotropic with an identity covariance matrix. Note that this does not imply that there is no structure left in the data. Whitening can be useful when analysing properties of the data which concern higher order statistics as in Independent Component Analysis, see Sec. 8.3.3.

8.3.2 PCA by Hebbian learning

It is instructive to study a very simple numerical procedure to compute $\mathbf{u}_1$, which corresponds to the *power method* or *von Mises iteration* [MP29, Wil65]. As we will see, it relates to Hebbian learning and can be extended to the computation of several leading eigenvectors from a neural network perspective.

Consider an initial vector $\mathbf{w}(0)$ which can be expanded in terms of the eigenvectors $\mathbf{u}_i$ of $\mathcal{C}$:

$$\mathbf{w}(0) = \sum_{j=1}^{N} a_j \, \mathbf{u}_j \quad \text{with coefficients } a_j \in \mathbb{R} \text{ and } a_1 \neq 0.$$

Repeated multiplication (from the left) with $\mathcal{C}$ leads to the t-th iterate

$$\begin{aligned} \mathbf{w}(t) \;&=\; \mathcal{C}^t \mathbf{w}(0) = \sum_{j=1}^{N} a_j \, \lambda_j^t \, \mathbf{u}_j = \lambda_1^t \left[a_1 \mathbf{u}_1 + \sum_{j=2}^{N} \left(\frac{\lambda_j}{\lambda_1} \right)^t a_j \, \mathbf{u}_j \right] \\ &\approx\; \lambda_1^t a_1 \mathbf{u}_1 \quad \text{for } t \to \infty, \end{aligned} \tag{8.18}$$

where we omit contributions that vanish like $\mathcal{O}\left(\left| \frac{\lambda_2}{\lambda_1} \right|^t \right)$.

Hence, assuming that $0 < \lambda_2 < \lambda_1$, the weights $\mathbf{w}(t)$ will be dominated by the leading eigenvector $\mathbf{u}_1$ for $t \to \infty$. In case of degeneracies, the argument can be extended to the m-dimensional subspace of leading eigenvalues and the role of λ_2 is taken over by λ_{m+1} accordingly.

We can obtain a modified iteration by replacing $\mathcal{C}$ by the shifted matrix $[I_N + \eta\mathcal{C}]$ with $\eta > 0$ where I_N is the N-dim. identity. This matrix has the same eigenvectors as C and ordered eigenvalues $1 + \eta\lambda_j$. The corresponding iteration reads

$$\mathbf{w}(t) = [I_N + \eta\mathcal{C}]\,\mathbf{w}(t-1) = \mathbf{w}(t-1) + \eta\,\mathcal{C}\mathbf{w}(t-1) = \mathbf{w}(t-1) + \eta\,\frac{1}{P}\sum_{\mu=1}^{P} y^\mu(t)\,\boldsymbol{\xi}^\mu \tag{8.19}$$

where $y^\mu(t) = \mathbf{w}(t-1)\cdot\boldsymbol{\xi}^\mu$ and we exploit that for general $\mathbf{w} \in \mathbf{R}^N$

$$[\mathcal{C}\mathbf{w}]_i = \sum_{j=1}^{N} C_{ij} w_j = \frac{1}{P}\sum_{\mu=1}^{P} \xi_i^\mu \sum_{j=1}^{N} \xi_j^\mu w_j = \frac{1}{P}\sum_{\mu=1}^{P} \xi_i^\mu y^\mu.$$

Instead of computing the sum over $\mu = 1, \ldots P$ in each iteration step (8.19) we can update the vector $\mathbf{w}$ in single steps of the form

$$\mathbf{w}(\tau) = \mathbf{w}(\tau-1) + \eta\, y^\mu(\tau)\,\boldsymbol{\xi}^\mu \tag{8.20}$$

where $\mathbf{w}(0)$ is the initial vector and the index μ on the r.h.s follows the sequence $\mu = 1, 2 \ldots P, 1, 2, \ldots$ which corresponds to the repeated presentation of all data in a fixed sequential order [5].

Note that the update (8.20) can be interpreted as *Hebbian Learning* in a linear neuron with inputs ξ_j and output $y = \mathbf{w}\cdot\boldsymbol{\xi}$. The update can also be derived as the gradient based maximization of cost function E_{var} in Eq. (8.9).

In practice, it makes sense to normalize the weight vector after each update step in order to avoid numerical problems of diverging or vanishing $|\mathbf{w}|$:

$$\mathbf{w}(\tau) = \frac{\mathbf{w}(\tau-1) + \eta\, y^\mu(\tau)\,\boldsymbol{\xi}^\tau}{|\mathbf{w}(\tau-1) + \eta\, y^\mu(\tau)\,\boldsymbol{\xi}^\tau|} \tag{8.21}$$

with $y^\mu(\tau) = \mathbf{w}(\tau-1)\cdot\boldsymbol{\xi}^\mu$. Assuming that the previous $\mathbf{w}(\tau-1)$ was already normalized, we note that

$$|\mathbf{w}(\tau-1) + \eta\, y^\mu(\tau)\,\boldsymbol{\xi}^\tau| = \sqrt{1 + 2\eta[y^\mu(\tau)]^2 + \mathcal{O}(\eta^2)}.$$

The last term in the square root can be neglected for small learning rates $\eta \to 0$. In the same limit $(1+\eta z)^{-1/2} \approx (1 + \eta z/2)$ and we therefore obtain

OJA'S RULE

$$\mathbf{w}(\tau) = \mathbf{w}(\tau-1) + \eta\left(y^\mu(\tau)\,\boldsymbol{\xi}^\tau - [y^\mu(\tau)]^2\mathbf{w}(\tau-1)\right). \tag{8.22}$$

[5] The change from Eq. (8.19) to (8.20) is reminiscent of the difference between *Gauss-Seidel* and *Jacobi* iterations for linear systems [Fle00], see Sec. 3.7.2 It also resembles the relation between batch and stochastic gradient descent, cf. see Appendix A.48.

Here terms $\mathcal{O}(\eta^2)$ have been omitted. The iteration (8.22) is known as *Oja's Rule* after Erkki Oja [Oja82]. It combines Hebbian learning with an appropriate weight decay that realizes an approximate normalization of $\mathbf{w}$ for small learning rates.

Further principal components

In principle, one could apply the power method also for the second and all following principal components. Theoretically, we could avoid contributions of $\mathbf{u}_1$ in the iteration (8.18) by starting from a $\mathbf{w}(0)$ with coefficient $a_1 = 0$. Analogously, we would set $a_1 = a_2 = \ldots a_{m-1} = 0$ when computing $\mathbf{u}_m$. Similarly, one could consider the modified matrices

$$\mathcal{C}_m = \mathcal{C} - \sum_{k=1}^{m-1} \lambda_k [\mathbf{u}_k \mathbf{u}_k^\top]$$

for the iteration of the vector $\mathbf{w}_m$. However, both ideas are at risk to fail in practice, as numerical inaccuracies will always introduce small but non-zero contributions of $\mathbf{u}_1$ which will blow up in the iterations, if not corrected for. It is more promising to iterate a set of vectors $\{\mathbf{w_j}\}_{j=1}^M$ in parallel and impose appropriate conditions after each step, e.g. by means of a Gram-Schmidt orthonormalization.

Two closely related extensions of Oja's Rule are based on this concept. We present here only the single step variants and refer the reader to the original literature [Oja89, San89]. For details and convergence proofs, see also [HKP91] for a more elaborate discussion.

OJA'S SUBSPACE ALGORITHM AND SANGER'S RULE (8.23)

Initialize a random $\mathbf{w}_m(0)$ with $|\mathbf{w}_m(0)| = 1$ for $m = 1, 2, \ldots M$
At discrete time step τ

- determine the index μ of the current example (sequential presentation)
- update the vectors $\mathbf{w}_m(\tau), m = 1, 2 \ldots M$

  according to OJA'S SUBSPACE ALGORITHM:

$$\mathbf{w}_m(\tau) = \mathbf{w}_m(\tau - 1) + \eta\, y_m^\mu(\tau) \left[\boldsymbol{\xi}^\mu - \sum_{k=1}^{M} y_k^\mu(\tau) \mathbf{w}_k(\tau - 1) \right] \tag{8.24}$$

  or according to SANGER'S RULE:

$$\mathbf{w}_m(\tau) = \mathbf{w}_m(\tau - 1) + \eta\, y_m^\mu(\tau) \left[\boldsymbol{\xi}^\mu - \sum_{k=1}^{m} y_k^\mu(\tau) \mathbf{w}_k(\tau - 1) \right] \tag{8.25}$$

  where $y_k^\mu(\tau) = \mathbf{w}_k(\tau - 1) \cdot \boldsymbol{\xi}^{\mu(\tau)}$.

The first terms in the brackets [...] of Eqs. (8.24, 8.25) correspond to the familiar Hebbian learning for linear units. The remaining terms can be motivated as approximate normalizations for $k = m$ as in (8.20), while the terms with $k \neq m$ are associated with the pair-wise orthogonalization of vectors.

Note that in Oja's subspace algorithm the sum in Eq. (8.24) is over all indices $k = 1, 2, \ldots M$, while in Sanger's rule (8.25) it is truncated to $k = 1, 2, \ldots m$ for the m-th vector. This imposes a hierarchy among the iterated weight vectors. In fact, one can show that Sanger's algorithm yields the ordered principal components, i.e. $\mathbf{w}_k(\tau) \to \mathbf{u}_k$ for large τ. On the contrary, in Oja's subspace algorithm the vectors $\{\mathbf{w}_k\}_{k=1}^{M}$ converge to form an arbitrary orthonormal basis of the same subspace. They do not necessarily become identical with $\mathbf{u}_k$ and do not display the same order with respect to the partial variances of the data.

Orthogonal vectors $\mathbf{w}_k \perp \mathbf{w}_l$ result in uncorrelated projections y_k^μ and y_l^μ. In fact, the updates (8.25) and (8.24) can also be derived as (single example) gradient maximization of cost function (8.9) complemented by penalty terms of the form $-\left[y_k^\mu\, y_l^\mu\right]^2$ for $k \neq l$.

8.3.3 Independent Component Analysis

As discussed above, PCA identifies directions in which uncorrelated projections display the largest variances. In Independent Component Analysis (ICA), the goal is to find orthogonal directions which are *independent* in a broader sense [HTF01, Sto04, HO00]. ICA is, for instance, used for *Blind Source Separation* to identify independent sources in a mixed signal [CJ10]. Algorithms based on Hebbian Learning have also been suggested for Independent Component Analysis, see e.g. [CLG08, LG19].

Typically, the problem is addressed by considering a proxy for statistical independence. Two popular concepts are

a) **Mutual information**
Here, projections are identified which carry as little information about each other as possible. For projections $x = \mathbf{w}^\top \boldsymbol{\xi}$ and $y = \mathbf{v} = \mathbf{v}^\top \boldsymbol{\xi}$ the mutual information can be determined as the relative entropy [HTF01]

$$I_{x,y} = \sum_{x \in \mathcal{X}} \sum_{y \in \mathcal{Y}} p_{x,y}(x,y) \log \left[\frac{p_{x,y}(x,y)}{p_x(x)\, p_y(y)} \right]$$

with the joint density $p_{x,y}$ in the domain $\mathcal{X} \times \mathcal{Y}$ and the marginal densities p_x and p_y. Minimizing $I_{x,y}$ identifies *independent directions* $\mathbf{w}$ and $\mathbf{v}$ in feature space.

b) **Deviation from Gaussianity**
Orthogonal directions in which the projections appear least Gaussian are expected to be most *interesting* and potentially relevant for the task at hand [HO00]. They would display, for instance, very skewed or multimodal histograms of projections.

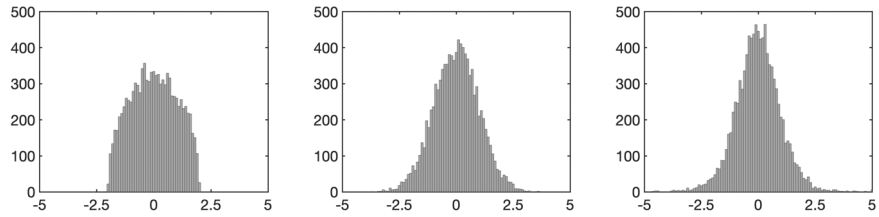


Figure 8.5: Left panel: three unimodal histograms with kurtosis ≈ -1 (left panel), kurtosis ≈ 0 (center panel, corresponding to a normal density), and kurtosis ≈ 2 (right panel), respectively.

As one example strategy for (b), we discuss here the maximization (in absolute value) of the *kurtosis* of projections $y^\mu = \mathbf{w} \cdot \boldsymbol{\xi}^\mu$ [HO00, HTF01]. It is defined as

$$\text{kurtosis} = \frac{\overline{(y - \overline{y})^4}}{\overline{(y - \overline{y})^2}^2} - 3, \tag{8.26}$$

where $\overline{(\ldots)}$ denotes means of the type $\overline{y} = \frac{1}{P} \sum_{\mu=1}^{P} y^\mu$. Very often, the data is centered and whitened in a first step, Eq. (8.17), appearing isotropic with unit variance and zero mean in any direction. In this case, Eq. (8.26) simplifies to

$$\text{kurtosis} = \overline{y^4} - 3. \tag{8.27}$$

The kurtosis as defined here is *zero* for normal densities.[6]

Projections $y^\mu = \mathbf{w}^\top \boldsymbol{\xi}$ with maximum kurtosis (in absolute value) can serve as a proxy for directions in which the data differs most from Gaussianity. Figure 8.5 displays histograms of a unimodal density with kurtosis< 0 (left panel), which appears *more bumpy* than a Gaussian (center panel), and a density with kurtosis> 0, which appears *pointier*. Similarly, other non-Gaussian densities, e.g. multimodal or skewed ones, also have a non-zero kurtosis.

8.4 Clustering and Vector Quantization

The identification and exploration of structures in a given data set can constitute a very useful preprocessing step. Quite often, methods of unsupervised learning are applied before the actual classification or regression scheme is implemented. The purpose can be to obtain insight into the complexity of the problem and into the difficulties that might be expected. If, for example, the data set contains several well-defined clusters or groups of similar feature vectors, this knowledge could be used in the design of a specific classifier. Similarly, density estimation techniques can be applied to obtain knowledge about the general statistical properties of the data.

[6] Note that the additive term -3 is sometimes omitted in the literature. Then, the kurtosis of a Gaussian density is obviously 3.

After a very brief discussion of elementary distance-based clustering methods, we present two prominent and related methods of unsupervised data analysis: Vector Quantization (VQ) by competitive learning and Gaussian Mixture Models (GMM) for density estimation.

8.4.1 Basic clustering methods

A variety of specific clustering techniques exist. Many of them are based on suitable distance measures in feature space like the familiar Euclidean metrics in the simplest case. In methods of *hierarchical clustering* the distance $D(C_a, C_b)$ between two clusters is derived from the pairwise $d(\mathbf{x_a}, \mathbf{x_b})$ of their individual elements $\mathbf{x}_a \in C_a$ and $\mathbf{x}_b \in C_b$. Prominent criteria for the computation of cluster distances are known as

- *single linkage* with
$$D(C_a, C_b) = \min \{d(\mathbf{x}_a, \mathbf{x}_b)\}_{\mathbf{x}_a \in C_a, \mathbf{x}_b \in C_b} ,$$
where the closest pair of feature vectors determines $D(C_a, C_b)$.

- *average linkage* where
$$D(C_a, C_b) = \frac{1}{|C_a||C_b|} \sum_{\mathbf{x}_a \in C_a} \sum_{\mathbf{x_b} \in C_b} d(\mathbf{x}_a, \mathbf{x_b})$$
is determined by the average pair-wise distance. Here, the number of vectors contained in a cluster C is denoted as $|C|$.

- *complete linkage* with
$$D(C_a, C_b) = \max \{d(\mathbf{x}_a, \mathbf{x}_b)\}_{\mathbf{x}_a \in C_a, \mathbf{x}_b \in C_b}$$
corresponds to the largest pairwise distance of vectors in C_a and C_b.

These and similar measures can be used in so-called hierarchical clustering approaches like

- *agglomerative clustering* (bottom up):
Here, every data point is initially considered a cluster. In every step the two clusters C_a and C_b, which are the closest according to some criterion $D(C_a, C_b)$, are merged into one cluster.

- *divisive clustering* (top down):
Here one starts by assigning all of the data to one cluster. In every step, one of the current clusters is selected and split into two sub-clusters. Various criteria can be considered in the cluster selection and to guide the actual division, see e.g. [DHS00].

Unlike K-means or similar methods, hierarchical clustering does not require a predefined number of clusters. On the contrary, the procedure generates a hierarchical tree of clusters. Deeper levels of this so-called *dendrogram* represent splits of the data set into an increasing number of sub-clusters. A particular configuration can be chosen once the dendrogram is constructed.

8.4.2 Competitive learning for Vector Quantization

One possible aim of unsupervised learning is the representation of a potentially large set of feature vectors by a few typical representatives. The term Vector Quantization (VQ) has been coined for this task. In VQ, so-called prototypes should reflect the frequency of observations in feature space. The goal could be to reduce storage needs or to reveal structures such as clusters in the data.

In the following we present a basic scheme for unsupervised Vector Quantization by competitive learning [Koh97, HKP91, Bis95a, HTF01, BHV16]. Technically, it resembles the methods of (supervised) Learning Vector Quantization discussed in Sec. 6.1 in that it also employs the concept of competitive learning. However, unsupervised VQ is applied to unlabeled feature vectors and its aim is the faithful representation of data sets, not an actual classification or regression task.

Like most unsupervised learning techniques, VQ is also guided by a cost function. Here, it is optimized in terms of the prototype vectors. The objective function measures the quality of a particular prototype configuration with respect to a given set of vectors $\mathbb{P} = \left\{\mathbf{x}^\mu \in \mathbb{R}^N\right\}_{\mu=1}^{P}$. In neuroscience jargon, the input vectors can be interpreted as *stimuli* which activate the neurons or units that are characterized by N-dim. vectors $\left\{\mathbf{w}^1, \mathbf{w}^2, \ldots \mathbf{w}^K\right\}$. The prototype vectors $\mathbf{w}^k \in \mathbb{R}^N$ can be seen as weight vectors, *exemplars* or *expected stimuli*, see [BHV16].

A very popular approach to VQ is based on the crisp assignment of any data point to the closest prototype, the so-called winner in terms of a pre-defined, fixed distance measure. We restrict the discussion to the use of simple squared Euclidean distance $d(\mathbf{x}, \mathbf{y}) = (\mathbf{x} - \mathbf{y})^2/2$ for $\mathbf{x}, \mathbf{y} \in \mathbb{R}^N$. Generalizations to alternative measures are in principle possible along the lines discussed in 6.1.

A corresponding, suitable cost function is given by the so-called quantization error [Bis95a, Bis06, HTF01, BHV16]:

$$H_{VQ} = \sum_{\mu=1}^{P} \tfrac{1}{2} \left(\mathbf{w}^{k(\mu)} - \mathbf{x}^\mu\right)^2 . \tag{8.28}$$

Here, $\mathbf{w}^{k(\mu)} \in \mathbb{R}^N$ denotes the winning prototype with the smallest Euclidean distance from $\mathbf{x}^\mu \in \mathbb{R}^N$:

$$d\left(\mathbf{w}^{k(\mu)}, \mathbf{x}^\mu\right) \leq d\left(\mathbf{w}^j, \mathbf{x}^\mu\right) \quad \text{for all } j = 1, 2, \ldots, K, \tag{8.29}$$

where ties are broken arbitrarily. Hence, H_{VQ} accumulates the quadratic distances of all feature vectors from their closest prototype. In this sense, the quantization error indicates how faithfully the set of feature vectors is represented by the prototypes.

Standard competitive VQ corresponds to the stochastic gradient descent based minimization of H_{VQ}, see Appendix A.48. The resulting algorithm is very intuitive and could be obtained on purely heuristic grounds: at each discrete time step, a single randomly selected feature vector $\mathbf{x}^\mu$ is drawn with equal probability from the data set. Then, the currently closest prototype $\mathbf{w}^{k(\mu)}$ is determined according to Eq. (8.29). Similar to the supervised LVQ1 algorithm

of Sec. 6.1, only the winner is adapted. However, in unsupervised VQ the winning prototype is always *moved closer to* the considered input vector; the update does not depend on additional information such as the labels in Learning Vector Quantization:

COMPETITIVE LEARNING (VECTOR QUANTIZATION)

- at time step t, select a single feature vector $\mathbf{x}^\mu$ randomly from the data set $\mathbb{P}$ with equal probability $1/P$
- identify the winning prototype, i.e.

$$\mathbf{w}^{k(\mu)} \text{ with } \quad d(\mathbf{w}^{k(\mu)}, \mathbf{x}^\mu) = \min \left\{ d(\mathbf{w}^j, \mathbf{x}^\mu) \right\}$$

- update only the winning prototype according to:

$$\mathbf{w}^{k(\mu)}(t+1) = \mathbf{w}^{k(\mu)}(t) + \eta(t) \left[\mathbf{x}^\mu - \mathbf{w}^{k(\mu)}(t) \right]. \tag{8.30}$$

The term *competitive learning* has been coined for this and several related training schemes as prototypes *compete* for updates [Koh97,Bis95a,Bis06,RMS92, BHV16]. The algorithm described above is an example of a Winner-Takes-All (WTA) scheme, extensions to the update of several prototypes at every time step have been considered also in the context of unsupervised Vector Quantization.

As in any stochastic gradient descent, convergence of the prototype vectors has to be guaranteed by employing a time-dependent or adaptive learning rate $\eta(t)$ which slowly approaches *zero* in the course of training, see the discussion of SGD in Appendix A.5.

Competitive Vector Quantization is closely related to the well-known Lloyd's algorithm or K-means algorithm [HTF01, Llo82, DHS00]. In contrast to the prescription (8.30) it considers all available data in each update of the system and alters all prototypes at a time:

K-MEANS ALGORITHM (LLOYD'S ALGORITHM)
Given a data set $\left\{\mathbf{x}^\mu \in \mathbb{R}^N\right\}$ $(\mu = 1, 2 \ldots P)$, initialize K prototype vectors $\left\{\mathbf{w}^j \in \mathbb{R}^N\right\}$ $(j = 1, 2, \ldots, N)$.
Repeat the following steps:

A) assign every data point $\mathbf{x}^\mu$ to the nearest prototype $\mathbf{w}^{k(\mu)}$ with

$$d\left(\mathbf{w}^{k(\mu)}, \mathbf{x}^\mu\right) = \min \left\{ d(\mathbf{w}^j, \mathbf{x}^\mu) \right\}.$$

B) compute updated prototypes (centers) $\mathbf{w}^j$ as the mean of all data points which were assigned to $\mathbf{w}^j$ in (A):

$$\mathbf{w}^j = \sum_{\mu=1}^P \mathbf{x}^\mu \delta_{j,k(\mu)} \Big/ \sum_{\mu=1}^P \delta_{j,k(\mu)}. \tag{8.31}$$

8.4.3 Practical issues and extensions of VQ

The quantization error H_{VQ} can be interpreted as a quality criterion when comparing different prototype configurations. However, this is only meaningful for systems with the same number of prototypes. In general, H_{VQ} will decrease with increasing K. Obviously, placing one prototype on each individual data point always gives the lowest possible quantization error $H_{VQ} = 0$.

A key difficulty of competitive VQ and the K-means algorithm is that the objective function H_{VQ} can display many sub-optimal local minima. Among other problems, this can lead to a strong dependence of the training outcome on the initial prototype positions. As an extreme example, placing a prototype in an empty region of feature space can prevent it from ever being identified as the winner for any of the data points, thus leaving it unchanged in a WTA training process. The term *dead unit* has been coined for such a prototype.

For competitive learning, rank-based schemes have been suggested which help to overcome this problem by updating not only the winner. Instead, prototypes are ranked according to their proximity to the presented feature vector, with e.g. the rank $r = 1$ of the winner, $r = 2$ of the second closest prototype etc. Generally, the magnitude of the update of a prototype is a decreasing function of r. A prominent example for the application of rank-based updates is the so-called *Neural Gas* algorithm [MBS93, RMS92, BHV16], which employs relatively large numbers of prototypes representing the density of data in feature space. The idea has also been extended in the context of supervised learning [HSV05].

In Self-Organizing Maps (SOM), competitive learning is also the key ingredient [Koh97]. In addition, prototypes are associated with a low-dimensional latent space, in which they form a grid. Updates affect the winning prototype as defined in conventional VQ, but neighbors of the winner in the grid are also updated. This way, the SOM can approximately preserve and visualize topologies, i.e. neighborhood relations, in the latent space. For more information on this popular method see e.g. [Koh97, RMS92, BHV16].

Vector Quantization vs. Clustering

Frequently, Vector Quantization is confused or even identified with clustering as indicated by the frequent use of the suggestive term *K-means clustering* in the literature. This is based on the idea that prototypes or centers should always represent pronounced clusters of similar data points. Then, the quantization error would correspond to the average within-cluster distances.

However, it is important to note that VQ is well-defined and useful even if there are no clusters present in the data. Figure 8.6 shows a few illustrative situations, where prototypes represent simple, two-dimensional data with minimal H_{VQ}. Panel (a) displays a single, elongated cluster represented by prototypes which characterize the variation of observed feature vectors. Here, the minimization of H_{VQ} does not correspond to the identification of a set of clusters. Panel (b) shows an idealized case of approximately spherical clusters. Each of the two clusters can be represented by one prototype. In panel (c) of

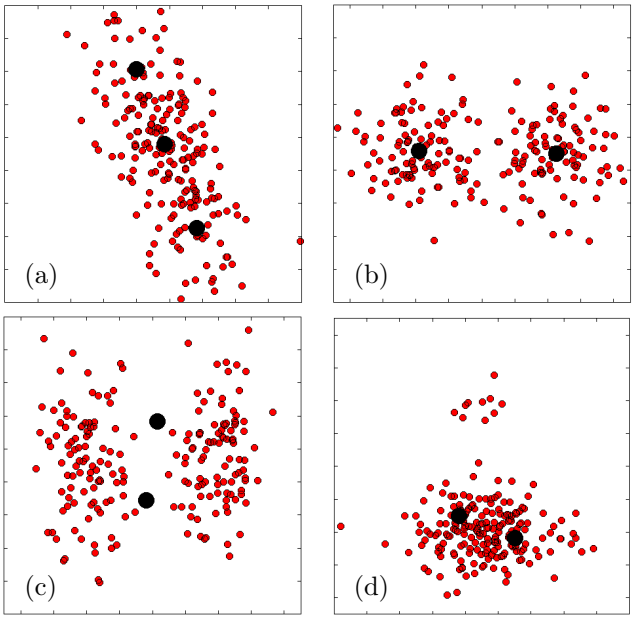


Figure 8.6: Vector Quantization: representation of two-dimensional data points by prototypes (schematic). In each panel, 200 data points are displayed as small (red) dots, prototype positions corresponding to minimal H_{VQ} are marked by filled black circles. The subpanels are referred to in the text of Sec. 8.4.3.

Fig. 8.6, clusters appear to be elongated along one of the axes in feature space. A minimal value of the quantization error can be achieved by placing the prototypes in the space between the apparent clusters, where hardly any examples are observed. Panel (c) illustrates the fact that the outcome of VQ can be highly sensitive to coordinate transformations, even if they are linear. Panel (d) of the Fig. 8.6 displays a situation in which a smaller, separated cluster is not *identified* at all. Although these few data points contribute large distances, their total contribution do not have a significant influence on the position of the prototypes when minimizing H_{VQ}.

The number of prototypes or clusters

As discussed in Sec. 2, unsupervised learning including clustering and Vector Quantization frequently lack simple performance measures which makes it difficult to select a model of suitable complexity. The goal of Vector Quantization can be formulated mathematically in terms of H_{VQ}, but the cost function is strictly speaking only suitable for comparing systems which have the same complexity, i.e. the same number of prototypes. Similar restrictions apply to explicit

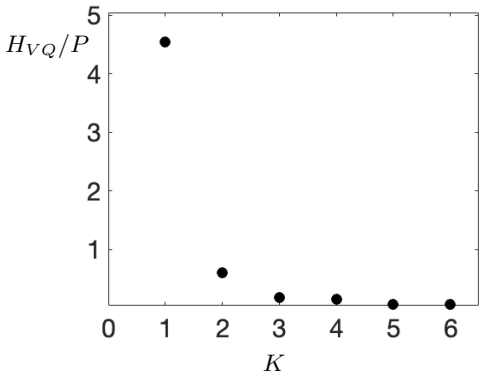


Figure 8.7: Illustration of the elbow method: the quantization error (per sample) as obtained from the Iris flower data set with $P = 150$ feature vectors [Fis36] as a function of K in the K-means algorithm.

methods of clustering and the related criteria.

In absence of additional information like domain knowledge providing a *ground truth*, it is impossible to infer the *correct* or *optimal* number of clusters or prototypes from the data set $\mathbb{P}$ alone. In this sense, clustering and VQ are ill-defined problems, ultimately their evaluation depends on the user's preferences and subjective quality criteria.

Several heuristic ideas have been suggested for the determination of a suitable number of clusters or prototypes. Here we illustrate the popular *elbow* method [Tho53] in terms of an application of the K-means algorithm. Fig. 8.7 displays the quantization error H_{VQ} as obtained in the Iris flower data set [Fis36] by applying the algorithm with different choices of K. In this simple example, we can conclude that $K = 3$ appears to be a reasonable choice. For $K = 1, 2$ the quantization is much higher, while for $K > 3$ no further, significant reduction of H_{VQ} is achieved. The curve displays a pronounced *elbow* shape that suggests $K = 3$, hence the term *elbow method*.[7]

Of course, the insight does not come as a surprise as the data set was already considered in Sec. 6.2.2 in the context of supervised learning. It contains samples from three different classes corresponding to a more or less pronounced cluster structure, see Fig. 6.2 for a discriminative visualization. In more general practical situations, the shape of the corresponding curves and the *elbow* is frequently much less conclusive.

Several related and alternative methods have been suggested, some of which are based on more rigorous statistical arguments. As one example, R. Tibshirani et *al.* suggested the so-called *gap statistics* as a criterion for the estimation of the number of clusters in K-means or other methods [TWH02].

[7] Depending on the actual quality criterion and the visual representation, the elbow could also be a *knee*.

8.5 Density estimation

A rather fundamental approach to obtaining insight into the properties of a given data set $\mathbb{P} = \{\boldsymbol{\xi}^\mu\}_{\mu=1}^P$ is to identify a model density that could have generated the observed data with high likelihood. Textbooks like [Bis95a, Bis06, HTF01] provide comprehensive overviews of relevant methods and theoretical background.

Several basic concepts can be applied in density estimation. In *parametric* methods, a particular statistical model is postulated and adapted. More precisely, a particular functional form of the density is assumed. Then, the optimization of its parameters can be formulated as an unsupervised learning process based on the observed feature vectors in $\mathbb{P}$.

So-called *non-parametric* methods [8] do not assume a specific functional form a priori, but aim to infer a descriptive density directly from the data [Bis95a]. Examples are so-called histogram methods or approaches using local kernels (not to be confused with kernel functions in the context of SVM and related methods).

Here we focus on the important family of *mixture models*, which are sometimes referred to as *semi-parametric* [Bis95a]. We first discuss some basic ideas behind parametric density estimation and then present the particular popular example of Gaussian Mixture Models (GMM), see e.g. [Bis95a, HTF01, DFO20].

8.5.1 Parametric density estimation

In the parametric approach we make explicit assumptions about a probability density that could explain the observed data set. Very frequently we will assume that feature vectors $\boldsymbol{\xi}^\mu \in \mathbb{R}^N$ can be interpreted as generated independently according to a specific identical N-dim. density. We furthermore assume an appropriate parametric form and, hence, we can write the likelihood and log-likelihood of the observed data as

$$L(\underline{\Theta}) = \prod_{\mu=1}^{P} p(\boldsymbol{\xi}^\mu|\underline{\Theta}) \quad \text{and} \quad \ell(\underline{\Theta}) = \ln(L(\underline{\Theta})) = \sum_{\mu=1}^{P} \ln p(\boldsymbol{\xi}^\mu|\underline{\Theta}). \tag{8.32}$$

Here, the vector $\underline{\Theta} \in \mathbb{R}^K$ concatenates the K parameters of the model density $p(\boldsymbol{\xi}|\underline{\Theta})$. The dimension K of $\underline{\Theta}$ as well as the type of parameters depend on the actual structure of the considered model.

The optimization of $\ell(\underline{\Theta})$ yields the corresponding maximum-likelihood model. Similar to the discussion in Sec. 2.13 we can extend the formalism by introducing a prior density $p_o(\underline{\Theta})$ and derive the resulting maximum a posteriori (MAP) model. Furthermore, Bayesian estimation techniques can be applied to obtain a probabilistic description of the model parameters, see Sec. 2.13 for a discussion

[8] Despite the suggestive name, non-parametric methods may very well comprise parameters and hyper-parameters.

in terms of linear regression. Here, we only follow the maximum likelihood approach and consider its application to a specific type of model densities in the next section.

8.5.2 Gaussian Mixture Models

As in any machine learning problem, model selection is a key difficulty also in the context of density estimation. The assumed model density should be appropriate to represent the complexity of the data. While the formalism is very powerful, it can be difficult to define suitable models which allow for an analytical or efficient numerical treatment. A particularly successful concept is the consideration of adaptive models which are defined as mixtures of specific basis functions. Gaussian densities $p(\boldsymbol{\xi}|\underline{\Theta})$ constitute a particular popular and convenient choice. Here we restrict our analysis to the particularly simple case with

$$\begin{aligned} p(\boldsymbol{\xi}|\underline{\Theta}) &= \sum_{m=1}^{M} p_m \; p(\boldsymbol{\xi}|m,\sigma_m,\mathbf{w}_m) \qquad (8.33) \\ \text{with} \qquad & p(\boldsymbol{\xi}|m,\sigma_m,\mathbf{w}_m) = \left(2\pi\sigma_m^2\right)^{-N/2} \exp\left[-\frac{1}{2\sigma_m^2}\left(\boldsymbol{\xi}-\mathbf{w}_m\right)^2\right] \end{aligned}$$

which also factorizes with respects to the components ξ_j. For simplicity we assume here that the contributing densities are isotropic with covariance matrices $\sigma_M I_N$. Extensions to more general N-dimensional Gaussian densities are of course possible.

The model parameters $\underline{\Theta}$ in (8.33) are given by the union of the sets $\left\{\sigma_m \in \mathbb{R}, \mathbf{w_m} \in \mathbb{R}^N, p_m \in \mathbb{R}\right\}$ for $m = 1, 2, \ldots M$. Hence, $\Theta \in \mathbb{R}^K$ with $K = M(N+2)$ in this special case. Note that the so-called mixing parameters p_m quantify the contribution of the individual Gaussians and have to satisfy the conditions

$$0 \le p_m \le 1 \;\; \text{for all } m \quad \text{and} \quad \sum_{m=1}^{M} p_m = 1. \qquad (8.34)$$

Given a particular realization of the model we assign a feature vector $\boldsymbol{\xi}$ to one of the contributing Gaussians with probability

$$Q_m = \frac{p_m \, p(\boldsymbol{\xi}|m,\sigma_m,\mathbf{w}_m)}{\sum_{k=1}^{M} p_k \, p(\boldsymbol{\xi}|k,\sigma_k,\mathbf{w}_k)} = \frac{\sigma_m^{-N} \, p_m \exp\left[-\frac{1}{2\sigma_m^2}\left(\boldsymbol{\xi}-\mathbf{w}_m\right)^2\right]}{\sum_{k=1}^{M} \sigma_k^{-N} \, p_k \exp\left[-\frac{1}{2\sigma_k^2}\left(\boldsymbol{\xi}-\mathbf{w}_k\right)^2\right]}. \qquad (8.35)$$

Note that also the Q_m are properly normalized with $\sum_{m=1}^{M} Q_m = 1$. Analogously, we define the quantities Q_m^μ by (8.35) evaluated in $\boldsymbol{\xi} = \boldsymbol{\xi}^\mu$.

For the specific model density (8.33), the log-likelihood (8.32) reads

$$\ell\left(\underline{\Theta}\right) = \sum_{\mu=1}^{P} \ln\left(\sum_{k=1}^{M} p_k \sigma_k^{-N} \exp\left[-\frac{1}{2\sigma_k^2}\left(\boldsymbol{\xi}^\mu-\mathbf{w}_m\right)^2\right]\right) + \text{const.} \qquad (8.36)$$

It is straightforward to work out the necessary conditions for a maximum of the log-likelihood [Bis95a]. They can be written in the suggestive form

$$\mathbf{w}_m = \sum_{\mu=1}^{P} Q_m^\mu \left(\frac{Q_m^\mu}{\sum_{\nu=1}^{P} Q_m^\nu} \right) \boldsymbol{\xi}^\mu \tag{8.37}$$

$$\sigma_m^2 = \sum_{\mu=1}^{P} \left(\frac{Q_m^\mu}{\sum_{\nu=1}^{P} Q_m^\nu} \right) (\boldsymbol{\xi}^\mu - \mathbf{w}_m)^2 \tag{8.38}$$

$$p_m = \frac{1}{P} \sum_{\mu=1}^{P} Q_m^\mu \tag{8.39}$$

with the assignment probabilities Q_m^μ defined in Eq. (8.35). Note that for the derivation of (8.39) the normalization $\sum_m p_m = 1$ has to be taken into account explicitly [Bis95a]. The equations can be solved self-consistently by means of the intuitive *natural iteration method*, see e.g. [Kik76]. If we interpret the r.h.s. as to define the next iterates of the quantities on the l.h.s. we obtain the following algorithm:

GAUSSIAN MIXTURE MODEL (isotropic Gaussian contributions)
- initialize $\mathbf{w}_m, \sigma_m, p_m$ for $m = 1, 2, \ldots, M$
- update the model parameters according the following iterative procedure:

$$\mathbf{w}_m(t+1) = \sum_{\mu=1}^{P} Q_m^\mu(t) \left(\frac{Q_m^\mu(t)}{\sum_{\nu=1}^{P} Q_m^\nu(t)} \right) \boldsymbol{\xi}^\mu \tag{8.40}$$

$$\sigma_m^2(t+1) = \sum_{\mu=1}^{P} \left(\frac{Q_m^\mu(t)}{\sum_{\nu=1}^{P} Q_m^\nu(t)} \right) \left(\boldsymbol{\xi}^\mu - \mathbf{w}_m(t+1) \right)^2 \tag{8.41}$$

$$p_m(t+1) = \frac{1}{P} \sum_{\mu=1}^{P} Q_m^\mu(t). \tag{8.42}$$

Note that here $\mathbf{w}_m(t+1)$ appears on the r.h.s. of (8.41), but the terms $Q_m^\mu(t)$ are evaluated by inserting the previous $\mathbf{w}_m(t)$ into Eq. (8.35). While this detail is not essential for the iteration to work, it is very interesting to note that this specific form can be derived as an *Expectation-Maximization* (EM) procedure for the optimization of the (log-)likelihood [Bis95a, HTF01, DFO20]. This methodological framework has been formalized by Dempster et *al.* for quite general problems involving *incomplete data*, see [DLR77]. In our density estimation problem we can interpret the assignment probabilities Q_m^μ as unknown *latent* variables. Detailed discussions of the very versatile EM-approach can be found in textbooks like [Bis95a] or [HTF01].

Figure 8.8 illustrates the application of the GMM algorithm in terms of a rather simple multi-modal density of two-dimensional data points (left panel).

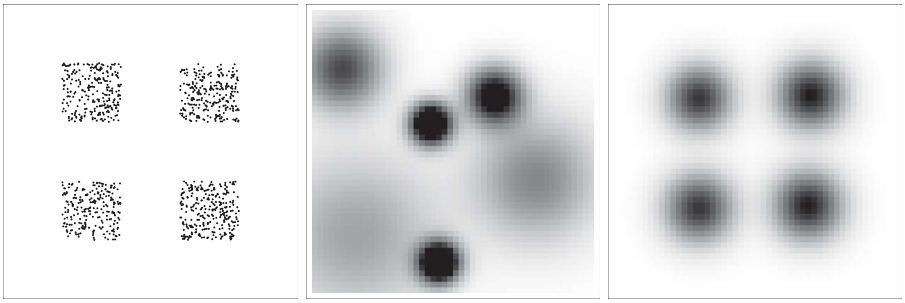


Figure 8.8: Illustration of density estimation by adaptation of a Gaussian Mixture Model. **Left panel:** 1000 two-dimensional data points drawn from a multimodal density. **Center panel:** initial density represented by a mixture of six Gaussians. **Right panel:** the model density after 20 EM-steps according to (8.40-8.42).

A mixture of six Gaussians is adapted to the data following the iteration (8.40-8.42), the center panel of Fig. 8.8 displays the initial configuration of the system with $p_k = 1/6$. After as few as 20 update steps, the density is approximated very well. Note that the model is overly complex with 6 Gaussians representing only 4 clusters in the data. However, this does not appear to constitute a problem in this simple setting. The complexity is reduced by coinciding Gaussians with equal means and variances or by eliminating redundant contributions by having mixing parameters $p_k \to 0$. The actual realization of the estimated density in terms of the model parameters will depend on the initialization in this example.

It is very instructive to consider the algorithm (8.40–8.42) in an extremely simplifying limit. Assume that the variances are equal and fixed in all components of the GMM, i.e. $\sigma_m^2 = \sigma^2$ which simplifies Eq. (8.35):

$$Q_m = \frac{p_m \exp\left[-\frac{1}{2\sigma^2}\left(\boldsymbol{\xi} - \mathbf{w}_m\right)^2\right]}{\sum_{k=1}^{M} p_k \exp\left[-\frac{1}{2\sigma^2}\left(\boldsymbol{\xi} - \mathbf{w}_k\right)^2\right]}. \tag{8.43}$$

If we now take the limit $\sigma \to 0$, the sum is dominated by the largest summand (the term with the smallest $(\boldsymbol{\xi} - \mathbf{w}_k)^2$), which is exponentially larger than all other ones. Consequently we obtain

$$\lim_{\sigma \to 0} Q_m = \begin{cases} 1 & \text{if } (\boldsymbol{\xi} - \mathbf{w}_m)^2 \leq (\boldsymbol{\xi} - \mathbf{w}_k)^2 \text{ for all } k \neq m \\ 0 & else. \end{cases} \tag{8.44}$$

Hence, in this limit, data points are assigned in a *crisp* way to the closest $\mathbf{w}_m$ according to Euclidean distance. In the GMM algorithm (8.40–8.42), the updates of the variances become obsolete and the p_m are simply computed as the fraction of crisp assignments to $\mathbf{w}_m(t)$, while the new $\mathbf{w}_m(t+1)$ are obtained as the means of the data points currently assigned to $\mathbf{w}_m(t)$.

In the considered limit, the GMM algorithm becomes identical with the intuitive K-means procedure presented in (8.31). This is yet another example for the observation made in Sec. 2.2 that many heuristic learning procedures can be interpreted as special cases or limits of more general statistical modelling schemes.

8.6 Missing values and imputation techniques

Many real world data sets are compromised by missing values, i.e. components of feature vectors that have not been observed, registered or communicated properly. Missing values can occur due to a variety of more or less complex reasons, see [Gho21] (chapter 3) and [GBB+23] for a comprehensive discussion. Quite frequently, the use of a certain spreadsheet based software tool leads to unwanted missingness or other artefacts and makes it difficult to excel. These are not explicitly discussed here, but always should be taken into consideration as a potential source of error.

In the literature, the following rather coarse categorization of missingness can be found [Lit88, GLSGFV10, LR02, Gho21, GBB+23], examples are taken from [Bla15][9]

- **MCAR:** *missing completely at random*
  Missingness is considered to be *completely at random* if the absence or presence of a feature does not depend on its value or on the value and/or missingness of other features. As an example, an accidentally damaged blood sample in a medical study would result in missing the observation of a particular feature.

- **MAR:** *missing at random*
  In this case, the fact that a feature is missing might be predictable from other information, but does not depend on the potential value of the missing feature. As an example, similar to the one presented in [Bla15], a person may miss an IQ test because they are ill on the day that the IQ test is given. This missingness relates to other available information about the person, but does not depend on the potential outcome of the test, i.e. the feature value itself.

- **MNAR:** *missing not at random*
  If missingness is specifically related to the feature that is missing, the somewhat vague term *not at random* is used. For instance, a person might have avoided a drug test resulting in a missing observation, because they took drugs and the outcome would be positive. Another example would be a feature that cannot be registered whenever it exceeds an allowed range of values, e.g. due to technical limitations of the measurement process.

The distinction of these types of missingness is not always very clear. Moreover, it can be difficult (if not impossible) to infer the underlying reason for

[9] See also https://www-users.york.ac.uk/~mb55/intro/typemiss4.htm

missingness from the provided data alone. The MCAR and MAR types, which are sometimes referred to as *ignorable*, are certainly the least difficult to handle. More systematic forms of missingness as in MNAR, would require sophisticated modelling techniques to take them properly into account. Consequently, the methods discussed in the following are most appropriate for data affected by MCAR or MAR type missingness.

8.6.1 Approaches without explicit imputation

Depending on the frequency and nature of missing features, simple strategies can be applied which ignore or eliminate the missingness beforehand or as an integral part of the training process.

∘ Deletion
The most straightforward idea to handle missingness is to include only complete feature vectors in the analysis and omit all others. This is appealing since it does not require explicit manipulations of the data. However, disregarding a subset of samples might introduce biases. In any case, simple deletion is only feasible if enough training data are available to begin with. In addition, one would have to reject all incomplete data in validation, test or working phase. A similar (also problematic) strategy is to omit features entirely if they appear to be missing in a significant fraction of the available samples.

∘ Training algorithms that can handle missingness
In some machine learning frameworks it is possible to restrict their application to the features that are present in a given observation. As an example, the computation and ranking of pairwise distances in a subset of samples can be restricted to the set of features that is present in all involved feature vectors. The concept could be applied, for instance, in Nearest Neighbor Classification. Similarly, in LVQ and other prototype based systems, cf. Chapter 6, the distances of an incomplete test or training sample from all prototypes can be computed and compared (excluding the missing components) for the identification of the closest prototype, see e.g. [GBV+17]. Another example of an algorithm that can handle missingness (and noise) without explicit imputation is the so-called Probabilistic Random Forest [RBS19].

8.6.2 Imputation based on available data

The most widely used approach to handle missing values is imputation, i.e. the replacement of missing features by more or less sophisticated estimates based on the available data. Several strategies and modifications of the basic idea are explained in the following. For details and references, see e.g. [Gho21, GBB+23].

∘ Naive estimates
A straightforward but naive idea is to replace a missing value in, say, feature

ξ_j by the corresponding mean or median in the data set as computed from all instances in which ξ_j is present. This can, however introduce or enhance a bias in imbalanced data sets.

As a seemingly more sophisticated choice, the class-wise mean or median of ξ_j could be used for imputation in a set of labeled training data. But this estimate would be unavailable for unlabeled feature vectors in the test or working phase.

◦ **Nearest-Neighbor imputation**
Applying a Nearest-Neighbor (NN) (or K-NN) approach partly solves the above mentioned problem and constitutes a popular tool for imputation. Given an incomplete feature vector $\boldsymbol{\xi}$ with missing value ξ_j, one can determine the nearest neighbor(s) of $\boldsymbol{\xi}$ in the training set e.g. in terms of the partial Euclidean distance w.r.t. the available dimensions. Then, the value of ξ_j in the nearest neighbor or an appropriate estimate based on the K nearest neighbors can be imputed.

◦ **Regression based imputation**
If a feature ξ_j can be expected to be correlated with other features ξ_k with $k \neq j$, imputation can be based on a regression scheme. From the available data we can infer this dependency by means of linear or non-linear regression and obtain a prediction for the missing value.

◦ **Cold deck and hot deck imputation**
The term *cold deck* imputation has been coined for situations in which a separate data set is used for imputing missing values in the training data or when testing/applying the trained system. On the contrary, in *hot deck* settings, the available (training) data set itself is used for determining the imputed values.

◦ **Generative modelling for imputation**
Methods discussed in Sec. 8.4, can be employed to estimate the density of data or a particular feature from a given data set. If the occurrence of feature ξ_j is modelled by e.g. a mixture of Gaussians, the resulting GMM can be used to generate random values for the imputation of missing values accordingly.

◦ **Multiple imputation**
Frequently, more than one imputation value is generated per missing feature. Generative approaches or randomized versions of regression based imputation can be used to provide multiple versions of the imputed data set. Then, training and validation can be performed on a number of versions in order to obtain reliable performance estimates.

As an example, the so-called *Multiple Imputation by Chained Equations* (MICE) has recently become popular [ASFL11, Gho21, GBB+23]. There, in a first step all missing values are replaced by simple mean or median imputation, with the exception of one selected feature dimension, say, ξ_k. Next, a regression based imputation is used to replace the missing values of ξ_k in the data set. Subsequently the missing values of a different feature $\xi_j (j \neq k)$ are

imputed by regression and the entire procedure is repeated until all missing features (and naive estimates) have been replaced by regression based imputation. The imputation of all missing values can be done with different selections of the initial feature ξ_k and by varying the order of features, thus obtaining several versions of the imputed data set.

8.7 Over- and undersampling, augmentation

Here we discuss the problems that arise when a classification problem is imbalanced in the sense that the prevalences of the individual classes are very different. Similar difficulties occur in regression problems with skewed and/or multimodal distributions of the target variable, but here we restrict the discussion to classification. For reviews of imbalance related strategies and further references consult, for instance [Cha09, HG09].

Class-imbalance can be handled in various ways when validating a given classifier, see the discussion in Sec. 7.4. However, it is often necessary to take the imbalance into account in the training phase already. Strong over-representation of certain classes in the training data can lead to very poor performance with respect to *minority classes*. Note that an imbalanced training set does not necessarily reflect the prevalences we can expect in the working phase.

8.7.1 Weighted cost functions

Assume that a training set comprises a total of $P = \sum_{j=1}^{C} P_j$ examples, with P_j of those representing class j in a C-class problem. Virtually all objective functions we have considered for training are of the general form

$$E(\underline{W}) = \frac{1}{P} \sum_{\mu=1}^{P} e^{\mu} \tag{8.45}$$

with the contribution e^{μ} of a single example in the data set. We can balance the influence of the different classes by considering the weighted cost function

$$E_{bal}(\underline{W}) = \sum_{j=1}^{C} \frac{1}{P_j} \sum_{\mu_j=1}^{P_j} e^{\mu_j} \tag{8.46}$$

where the partial sums over $\mu_j = 1, 2, \ldots P_j$ contain only examples from class $j \in \{1, 2, \ldots C\}$. Note that a gradient descent optimization of E_{bal} leads to updates that are equivalent to descent in the original E, Eq. (8.45), with class-specific learning rates $\eta_j \propto 1/P_j$.

8.7.2 Undersampling

Another straightforward way to compensate for class imbalances is to perform the actual training on sets with balanced class composition. This can

be achieved by randomly selecting the same number of examples from each class. Hence, the training set size will be limited to at most $\min\{P_1, P_2, \ldots P_C\}$ examples per class. Undersampling can be limited to the actual training set while validation and test sets can remain unbalanced, but the evaluation criteria should be robust against imbalance, like the BAC or other measures discussed in Sec. 7.4.2.

The random selection could be done once, disregarding the remaining data. However, this strategy would not make use of all available information and bears the risk of selecting atypical cases. Instead, the random undersampling should be performed repeatedly, which enables the computation of averaged performance measures.

8.7.3 Oversampling

One can also aim at increasing the effective influence of the underrepresented *minority* classes by generating additional examples for training. The two main ideas are described in the following.

Random oversampling

In this simple approach, the training set is augmented by exact copies of randomly selected examples in the underrepresented classes. It can be realized by random selection of examples with replacement.

For loss functions of the general form (8.45), the effect of including copied examples in the training data is very similar to weighting the classes as in Eq. (8.46). However, in the corrected cost function every example from a given class has the same weight.

In practice, random oversampling in the underrepresented classes and undersampling of the overrepresented ones are often combined.

Generative oversampling

Instead of using exact copies of available examples, one can aim at generating synthetic data which reflect the statistical properties of the minority classes.

The simplest idea would be to augment the training set by noisy copies of randomly selected feature vectors. In more sophisticated approaches one performs a suitable density estimation and uses the resulting model for generating additional samples.

The *Synthetic Minority Oversampling Technique* (SMOTE) [CBHK02] follows a similar, yet simpler concept. The basic scheme is summarized in the following:

SMOTE (Synthetic Minority Oversampling Technique)

- randomly select an example from the minority class
- determine its k nearest neighbors in the same class, e.g. according to Euclidean distance
- select one of the neighbors with equal probability $1/k$
- generate a new data point at a random position on the line connecting the two selected feature vectors, assign it to the same class.

Several modifications and extensions can be considered. For example, the neighbor identification and construction of the synthetic data point could be based on different distance measures, see [GBV+17] for an example of applying *geodesic* SMOTE in angle-based classification.

Practical issues

The suitability of the approaches discussed in Sec. 8.7.1–8.7.3 clearly depends on the availability of example data from all classes and on the details of the problem at hand.

Both the weighting of classes in the cost function and undersampling are easy to realize since they do not rely on generating synthetic data. Clearly, undersampling is only feasible for data sets with a reasonably large number of examples for each class to begin with. However, if that is not the case, the application of machine learning is questionable anyway.

Generative oversampling cannot really create fundamentally novel information, since the augmented feature vectors more or less faithfully reflect the properties of the original data set. As a consequence, the oversampling and subsequent training bears the risk of overfitting to the available minority class data. In addition, generative oversampling introduces problems of model and parameter selection, e.g. concerning the number of neighbors in SMOTE or the choice of a model density to begin with.

Imbalanced data can constitute a significant challenge in practical applications of machine learning. Measures to compensate the imbalance should be applied with care and they should always be evaluated in a proper validation process.

8.7.4 Data augmentation

In practical applications, even balanced data sets are often *augmented* by additional, artificial training data. Data augmentation can be naively motivated as a way to improve the training by providing more example data. In principle, one can employ all methods discussed for oversampling in the previous section also for the generation of additional data. The concept of data augmentation

is discussed briefly in [GBC16] with additional references given. One survey on image data augmentation for Deep Learning is provided in [SK19].

Obviously, some of the risks mentioned in Sec. 8.7.3 are also relevant for data augmentation. In principle, it is not possible to generate genuinely novel information from a given data set alone. However, a few cases in which augmentation appears justifiable can be identified:

- **domain knowledge based models**
  In specific applications, generative models of the expected data in the working phase may be available, which are based on explicit domain knowledge. Such models can be used to generate surrogate data for training and testing. Assume that the characteristics of a novel instrument, say a telescope in astronomy, are known in detail. If, in addition, key properties of the objects of interest are known, artificial data can be generated by means of simulations of the observation process and subsequently used to (pre-) train a classifier or regression system, see [RMBJ21] for just one recent example. Of course, systems trained on surrogate data should eventually be validated and tested in a real world setting.

- **increased robustness against noise**
  The robustness of a classifier w.r.t. input noise can be increased by complementing the training set by noisy copies of the original data. In fact, this strategy can be viewed as a regularization technique [Bis95b, GBC16]. Obviously the noise must not be too strong and should reflect the expected level in real data.

- **imposing invariances**
  Analogous strategies can be applied in order to achieve robustness with respect to other expected variations in feature space. This plays an important role in image processing tasks. The original data is often subjected to systematic variations, such as rotation, scaling, or skewing of objects. A properly designed training set will result in a classifier or regression system that displays the desired invariances.

  It is essential that the applied variations are suitable for the task at hand: while e.g. galaxy classification [NWB+19] will not be affected by arbitrary rotations in the image plane, as all orientations should occur in the data anway. On the contrary, handwritten character recognition is insensitive to rotations of the symbols only in a very small range of angles.

In many cases, data set augmentation appears to be a cheap but efficient way to increase the performance of the trained system. However, systematic alternatives should always be considered. In particular, invariances could be imposed through preprocessing, i.e. the extraction of invariant features. Even more elegantly, they can be achieved by appropriate network design and modified loss functions which realize invariant functions, see [WBJH18] for an example and further references.

Concluding quote

Everybody right now, they look at the current technology, and they think, "OK, that's what artificial neural nets are." And they don't realize how arbitrary it is. We just made it up! And there's no reason why we shouldn't make up something else.

— Geoffrey Hinton

Appendix A

Optimization

There is nothing objective about objective functions.

— James L. McClelland

Here we summarize some essential mathematical concepts concerning real-valued functions of multi-dimensional arguments and their optimization. In particular, we consider local extrema and gradient-based search strategies. We also discuss basic aspects of constrained optimization.

Note that we refrain from providing the precise (yet usually mild) mathematical conditions for the validity of expansions and applicability of theorems. For instance, we assume implicitly that all considered functions are continuous and differentiable, that second derivatives are symmetric, etc. For more rigorous presentations we refer the reader to the mathematical literature, e.g [Fle00, PP12, PAH19, Str19, DFO20].

A.1 Multi-dimensional Taylor expansion

Consider a real-valued function of the form

$$f:\ \mathbf{x} \in \mathbb{R}^d \to\ f(\mathbf{x}) \in \mathbb{R}. \tag{A.1}$$

If the value of the function $f(\mathbf{x}_o)$ and its derivatives are known in some point $\mathbf{x}_o$, we can perform a d-dimensional Taylor expansion to obtain the approximation

$$f(\mathbf{x}_o + \mathbf{h}) \ \approx\ f(\mathbf{x}_o) \ +\ \mathbf{h}^\top \nabla f(\mathbf{x}_o) \ +\ \frac{1}{2}\ \mathbf{h}^\top\, H(\mathbf{x}_o)\, \mathbf{h} + \mathcal{O}\left(|\mathbf{h}|^3\right) \tag{A.2}$$

in the vicinity of $\mathbf{x}_o$. Here, we assume that the norm of the deviation $\mathbf{h} \in \mathbb{R}^d$ is small: $|\mathbf{h}| \ \approx\ 0$. The first term merely corresponds to the simple estimate $f(\mathbf{x}) \approx f(\mathbf{x}_o)$ close to the reference point. The second term takes into account

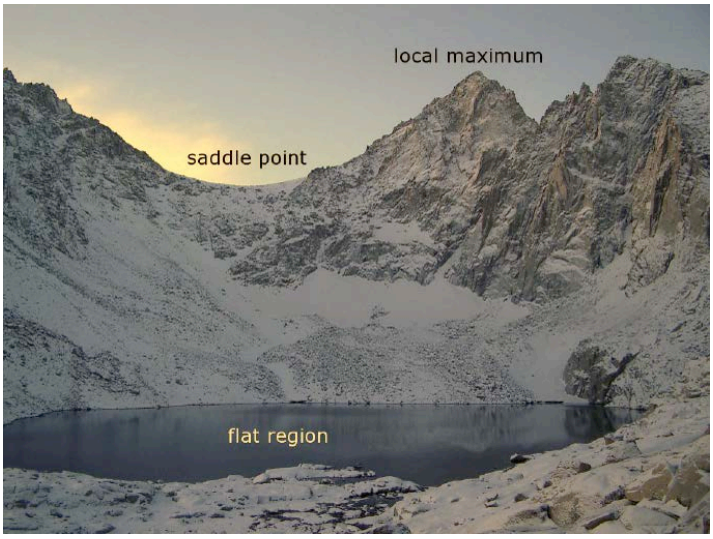


Figure A.1: Real world illustration of extrema and saddle points of a function in $d = 2$ dimensions (elevation $z(x, y)$). *Zero* gradients can correspond to maxima, minima, saddle points or extended flat regions. Photo taken on the Mt. Whitney trail (https://en.wikipedia.org/wiki/Mount_Whitney_Trail).

the first (partial) derivatives of the function with respect to the coordinates $\mathbf{x}$. With the formal vector

$$\nabla = \left(\frac{\partial}{\partial x_1}, \frac{\partial}{\partial x_2}, \dots \frac{\partial}{\partial x_d} \right)^\top \tag{A.3}$$

we obtain the gradient of f in $\mathbf{x}_o$, i.e. the vector of partial derivatives

$$\nabla f(\mathbf{x}_o) = \left(\frac{\partial f}{\partial x_1}, \frac{\partial f}{\partial x_2}, , \dots, \frac{\partial f}{\partial x_d} \right)^\top \Bigg|_{\mathbf{x}=\mathbf{x}_o} \quad \text{and} \quad \mathbf{h}^\top \nabla f(\mathbf{x}_o) = \sum_{j=1}^{d} h_j \left. \frac{\partial f}{\partial x_j} \right|_{\mathbf{x}=\mathbf{x}_o} . \tag{A.4}$$

The third term in (A.2) involves the $(d \times d)$-dimensional Hesse matrix or Hessian $H(\mathbf{x}_o)$ of second derivatives with elements

$$H_{ij}(\mathbf{x}_o) = H_{ji}(\mathbf{x}_o) = \left. \frac{\partial^2}{\partial x_i \partial x_j} H \right|_{\mathbf{x}=\mathbf{x}_o} \quad \text{and} \quad \mathbf{h}^\top H(\mathbf{x}_o)\, \mathbf{h} = \sum_{i,j=1}^{d} h_i H_{ij}(\mathbf{x}_o) h_j . \tag{A.5}$$

Higher order terms are obviously more involved than the quadratic approximation (A.2). Here we assume implicitly, that a quadratic approximation is suitable for the extrema under consideration.

A.2 Local extrema and saddle points

First, we consider unconstrained problems of the form

$$\underset{\mathbf{x} \in \mathbb{R}^d}{\text{minimize}} \; f(\mathbf{x})$$

with the real-valued objective function f. Obviously, analogous results are obtained for local maxima by considering the minima of $-f(\mathbf{x})$. First, we assume that $\mathbf{x}$ can be chosen anywhere in $\mathbb{R}^d$ without any restrictions. Hence, we do not have to consider minima at the border of *allowed regions*

Note that the term *local* minimum is not synonymous with *suboptimal* minimum, it only implies that the function locally increases whenever moving away from the minimum. In this sense, a global minimum is also a local minimum.

A.2.1 Necessary and sufficient conditions

An obvious, necessary condition for the presence of a local minimum of f in the point $\mathbf{x}^*$ is that all first derivatives vanish:

$$\nabla f(\mathbf{x}^*) = 0. \tag{A.6}$$

This follows immediately from the Taylor expansion (A.2) up to first order: If $\nabla f(\mathbf{x}^*) \neq 0$, a small displacement with $\mathbf{h} = -\eta \nabla f(\mathbf{x}^*)$ with $\eta > 0$ would result in

$$f(\mathbf{x}^* + \mathbf{h}) \approx f(\mathbf{x}^*) - \eta |\nabla f(\mathbf{x})|^2 < f(\mathbf{x}^*)$$

and, hence, $\mathbf{x}^*$ could not be a local minimum.

Assuming that (A.6) is satisfied, we have with the shorthand $H^* = H(\mathbf{x}^*)$

$$f(\mathbf{x}^* + \mathbf{h}) \approx f(\mathbf{x}^*) + \frac{1}{2} \mathbf{h}^\top H^* \mathbf{h} \tag{A.7}$$

$$\nabla f(\mathbf{x}^* + \mathbf{h}) \approx H^* \mathbf{h}. \tag{A.8}$$

From Eq. (A.7) we obtain the sufficient condition

$$\mathbf{x}^\top H^* \mathbf{x} > 0 \;\text{ for all }\; \mathbf{x} \in \mathbb{R}^d \tag{A.9}$$

in the presence of a local minimum in $\mathbf{x}^*$. If this is the case, small steps away from $\mathbf{x}^*$ will always lead *up-hill*, increasing the function value. This is obviously only correct as long as $|\mathbf{h}|$ is small enough and the higher-order corrections $\mathcal{O}(|\mathbf{h}|^3)$ can be neglected.

The Hessian H^* has to be positive definite, which is equivalent to the condition that all its eigenvalues are positive, i.e.

$$H^* \mathbf{u}_i = \rho_i^* \mathbf{u}_i \;\text{ with } \rho_i^* > 0 \;\; (\text{i=1,2,...d}) \tag{A.10}$$

Note that – due to the symmetry of H^* – all eigenvalues are real and we can construct an orthonormal set of eigenvectors $\{\mathbf{u}_i\}_{i=1}^d$ as a basis in $\mathbb{R}^d$ [PP12, Str19, DFO20].

Note also that an eigenvalue ρ_i relates to the curvature (second derivative) of the function f in x^* along the direction of the normalized eigenvector $\mathbf{u}_i$. Defining $\widehat{f}_i(\gamma) = f(\mathbf{x}^* + \gamma \mathbf{u}_i)$ and using $\mathbf{u}_i^\top \mathbf{u}_i = 1$ we have

$$\widehat{f}_i(\gamma) \approx (\mathbf{x}^*) + (\gamma \mathbf{u}_i)^\top H^*(\gamma \mathbf{u}_i) = \widehat{f}_i(0) + \gamma^2 \rho_i^*, \quad \text{i.e.} \quad \left.\frac{\partial^2 \widehat{f}_i}{\partial \gamma^2}\right|_{\gamma=0} = \rho_i^*$$

by comparison with the conventional, one-dimensional Taylor expansion with vanishing linear term.

If the Hessian is indefinite, i.e. if it has positive and negative eigenvalues, the point $\mathbf{x}^*$ corresponds to a saddle point: in some of the eigendirections $\mathbf{u}_i^*$ it displays a local maximum, while in others it behaves like a one-dimensional local minimum, cf. Fig. A.1.

Semi-definite H^* with $\rho_i^* \geq 0$ (or $\rho_i^* \leq 0$) require more careful considerations: Eigenvalues $\rho_i^* = 0$ can correspond either to an extended *flat region* (along $\mathbf{u}_i$) around a local extremum or to a saddle point in $\mathbf{x}^*$. We refrain from discussing this subtlety in detail and refer the reader to e.g. [Fle00, Str19, DFO20].

A.2.2 Example: unsolvable systems of linear equations

We consider a set of P linear equations in N variables $\mathbf{w} \in \mathbb{R}^N$ of the form

$$\mathbf{w}^\top \boldsymbol{\xi}^\mu \stackrel{!}{=} y^\mu \quad \text{for} \quad \mu = 1, 2, \ldots, P. \tag{A.11}$$

For $P > N$ the system is overdetermined and, in general, inconsistent, i.e. not solvable. Here, the notation is chosen to resemble the machine learning settings that we are considering throughout, where the coefficients and r.h.s. of the system are given by a data set of the familiar form

$$\mathbb{D} = \left\{ \boldsymbol{\xi}^\mu \in \mathbb{R}^N, y^\mu \in \mathbb{R} \right\}_{\mu=1}^P .$$

Using the matrix and vector notation introduced in Sec. 2.2.2, we define

$$Y = \left(y_1, y_2, \ldots y^P\right)^\top \in \mathbb{R}^P \quad \text{and} \quad \chi = \left[\boldsymbol{\xi}^1, \boldsymbol{\xi}^2, \ldots \boldsymbol{\xi}^P\right]^\top \in \mathbb{R}^{P \times N}. \tag{A.12}$$

Now Eq. (A.11) is conveniently written as

$$\chi \, \mathbf{w} \stackrel{!}{=} Y. \tag{A.13}$$

If χ happens to be an invertible square matrix with $N = P$, the solution of the problem is obviously $\mathbf{w}^* = \chi^{-1} Y$. However, if the set of equations is overdetermined and cannot be satisfied exactly, we can resort to approximative solutions. A natural and popular choice is to minimize the corresponding *Sum of Squared Error* (SSE), cf. Eq. (2.5), in the sense of a linear regression

$$\begin{aligned} E^{SSE}(\mathbf{w}) &= \frac{1}{2} \sum_{\mu=1}^{P} \left(\mathbf{w}^\top \boldsymbol{\xi}^\mu - y^\mu\right)^2 = \frac{1}{2} \left[\chi \mathbf{w} - Y\right]^2 \\ &= \frac{1}{2} \mathbf{w}^\top \left[\chi^\top \chi\right] \mathbf{w} - \mathbf{w}^\top \chi^\top Y + \frac{1}{2} Y^\top Y, \end{aligned} \tag{A.14}$$

where the last term is independent of $\mathbf{w}$. We proceed as in (2.2.2) by considering the first order, necessary conditions (A.6) for a solution $\mathbf{w}^*$:

$$\nabla_w E^{SSE} \;=\; [\chi^\top \chi]\mathbf{w}^* \,-\, \chi^\top Y \stackrel{!}{=} 0. \tag{A.15}$$

In Sec. 2.2.2 we have already presented the formal solution of (A.15) in terms of the left pseudoinverse [PP12, BH12]

$$\mathbf{w}^* = \chi^+_{left} Y \;\text{ with }\; \chi^+_{left} = \left(\left[\chi^\top \chi\right]^{-1} \chi^\top \right) \tag{A.16}$$

under the condition that $[\chi^\top \chi]$ is invertible. Here, the $(N \times P)$-dim. matrix χ^+_{left} satisfies $\chi^+_{left}\chi = I_{N\times N}$ with the N-dim. identity.

The assumption that $[\chi^\top \chi] \in \mathbb{R}^{N\times N}$ is invertible is consistent with the assumption that the equations $\{\mathbf{w}^{*\top}\boldsymbol{\xi}^\mu = y^\mu\}_{\mu=1,2,\ldots P}$ cannot be satisfied simultaneously, which - after all - was the motivation for minimizing E^{SSE}.

In our example we obtain the Hesse matrix of second derivatives in $\mathbf{w}^*$ as

$$H^*_{ik} = \frac{\partial^2 E^{SSE}}{\partial w_i \partial w_k} \;=\; \sum_{\mu=1}^{P} \xi_i^\mu \xi_k^\mu \;\text{ or }\; H^* \,=\, \chi^\top \chi \,\in \mathbb{R}^{N\times N}. \tag{A.17}$$

Note that for the quadratic E^{SSE} a Taylor expansion up to second order would be exact in the minimum $\mathbf{w}^*$, of course.

The sufficient second order condition (A.9) corresponds to positive definite H^*. Here, it is guaranteed that H^* is at least positive semi-definite:

$$\mathbf{u}^\top \chi^\top \chi \mathbf{u} = [\chi \mathbf{u}]^2 \geq 0 \;\text{ for any } \mathbf{u} \in \mathbb{R}^N.$$

In order to have positive definite H^*, the condition $P \geq N$ must be fulfilled[1]. For $P < N$, the $(N \times N)$-dim. matrix $H^* = [\chi^\top \chi]$ cannot have full rank and is bound to have *zero* eigenvalues. Correspondingly, the system (A.11) has many solutions. We will address this case in the discussion of constrained optimization in the next section.

Strict positive definiteness implies that $H^* = [\chi^\top \chi]$ is invertible. We conclude that – under this condition – the solution (A.16) exists and corresponds to a local minimum, which is also a global minimum since unrestricted quadratic costs cannot display other, local minima.

In Sec. 2.2.2 we have briefly considered the case of singular $[\chi^\top \chi]$ which cannot be inverted. There we solved the problem by introducing a non-singular $[\chi^\top \chi + \lambda I_N]$ with $\lambda > 0$, which corresponds to a simple form of *regularization*. Alternatively, we can make use of the *right pseudo-inverse* which is introduced and discussed below in Sec. A.3.2.

[1] $P \geq N$ is necessary, not sufficient. Correlations in the data set can still yield singular H^*.

A.3 Constrained optimization

A.3.1 Equality constraints

Frequently, one encounters optimization problems of the form

$$\underset{\mathbf{x}\,\in\,\mathbb{R}^d}{\text{minimize}}\; f(\mathbf{x}) \;\text{ subject to } n \text{ equality constraints } \{g_i(\mathbf{x})=0\}_{i=1}^{n}\,, \quad \text{(A.18)}$$

where the real-valued functions g_i define additional conditions under which $f(\mathbf{x})$ has to be minimized: The constraints $\{g_i(\mathbf{x})\}_{i=1}^{n}$ define the set of allowed arguments $\mathbf{x}$.

As a consequence, solutions of the problem (A.18) do not necessarily correspond to local minima of the (unrestricted) objective function f discussed in Sec. A.2. If it is possible to eliminate the conditions explicitly, one can transform the problem to an unconstrained one and proceed as before.

Formally, constraints given in the form of equations as in (A.18) can be dealt with systematically by introducing Lagrange multipliers $\{\lambda_i \in \mathbb{R}\}_{i=1}^{n}$. We define the Lagrange function or Lagrangian

$$\mathcal{L}\Big(\mathbf{x},\,\{\lambda_i\}_{i=1}^{n}\Big) = f(\mathbf{x}) - \sum_{i=1}^{n} \lambda_i\; g_i(\mathbf{x}) \qquad \text{(A.19)}$$

with the real-valued multipliers λ_i. Solutions of the constrained problem (A.18) satisfy the first order necessary conditions

$$\left\{ \left.\frac{\partial \mathcal{L}}{\partial x_j}\right|_{*} = 0 \right\}_{j=1}^{d} \quad \text{and} \quad \left\{ \left.\frac{\partial \mathcal{L}}{\partial \lambda_i}\right|_{*} = 0. \right\}_{i=1}^{n} \qquad \text{(A.20)}$$

where we use the shorthand $(...)|_*$ for the evaluation in $\mathbf{x}=\mathbf{x}^*$ and $\{\lambda_i = \lambda_i^*\}$. The second set of conditions merely corresponds to the original constraints $g_i(\mathbf{x})=0$.

Sufficient conditions for the presence of an optimum are non-trivial to formulate in the presence of constraints, see [Fle00, PAH19] for a detailed discussion. Note that the extended $(d+n)\times(d+n)$-dimensional Hessian of the Lagrange function $\mathcal{L}$ is indefinite, in general.

The conditions (A.20) can often be exploited in order to eliminate some of the variables or, in fact, the multipliers and to simplify the structure of the optimization problem significantly.

In Sec. 3.7.2 we present an important example of the above in terms of the Adaline problem, i.e. Widrow's *Adaptive Linear Neuron* [WH60, WL90]. There, we consider the minimization of the norm $|\mathbf{w}|^2$ under linear equality constraints $\{\mathbf{w}^\top \boldsymbol{\xi}^\mu S_T^\mu = 1\}_{\mu=1}^{P}$. The actual perceptron weights can be eliminated by making use of the first order conditions, while the Lagrange multipliers play the role of the embedding strengths $\vec{x} \in \mathbb{R}^P$. The resulting optimization problem corresponds to an unconstrained maximization of the modified cost function (3.77).

A.3.2 Example: under-determined linear equations

We revisit the set of linear equations considered in Sec. A.2.2

$$\chi\,\mathbf{w} \;=\; Y. \tag{A.21}$$

If it represents $P < N$ equations for the N unknowns $\mathbf{w} \in \mathbb{R}^N$, the system can have many solutions. Then, a number i_o of vectors $\mathbf{v}_i \neq 0$ of χ exist with $\chi\mathbf{v}_i = 0$. Consequently, for any given solution $\mathbf{w}$ of (A.21) we can construct a continuum of solutions

$$\mathbf{w} + \sum_{i=1}^{i_o} c_i\,\mathbf{v}_i \quad \text{with arbitrary coefficients } c_i \in \mathbb{R}.$$

The solution of minimal norm $\mathbf{w}^*$ is of particular interest. We can formulate the search for $\mathbf{w}^*$ as a quadratic optimization problem with linear equality constraints:

EXAMPLE: MINIMAL NORM SOLUTION OF LINEAR EQUATIONS

$$\underset{\mathbf{w}\in\mathbb{R}^N}{\text{minimize}} \;\; \frac{1}{2}\,\mathbf{w}^2 \quad \text{subject to} \;\; \chi\,\mathbf{w} = Y. \tag{A.22}$$

Introducing multipliers $\vec{\lambda} \in \mathbb{R}^P$, we obtain the Lagrange function

$$\mathcal{L}\big(\mathbf{w},\,\vec{\lambda}\big) = \frac{1}{2}N\mathbf{w}^2 - \lambda^\top\,(\chi\mathbf{w} - Y) \tag{A.23}$$

The first order necessary conditions (A.20) become

$$\mathbf{w} \;-\; \chi^\top\vec{\lambda} \;\stackrel{!}{=}\; 0 \quad \text{and} \quad \chi\mathbf{w} \;-\; Y \;\stackrel{!}{=}\; 0.$$

While the second condition is obvious, the first one suggests to eliminate $\mathbf{w}$. We obtain the modified cost function

$$\widetilde{\mathcal{L}} \;=\; -\frac{1}{2}\vec{\lambda}^\top[\chi\chi^\top] + \vec{\lambda}^\top Y \quad \text{and the stationarity condition} \quad -\chi\chi^\top\,\vec{\lambda} + Y \;\stackrel{!}{=}\; 0.$$

Note that here $[\chi\chi^\top] \in \mathbb{R}^{P\times P}$ is the counterpart of the $(N\times N)$-dim. matrix $[\chi^\top\chi]$ that we have encountered earlier in Eqs. (A.14 ff).

Here, we assume that $P < N$ and that $[\chi\chi^\top]$ is invertible. We therefore get immediately

$$\vec{\lambda} = [\chi\chi^\top]^{-1}\,Y \;\;\Rightarrow\;\; \mathbf{w} \;=\; \chi^\top[\chi\chi^\top]^{-1}Y.$$

This motivates the definition of the so-called *right pseudoinverse* [PP12, BH12]

$$\chi^+_{right} \;=\; \chi^\top[\chi\chi^\top]^{-1} \;\;\Rightarrow\;\; \chi\,\chi^+_{right} = I_{P\times P}, \tag{A.24}$$

with the P-dim. identity matrix $I_{P\times P}$.

Remark: A unified treatment
Unsolvable over-determined and solvable under-determined systems can be treated in a unified way [BH12]. Consider the limits

$$\lim_{\gamma\to 0^+} [\chi^\top \chi + \gamma\, I_N]^{-1}\, \chi^\top \quad \text{and} \quad \lim_{\gamma\to 0^+} \chi^\top\, [\chi\, \chi^\top + \gamma\, I_P]^{-1} \tag{A.25}$$

with the P-dim. and N-dim. identity matrices I_P and I_N, respectively. Both limits exist even if $\chi^\top\chi$ or $\chi\chi^\top$ is singular. The limits either coincide with the left pseudoinverse (2.8) or with χ^+_{right} defined above. Note also that Eq. (A.25, left) corresponds to a limit of the regularization term (2.9) that we introduced heuristically in Sec. 2.2.2.

A.3.3 Inequality constraints

The concept of Lagrange multipliers has been extended to the presence of inequality constraints in optimization problems of the form

$$\underset{\mathbf{x}\,\in\,\mathbb{R}^d}{\text{minimize}}\ f(\mathbf{x}) \ \text{ subject to } n \text{ constraints } \ \{g_i(\mathbf{x}) \geq 0\}_{i=1}^n\,. \tag{A.26}$$

We refrain from discussing the more general combination of inequality and equality constraints, which leads to fairly obvious extensions of the following. For details we refer the reader to [Fle00,PAH19,Str19,DFO20]. Note that, in principle, we could introduce pairs of inequality constraints $g_i(\mathbf{x}) \geq 0$ and $-g_i(\mathbf{x}) \geq 0$ simultaneously in order to include equality constraints, effectively.

Formally, the corresponding Lagrange function (A.19)

$$\mathcal{L}\Big(\mathbf{x},\, \{\lambda_i\}_{i=1}^n\Big) = f(\mathbf{x}) - \sum_{i=1}^n \lambda_i\; g_i(\mathbf{x})$$

is the starting point. First order necessary conditions for a solution of (A.26) are given by the Kuhn-Tucker (KT) Theorem of optimization theory [Fle00,PAH19]. Here, they read

KUHN-TUCKER CONDITIONS (only inequality constraints)

$$\nabla_x \mathcal{L}|_* = 0 \quad \Leftrightarrow \quad \nabla_x f|_* = \sum_{i=1}^n \lambda_i^*\ \nabla_x g_i|_* \qquad \text{(stationarity)} \tag{A.27}$$

$$g_i(\mathbf{x}^*) \geq 0 \quad \text{for} \quad i = 1, 2, \ldots n \qquad \text{(constraints)} \tag{A.28}$$

$$\lambda_i^* \geq 0 \quad \text{for} \quad i = 1, 2, \ldots n \qquad \text{(non-neg. multipliers)} \tag{A.29}$$

$$\lambda_i^*\, g_i(\mathbf{x}^*) = 0 \quad \text{for} \quad i = 1, 2, \ldots n. \qquad \text{(complementarity)} \tag{A.30}$$

where we use the shorthand $(...)|_*$ for the evaluation in $\mathbf{x} = \mathbf{x}^*$ and $\{\lambda_i = \lambda_i^*\}$.

The first condition (A.27) corresponds to the stationarity of the Lagrangian with respect to the variables $\mathbf{x}$. Condition (A.28) simply represents the original constraints, while (A.29) states that all multipliers are non-negative in the optimum. Individual multipliers $\lambda_i > 0$ correspond to so-called *active* constraints with $g_i(\mathbf{x}) = 0$, which follows from the complementarity condition (A.30). On the contrary, if $g_i(\mathbf{x}) > 0$ is satisfied with $\lambda_i = 0$, the constraint does not have to be enforced explicitly and is termed *inactive*.

More detailed interpretations and discussions of the KT conditions can be found in the literature, see for instance [Fle00, PAH19].

Example: optimal stability in the perceptron

As an important application of the KT theorem we exploit the necessary conditions (A.27–A.30) in the problem of maximum stability for the perceptron, see Sec. 3.7.3:

$$\underset{\mathbf{w} \in \mathbb{R}^N}{\text{minimize}} \quad \frac{N}{2}\mathbf{w}^2 \quad \text{subject to inequality constraints} \quad \{E^\mu \geq 1\}_{\mu=1}^P .$$

The KT conditions are based on the Lagrangian

$$\mathcal{L}\left(\mathbf{w}, \vec{\lambda}\right) = \frac{N}{2}\mathbf{w}^2 - \sum_{\mu=1}^{P} \lambda^\mu \left(\mathbf{w}^\top \boldsymbol{\xi}^\mu S_T^\mu - 1 \right).$$

which is the same as for the Adaline problem with equality constraints as given in Eq. (3.70). We work out the gradient

$$\nabla_w \mathcal{L} \;=\; N\,\mathbf{w} - \sum_{\mu=1}^{P} \lambda^\mu\, \boldsymbol{\xi}^\mu S_T^\mu$$

and obtain from (A.27-A.30) the following set of necessary conditions, which were already presented in Sec. 3.7:

KUHN-TUCKER CONDITIONS (optimal stability)

$$\mathbf{w}^* \;=\; \frac{1}{N}\sum_{\mu=1}^{P} \lambda^{*\mu}\, \boldsymbol{\xi}^\mu S_T^\mu \qquad \text{(embedding strengths } \lambda^\mu\text{)} \qquad \text{(A.31)}$$

$$E^{*\mu} \;=\; \mathbf{w}^{*\top} \boldsymbol{\xi}^\nu S_T^\mu \geq 1 \;\text{ for all } \mu \qquad \text{(linear separability)} \qquad \text{(A.32)}$$

$$\lambda^{*\mu} \;\geq\; 0 \;\; (\text{not all } \lambda^{*\mu} = 0) \qquad \text{(non-negative multipliers)} \qquad \text{(A.33)}$$

$$\lambda^{*\mu}\;(E^{*\mu} - 1) = 0 \;\text{ for all } \mu \qquad \text{(complementarity).} \qquad \text{(A.34)}$$

As outlined in Sec. 3.7, the first condition shows that the Lagrange multipliers play the role of embedding strengths. The weights can be interpreted as to result from iterative Hebbian learning and can, in fact, be eliminated from the optimization problem.

A.3.4 The Wolfe Dual for convex problems

The concept of duality plays an important role in the analysis and solution of optimization problems. The idea is typically to derive an alternative formulation of a given problem, which is then easier to handle numerically or even analytically. A particularly powerful framework is that of the so-called *Wolfe Dual* for quite general problems [Wol61], which is discussed in some detail in [Fle00] and [PAH19], for instance.

The Wolfe Dual simplifies significantly in the case of so-called *convex problems* which are of the form (A.26) with the additional requirements that

a) the set $I\!K = \{\mathbf{x} \,|g_i(\mathbf{x}) \geq 0 \;\text{ for } i = 1, 2, \ldots n\}$ is convex
b) the objective function $f(\mathbf{x})$ is a convex function on $I\!K$.

In particular, condition (a) is satisfied if all functions $g_i(\mathbf{x})$ are convex or even linear. One of the most important results for convex optimization problem states that every local solution of the problem is also a global solution [Fle00, PAH19].

Under rather mild assumptions[2] one can show that if $\mathbf{x}^*$ is a solution of (A.26), then $\left\{\mathbf{x}^*, \vec{\lambda}^*\right\}$ with the vector notation $\vec{\lambda} = \{\lambda_1, \lambda_2, \ldots \lambda_n\}$ solves the following problem:

WOLFE DUAL of a convex problem of the form (A.26):

$$\underset{\mathbf{x}\in\mathbb{R}^d,\, \vec{\lambda}\in\mathbb{R}^n}{\text{maximize}} \;\; \mathcal{L}\left(\mathbf{x}, \vec{\lambda}\right) \;\; \text{subject to } \nabla_x \mathcal{L}(\mathbf{x}, \vec{\lambda}) = 0 \;\text{ and }\; \vec{\lambda} \geq 0 \qquad \text{(A.35)}$$

with the Lagrangian $\mathcal{L}$ as defined in Eq. (A.19). Moreover, the solution satisfies $f(\mathbf{x}^*) = \mathcal{L}(\mathbf{x}^*, \vec{\lambda}^*)$.

Note that while in the original problem (A.26) f is minimized with respect to $\mathbf{x}$, here the maximization refers to λ and $\mathbf{x}$ under the constraint that $\nabla_x \mathcal{L} = 0$. For a more detailed discussion of this subtlety and the Wolfe Dual of more general problems, see [Fle00, PAH19].

Example: quadratic optimization under linear constraints

Frequently, the condition $\nabla_x \mathcal{L}(\mathbf{x}, \vec{\lambda}) = 0$ can be used to eliminate the original variables $\mathbf{x}$, resulting in a simplified optimization problem. As an example consider a quadratic problem with linear inequality constraints, which involves the vector of variables $\vec{x} \in \mathbb{R}^n$, i.e. $d = n$, a symmetric matrix $C \in \mathbb{R}^{n\times n}$ and a vector of constants $\vec{b} \in \mathbb{R}^n$:

[2] Most importantly, the functions f and g_i should be continuously differentiable. An additional so-called regularity assumption is discussed in [Fle00], sections 9.4 and 9.5.

EXAMPLE:

$$\underset{\vec{x}\in\mathbb{R}^n}{\text{minimize}} \quad \frac{1}{2}\vec{x}^\top C\,\vec{x} \quad \text{subject to} \quad C\vec{x} \geq \vec{b}. \tag{A.36}$$

A particular property of this example problem is that the same matrix C defines the quadratic form and the linear constraints. More general examples for the application of duality are presented in, e.g., [Fle00, PAH19]. Here, the corresponding Wolfe Dual (A.35) reads

EXAMPLE: Wolfe Dual of the quadratic problem (A.36)

$$\underset{\vec{x}\in\mathbb{R}^n,\,\vec{\lambda}\,\in\,\mathbb{R}^n}{\text{maximize}} \quad \frac{1}{2}\vec{x}^\top C\,\vec{x} - \vec{\lambda}^\top (C\vec{x} - \vec{b}) \tag{A.37}$$

$$\text{subject to} \quad C\vec{x} = C\vec{\lambda} \quad \text{and} \quad \vec{\lambda} \geq 0.$$

Now we can conveniently exploit the constraint $C\vec{x} = C\vec{\lambda}$ in order to eliminate $\vec{x}$. While it does not necessarily imply $\vec{x} = \vec{\lambda}$, we can still simplify the problem and obtain

EXAMPLE: Simplification of the Wolfe Dual (A.37)

$$\underset{\vec{\lambda}\,\in\,\mathbb{R}^n}{\text{maximize}} \quad -\frac{1}{2}\vec{\lambda}^\top C\,\vec{\lambda} + \vec{\lambda}^\top \vec{b} \quad \text{subject to} \quad \vec{\lambda} \geq 0. \tag{A.38}$$

Note that this specific example has the exact same mathematical structure as the problem of optimal stability discussed in Section 3.7, if we set $n = P$ and $\vec{b} = \vec{1} \in \mathbb{R}^P$ and moreover assume that $C \in \mathbb{R}^{P\times P}$ is defined according to Eq. (3.74). Renaming $\vec{\lambda} = \vec{x}$ we recover the Wolfe Dual (3.97) for the problem of optimal stability, see Sec. 3.7.

A.4 Gradient based optimization

The gradient is the basis or at least an important component of many practical optimization techniques. Gradient descent persists to be one of the most popular and most successful method for the training of neural networks and more general machine learning setups. We discuss the reasons for this perhaps somewhat surprising fact in Chapter 5.

A.4.1 Gradient and directional derivative

The gradient as defined in Eq. (A.4) is closely related to so-called directional derivative. Consider a normalized vector $\mathbf{a} \in \mathbb{R}^d$ with $|\mathbf{a}| = 1$. According to the

Taylor expansion (A.2), the corresponding directional derivative in $\mathbf{x}_o$ can be written as

$$\lim_{\alpha\to 0} \frac{f(\mathbf{x_o}+\alpha\mathbf{a})-f(\mathbf{x})}{\alpha} = \mathbf{a}^\top \nabla f(\mathbf{x}_o).$$

Obviously the conventional partial derivatives are recovered by setting $\mathbf{a} = \mathbf{e}^i$, i.e. by taking the directional derivative along the coordinate unit vectors $\mathbf{e}^i$ with $e^i_k = \delta_{ik}$.

In any direction with $\mathbf{a}^\top \nabla f(\mathbf{x}_o) < 0$ the function decreases, while $\mathbf{a}$ with $\mathbf{a}^\top \nabla f(\mathbf{x}_o > 0$ marks directions of ascent.

In a given point $\mathbf{x}_o$ the magnitude of the directional derivative quantifies how rapidly the function increases or decreases in the direction of $\mathbf{a}$. For a given $\nabla f(\mathbf{x}_o)$ we see immediately that we obtain the directions of ...

$$\begin{aligned} \text{... steepest ascent for} \quad & \mathbf{a} \propto +\nabla f(\mathbf{x}_o) \\ \text{... steepest descent for} \quad & \mathbf{a} \propto -\nabla f(\mathbf{x}_o) \\ \text{... stationarity for} \quad & \mathbf{a} \perp \nabla f(\mathbf{x}_o). \end{aligned}$$

The gradient is always perpendicular to the *level directions* along which f is locally constant.

A.4.2 Gradient descent

The properties of the gradient can be used for the numerical minimization of a function $f(\mathbf{x})$. Starting from an initial vector $\mathbf{x}(t=0)$, we consider an iteration of the form

$$\mathbf{x}(t+1) = \mathbf{x}(t) - \eta \nabla f(\mathbf{x}(t)). \tag{A.39}$$

If the step size $\eta > 0$ is sufficiently small, the Taylor expansion of f in $\mathbf{x}(t)$ is dominated by the linear term and we have

$$f(\mathbf{x}(t+1)) \approx f(\mathbf{x}(t) - \eta|\nabla f(\mathbf{x}(t))|^2 < f(\mathbf{x}(t)) \tag{A.40}$$

in every step of the sequence $\mathbf{x}(0) \to \mathbf{x}(1) \ldots \to \mathbf{x}(t) \to \mathbf{x}(t+1) \ldots$
Consequently f can only decrease and the gradient descent (A.39) should approach a local minimum. Analogously, we can devise gradient ascent with updates along $+\nabla f(\mathbf{x}(t))$ in order to approach a local maximum.

It is important that the step size η is *small enough* in order to guarantee validity of Eq. (A.40) and ensure convergence of the iteration. We will make this statement more precise in the following.

The role of the step size

We can investigate the behavior of gradient descent in greater detail by assuming that $\mathbf{x}(t)$ is already close to a local minimum $\mathbf{x}^*$ with

$$f(\mathbf{x}(t)) \approx f(\mathbf{x}^*) + \frac{1}{2}(\mathbf{x}(t)-\mathbf{x}^*)^\top H^*(\mathbf{x}(t)-\mathbf{x}^*) \quad \text{and} \quad \nabla f(\mathbf{x}(t)) \approx H^*(\mathbf{x}(t)-\mathbf{x}^*). \tag{A.41}$$

Introducing the shorthand $\boldsymbol{\delta}_t = \mathbf{x}(t) - \mathbf{x}^*$ for the deviation from the optimum, the update (A.39) at step t of the descent satisfies approximately

$$\boldsymbol{\delta}_t \approx \boldsymbol{\delta}_{t-1} - \eta H^* \boldsymbol{\delta}_{t-1} \approx [I - \eta H^*]\, \boldsymbol{\delta}_{t-1} \approx [I - \eta H^*]^2\, \boldsymbol{\delta}_{t-2} \ \ldots \approx [I - \eta H^*]^t\, \boldsymbol{\delta}_0$$

with the N-dim. identity matrix I. Here, we have subtracted $\mathbf{x}^*$ on both sides of Eq. (A.39) and approximated the gradient according to (A.41).

Now we can exploit the fact that an orthonormal basis of $\mathbb{R}^d$ can be constructed from the eigenvectors $\mathbf{u}_i$ of H^*. Hence, we can expand the initial deviation as

$$\boldsymbol{\delta}_0 = \sum_{i=1}^{d} c_i\, \mathbf{u}_i \quad \text{with coefficients} \quad c_i \in \mathbb{R}$$

in terms of the eigenvectors. We obtain immediately

$$\boldsymbol{\delta}_t \approx [I - \eta H^*]^t\, \boldsymbol{\delta}_0 = \sum_{i=1}^{d} c_i\, [1 - \eta \rho_i]^t\, \mathbf{u}_i$$

The corresponding squared magnitude reads

$$|\boldsymbol{\delta}_t|^2 \approx \sum_{i,j=1}^{d} c_i c_j\, [1 - \eta \rho_i]^t\, [1 - \eta \rho_j]^t\, \mathbf{u}_i^\top\, \mathbf{u}_j = \sum_{i=1}^{d} c_i^2\, [1 - \eta \rho_i]^{2t}$$

where we have used that $\mathbf{u}_i^\top \mathbf{u}_i = 1$ and $\mathbf{u}_i^\top \mathbf{u}_j = 0$ for $i \neq j$. We obtain that

$$\lim_{t \to \infty} |\boldsymbol{\delta}_t|^2 = 0 \quad \text{if and only if} \quad |1 - \eta \rho_i| < 1 \quad \text{for all } i.$$

Since all eigenvalues are positive and therefore $1 - \eta \rho_i < 1$ for all i, the $(t \to \infty)$ asymptotic behavior of $\boldsymbol{\delta}_t$ is dominated by the largest eigenvalue of the Hessian $\rho_{max} = \max\{\rho_1, \rho_2 \ldots, \rho_N\}$: The condition for convergence of the iteration to the local minimum is that

$$-1 < 1 - \eta \rho_{max} \ \Leftrightarrow \ \eta < \eta_{max} = \frac{2}{\rho_{max}}. \tag{A.42}$$

For a finite range of step sizes $0 < \eta < \eta_{max}$, convergence is guaranteed, once the iteration is sufficiently close to the minimum. Moreover, we note that the factor $(1 - \eta \rho_{max})$ changes sign for $\eta = 1/\rho_{max}$. Hence, for $\eta < \eta_{max}/2$ the iteration will approach the minimum smoothly, while for $\eta_{max}/2 < \eta < \eta_{max}$ the orientation of $\boldsymbol{\delta}_t$ flips in every gradient step and $\mathbf{x}(t)$ displays an oscillatory behavior while still approaching the minimum as $t \to \infty$.

Finally, for $\eta > \eta_{max}$, the iteration diverges and moves away from $\mathbf{x}^*$ in at least one eigendirection of H^*. The three different regimes are illustrated and summarized in Figure 5.6 of Chapter 5.

It is important to realize that the above result is valid only in the vicinity of a given, local optimum. It does not enable us to make statements concerning the behavior or the role of the step size far away from an optimum.

The fact that convergence can be achieved with a constant, non-zero value of η constitutes an important insight. However, the practical usefulness of this insight is limited: the properties of the Hessian can be very different for every local minimum and they are obviously not available in advance. To a large extent, the initialization $\mathbf{x}(0)$ of a gradient procedure will determine which of the (potentially many) local minima will be approached.

Close to a specific minimum, the local Hessian $H(\mathbf{x}(t))$ can be seen as an estimate of H^* and used to select a suitable step size. More sophisticated higher order optimization techniques like Newton- or Quasi-Newton methods use $H(\mathbf{x}(t))$ or an approximation thereof explicitly in the update, see for instance [Fle00,PAH19]. We refrain from introducing these more involved techniques here and restrict the discussion to relatively simple gradient-based methods which are frequently used in the context of machine learning.

A.4.3 The gradient under coordinate transformations

A frequently ignored property of the gradient is that the direction of steepest descent or ascent behaves non-trivially under coordinate transformations.

In the context of the Adaline algorithm we see in Sec. 3.7.2 that gradient descent for the cost function of Eq. (3.77) in the space of embedding strengths $\vec{x} \in \mathbb{R}^P$ is not equivalent to gradient descent in terms of the weights $\mathbf{w} \in \mathbb{R}^N$.[3] Here we consider even simpler linear transformations from $\mathbf{x} \in \mathbb{R}^d$ to $\mathbf{z} \in \mathbb{R}^d$:

$$\mathbf{z} = A\mathbf{x} \quad \text{with components } z_i = \sum_{j=1}^{d} A_{ij} x_j \quad \text{and } A \in \mathbb{R}^{d\times d}.$$

In the following we use the notation $\nabla_x = (\partial/\partial x_1, \partial/\partial x_2, \dots \partial/\partial x_d)^\top$ and ∇_z analogously to indicate which set of variables the gradient refers to.

Interpreting the objective as a function of $\mathbf{z}$ we obtain with the chain rule

$$\frac{\partial f}{\partial x_k} = \sum_{i=1}^{d} \frac{\partial f}{\partial z_i} \frac{\partial z_i}{\partial x_k} = \sum_{i=1}^{d} A_{ik} \frac{\partial f}{\partial z_i} \qquad \text{i.e.} \quad \nabla_x f = A^\top \nabla_z f, \tag{A.43}$$

or, assuming that A is invertible: $\nabla_z f = A^{-\top} \nabla_x f$, while $\mathbf{z} = A\mathbf{x}$.

The transformation of the gradient is different from that of "ordinary" vectors, see [Tou12] for a brief discussion. The direction of steepest descent in the transformed coordinates does not coincide with the naive projection of the original gradient, in general.

Therefore, we should be aware that a gradient descent procedure found for one coordinate system does not necessarily follow the steepest descent in another. This seems to cast some doubt on the distinguished role of gradient descent. We note however, reassuringly, that directions of descent need not be the *steepest* in order to be useful in numerical minimization procedures.

[3] It can be formulated, however, as gradient descent w.r.t. a different cost function in $\mathbf{w}$.

For a more detailed discussion and an introduction of the so-called *co-variant* or *natural gradient* we refer to the lecture notes of M. Toussaint as a starting point [Tou12]. The corresponding method of Natural Gradient Descent can be related to information theoretic metrics and was pioneered by Shun-ichi Amari in the context of machine learning, see [Ama98] for a discussion and further references.

A.5 Variants of gradient descent

The previous section presents and discusses plain "classical" gradient descent for the minimization of cost functions in unrestricted optimization problems.

The basic idea of gradient descent can of course be modified and adapted to the specific properties and needs of a given problem. In the following we briefly discuss a number of variants which are particularly relevant in machine learning problems. This is the case, for instance, for the popular Stochastic Gradient Descent and its modifications and extensions.

A.5.1 Coordinate descent

Quite generally we consider the minimization of a continuously differentiable function $f(\mathbf{x})$ of d-dimensional arguments $\mathbf{x} = (x_1, x_2, \dots x_d)$. As discussed, the negative gradient $-\nabla_x f|_{\mathbf{x}=\mathbf{x}(t)}$ marks the direction of the steepest descent in a given point $\mathbf{x}(t)$. We use the shorthand $(\dots)|_t$ for $(\dots)|_{\mathbf{x}=\mathbf{x}(t)}$ in the following.

From a practical point of view it is often advantageous to resort to *some* direction of descent $\mathbf{a}(t)$ which is not necessarily the steepest and, therefore, only has to satisfy

$$\mathbf{a}(t)^\top \nabla f|_t < 0. \tag{A.44}$$

A corresponding iterative procedure of the form $\mathbf{x}(t+1) = \mathbf{x}(t) + \eta\, \mathbf{a}(t)$ can be used to approach a (local) minimum of f, provided a suitable step size η is chosen.

A particularly simple example is the sequential performance of steps along one of the coordinate axes with $\mathbf{a}(t) \propto \mathbf{e}_{i(t)}$. It is straightforward to see that we can easily satisfy Eq. (A.44) by setting

$$\mathbf{a}(t) = -\left[\mathbf{e}_{i(t)}^\top \nabla f|_t\right] \mathbf{e}_{i(t)} = -\left.\frac{\partial f}{\partial x_{i(t)}}\right|_t \mathbf{e}_{i(t)} = \left(0, 0, \dots \left.\frac{-\partial f}{\partial x_{i(t)}}\right|_t, 0, \dots 0\right).$$

Hence, the iteration step changes $\mathbf{x}$ only in one component, $i(t)$, and it is determined by the corresponding partial derivative of f.
In deterministic-sequential coordinate descent a recursion of the form

$$i(0) = 1;\ i(t) = [i(t-1) \bmod d] + 1 \text{ for } \ t = 1, 2, 3, \dots$$

generates a cyclic sequence of the form $i(t) = 1, 2, 3, \dots, d, 1, 2, 3, \dots, d, 1, 2, \dots$

Essentially, the structure of the algorithm is the same as for a conventional gradient descent step when it is performed as a loop over the d coordinates. Here, however, in each component we make use of the previously updated coordinates, while in conventional gradient descent the values from the previous loop are used.

The resulting, so-called *coordinate-wise descent* or *coordinate descent* for short, is a simple, sometimes surprisingly efficient method for minimization[4]. A recent review of this classical approach can be found in, e.g., [Wri15]. Obviously, one could also consider modifications with randomized selection of the updated coordinate, etc.

A.5.2 Constrained problems and projected gradients

In general, the solution of constrained problems by means of gradient descent techniques requires considerable refinement.

One important approach in this context is *gradient projection* [Fle00]: Assume that the search space is restricted to a region $\mathcal{D}$ by equality and/or inequality constraints. If a step according to naive, unconstrained gradient descent yields $\widetilde{\mathbf{x}}(t+1) = \mathbf{x}(t) - \eta \nabla f \in \mathcal{D}$, the step is accepted and $\mathbf{x}(t+1) = \widetilde{\mathbf{x}}(t)$. Otherwise one determines $\mathbf{x}(t+1)$ as a projection into $\mathcal{D}$ as $\mathbf{x}(t+1) = \text{argmin}_{\mathcal{D}} \, ||\mathbf{x} - \widetilde{\mathbf{x}}||$ in an auxiliary optimization step. Depending on the precise nature of the constraints, the computation of the projected gradient can be costly.

In the case of simpler constraints, for instance as given by linear inequalities, one can often resort to so-called *active set* methods [Fle00, PAH19]. There, the iteration proceeds by unconstrained gradient descent or ascent until one or several constraints would be violated and, therefore, become active. Temporarily, active inequalities are treated as equality constraints in the following steps which *move along* the corresponding planes.

A.5.3 Stochastic gradient descent

A specific modification of gradient descent may be applied when the cost function can be written as a sum over a number of individual functions as in

$$f(\mathbf{x}) = \sum_{m=1}^{M} h_m(\mathbf{x}) \quad \text{with} \quad \nabla_x f(\mathbf{x}) = \sum_{m=1}^{M} \nabla_x h_m(\mathbf{x}), \tag{A.45}$$

where the second identify follows from the linearity of the gradient operator.

In machine learning, very often the training process is guided by a cost function which can be written as a sum over a given set of example data. In supervised learning, e.g. regression, it typically corresponds to an error measure evaluated with respect to the individual examples in the training set

$$\mathbb{D} = \left\{ \boldsymbol{\xi}^\mu \in \mathbb{R}^N, y^\mu \in \mathbb{R} \right\}_{\mu=1}^{P} .$$

[4]Of course, *coordinate ascent* can be formulated for maximization analogously.

A corresponding cost function can be written in the form

$$E(\underline{W}) = \frac{1}{P}\sum_{\mu=1}^{P} e^{\mu}(\underline{W}) \quad \text{where } e^{\mu}(\underline{W}) = e(\boldsymbol{\xi}^{\mu}, y^{\mu}|\,\underline{W}) \tag{A.46}$$

quantifies the contribution of an individual example data to the total costs. Quite generally, the vector $\underline{W}$ is meant to concatenate all degrees of freedom in the trained system, e.g. all adaptive weights and thresholds of a neural network. Obviously this is of the form (A.45) with variables $\underline{W}$ and individual functions $e^{\mu}(\underline{W})$. The discussion in the following refers to this machine learning setup and notation, but carries over to more general problems of the type (A.45),

Of course, the numerical minimization of E could be aimed at by applying standard gradient descent as for any other, more general objective function. In analogy to Eq. (A.39) the basic form of updates would be

$$\underline{W}(t+1) = \underline{W}(t) - \eta\,\nabla_W E(\underline{W}(t)). \tag{A.47}$$

We note, however, that costs of the form (A.46) can be interpreted as an empirical average $\overline{(\ldots)} = \sum_{\mu}(\ldots)/P$ over the data set, which corresponds to drawing examples from $\mathbb{D}$ with equal probability $1/P$:

$$E(\mathbf{w}) = \overline{e^{\mu}(\underline{W})}.$$

Accordingly, the gradient of E w.r.t. $\underline{W}$ can be written as a mean of individual gradients:

$$\nabla_w E(\underline{W}) \;=\; \frac{1}{P}\sum_{\mu=1}^{P} \nabla_W e^{\mu}(\underline{W}) \;=\; \overline{\nabla_W e^{\mu}(\underline{W})}.$$

This suggests to approximate the full gradient of E by computing a restricted empirical mean over a random subset of examples from $\mathbb{D}$. As an extreme case, we consider one randomly selected, single example in an individual training step:

$$\begin{aligned} \mu(t) &\in \{1,2,\ldots,P\} \quad \text{(randomly selected with equal probability } 1/P) \\ \underline{W}(t+1) &= \underline{W}(t) - \widehat{\eta}\,\nabla_W e^{\mu(t)}(\underline{W}(t)) \end{aligned} \tag{A.48}$$

where we denote the learning rate by $\widehat{\eta}$ to potentially distinguish it from η in batch gradient descent. Obviously, a single update step is, in general, computationally cheaper than the full gradient descent (A.47) for which a sum over all examples has to be performed.

Individual update steps follow a rough, stochastic approximation of the true gradient of E, hence the term stochastic gradient descent (SGD) has been coined for the iterative procedure. In practice, we could actually draw (with replacement) a random example from $\mathbb{D}$ independently in each step. Frequently, updates are organized in epochs, e.g. by generating a random permutation of $\{1,2,\ldots P\}$ and presenting the entire $\mathbb{D}$ in this order before moving on to the next epoch with a novel randomized order of the examples.

On average over the random selection process, an SGD update is guided by the true negative gradient $-\nabla_W E$ and, therefore, we can expect that the cost functions typically decreases over many steps for suitable choices of $\widehat{\eta}$. However, single training steps may actually increase the objective function temporarily, as individual terms $\nabla_W e^\mu$ can point *uphill* w.r.t. E with $(\nabla_W e^\mu) \cdot \nabla_W E > 0$.

Let us now study the behavior of SGD near or in a local minimum $\underline{W}^*$ with $\nabla_W E(\underline{W}^*) = 0$. Assuming that $\underline{W}(t) = \underline{W}^*$ we have that

$$\underline{W}(t+1) = \underline{W}(t) + \Delta\underline{W}(t) \quad \text{with} \quad \Delta\underline{W} = -\widehat{\eta}\,\nabla_W e^\mu(\underline{W}^*)$$

for a randomly selected $\mu \in \{1, 2, \ldots, P\}$. It follows that on average over the selection process

$$\langle \Delta\underline{W}(t) \rangle_{\mathbb{D}} = -\widehat{\eta}\, E(\underline{W}^*) \; = 0.$$

In this sense, the SGD training becomes stationary in a local minimum of E and $\langle \Delta\underline{W} \rangle_{\mathbb{D}} \to 0$ as $\mathbf{w} \to \underline{W}^*$. However, it is important to realize that, generally, in an individual update step, the weight vector will change even if $\underline{W} = \underline{W}^*$ is exactly satisfied. This can be seen from the average magnitude of the update step

$$\left\langle (\Delta\underline{W})^2 \right\rangle_{\mathbb{D}} \; = \; \widehat{\eta}^2 \left\langle \left(e^\mu(\underline{W}^*) \right)^2 \right\rangle_{\mathbb{D}} = \widehat{\eta}^2 \frac{1}{P} \sum_{\mu=1}^{P} \left(\nabla_W e^\mu(\underline{W}^* \right)^2 \geq 0.$$

Note that $\langle (\Delta\underline{W})^2 \rangle_{\mathbb{D}} = 0$ is possible iff all individual terms $\nabla_W e^\mu(\underline{W}^*) = 0$, which could mean that each contribution e^μ is minimized in $\underline{W}^*$ individually. If, for instance, a quadratic error measure of the form $e^\mu = (\sigma^\mu - \tau^\mu)^2/2$ with $\nabla_W e^\mu = (\sigma^\mu - \tau^\mu)$ is employed, this would imply that $\sigma^\mu = \tau^\mu$ for all examples simultaneously, representing a perfect solution with $E(\underline{W}^*) = 0$.

In general, however, $\langle (\Delta\underline{W})^2 \rangle_{\mathbb{D}} > 0$ in the local minimum $\underline{W}^*$. This indicates that for constant learning rate $\widehat{\eta} > 0$, the adaptive quantities $\underline{W}(t)$ will persist to fluctuate in the vicinity of a local minimum, even in the limit $t \to \infty$.

The generic behavior for $\widehat{\eta} > 0$ is illustrated in Fig. 5.7 of Chapter 5. As discussed there, Robbins and Monro [RM51], see also [Bis95a, HTF01], have shown that convergence can be achieved with a time dependent *learning rate schedule* with $\lim_{t\to\infty} \widehat{\eta}(t) = 0$ which satisfies the conditions (5.16):

$$\textbf{(I)} \; \lim_{T\to\infty} \sum_{t=0}^{T} \widehat{\eta}(t)^2 \; < \; \infty \quad \text{and} \quad \textbf{(II)} \; \lim_{T\to\infty} \sum_{t=0}^{T} \widehat{\eta}(t) \; \to \; \infty. \tag{A.49}$$

Intuitively, the first condition (I) states that $\widehat{\eta}(t)$ has to decrease *fast enough* in order to achieve a stationary configuration, eventually. Condition (II) implies that the decrease is *slow enough* so that the entire search space can be explored efficiently without stopping the iteration too early.

Simple schedules which reduce the learning rate asymptotically like $\widehat{\eta}(t) \propto 1/t$ for large t satisfy both conditions in (5.16). Just one possible and popular realization of such a decrease is of the form

$$\widehat{\eta}(t) \; = \; \frac{a}{b+t} \quad \text{with parameters} \quad a, b > 0.$$

Sophisticated schemes have been devised in which the learning rate is not explicitly time dependent, but is adapted in the course of training. The adaptation can be based on (estimated) second order derivatives or on the observed variance of the gradient over several update steps, for instance. Learning rate adaptation is frequently combined with so-called *momentum terms* which comprise information about previous updates. Up-to-date textbooks such as [GBC16] provide more details and further references. For a brief discussion, see Chapter 5, where also other modifications of SGD are introduced.

A.6 Example calculation of a gradient

In order to exemplify the computation of gradients in a feed-forward network, we consider here a specific two-layer architecture with N-dimensional input, K hidden units and a single output

$$\sigma(\boldsymbol{\xi}) = h\left(\sum_{j=1}^{K} v_j\, g\left(\mathbf{w}^{(j)}\cdot\boldsymbol{\xi}\right)\right). \tag{A.50}$$

All input-to-hidden weight vectors $\mathbf{w}^{(j)}$ and hidden-to-output weights v_j are assumed to be adaptive, while for simplicity local thresholds are not considered here. The output activation $h(\ldots)$ is taken to be (potentially) different from the hidden unit activations $g(\ldots)$.

For a given data set $\mathbb{D} = \{\boldsymbol{\xi}^\mu, \tau^\mu\}_{\mu=1}^P$, we consider the familiar quadratic deviation

$$E = \frac{1}{P}\sum_{\mu=1}^{P} e^\mu \quad \text{with single example terms} \quad e^\mu = \frac{1}{2}\big(\sigma(\boldsymbol{\xi}^\mu) - \tau^\mu\big)^2 \tag{A.51}$$

Note that only σ depends on the weights, the target values τ are fixed and given in the data set. In the following we work out derivatives for one example term e^μ only. For convenience, we omit the index μ and write σ in short for $\sigma(\boldsymbol{\xi})$.

First we compute the derivative with respect to one of the hidden-to-output weights. Only one term in the sum $\sum_{j=1}^K \ldots$ depends on v_k and we obtain

$$\frac{\partial e}{\partial v_k} = (\sigma-\tau)\,\frac{\partial \sigma}{\partial v_k} = \underbrace{(\sigma-\tau)\,h'\left(\sum_{j=1}^{K} v_j g(\mathbf{w}^{(j)}\cdot\boldsymbol{\xi})\right)}_{\text{shorthand: } \delta}\, g(\mathbf{w}^{(k)}\cdot\boldsymbol{\xi}). \tag{A.52}$$

Of course, h' has to be specified in a concrete setting. For instance, if $h(x) = \tanh(\gamma x)$ we have $h'(x) = \gamma\left(1-\tanh^2(\gamma x)\right)$.

Next, we take the derivative with respect to a single input-to-hidden weight $w_n^{(m)}$, i.e. the n-th component of the m-th input-to-hidden weight vector. With the same shorthand δ as defined above we obtain

$$\frac{\partial e}{\partial w_n^{(m)}} = (\sigma-\tau)\,\frac{\partial \sigma}{\partial w_n^{(m)}} = \delta\, v_m\, g'\left(\mathbf{w}^{(m)}\cdot\boldsymbol{\xi}\right)\,\xi_n. \tag{A.53}$$

The term $v_m\, g'(\ldots)$ corresponds to the derivative of the only term in the sum $\sum_{j=1}^{K} \ldots$ that contains the weight vector $\mathbf{w}^{(m)}$. Finally the factor ξ_n appears because

$$\mathbf{w}^{(m)} \cdot \boldsymbol{\xi} = \sum_{j=1}^{N} w_j^{(m)} \xi_j \quad \text{and thus} \quad \frac{\partial(\mathbf{w}^{(m)} \cdot \boldsymbol{\xi})}{\partial w_n^{(m)}} = \xi_n.$$

Note that the r.h.s. of Eqs. (A.52,A.53) are just numbers as they correspond to derivatives w.r.t. single weights. We could build a single gradient vector from the component-wise results by combining all adaptive quantities into a vector $\underline{W}$ as introduced in the previous section. It appears more natural, however, to write for $m = 1, 2, \ldots K$:

$$\nabla_{w^{(m)}} E = \delta\, v_m\, g'\left(\mathbf{w}^{(m)} \cdot \boldsymbol{\xi}\right) \boldsymbol{\xi} \tag{A.54}$$

where the notation $\nabla_{w^{(m)}}$ stands for the gradient with respect to the m-th input-to-hidden weight vector. Note that both sides of the equation obviously correspond to N-dim. vectors. The partial derivatives w.r.t. the v_k are given by the K additional equations (A.52).

For the gradient of the full cost function we have to perform sums over all examples. Note that the abbreviation δ is defined for a particular single example, as well. Hence we get

$$\frac{\partial E}{\partial v_k} = \frac{1}{P} \sum_{\mu=1}^{P} \underbrace{\left(\sigma(\boldsymbol{\xi}^\mu) - \tau^\mu\right) h'\left(\sum_{j=1}^{K} v_j g(\mathbf{w}^{(j)} \cdot \boldsymbol{\xi}^\mu)\right)}_{\text{shorthand: } \delta^\mu} g\left(\mathbf{w}^{(k)} \cdot \boldsymbol{\xi}^\mu\right) \tag{A.55}$$

$$\nabla_{w^{(m)}} E = \frac{1}{P} \sum_{\mu=1}^{P} \delta^\mu\, v_m\, g'\left(\mathbf{w}^{(m)} \cdot \boldsymbol{\xi}^\mu\right) \boldsymbol{\xi}^\mu. \tag{A.56}$$

In a network with more layers, the chain rule has to be applied several times and in each layer terms similar to δ from previous layers appear. While the network response σ is determined by propagating an input towards the output, the gradient is computed by propagating the deviation $(\sigma - \tau)$ backwards through the network. In both operations the same weights play the role of coefficients. This is the basic concept behind the famous *Backpropagation of Error* for efficient gradient calculation in multi-layered networks [RHW86].

List of figures

List of algorithms

Abbrev. and acronyms

Adaline adaptive linear element, adaptive linear neuron
AdaTron adaptive perceptron (algorithm)
AUC area under the curve
AUROC area under the receiver operating characteristics curve
BAC balanced accuracy
CM committee machine
CNN convolutional neural network
CoD coefficient of determination
ELM extreme learning machine
EM expectation-maximization (algorithm)
FP, FN false positives, false negatives (counts)
fpr, fnr false positive rate, false negative rate
GLVQ generalized learning vector quantization
GMLVQ generalized matrix relevance learning vector quantization
GMM gaussian mixture model
hom. homogeneous(ly)
inh. inhomogeneous(ly)
ICA independent component analysis
IQR interquartile range
KL Kullback-Leibler (divergence)
KT Kuhn-Tucker (e.g. KT conditions, KT theorem, KT point)
l.h.s. left hand side
lin. sep. linearly separable
LMS least mean squares
LVQ learning vector quantization
MAE mean absolute error
MAR missing at random
MAP maximum a posteriori (probability)
MCAR mis ing comp etely at r ndom
MNAR missing not at random
MDS multi-dimensional scaling
MICE multiple imputation by chained equations
MSE mean squared error
NPC nearest prototype classifier {classification}
OBD optimal brain damage
OBS optimal brain surgeon
ODE ordinary differential equation
PM parity machine

PCA principal component analysis
PCT perceptron convergence theorem
PR precision-recall
Prec precision
PSP perceptron storage problem
RBF radial basis functions
Rec recall
ReLU rectified linear unit
r.h.s. right hand side
ROC receiver operating characteristics
ROI region of interest
RSLVQ robust soft learning vector quantization
SENS sensitivity
SPEC specificity
SGD stochastic gradient descent
SMOTE synthetic minority oversampling technique
SNE stochastic neighborhood embedding
SOM self-organizing map
SSE sum of squared errors
SVM support vector machine
t-SNE t-distributed stochastic neighborhood embedding
TP, TN true positives, true negatives (counts)
tpr, tnr true positive rate, true negative rate
UMAP uniform manifold approximation and projection
VC Vapnik-Chervonenkis, e.g. in VC-dimension
VQ vector quantization
w.r.t. with respect to
WTA winner-takes-all

Notes on the bibliography

References are sorted alphabetically by their BibTeX keys for ease of browsing. The BibTeX source file is available upon request from the author or at `www.cs.rug.nl/~biehl`.

Most online sources point to the publisher's final versions, some of which might not be publicly available. Where possible, links to accessible preprint versions are provided as an alternative.

All of the provided online sources have been **accessed in April 2023**. However, the author cannot guarantee the correctness of the links and is not liable for potential copyright infringements on the corresponding websites.

Bibliography

[AB89] J.K. Anlauf and M. Biehl. The AdaTron: an adaptive perceptron algorithm. *Europhys. Lett.*, 10(7):687–692, 1989. Online: https://iopscience.iop.org/article/10.1209/0295-5075/10/7/014,
see also *Europhys. Lett.* 11(4):387 for an Erratum: https://doi.org/10.1209/0295-5075/11/4/016.

[ABR64] M. A. Aizerman, E. A. Braverman, and L. Rozonoer. Theoretical foundations of the potential function method in pattern recognition learning. In *Automation and Remote Control,*, number 25 in Automation and Remote Control, pages 821–837, 1964. Online: https://cs.uwaterloo.ca/~y328yu/classics/kernel.pdf.

[ABS99] M. Ahr, M. Biehl, and Schlösser. Weight decay induced phase transitions in multilayer neural networks. *Journal of Physics A: Mathematical and General*, 32:5003–5008, 1999. Preprint: https://arxiv.org/pdf/cond-mat/9901179.pdf.

[AK13] M.G. Augasta and T. Kathirvalavakumar. Pruning algorithms of neural networks - a comparative study. *Central European Journal of Computer Science*, 3:105–115, 2013. https://doi.org/10.2478/s13537-013-0109-x.

[Ama93] S. Amari. Backpropagation and stochastic gradient descent method. *Neurocomputing*, 5(4):185–196, 1993. Online: https://doi.org/10.1016/0925-2312(93)90006-O.

[Ama98] S. Amari. Natural Gradient Works Efficiently in Learning. *Neural Computation*, 10:251–276, 1998. Online: https://doi.org/10.1162/089976698300017746.

[AMB+18] B. Aubin, A. Maillard, J. Barbier, F. Krzakala, F. Macris, and L. Zdeborová. The committee machine: Computational to statistical gaps in learning two-layers neural network. In *Proc. Conf. on Neural Information Processing Systems (NeurIPS 2018)*, pages 3223–3234. Curran Associates Inc., 2018. Preprint: https://arxiv.org/abs/1806.05451.

[AN20] M. Ahmed and A.K.M. Najmul Islam. Deep Learning: Hope or Hype. *Ann. Data. Sci.*, 7:427–432, 2020. Online: https://doi.org/10.1007/s40745-019-00237-0.

[APW+09] A. Airola, T. Pahikkala, W. Waegeman, B. De Baets, and T. Salakoski. A comparison of auc estimators in small-sample studies. In S. Dzeroski, P. Guerts, and J. Rousu, editors, *Proceedings of the third International Workshop on Machine Learning in Systems Biology*, volume 8

of *Proceedings of Machine Learning Research*, pages 3–13, 2009. Online: https://proceedings.mlr.press/v8/airola10a.html.

[ASFL11] M.J. Azur, E.A. Stuart, C. Frangakis, and P.J. Leaf. Multiple imputation by chained equations: what is it and how does it work? *International Journal of Methods in Psychiatric Research*, 20(1):40–49, 2011. Online: https://doi.org/10.1002/mpr.329.

[BAK91] M. Biehl, J.K. Anlauf, and W. Kinzel. Perceptron learning by constrained optimization: the AdaTron algorithm. In F. Pasemann and H.D. Doebner, editors, *Neurodynamics: Proc. 9th Summer Workshop on Math. Physics, Arnold Sommerfeld Institut, Clausthal, 1990*, pages 194–210. World Scientific, 1991. Online: https://www.worldscientific.com/worldscibooks/10.1142/1526.

[BAP+12] A. Backhaus, P. Ashok, B. Praveen, K. Dholakia, and U. Seiffert. Classifying Scotch Whisky from near-infrared Raman spectra with a Radial Basis Function Network with Relevance Learning. In M. Verleysen, editor, *Proceedings of the European Symposium on Artificial Neural Networks ESANN 2012*, pages 411–416. d-side publishing, 2012. Online: https://www.esann.org/sites/default/files/proceedings/legacy/es2012-139.pdf.

[Bar19] D. Baron. Machine Learning in Astronomy: A Practical Overview, 2019. In [MSK19]. Online: http://research.iac.es/winterschool/2018/media/summaries/ml_summary_dbaron.pdf.

[BB00] K.P. Bennett and E.J. Bredensteiner. Geometry in Learning. In C.A. Gorini, editor, *Geometry at Work*, volume 53 of *MAA Notes*, pages 132–148. Mathematical Association of America, 2000. Online: https://www.researchgate.net/publication/2407543_Geometry_in_Learning.

[BBH12] K. Bunte, M. Biehl, and B. Hammer. A general framework for dimensionality-reducing data visualization mapping. *Neural Computation*, 24(3):771–804, 2012. Preprint: https://www.cs.rug.nl/~biehl/Preprints/2011-NECO-mapping.pdf.

[BBK20] J.N. Böhm, P. Berens, and D. Kobak. A unifying perspective on neighbor embeddings along the attraction-repulsion spectrum. *ArXiv*, 2020. Online: https://arxiv.org/abs/2007.08902.

[BBL07] M. Biehl, R. Breitling, and Y. Li. Analysis of Tiling Microarray Data by Learning Vector Quantization and Relevance Learning. In H. Yin, P. Tino, E. Corchado, W. Byrne, and X. Yao, editors, *Proc. Intelligent Data Engineering and Automated Learning, IDEAL*, volume 4881 of *Lecture Notes in Computer Science*, pages 880–889. Springer, Berlin, 2007. Online: https://pure.rug.nl/ws/files/10196444/2007LNCSBiehl.pdf.

[BBVZ17] G. Bhanot, M. Biehl, T. Villmann, and D. Zühlke. Biomedical data analysis in translational research: Integration of expert knowledge and interpretable models. In M. Verleysen, editor, *Proc. of the European Symposium on Artificial Neural Networks (ESANN 2017)*, pages 177–186. i6doc.com, 2017. Preprint: https://pure.rug.nl/ws/portalfiles/portal/41806420/2017_ESANN_biomedical_session.pdf.

[BC91] P. Baldi and Y. Chauvin. Temporal Evolution of Generalization during Learning in Linear Networks. *Neural Computation*, 3(4):589–603, 12 1991. Online: https://ieeexplore.ieee.org/document/6796708.

[BCLC15] M. Boareto, J. Cesar, V. Leite, and N. Caticha. Supervised variational relevance learning, an analytic geometric feature selection with applications to omic data sets. *IEEE/ACM Trans. Computational Biology and Bioinformatics*, 12(3):705–711, 2015. Online: https://www.researchgate.net/publication/271727720.

[BCS+22] M. Baillie, S. le Cessie, C.O. Schmidt, L. Lusa, M. Huebner, and the Topic Group Initial Data Analysis of the STRATOS Initiative. Ten simple rules for initial data analysis. *PLOS Computational Biology*, 18(2):1–7, 2022. Online: https://doi.org/10.1371/journal.pcbi.1009819.

[Bel57] R. Bellman. *Dynamic Programming*. Rand Corporation research study. Princeton University Press, 1957. Google books: https://books.google.de/books?id=wdtoPwAACAAJ.

[BGFG20] D. Blalock, J.J. Gonzalez Ortiz, J. Frankle, and J. Guttag. What is the state of neural network pruning? In I. Dhillon, D. Papailiopoulos, and V. Sze, editors, *Proceedings of Machine Learning and Systems*, volume 2, pages 129–146, 2020. Preprint: https://arxiv.org/pdf/2003.03033.pdf.

[BGH07] M. Biehl, A. Ghosh, and B. Hammer. Dynamics and generalization ability of LVQ algorithms. *Journal of Machine Learning Research*, 8:323–360, 2007. Online: https://www.jmlr.org/papers/v8/biehl07a.html.

[BGV92] B.E. Boser, I.M. Guyon, and V.N. Vapnik. A training algorithm for optimal margin classifiers. In *Proc. 5th Ann. Workshop on Computational Learning Theory (COLT '92)*, pages 144–152. ACM Press, NY, 1992. Preprint: https://www.svms.org/training/BOGV92.pdf.

[BH12] J.C.A. Barata and M.S. Hussein. The Moore-Penrose Pseudioinverse: A Tutorial Review of the Theory. *Brazilian Journal of Physics*, 42(1):146–165, 2012. Preprint: https://arxiv.org/abs/1110.6882v1.

[BHMM19] M. Belkin, D. Hsu, S. Ma, and S. Mandal. Reconciling modern machine-learning practice and the classical bias-variance trade-off. *Proc. Natl. Acad. Sci. USA*, 116(32):15849–15854, 2019. Preprint: https://arxiv.org/abs/1812.11118.

[BHS+16] M. Biehl, B. Hammer, F.-M. Schleif, P. Schneider, and T. Villmann. Stationarity of Matrix Relevance LVQ. In *Proc. IEEE International Joint Conference on Neural Networks (IJCNN 2015)*. IEEE, 2016. Preprint: https://www.cs.rug.nl/~biehl/Publications/PDFS/biehl-ijcnn2015.pdf.

[BHT23] S. Bates, T. Hastie, and R. Tibshirani. Cross-validation: what does it estimate and how well does it do it? *Journal of the American Statistical Association*, pages 1–22, 2023. Preprint: https://arxiv.org/abs/2104.00673.

[BHV14] M. Biehl, B. Hammer, and T. Villmann. Distance measures for prototype based classification. In L. Grandinetti, T. Lippert, and N. Petkov, editors, *Brain-Inspired Computing, BrainComp2013*, volume 8603 of *Lecture Notes in Computer Science*, pages 110–116. Springer, Berlin, 2014. Preprint: https://www.cs.rug.nl/~biehl/Preprints/2013-Cetraro-distances.pdf.

[BHV16] M. Biehl, B. Hammer, and T. Villmann. Prototype-based models in machine learning. *Wiley Interdisciplinary Reviews: Cognitive Science*,

7(2):92–111, 2016. Online: https://pure.rug.nl/ws/portalfiles/portal/172538141/wcs.1378.pdf.

[Bie17] M. Biehl. Biomedical applications of prototype based classifiers and relevance learning. In D. Figueiredo, C. Martin-Vide, D. Pratas, and M.A. Vega-Rodriguez, editors, *AlCoB: 4th International Conference on Algorithms for Computational Biology*, volume 10252, pages 3–23. Springer LNCS, 2017. Preprint: https://www.cs.rug.nl/~biehl/Preprints/2017-AlCoB-biehl.pdf.

[Bie19] M. Biehl. Supervised Learning - An Introduction, 2019. In [MSK19]. Online: http://research.iac.es/winterschool/2018/media/summaries/Winter-School-Biehl-notes-12-03-19.pdf.

[Bis95a] C.M. Bishop. *Neural Networks for Pattern Recognition*. Oxford University Press, Inc., New York, NY, USA, 1995. Google books: https://books.google.de/books/about/Neural_Networks_for_Pattern_Recognition.html?id=b8GuQgAACAAJ.

[Bis95b] C.M. Bishop. Training with Noise is Equivalent to Tikhonov Regularization. *Neural Computation*, 7(1):108–116, 1995. Preprint: https://www.microsoft.com/en-us/research/wp-content/uploads/2016/02/bishop-tikhonov-nc-95.pdf.

[Bis06] C.M. Bishop. *Pattern Recognition and Machine Learning (Information Science and Statistics)*. Springer, Heidelberg, Germany, 2006. Google books: https://books.google.nl/books/about/Pattern_Recognition_and_Machine_Learning.html?id=kTNoQgAACAAJ.

[BL88] D.S. Broomhead and D. Lowe. Multivariable functional interpolation and adaptive networks. *Complex Systems*, 2:321–355, 1988. Online: https://sci2s.ugr.es/keel/pdf/algorithm/articulo/1988-Broomhead-CS.pdf.

[BL89] S. Becker and Y. Le Cun. Improving the convergence of back-propagation learning with second order methods. In D. Touretzky, G. Hinton, and T. Sejnowski, editors, *Proc. Connectionist Models Summer School*, pages 29–37, 1989. Online: https://www.researchgate.net/publication/216792889_Improving_the_Convergence_of_Back-Propagation_Learning_with_Second-Order_Methods.

[BL13] V. Van Belle and P. Lisboa. Research directions in interpretable machine learning models. In M. Verleysen, editor, *Proc. of the European Symp. on Artificial Neural Networks (ESANN 2013)*, pages 533–431. d-side, 2013. Online: https://web.archive.org/web/20170829075124id_/https://www.elen.ucl.ac.be/Proceedings/esann/esannpdf/es2013-14.pdf.

[Bla15] M. Bland. *An Introduction to Medical Statistics*. Oxford University Press, 4 edition, 2015. selected material: https://www-users.york.ac.uk/~mb55/intro/introcon4.htm.

[BM01] E. Bingham and H. Mannila. Random projection in dimensionality reduction: Applications to image and text data. In *Proceedings of the Seventh ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, KDD '01, pages 245Ă–250, New York, NY, USA, 2001. Association for Computing Machinery. Online: https://doi.org/10.1145/502512.502546.

[BMR+20] T.B. Brown, B. Mann, N. Ryder, M. Subbiah, J. Kaplan, P. Dhariwal, A. Neelakantan, P. Shyam, G. Sastry, A. Askell, S. Agarwal, A. Herbert-Voss, G. Krueger, T. Henighan, R. Child, A. Ramesh, D.M. Ziegler, J. Wu, C. Winter, C. Hesse, M. Chen, E. Sigler, M. Litwin, S. Gray, B. Chess, J. Clark, C. Berner, S. McCandish, A. Radford, I. Sugskever, and D. Amodei. Language models are few-shot learners. In H. Larochelle, M. Ranzato, R. Hadsell, M.F. Balcan, and H. Lin, editors, *Advances in Neural Information Processing Systems, Proc. of NEURIPS 2020*, pages 1877–1901. Curran Associates, Inc., 2020. Preprint: https://arxiv.org/pdf/2005.14165.pdf.

[BN03] M. Belkin and P. Niyogi. Laplacian eigenmaps for dimensionality reduction and data representation. *Neural Computation*, 15:1373–1396, 2003. Online: https://www2.imm.dtu.dk/projects/manifold/Papers/Laplacian.pdf.

[BO91] M. Biehl and M. Opper. Tilinglike learning in the parity machine. *Physical Review A*, 44(10):6888–6894, 1991. Online: https://journals.aps.org/pra/abstract/10.1103/PhysRevA.44.6888.

[Bös96] S. Bös. Optimal weight decay in a perceptron. In C. van der Malsburg, W. von Seelen, J.C. Vorbrüggen, and B. Sendhoff, editors, *Artificial Neural Networks - ICANN 1996*, volume 1112 of *Lecture Notes in Computer Science*, pages 551–556, Berlin, 1996. Springer. Preprint: https://citeseerx.ist.psu.edu/viewdoc/download?doi=10.1.1.54.86.

[Bot91] L. Bottou. Stochastic gradient learning in neural networks. In *Proc. of Neuro-Nimes 91*. EC2 editions, 1991. Online: https://leon.bottou.org/publications/pdf/nimes-1991.pdf.

[Bot04] L. Bottou. Stochastic learning. In O. Bousquet and U. von Luxburg, editors, *Advanced Lectures on Machine Learning*, Lecture Notes in Artificial Intelligence, LNAI 3176, pages 146–168. Springer Verlag, Berlin, 2004. Preprint: https://leon.bottou.org/publications/pdf/mlss-2003.pdf.

[Bot19] A. Botchkarev. A new typology design of performance metrics to measure errors in machine learning regression algorithms. *Interdisciplinary Journal of Information, Knowledge, and Management*, 14:045–076, 2019. Online: https://dx.doi.org/10.28945/4184.

[Bre01] L. Breiman. Random Forests. *Machine Learning*, 45:5–32, 2001. https://www.stat.berkeley.edu/~breiman/randomforest2001.pdf.

[BSH+12] K. Bunte, P. Schneider, B. Hammer, F.-M. Schleif, T. Villmann, and M. Biehl. Limited rank matrix learning, discriminative dimension reduction, and visualization. *Neural Networks*, 26:159–173, 2012. https://www.techfak.uni-bielefeld.de/~fschleif/pdf/nn_2012.pdf.

[BSS94] N. Barkai, S. Seung, and H. Sompolinsky. On-line Learning of Dichotomies. In J.D. Cowan, G. Tesauro, and J. Alspector, editors, *Adv. in Neural Information Processing Systems. Proc. NIPS 2 (1993)*, volume 6, pages 303–310. MIT Press, 1994. Online: https://proceedings.neurips.cc/paper/1994/file/9c01802ddb981e6bcfbec0f0516b8e35-Paper.pdf.

[Bun11] K. Bunte. *Adaptive dissimilarity measures, dimension reduction and visualization*. PhD thesis, University of Groningen, 2011. Online: https://www.rug.nl/research/portal/files/14550974/kbunte_thesis.pdf.

[Bur89] P. Burman. A comparative study of ordinary cross-validation, v-fold cross validation and the repeated learning testing-model methods. *Biometrika*, 76:503–514, 1989. Online: https://www.jstor.org/stable/2336116.

[BW88] E.B. Baum and F. Wilczek. Supervised learning of probability distributions by neural networks. In D.Z. Anderson, editor, *Adv. in Neural Information Processing Systems*, pages 52–61. American Inst. of Physics, 1988. Online: https://proceedings.neurips.cc/paper/1987/file/eccbc87e4b5ce2fe28308fd9f2a7baf3-Paper.pdf.

[CBHK02] N.V. Chawla, K.W. Bowyer, L.O. Hall, and W.P. Kegelmeyer. SMOTE: Synthetic Minority Over-sampling Technique. *Journal of Artificial Intelligence Research*, 19:321–357, 2002. Online: https://www.jair.org/index.php/jair/article/view/10302/24590.

[CCST99] N. Cristianini, C. Campbell, and J. Shawe-Taylor. Dynamically adapting kernels in Support Vector Machines. In *Advances in Neural Information Processing Systems*, pages 204–210, 1999. Online: https://proceedings.neurips.cc/paper/1998/file/7fb8ceb3bd59c7956b1df66729296a4c-Paper.pdf.

[CGBNT03] K. Crammer, R. Gilad-Bachrach, A. Navot, and A. Tishby. Margin analysis of the LVQ algorithm. In S. Becker, S. Thrun, and K. Obermayer, editors, *Advances in Neural Information Processing Systems*, volume 15, pages 462–469. MIT Press, Cambridge, MA, 2003. Online: https://proceedings.neurips.cc/paper/2002/file/bbaa9d6a1445eac881750bea6053f564-Paper.pdf.

[CH67] T. Cover and P. Hart. Nearest neighbor pattern classification. *IEEE Trans. Information Theory*, 13:21–27, 1967. Online: https://isl.stanford.edu/~cover/papers/transIT/0021cove.pdf.

[Cha09] N.V. Chawla. Data Mining for Imbalanced Datasets. In M.A. Arbib, editor, *Data Mining and Knowledge Discovery Handbook*, pages 875–886. Springer, Boston (MA), 2009. Google books: https://books.google.de/books/about/Data_Mining_and_Knowledge_Discovery_Hand.html?id=S-XvEQWABeUC.

[Chu71] L.O. Chua. Memristor – The Missing Circuit Element. *Transaction on Circuit Theory*, CT-18(5), 1971. Online: https://citeseerx.ist.psu.edu/doc/10.1.1.189.3614.

[CJ10] P. Comon and C. Jutten. *Handbook of Blind Source-Separation*. Academic Press, Oxford, 2010. Online: https://www.gipsa-lab.grenoble-inp.fr/~pierre.comon/FichiersPdf/HandBook.pdf.

[CL04] B.W. Connors and M.A. Long. Electrical synapses in the mammalian brain. *Annu. Rev. Neurosci.*, 27:393–418, 2004. Online: https://longlab.med.nyu.edu/wp-content/uploads/2017/10/20.pdf.

[CLG08] C. Clopath, A. Longtin, and W Gerstner. An online Hebbian learning rule that performs Independent Component Analysis. *BMC Neuroscience*, 9(O13), 2008. Online: https://doi.org/10.1186/1471-2202-9-S1-O13.

[CMB00] J.L. Castro, C.J. Mantas, and J.M. Benitez. Neural networks with a continuous squashing function in the output are universal approximators. *Neural Networks Letter*, 13:561–563, 2000. Online: https://sci2s.ugr.es/sites/default/files/ficherosPublicaciones/0820_2000-benitez-NN.pdf.

[Cov65] T.M. Cover. Geometrical and Statistical Properties of Systems of Linear Inequalities with Applications in Pattern Recognition. *IEEE Trans. Electronic Computers*, pages 326–334, 1965. Online: https://isl.stanford.edu/~cover/papers/paper2.pdf.

[CR95] Y. Chauvin and D.E. Rumelhart. *Backpropagation: Theory, Architectures, and Applications*. Psychology Press, 1995. Online: https://www.taylorfrancis.com/books/mono/10.4324/9780203763247.

[CS14] G. Chandrashekar and F. Sahin. A survey on feature selection methods. *Computers & Electrical Engineering*, 40(1):16–28, 2014. Online: https://www.researchgate.net/publication/305952742_A_Survey_on_Feature_Selection,.

[CST00] N. Cristianini and J. Shawe-Taylor. *An Introduction to Support Vector Machines and Other Kernel- based Learning Methods*. Cambridge University Press, Cambridge, UK, 2000. Online: https://doi.org/10.1017/CBO9780511801389.

[CSZ06] O. Chapelle, B. Schölkopf, and A. Zien. *Semi-supervised learning*. MIT Press, Cambridge, MA, 2006. Online: https://www.molgen.mpg.de/3659531/MITPress--SemiSupervised-Learning.pdf.

[CUH16] D.-A. Clever, T. Unterhiner, and S. Hochreiter. Fast and Accurate Deep Network Learning by Exponential Linear Units (ELUs). In Y. Bengio and Y. LeCun, editors, *Proc. of the 4th International Conference on Learning Representations, ICLR 2016*, 2016.

[CV95] C. Cortes and V. Vapnik. Support-Vector Networks. *Machine Learning*, 20(3):273–297, 1995. Online: https://link.springer.com/article/10.1007/bf00994018.

[Cyb89] G. Cybenko. Approximations by Superpositions of a Sigmoidal Function. *Math. Control Signals Systems*, 2:303–314, 1989. Online: https://cognitivemedium.com/magic_paper/assets/Cybenko.pdf.

[Dav66] Ray Davies. Dedicated Follower of Fashion, 1966. The Kinks, single, online available at https://www.youtube.com/watch?v=nxTnJPIl20U.

[DFO20] M.P. Deisenroth, A.A. Faisal, and C.S. Ong. *Mathematics for Machine Learning*. Cambridge University Press, 2020. Online: https://mml-book.com.

[DG06] J. Davis and M. Goadrich. The Relationship Between Precision-Recall and ROC Curves. In *Proc. of the 23rd International Conference on Machine Learning (ICML)*, pages 233–240. ACM, New York, USA, 2006. Online: https://doi.acm.org/10.1145/1143844.1143874.

[DHS00] R.O. Duda, P.E. Hart, and D.G. Stork. *Pattern Classification*. Wiley, New York, 2000. Google books: https://books.google.nl/books?id=Br33IRC3PkQC.

[Die00] R. Dietrich. *Statistical Mechanics of Neural Networks: Enhancement by Weighting of Examples*. PhD thesis, Julius-Maximilians-Universität Würzburg, Germany, 2000. Online: https://citeseerx.ist.psu.edu/doc/10.1.1.15.4844.

[DLR77] A.P. Dempster, N.M. Lair, and D.B. Rubin. Maximum Likelihood from Incomplete Data via the EM Algorithm. *J. of the Royal Statistical Society. Series B (Methodological)*, 39(1):1–38, 1977. Online: https://www.ece.iastate.edu/~namrata/EE527_Spring08/Dempster77.pdf.

[DM92] C. Darken and J. Moody. Towards faster stochastic gradient search. In Hanson Moody, J.E, S.J., and R.P. Lippmann, editors, *Advances in Neural Information Processing Systems 4*, volume 4, pages 1009–1016, 1992. Online: https://proceedings.neurips.cc/paper/1991/file/e2230b853516e7b05d79744fbd4c9c13-Paper.pdf.

[DO87] S. Diederich and M. Opper. Learning of correlated patterns in spin-glass networks by local learning rules. *Physical Review Letters*, 58(9):949–952, 1987. Online: https://journals.aps.org/prl/abstract/10.1103/PhysRevLett.58.949.

[Dom00] P. Domingos. A unified bias-variance decomposition and its applications. In *Proc. 17th Intl. Conf. on Machine Learning (ICML)*, pages 231–238. Morgan Kaufmann, 2000. Preprint: https://homes.cs.washington.edu/~pedrod/papers/mlc00a.pdf.

[DRAP15] G. Ditzler, M. Roveri, C. Alippi, and R. Polikar. Learning in nonstationary environment: A survey. *Comput. Intell. Mag.*, 10:12–25, 2015. Online: https://www.researchgate.net/publication/282907128.

[DS98] N.R. Draper and H. Smith. *Applied Regression Analysis*. Wiley, 3 edition, 1998. Google books: https://books.google.nl/books?id=uSReBAAAQBAJ.

[EB01] A. Engel and C. van den Broeck. *The Statistical Mechanics of Learning*. Cambridge University Press, Cambridge, UK, 2001. Google books: https://books.google.nl/books/about/Statistical_Mechanics_of_Learning.html?id=qVo4IT9ByfQC.

[Efr83] B. Efron. Estimating the error rate of a prediction rule: Improvement on cross-validation. *Journal of the American Statistical Association*, 78(382):316–331, 1983. Online: https://doi.org/10.1080/01621459.1983.10477973.

[EHT20] M. Espadoto, N.S.T. Hirata, and A.C. Telea. Deep learning multidimensional projections. *Information Visualization*, 19(3):247–269, 2020. Preprint: https://arxiv.org/abs/1902.07958.

[ER97] B. Efron and R.Tibshirani. Improvements on cross-validation: The .632+ bootstrap method. *Journal of the American Statistical Association*, 92(438):548–560, 1997. Online: https://sites.stat.washington.edu/courses/stat527/s13/readings/EfronTibshirani_JASA_1997.pdf.

[EYG92] S. Eger, P. Youssef, and I. Gurevych. Is it Time to Swish? Comparing Deep Learning Activation Functions Across NLP Tasks. In *Proceedings of the 2018 Conference on Empirical Methods in Natural Language Processing*, pages 4415–4424, Brussels, Belgium, 1992. Association for Computational Linguistics. Online: https://www.aclweb.org/anthology/D18-1472.

[Faw06] T. Fawcett. An introduction to ROC analysis. *Pattern Recognition Letters*, 27:861–874, 2006. Online: https://www.researchgate.net/publication/222511520_Introduction_to_ROC_analysis.

[FCC98] T. Friess, N. Cristianini, and C. Campbell. The Kernel-AdaTron Algorithm: a Fast and Simple Learning Procedure for Support Vector Machines. In *Machine Learning: Proc. 15th Intl. Conf. (ICML)*. Morgan Kaufmann, San Francisco, CA, 1998. Online: https://citeseerx.ist.psu.edu/viewdoc/download?doi=10.1.1.42.2060.

[Fis36] R. Fisher. The use of multiple measurements in taxonomic problems. *Annual Eugenics*, 7:179–188, 1936. Online: https://onlinelibrary.wiley.com/doi/pdf/10.1111/j.1469-1809.1936.tb02137.x.

[FL90] S. Fahlman and C. Lebiere. The cascade-correlation learning architecture. In D. Touretzky, editor, *Advances in Neural Information Processing Systems*, volume 2, pages 524–532. Morgan-Kaufmann, 1990. Online: https://proceedings.neurips.cc/paper/1989/file/69adc1e107f7f7d035d7baf04342e1ca-Paper.pdf.

[Fle00] R. Fletcher. *Practical Methods of Optimization (2nd Edition)*. Wiley, 2000. Online: https://onlinelibrary.wiley.com/doi/book/10.1002/9781118723203.

[FP96] J. C. Fort and G. Pages. Convergence of stochastic algorithms: from the Kushner & Clark theorem to the Lyapounov functional. *Advances in applied probability*, 28:1072–1094, 1996. Online: https://www.jstor.org/stable/1428165.

[Fre90] M. Frean. The upstart algorithm: A method for constructing and training feedforward neural networks. *Neural Computation*, 2(2):198–209, 1990. Online: https://homepages.ecs.vuw.ac.nz/foswiki/pub/Users/Marcus/MarcusFreanPublications/Frean90-Upstart-Algorithm.pdf.

[Fri94] J.H. Friedman. An Overview of Predictive Learning and Function Approximation. In *From Statistics to Neural Networks*, volume 136 of *NATO ASI Series F: Computer Systems Sciences*, pages 1–61. Springer, Berlin, Heidelberg, 1994. Online: https://purl.stanford.edu/vz166tw6964.

[Fuk80] K. Fukushima. Neocognitron: A self-organizing neural network model for a mechanism of pattern recognition unaffected by shift in position. *Biol. Cybernetics*, 36:193–202, 1980. Online: https://doi.org/10.1007/BF00344251.

[Fuk88] K. Fukushima. Neocognitron: A hierarchical neural network capable of visual pattern recognition. *Neural Networks*, 1(2):119–130, 1988. Online: https://doi.org/10.1016/0893-6080(88)90014-7.

[Fuk19] K. Fukushima. Recent advances in the deep CNN neocognitron. *Nonlinear Theory and Its Applications, IEICE*, 10(4):304–321, 2019. Online: https://doi.org/10.1587/nolta.10.304.

[FV10] B. Frénay and M. Verleysen. Using SVMs with randomized feature spaces: an extreme learning approach. In M. Verleysen, editor, *Proc. of the European Symosium on Artifical Neural Networks (ESANN 2010)*, pages 315–320. d-side, 2010. Online: https://perso.uclouvain.be/michel.verleysen/papers/esann10bf.pdf.

[Gal90] S.I. Gallant. Perceptron-based learning algorithms. *IEEE Transactions on Neural Networks*, 1(2):179–191, 1990. Online: https://www.ling.upenn.edu/courses/Fall_2007/cogs501/Gallant1990.pdf.

[GBB11] X. Glorot, A. Bordes, and Y. Bengio. Deep sparse rectifier neural networks. In G. Gordon, D. Dunson, and M. Dudáŋk, editors, *Proceedings of the Fourteenth International Conference on Artificial Intelligence and Statistics*, volume 15 of *Proceedings of Machine Learning Research*, pages 315–323, Fort Lauderdale, FL, USA, 2011. JMLR Workshop and Conference Proceedings. Online: https://proceedings.mlr.press/v15/glorot11a/glorot11a.pdf.

[GBB+23] S. Ghosh, E.S. Baranowski, M. Biehl, W. Arlt, P. Tino, and K. Bunte. Interpretable models capable of handling systematic missingness in imbalanced classes and heterogeneous datasets. 2023. In preparation.

[GBC16] I.J. Goodfellow, Y. Bengio, and A. Courville. *Deep Learning*. MIT Press, Cambridge, MA, USA, 2016. Online: https://www.deeplearningbook.org.

[GBV+17] S. Ghosh, E.S. Baranowski, R. van Veen, G.-J. de Vries, M. Biehl, W. Arlt, P. Tino, and K. Bunte. Comparison of strategies to learn from imbalanced classes for computer aided diagnosis of inborn steroidogenic disorders. In M. Verleysen, editor, *25th European Symposium on Artificial Neural Networks, Computational Intelligence and Machine Learning, ESANN 2017*, pages 199–205. i6doc.com, 2017. Online: https://www.esann.org/sites/default/files/proceedings/legacy/es2017-94.pdf.

[GE03] I. Guyon and A. Elisseeff. An introduction to variable and feature selection. *J. Mach. Learn. Res.*, 3:1157–1182, 2003. Online: https://www.jmlr.org/papers/volume3/guyon03a/guyon03a.pdf.

[GG91] M. Griniasty and H. Gutfreund. Learning and retrieval in attractor neural networks above saturation. *J. of Phys. A: Math. Gen.*, 24:715–734, 1991. Online: https://www.sciencedirect.com/science/article/abs/pii/0378437195001827.

[GG16] Y. Gal and Z. Ghahramani. Dropout as a bayesian approximation: Representing model uncertainty in deep learning. ICML'16, pages 1050–1059. JMLR.org, 2016. Online: https://proceedings.mlr.press/v48/gal16.pdf.

[Gho21] S. Ghosh. *Intrinsically Interpretable Machine Learning In Computer Aided Diagnosis*. PhD thesis, University of Groningen, 2021. Online: https://doi.org/10.33612/diss.175627883.

[GLSGFV10] P.J. Garcia-Laencina, JL. Sancho-Gomez, and A.R. Figueiras-Vidal. Pattern classification with missing data: a review. *Neural Comput & Applic*, pages 263–282, 2010. Online: https://doi.org/10.1007/s00521-009-0295-6.

[GM90] M. Golea and M. Marchand. A growth algorithm for neural network decision trees. *Europhysics Letters*, 12(3):205–210, 1990. Online: https://www.researchgate.net/publication/231136885.

[GP90] F. Girosi and T. Poggio. Networks and the best approximation property. *Biol. Cybernetics*, 63:169–176, 1990. Online: http://cbcl.mit.edu/people/poggio/journals/girosi-poggio-BiolCybernetics-1990.pdf.

[GT18] A.N. Gorban and I.Y. Tyukin. Blessing of dimensionality: mathematical foundations of the statistical physics of data. *Phil. Trans. R. Soc. A*, 376(20170237), 2018. Online: https://doi.org/10.1098/rsta.2017.0237.

[Gue97] K. Guerney. *An Introduction to Neural Networks*. UCL Press, London, 1997. Online: http://www.macs.hw.ac.uk/~yjc32/project/ref-NN/Gurney_et_al.pdf.

[Guy16] I. Guyon. Data mining history: The invention of Support Vector Machines, 2016. Online: https://www.kdnuggets.com/2016/07/guyon-data-mining-history-svm-support-vector-machines.html.

[GW10] O. Golubitsky and S. Watt. Distance-based classification of handwritten symbols. *Int. J. on Document Analysis and Recognition (IJDAR)*, 13:133–146, 2010. Preprint: https://www.csd.uwo.ca/~watt/pub/reprints/2009-ijdar-similarity.pdf.

[gwe09] gwern.net. The Neural Net Tank Urban Legend, 2009. Online: https://www.gwern.net/Tanks.

[Har68] P. Hart. The condensed nearest neighbor rule. *IEEE Trans. Information Theory*, 14:515–516, 1968. Online: https://sci2s.ugr.es/keel/pdf/algorithm/articulo/hart1968.pdf.

[Hay09] S. Haykin. *Neural Networks and Learning Machines*. Pearson Education, Upper Saddle River, NJ, USA, third edition, 2009. Online: https://cours.etsmtl.ca/sys843/REFS/Books/ebook_Haykin09.pdf.

[HB87] S.J. Hanson and David J. Burr. Minkowski-r Back-Propagation: Learning in Connectionist Models with Non-Euclidian Error Signals. In *Proc. Neural Information Processing Systems 1987*, pages 348–357, Cambridge, MA, USA, 1987. MIT Press. Online: https://proceedings.neurips.cc/paper/1987/file/fc490ca45c00b1249bbe3554a4fdf6fb-Paper.pdf.

[Heb49] D.O. Hebb. *The Organization of Behavior*. Erlbaum, 1949. Reprinted 2002, 335 pages. Online: https://pure.mpg.de/rest/items/item_2346268_3/component/file_2346267/content.

[Her02] R. Herbrich. *Learning Kernel Classifiers, Theory and Algorithms*. MIT Press, Cambridge, MA, 2002. Online: https://mitpress.mit.edu/9780262546591/learning-kernel-classifiers.

[HG09] H. He and E.A. Garcia. Learning from imbalanced data. *IEEE Trans. on Knowledge and Data Engineering*, 21(9):1263–1284, 2009. Online: https://ieeexplore.ieee.org/document/5128907,.

[Hin86] G.E. Hinton. Learning Distributed Representations of Concepts. *Proc. of the Cognitive Science Society*, pages 1–12, 1986. Online: https://citeseerx.ist.psu.edu/viewdoc/download?doi=10.1.1.408.7684.

[HK98] H. Horner and R. Kühn. Neural Networks. In U. Ratsch, M. Richter, and I.O. Stametescu, editors, *Intelligence and Artificial Intelligence, an Interdisciplinary Debate*, pages 125–161. Springer, Heidelberg, Germany, 1998. Online: https://link.springer.com/chapter/10.1007/978-3-662-03667-9_8.

[HKK+10] M.E. Houle, H.-P. Kriegel, P. Kröger, E. Schubert, and A. Zimek. Can shared-neighbor distances defeat the curse of dimensionality?

In M. Gertz and B. Ludäscher, editors, *Scientific and Statistical Database Management*, pages 482–500, Berlin, Heidelberg, 2010. Springer Berlin Heidelberg. Preprint: https://imada.sdu.dk/~zimek/publications/SSDBM2010/SNN-SSDBM2010-preprint.pdf.

[HKP91] J.A. Hertz, A.S. Krogh, and R.G. Palmer. *Introduction To The Theory Of Neural Computation*. Addison-Wesley, Reading, MA, USA, 1991. Online: https://www.taylorfrancis.com/books/mono/10.1201/9780429499661.

[HO00] A. Hyvärinen and E. Oja. Independent component analysis: algorithms and applications. *Neural Networks*, 13(4):411–430, 2000. Online: http://www.cse.msu.edu/~cse902/S03/icasurvey.pdf.

[Hop82] J J Hopfield. Neural networks and physical systems with emergent collective computational abilities. *Proceedings of the National Academy of Sciences*, 79(8):2554–2558, 1982. Online: https://www.pnas.org/doi/abs/10.1073/pnas.79.8.2554.

[Hop87] J.J. Hopfield. Learning algorithms and probability distributions in feed-forward and feed-back networks. *Proc. of the National Academy of Sciences*, 84:8429–8433, 1987. Online: https://www.ncbi.nlm.nih.gov/pmc/articles/PMC299557/pdf/pnas00338-0264.pdf.

[Hor89] K. Hornik. Multilayer Feedforward Networks are Universal Approximators. *Neural Networks*, 2:359–366, 1989. Online: https://www.cs.cmu.edu/~epxing/Class/10715/reading/Kornick_et_al.pdf.

[HR03] G. Hinton and S. Roweis. Stochastic neighbor embedding. In S. Becker, S. Thrun, and K. Obermayer, editors, *Advances in Neural Information Processing Systems*, volume 15, pages 833–840. MIT Press, 2003. Online: https://www.cs.toronto.edu/~hinton/absps/sne.pdf.

[HR04] A. Herschtal and B. Raskutti. Optimising area under the ROC curve using gradient descent. In *Proc. of the 21st International Conference on Machine Learning*, page 8, 2004. Online: https://icml.cc/Conferences/2004/proceedings/papers/132.pdf.

[HRM+60] J.C. Hay, F. Rosenblatt, A.E. Murray, A. Stieber, and R.A Wolf. *Mark I Perceptron Operator's Manual, Report No. VG-1196-G-5*. Cornell Aeronautical Laboratory Inc., Buffalo, NY, USA, 1960. Online: https://apps.dtic.mil/sti/pdfs/AD0236965.pdf.

[HSK+12] G.E. Hinton, N. Srivastava, A. Krizhevsky, I. Sutskever, and R.R. Salakhutdinov. Improving neural networks by preventing co-adaptation of feature detectors. *CoRR*, 2012. Preprint: https://arxiv.org/abs/1207.0580.

[HSV05] B. Hammer, M. Strickert, and T. Villmann. Supervised neural gas with general similarity measure. *Neural Processing Letters*, 21(1):21–44, 2005. Online: https://www.researchgate.net/publication/2873240.

[HSW93] B. Hassibi, D.G. Stork, and G.J. Wolff. Optimal Brain Surgeon and general network pruning. In *IEEE International Conference on Neural Networks*, volume 1, pages 293–299, 1993. Online: https://doi.org/10.1109/ICNN.1993.298572.

[HT01] D.J. Hand and R.J. Till. A simple generalisation of the area under the ROC curve for multiple class classification problems. *Machine Learning*, 45(2):171–186, 2001. Online: https://link.springer.com/article/10.1023/A:1010920819831.

[HTF01] T. Hastie, R. Tibshirani, and J. Friedman. *The Elements of Statistical Learning*. Springer Series in Statistics. Springer, New York, NY, USA, 2001. Online: https://hastie.su.domains/Papers/ESLII.pdf.

[Hub29] E. Hubble. A relation between distance and radial velocity among extra-galactic nebulae. *Proceedings of the National Academy of Sciences (PNAS)*, 15(3):168–173, 1929. Online: https://www.pnas.org/content/15/3/168.

[Huc18] J. Huchra. home page, 2018. Online: https://www.cfa.harvard.edu/~dfabricant/huchra.

[Hue19] M. Huertas-Company. Deep Learning, 2019. In: [MSK19]. Online: http://research.iac.es/winterschool/2018/media/summaries/iac_winter_syllabus_MHC.pdf.

[HV02] B. Hammer and T. Villmann. Generalized Relevance Learning Vector Quantization. *Neural Networks*, 15(8-9):1059–1068, 2002. Online: https://www.sciencedirect.com/science/article/abs/pii/S0893608002000795.

[HV05] B. Hammer and T. Villmann. Classification using non-standard metrics. In M. Verleysen, editor, *Proc. Europ. Symp. on Artificial Neural Networks (ESANN)*, pages 303–316. d-side publishing, 2005. Online: https://www.researchgate.net/publication/221165820.

[HW59] D.H. Hubel and T.N. Wiesel. Receptive fields of single neurons in the cat's striate cortex. *J. Physiol.*, 148(3):574–591, 1959. Online: https://doi.org/10.1113/jphysiol.1959.sp006308.

[HZRS15] K. He, X. Zhang, S. Ren, and J. Sun. Delving deep into rectifiers: Surpassing human-level performance on imagenet classification. In *2015 IEEE International Conference on Computer Vision (ICCV)*, pages 1026–1034, 2015. Preprint: https://arxiv.org/abs/1502.01852.

[HZS06] G.-B. Huang, Q.-Y. Zhu, and C.-K. Siew. Extreme learning machine: Theory and applications. *Neurocomputing*, 70(1):489–501, 2006. Online: https://www.sciencedirect.com/science/article/abs/pii/S0925231206000385.

[JBB15] A. Jovic, K. Brkic, and N. Bogunovic. A review of feature selection methods with applications. In *38th International Convention on Information and Communication Technology, Electronics and Microelectronics (MIPRO)*, pages 1200–1205, 2015. Online: https://doi.org/10.1109/MIPRO.2015.7160458.

[Kat66] B. Katz. *Nerve, Muscle, and Synapse*. McGraw-Hill, New York, 1966.

[KEP17] T. Kautz, B.M. Eskofier, and C.F. Pasluosta. Generic performance measure for multiclass classifiers. *Pattern Recognition*, 68:1125, 2017. Online: https://www.researchgate.net/publication/314272452_Generic_Performance_Measure_for_Multiclass-Classifiers.

[KHV14] M. Kaden, W. Hermann, and T. Villmann. Optimization of general statistical accuracy measures for classification based on learning vector quantization. In M. Verleysen, editor, *Proc. of the European Symposium on Artificial Neural Networks ESANN 2014*, pages 47–52. i6doc.com, 2014. Online: https://citeseerx.ist.psu.edu/viewdoc/download?doi=10.1.1.431.5465.

[Kik76] R. Kikuchi. Natural iteration method and boundary free energy. *J. Chemical Physics*, 65:4545, 1976. Online: https://doi.org/10.1063/1.432909.

[KLS20] D. Kobak, J. Lomond, and B. Sanchez. The optimal ridge penalty for real-world high-dimensional data can be zero or negative due to the implicit ridge regularization. *J. Machine Learning Research*, 21:1–16, 2020. Online: https://jmlr.org/papers/volume21/19-844/19-844.pdf.

[KM87] W. Krauth and M. Mezard. Learning algorithms with optimal stability in neural networks. *J. Phys. A: Math. Gen.*, 20(11):L745–L752, 1987. Online: https://iopscience.iop.org/article/10.1088/0305-4470/20/11/013.

[Kob97] S. Kobe. Ernst Ising - Physicist and Teacher. *J. Stat. Phys.*, 88:991–995, 1997. Online: https://link.springer.com/article/10.1023/B:JOSS.0000015184.19421.03.

[Koc98] C. Koch. *Biophysics of Computation: Information Processing in Single Neurons*. Oxford University Press, New York, NY, USA, 1998. Selected chapters: https://christofkoch.com/biophysics-book/.

[Koh90] T. Kohonen. Improved versions of Learning Vector Quantization. *In Proc. of the International Joint conference on Neural Networks (San Diego, 1990)*, 1:545–550, 1990. Online: https://ieeexplore.ieee.org/document/5726582.

[Koh95] T. Kohonen. Learning Vector Quantization. In M.A. Arbib, editor, *The Handbook of Brain Theory and Neural Networks.*, pages 537–540. MIT Press, Cambridge, MA, 1995. Online: https://mitpress.mit.edu/9780262511025/the-handbook-of-brain-theory-and-neural-networks/.

[Koh97] T. Kohonen. *Self-Organizing Maps*. Springer, Berlin, Germany, 1997. Online: https://link.springer.com/book/10.1007/978-3-642-56927-2.

[KSV88] A.H. Kramer and A. Sangiovanni-Vincentelli. Efficient parallel learning algorithms for neural networks. In *Proceedings of the 1st International Conference on Neural Information Processing Systems*, NIPS'88, pages 40–48, Cambridge, MA, USA, 1988. MIT Press. Online: https://papers.nips.cc/paper/1988/hash/02522a2b2726fb0a03bb19f2d8d9524d-Abstract.html.

[KY97] T.-Y. Kwok and D.-Y. Yeung. Constructive algorithms for structure learning in feedforward neural networks for regression problems. *Neural Networks, IEEE Transactions on*, 8:630 – 645, 06 1997. Online: https://repository.ust.hk/ir/Record/1783.1-52.

[LB89] B.E. Lautrup and S. Brunak. *Neural Networks: Computers with Intuition*. World Scientific, 1989. Online: https://www.worldscientific.com/worldscibooks/10.1142/0878.

[LBD+89] Y. LeCun, B. Boser, J. S. Denker, D. Henderson, R. E. Howard, W. Hubbard, and L. D. Jackel. Backpropagation Applied to Handwritten Zip Code Recognition. *Neural Computation*, 1(4):541–551, 12 1989. Online: https://doi.org/10.1162/neco.1989.1.4.541.

[LBH18] Y. LeCun, Y. Bengio, and G. Hinton. Deep Learning. *Nature*, 521(7553):436–444, 2018. Online: https://www.nature.com/articles/nature14539.

[LBV17] M. LeKander, M. Biehl, and H. de Vries. Empirical Evaluation of Gradient Methods for Matrix Learning Vector Quantization. In *Proc. 12th Intl. Workshop on Self-Organizing Maps, Learning Vector Quantization and Visualization (WSOM+), Nancy/France*. IEEE Xplore, 2017. 8 pages. Online: https://ieeexplore.ieee.org/document/8020027.

[LDS90] Y. LeCun, J. Denker, and S.A. Solla. Optimal Brain Damage. In D. Touretzky, editor, *Advances in Neural Information Processing Systems*, volume 2, pages 598–605. Morgan-Kaufmann, 1990. Online: https://proceedings.neurips.cc/paper/1989/file/6c9882bbac1c7093bd25041881277658-Paper.pdf.

[LG19] M. Lange-Geisler. *Hebbian learning approaches based on general inner products and distance measures in non-Euclidean spaces*. PhD thesis, University of Groningen, 2019. Online: https://pure.rug.nl/ws/portalfiles/portal/77221420/Complete_thesis.pdf.

[LHC06] P. Li, T.J. Hastie, and K.W. Church. Very sparse random projections. In *Proceedings of the 12th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, KDD '06, pages 287–296, New York, NY, USA, 2006. Association for Computing Machinery. Online: https://doi.org/10.1145/1150402.1150436.

[Lic13] M. Lichman. UCI Machine Learning Repository, 2013. Online: https://archive.ics.uci.edu/ml,.

[Lit74] W.A. Little. The existence of persistent states in the brain. *Mathematical Biosciences*, 19(1):101–120, 1974. Online: https://www.sciencedirect.com/science/article/pii/0025556474900315.

[Lit88] R.J.A. Little. A test of missing completely at random for multivariate data with missing values. *Journal of the American Statistical Association*, 83(404):1198–1202, 1988. Online: https://doi.org/10.1080/01621459.1988.10478722.

[LJ09] M. Lukoševičius and H. Jaeger. Reservoir computing approaches to recurrent neural network training. *Computer Science Review*, 3(3):127–149, 2009. Online: https://doi.org/10.1016/j.cosrev.2009.03.005.

[Llo82] S.P. Lloyd. Least square quantization in PCM. *IEEE Transactions on Information Theory*, 28:29–137, 1982. First published in Bell Telephone Laboratories Paper 1957. Online: https://cs.nyu.edu/~roweis/csc2515-2006/readings/lloyd57.pdf.

[LLPS93] M. Leshno, V.Y. Lin, A. Pinkus, and S. Schocken. Multilayer Feedforward Networks With a Nonpolynomial Activation Function Can Approximate Any Function. *Neural Networks*, 6:861–867, 1993. Online: https://www.sciencedirect.com/science/article/abs/pii/S0893608005801315.

[LR02] R.J.A. Little and D.B. Rubin. *Statistical Analysis with Missing Data*. Wiley Series in Probability and Statistics. John Wiley & Sons, 2 edition, 2002. Online: https://doi.org/10.1002/9781119013563.

[LV07] J.A. Lee and M. Verleysen. *Nonlinear Dimensionality Reduction*. Springer, 2007. Online: https://link.springer.com/book/10.1007/978-0-387-39351-3.

[LVM+20] M. Loog, T. Viering, A. Mey, J.H. Kreijthe, and D.M.J. Tax. A brief prehistory of double descent. *Proc. Natl. Acad. Sci. USA*, 117(20):10625–10626, 2020. Preprint: https://arxiv.org/abs/2004.04328.

[Mah36] P. Mahalanobis. On the generalised distance in statistics. *Proc. of the National Inst. of Sciences of India*, 2:49–55, 1936. Online: http://bayes.acs.unt.edu:8083/BayesContent/class/Jon/MiscDocs/1936_Mahalanobis.pdf.

[Man69] O.L. Mangasarian. *Nonlinear Programming*. McGraw-Hill, New York, 1969. Online: https://epubs.siam.org/doi/book/10.1137/1.9781611971255.

[Mar18] G. Marcus. Deep learning: A critical appraisal. *arXiv e-prints*, 1801.00631, 2018. Preprint: https://arxiv.org/abs/1801.00631.

[MBS93] T. Martinetz, S. Berkovich, and K. Schulten. Neural Gas Network for Vector Quantization and its Applications to Time-Series Prediction. *IEEE Trans. on Neural Networks*, 4:558–569, 1993. Online: https://ieeexplore.ieee.org/document/238311.

[MD89a] G.J. Mitchison and R.M. Durbin. Bounds on the learning capacity of some multi-layer networks. *Biological Cybernetics*, 60:345–356, 1989. Online: https://link.springer.com/article/10.1007/BF00204772.

[MD89b] J.E. Moody and C.J. Darken. Fast learning in networks of locally-tuned processing units. *Neural Computation*, 1(2):281–294, 1989. Online: https://citeseerx.ist.psu.edu/viewdoc/download?doi=10.1.1.830.1813.

[MD09] R. Mabry and P. Deiermann. Of Cheese and Crust: A Proof of the Pizza Conjecture and Other Tasty Results. *The American Mathematical Monthly*, 116(5):423–438, 2009. Online: https://www.cs.umd.edu/~gasarch/BLOGPAPERS/pizza.pdf.

[Mea92] A. Mead. Review of the development of multidimensional scaling methods. *Journal of the Royal Statistical Society, Series D (The Statistician)*, 41(1):27–39, 1992. Online: https://www.jstor.org/stable/2348634.

[Mer09] J. Mercer. Functions of positive and negative type, and their connection with the theory of integral equations. *Philosophical Transactions of the Royal Society of London. Series A, Containing Papers of a Mathematical or Physical Character*, 209(441-458):415–446, 1909. Online: https://royalsocietypublishing.org/doi/abs/10.1098/rsta.1909.0016.

[MH08] L. van der Maaten and G Hinton. Visualizing data using t-sne. *Journal of Machine Learning Research*, 9:2579–2605, 2008. Online: https://www.jmlr.org/papers/v9/vandermaaten08a.html.

[MHM20] L. McInnes, J. Healy, and J. Melville. UMAP: Uniform manifold approximation and projection for dimension reduction. *Journal of Open Source Software*, 3(29):861, 2020. Preprint: https://arxiv.org/abs/1802.03426.

[MM92] H.N. Mhaskar and C.A. Micchelli. Approximation by Superposition of Sigmoidal and Radial Basis Functions. *Adv. in Appl. Mathematics*, 13:350–373, 1992. Online: https://www.sciencedirect.com/science/article/pii/019688589290016P.

[MN89] M. Mezard and J.-P. Nadal. Learning in feedforward layered networks: the tiling algorithm. *J. of Physics A: Math. Gen.*, 22(12):2191–2203, 1989. Online: https://www.researchgate.net/publication/230980679.

[MP29] R. von Mises and H. Pollaczek-Geiringer. Praktische Verfahren der Gleichungsauflösung. *ZAMM - Zeitschrift für Angewandte Mathematik und Mechanik*, 9:152–164, 1929. Online: https://onlinelibrary.wiley.com/doi/abs/10.1002/zamm.19290090206.

[MP69] M. Minsky and S. Papert. *Perceptrons: An Introduction to Computational Geometry*. MIT Press, Cambridge, MA, USA, 1969. Online: https://direct.mit.edu/books/book/3132/PerceptronsAn-Introduction-to-Computational.

[MPCB14] G. Montufar, R. Pascanu, K. Cho, and Y. Bengio. On the Number of Linear Regions of Deep Neural Networks. In Z. Ghahramani, M. Welling, C. Cortes, N.D. Lawrence, and K.Q. Weinberger, editors, *Adv. in Neural Information Processing Systems. Proc. NIPS (2014)*, volume 27, pages 2924–2932. Curran Associates Inc., 2014. Preprint: https://arxiv.org/abs/1402.1869.

[MPH09] L. van der Maaten, E. Postma, and J. van den Herik. Dimensionality reduction: a comparative review. page 13, 2009. Online: https://lvdmaaten.github.io/publications/papers/TR_Dimensionality_Reduction_Review_2009.pdf.

[MSK19] A. Monreal Ibero, J. Sánchez Almeida, and J. Knapen, editors. *Canary Islands Winter School of Astrophysics: Big Data Analysis in Astronomy*. Instituo de Astrofisica de Canarias, Tenerife/Spain, 2019. E-book: http://research.iac.es/winterschool/2018/pages/book-ws2018.php.

[MSS+11] E. Mwebaze, P. Schneider, F.-M. Schleif, J. R. Aduwo, J. A. Quinn, S. Haase, T. Villmann, and M. Biehl. Divergence based classification and Learning Vector Quantization. *Neurocomputing*, 74:1429–1435, 2011. Online: https://pure.rug.nl/ws/files/2489124/2011NeurocompMwebaze.pdf.

[MSV89] S. Makram-Ebeid, J.-A. Sirat, and J.-R. Viala. A Rationalized Back-Propagation Learning Algorithm. In *International Joint Conference on Neural Networks*, volume 2, pages 373–380. IEEE, New York, 1989. Online: https://ieeexplore.ieee.org/document/118725.

[Mur22] K.M. Murphy. *Probabilistic Machine Learning: An Introduction*. MIT Press, 2022. Online: https://probml.github.io/pml-book/book1.html.

[Mwe14] E. Mwebaze. *Divergences for prototype-based classification and causal structure discovery: Theory and application to natural datasets*. PhD thesis, University of Groningen, 2014. Online: https://hdl.handle.net/11370/7c4e0ebf-76cf-4f64-aeec-7096aa919559.

[MZ95] R. Monasson and R. Zecchina. Learning and generalization theories of large committee-machines. *Modern Physics Letters B*, 9(30):887–897, 1995. Preprint: https://www.phys.ens.fr/~monasson/Articles/a15.pdf.

[ND91] D. Nabutovsky and E. Domany. Learning the unlearnable. *Neural Computation*, 3(4):604–616, 1991. Online: https://doi.org/10.1162/neco.1991.3.4.604.

[NE14] D. Nova and P.A. Estévez. A review of Learning Vector Quantization classifiers. *Neural Computing and Applications*, 25(3-4):511–524, 2014. Preprint: https://arxiv.org/abs/1509.07093.

[Neu02] Neural Networks Research Centre, Helsinki. Bibliography on the Self-Organizing Maps (SOM) and Learning Vector Quantization (LVQ). *Otaniemi: Helsinki Univ. of Technology*, 2002. Repository: https://liinwww.ira.uka.de/bibliography/Neural/{SOM.LVQ}.html.

[NH92a] S. Nowlan and G.E. Hinton. Adaptive soft weight tying using gaussian mixtures. In J. Moody, S. Hanson, and R. P. Lippmann, editors, *Advances in Neural Information Processing Systems*, volume 4, pages 993–1000. Morgan-Kaufmann, 1992. Online: https://proceedings.neurips.cc/paper/1991/file/05f971b5ec196b8c65b75d2ef8267331-Paper.pdf.

[NH92b] S.J. Nowlan and G.E. Hinton. Simplifying neural networks by soft weight-sharing. *Neural Computation*, 4(4):473–493, 1992. Online: https://doi.org/10.1162/neco.1992.4.4.473.

[NH10] V. Nair and G.E. Hinton. Rectified linear units improve restricted boltzmann machines. In *Proc. of ICML 2010*, pages 807–814, Madison, WI, USA, 2010. Omnipress. Preprint: https://www.cs.toronto.edu/~fritz/absps/reluICML.pdf.

[Nik17] E. Nikolaychuk. Dogs, wolves, data science, and why machines must learn like humans do, 2017. Online: https://hackernoon.com/dogs-wolves-data-science-and-why-machines-must-learn-like-humans-do-41c43bc7f982.

[NMB+18] B. Neal, S. Mittal, A. Baratin, V. Tantia, M. Scicluna, S. Lacoste-Julien, and I. Migliagkas. A modern take on the bias-variance tradeoff in neural networks. *arxiv e-prints*, 1810.08591, 2018. Preprint: https://arxiv.org/abs/1810.08591.

[NWB18] A. Nolte, L. Wang, and M. Biehl. Prototype-based analysis of GAMA galaxy catalogue data. In M. Verleysen, editor, *26th European Symposium on Artificial Neural Networks, Computational Intelligence and Machine Learning, ESANN 2018*, pages 339–344. i6doc.com, 2018. Online: https://pure.rug.nl/ws/files/58483665/ESANN2018_115.pdf.

[NWB+19] A. Nolte, L. Wang, M. Bilicki, B. Holwerda, and M. Biehl. Galaxy classification: A machine learning analysis of GAMA catalogue data. *Neurocomputing*, 342:172–190, 2019. Preprint: https://arxiv.org/abs/1903.07749.

[OB21] J. Opitz and S. Burst. Macro F1 and Macro F1. *arXiv eprints*, 1911.03347, 2021. Online: https://arxiv.org/abs/1911.03347.

[OH91] M. Opper and D. Haussler. Generalization performance of Bayes optimal classification algorithm for learning a perceptron. *Phys. Rev. Lett.*, 66:2677–2680, 1991. Online: https://journals.aps.org/prl/abstract/10.1103/PhysRevLett.66.2677.

[Oja82] E. Oja. Simplified neuron model as a principal component analyzer. *J. Math. Biology*, 15:267–273, 1982. Online: https://doi.org/10.1007/BF00275687.

[Oja89] E. Oja. Neural Networks, Principal Components, and Subspaces. *International Journal of Neural Systems*, 1(1):61–68, 1989. Online: https://www.worldscientific.com/doi/abs/10.1142/S0129065789000475.

[OKKN90] M. Opper, W. Kinzel, J. Kleinz, and R. Nehl. On the ability of the optimal perceptron to generalise. *J. of Physics A: Math. and Gen.*, 23(11):L581–L586, 1990. Online: https://www.semanticscholar.org/paper/On-the-ability-of-the-optimal-perceptron-to-Opper-Kinzel/b2e7dc93f6827606e153219b5767d74ecbc5bded.

[Ola96] M. Olazaran. A sociological study of the official history of the perceptron controversy. *Social Studies of Science*, 26:611–659, 1996. Online: https://journals.sagepub.com/doi/10.1177/030631296026003005.

[OLLB20] J. Ott, E. Linstead, N. LaHaye, and P. Baldi. Learning in the machine: To share or not to share? *Neural Networks*, 126:235–249, 2020. Online: https://doi.org/10.1016/j.neunet.2020.03.016.

[OM99] D. Opitz and R. Maclin. Popular ensemble methods: an empirical study. *J. of Artificial Intelligence Research*, 11:169–198, 1999. Preprint: https://arxiv.org/abs/1106.0257.

[Opp90] M. Opper, 1990. private communication.

[Opp94] M. Opper. Learning and generalization in a two-layer neural network: The role of the Vapnik-Chervonvenkis dimension. *Phys. Rev. Lett.*, 72:2113–16, 1994. Online: https://journals.aps.org/prl/abstract/10.1103/PhysRevLett.72.2113.

[OSB20] E. Oostwal, M. Straat, and M. Biehl. Hidden unit specialization in layered neural networks: ReLU vs. sigmoidal activation. *Physica A: Statistical Mechanics and its Applications*, 564:125517, 2020. Preprint: https://arxiv.org/abs/1910.07476.

[PAH19] B. Paassen, A. Artelt, and B. Hammer. *Lecture Notes on Applied Optimization*. Bielefeld University, Germany, 2019. Online: https://pub.uni-bielefeld.de/record/2935200.

[PBB11] G. Papari, K. Bunte, and M. Biehl. Waypoint averaging and step size control in learning by gradient descent. In F.-M. Schleif and T. Villmann, editors, *Proc. Mittweida Workshop on Computational Intelligence MIWOCI 2011, Machine Learning Reports MLR-2011-06*, pages 16–26. Bielefeld University, Germany, 2011. Online: https://www.techfak.uni-bielefeld.de/~fschleif/mlr/mlr_06_2011.pdf#page=17.

[PBG+94] A. Priel, M. Blatt, T. Grossmann, E. Domany, and I. Kanter. Computational capabilities of restricted two-layered perceptrons. *Phys. Rev. E*, 50:577–595, Jul 1994. Online: https://link.aps.org/doi/10.1103/PhysRevE.50.577.

[PHL96] M. Pedersen, L. Hansen, and J. Larsen. Pruning with generalization based weight saliencies: γOBD, γOBS. In D. Touretzky, M. C. Mozer, and M. Hasselmo, editors, *Advances in Neural Information Processing Systems*, volume 8, pages 521–527. MIT Press, 1996. Online: https://citeseerx.ist.psu.edu/viewdoc/download?doi=10.1.1.29.3425.

[PJ92] B.T. Polyak and A.B. Juditsky. Acceleration of stochastic approximation by averaging. *SIAM J. Control Optim.*, 30(4):838–855, 1992. Online: https://www.researchgate.net/publication/236736831.

[PJS17] J. Peters, D. Janzing, and B. Schölkopf. *Elements of Causal Inference: Foundations and Learning Algorithms*. MIT Press, Cambridge, MA, 2017. Online: https://mitpress.mit.edu/books/elements-causal-inference.

[Pla98] J.C. Platt. Sequential minimal optimization: A fast algorithm for training support vector machines. Technical report, MSR-TR-98-14, Microsoft Research, 1998. Online: https://citeseerx.ist.psu.edu/viewdoc/summary?doi=10.1.1.43.4376.

[PMB14] R. Pascanu, G. Montufar, and J. Bengio. On the number of response regions of deep feedforward networks with piecewise linear activations. In *Second International Conference on Learning Representations - ICLR 2014*. ICLR, Banff, 2014. Preprint: https://arxiv.org/abs/1312.6098.

[PNH86] D. Plaut, S. Nowlan, and G. Hinton. Experiments on learning by back propagation. Technical report, CMU-CS-86-126, Dept. of Comp. Science, Carnegie Mellon University, 1986. Online: https://ni.cmu.edu/~plaut/papers/pdf/PlautNowlanHinton86TR.backprop.pdf.

[PP12] K.B. Petersen and M.S. Pedersen. The Matrix Cookbook, 2012. Version 20121115, Online: https://www2.imm.dtu.dk/pubdb/p.php?3274,.

[Pre97] L. Prechelt. Early Stopping – But When? In Montavon, G. and Orr, G.B. and Müller, K.R., editor, *Neural Networks: Tricks of the Trade*, volume 7700 of *Lecture Notes in Computer Science*, pages 53–67. Springer, Heidelberg, 1997. Online: https://link.springer.com/chapter/10.1007/978-3-642-35289-8_5.

[Pre98] L. Prechelt. Automatic early stopping using cross validation: quantifying the criteria. *Neural networks : the official journal of the International Neural Network Society*, 11(4):761–767, 1998. Online: https://www.sciencedirect.com/science/article/abs/pii/S0893608098000100.

[Pre21] K. Pretz. Stop Calling Everything AI, Machine Learning Pioneer Says. *IEEE Spectrum*, March 2021. Online: https://spectrum.ieee.org/stop-calling-everything-ai-machinelearning-pioneer-says.

[QRK+05] R. Q. Quiroga, L. Reddy, G. Kreiman, C. Koch, and I. Fried. Invariant visual representation by single neurons in the human brain. *Nature*, 435:1102–1107, 2005. Online: https://www.researchgate.net/publication/7770938.

[Ras18] S. Raschka. Model evaluation, model selection, and algorithm selection in machine learning. *CoRR*, abs/1811.12808, 2018. Preprint: https://arxiv.org/abs/1811.12808.

[RBS19] I. Reis, D. Baron, and S. Shahaf. Probabilistic Random Forest: a machine learning algorithm for noisy data sets. *The Astronomical Journal*, 157(16), 2019. Online: https://iopscience.iop.org/article/10.3847/1538-3881/aaf101.

[Ree93] R. Reed. Pruning algorithms - a survey. *IEEE Trans. on Neural Networks*, 4:740–747, 1993. Online: https://axon.cs.byu.edu/~martinez/classes/678/Papers/Reed_PruningSurvey.pdf.

[RHW86] D.E. Rumelhart, G.E. Hinton, and R.J. Williams. Learning representations by back-propagating errors. *Nature*, 323:533–536, 1986. Online: https://www.nature.com/articles/323533a0.

[RKSV20] J. Ravichandran, M. Kaden, S. Saralajew, and T. Villmann. Variants of dropconnect in learning vector quantization networks for evaluation of classification stability. *Neurocomputing*, 403:121–132, 2020. Online: https://www.sciencedirect.com/science/article/pii/S0925231220305014.

[RM51] H. Robbins and S. Monro. A stochastic approximation method. *The Ann. of Mathematical Statistics*, 22:405, 1951. Online: http://www.columbia.edu/~ww2040/8100F16/RM51.pdf.

[RM86] D.E. Rumelhart and J.L. McClelland. *Parallel Distributed Processing. Explorations in the Microstructure of Cognition*. MIT Press, Cambridge, MA, 1986. Online: https://ieeexplore.ieee.org/book/6276825.

[RMBJ21] S. Rezaei, J.P. McKean, M. Biehl, and A. Javadpour. DECORAS: detection and characterization of radio-astronomical sources using deep learning. *Monthly Notices of the Royal Astronomical Society*, 510(4):5891–5907, 12 2021. Online: https://academic.oup.com/mnras/article-pdf/510/4/5891/42297317/stab3519.pdf.

[RMS92] H. Ritter, T. Martinetz, and K. Schulten. *Neural Computation and Self-Organizing Maps*. Addison-Wesley, New York, NY, USA, 1992. Google books: https://books.google.nl/books/about/Neural_Computation_and_Self_organizing_M.html?id=x1NjtAEACAAJ.

[Roj96] P. Rojas. *Neural Networks - A Systematic Introduction*. Springer, Berlin, Germany, 1996. Online: https://page.mi.fu-berlin.de/rojas/neural/.

[Roj03] R. Rojas. Networks of width one are universal classifiers. In *Proc. of the International Joint Conference on Neural Networks*, volume 4, pages 3124–3127, 2003. Online: https://ieeexplore.ieee.org/document/1224071.

[Roj17] P. Rojas. Deepest Neural Networks. *arXiv:1707.02617v1*, 2017. 9 pages. Preprint: https://arxiv.org/abs/1707.02617.

[Ros58] F. Rosenblatt. The Perceptron: A Probabilistic Model for Information Storage and Organization in the Brain. *Psychological Review*, pages 65–386, 1958. Online: https://citeseerx.ist.psu.edu/viewdoc/download?doi=10.1.1.335.3398.

[Ros61] F. Rosenblatt. *Principles of Neurodynamics. Perceptrons and the Theory of Brain Mechanisms*. Cornell Aeronautical Laboratory Inc., Buffalo, NY, USA, 1961. Online: https://safari.ethz.ch/digitaltechnik/spring2018/lib/exe/fetch.php?media=neurodynamics1962rosenblatt.pdf.

[Ros16] M. van Rossum. *Neural Computation*. Univ. of Edinburgh, UK, 2016. Online: https://www.inf.ed.ac.uk/teaching/courses/nc/ln_all.pdf.

[RS00] S.T. Roweis and L.K. Saul. Nonlinear dimensionality reduction by local linear embedding. *Science*, 290(5500):2323–2326, 2000. Online: https://www.science.org/doi/10.1126/science.290.5500.2323.

[RSG16] M.T. Ribeiro, S. Singh, and C. Guestrin. "Why Should I Trust You?": Explaining the Predictions of Any Classifier. In *Proc. of the 22nd ACM SIGKDD Intl. Conf. on Knowledge Discovery and Data Mining*, KDD

'16, pages 1135–1144, New York, NY, USA, 2016. Association for Computing Machinery. Online: https://www.kdd.org/kdd2016/papers/files/rfp0573-ribeiroA.pdf.

[Ruj93] P. Ruján. A fast method for calculating the perceptron with maximal stability. *J. de Physique I*, 3(2):277–290, 1993. Online: https://hal.archives-ouvertes.fr/file/index/docid/246719/filename/ajp-jp1v3p277.pdf.

[Ruj97] P. Ruján. Playing billiards in version space. *Neural Computation*, 9(1):99–122, 1997. Online: https://doi.org/10.1162/neco.1997.9.1.99.

[Rup88] D. Ruppert. Efficient estimations from a slowly convergent Robbins-Monro process, 1988. Technical Report, Cornell University Operations Research and Industrial Engineering. Online: https://www.researchgate.net/publication/246031929.

[Saa99] D. Saad, editor. *Online learning in neural networks*. Cambridge University Press, Cambridge, UK, 1999. Online: https://www.cambridge.org/core/books/online-learning-in-neural-networks/7D901F98C2A4F1CF69648FDAEF6877CD#.

[Sam69] J. W. Sammon. A nonlinear mapping for data structure analysis. *IEEE Transactions on Computers*, C-18(5):401–409, 1969. Online: http://syllabus.cs.manchester.ac.uk/pgt/2017/COMP61021/reference/Sammon.pdf.

[San89] T.D. Sanger. Optimal unsupervised learning in a single-layer linear feedforward neural network. *Neural Networks*, 2(6):459–473, 1989. Online: https://www.sciencedirect.com/science/article/pii/0893608089900440.

[SBH09] P. Schneider, M. Biehl, and B. Hammer. Adaptive relevance matrices in Learning Vector Quantization. *Neural Comput.*, 21:3532–3561, 2009. Preprint: https://www.cs.rug.nl/biehl/Preprints/mlmatrix.pdf.

[SBS+10] P. Schneider, K. Bunte, H. Stiekema, B. Hammer, T. Villmann, and M. Biehl. Regularization in matrix relevance learning. *Neural Networks, IEEE Transactions on*, 21(5):831–840, 2010. Online: https://research.rug.nl/en/publications/regularization-in-matrix-relevance-learning.

[SC10] S.K. Sharma and P. Chandra. Constructive Neural Networks: A Review. *Intl. J. of Engineering Science and Technology*, 2(12):7847–7855, 2010. Online: https://www.researchgate.net/publication/50384469.

[Sch01] L. Schläfli. *Theorie der vielfachen Kontinuitat*. Zürcher & Furrer, Zürich, Switzerland, 1901. Reprint: https://babel.hathitrust.org/cgi/ssd?id=coo.31924059156582.

[Sch93] C. Schaffer. Overfitting avoidance as bias. *Machine Learning*, 10:153–178, 1993. Online: https://link.springer.com/article/10.1023/A:1022653209073.

[Sch01] B. Schölkopf. The kernel trick for distances. In *Proc. Adv. in neural information processing systems*, pages 301–307, 2001. Online: https://proceedings.neurips.cc/paper/2000/file/4e87337f366f72daa424dae11df0538c-Paper.pdf.

[Sch10] P. Schneider. *Advanced methods for prototype-based classification*. PhD thesis, University of Groningen, 2010. Online: https://www.rug.nl/research/portal/files/14618216/thesis.pdf.

[Sch15] J. Schmidhuber. Deep learning in neural networks: an overview. *Neural Networks*, 61:85–117, 2015. Preprint: https://arxiv.org/abs/1404.7828.

[Sej20] T.J. Sejnowski. The unreasonable effectiveness of deep learning in artificial intelligence. *Proc. National Academy of Sciences of the Unitede States of America (PNAS)*, 117(48):30033–30038, 2020. Preprint: https://arxiv.org/abs/2002.04806.

[SH93] H. Schwarze and J. Hertz. Generalization in fully connected committee machines. *Europhysics Letters*, 21(7):785–790, 1993. Online: https://iopscience.iop.org/article/10.1209/0295-5075/21/7/012.

[SHH20] A. Schulz, F. Hinder, and B. Hammer. Deepview: Visualizing classification boundaries of deep neural networks as scatter plots using discriminative dimensionality reduction. In *Proceedings of the Twenty-Ninth International Joint Conference on Artificial Intelligence,{IJCAI-20}*, 2020. Preprint: https://arxiv.org/abs/1909.09154,.

[SHK+14] N. Srivastava, G.E. Hinton, A. Krizhevsky, I. Sutskever, and R. Salakhutdinov. Dropout: A simple way to prevent neural networks from overfitting. *Journal of Machine Learning Research*, 15(56):1929–1958, 2014. Online: https://jmlr.org/papers/v15/srivastava14a.html.

[SHRV18] S. Saralajew, L. Holdijk, M. Rees, and T. Villmann. Prototype-based neural network layers: Incorporating vector quantization. *CoRR*, abs/1812.01214, 2018. Online: https://arxiv.org/abs/1812.01214.

[SK99] P. Sommervuo and T. Kohonen. Self-organizing maps and learning vector quantization for feature sequences. *Neural Processing Letters*, 10(2):151–159, 1999. Online: http://cis.legacy.ics.tkk.fi/panus/papers/dtwsom.pdf.

[SK19] C. Shorten and T.M. Khoshgoftaar. A survey on Image Data Augmentation for Deep Learning. *J. of Big Data*, 6:60, 2019. Online: https://doi.org/10.1186/s40537-019-0197-0.

[SL92] J. Sjöberg and L. Ljung. Overtraining, regularization, and searching for minimum in neural networks. *IFAC Proceedings Volumes*, 25(14):73–78, 1992. Preprint: https://citeseerx.ist.psu.edu/viewdoc/download?doi=10.1.1.51.298&rep=rep1&type=pdf.

[SL09] M. Sokolova and G. Lapalme. A systematic analysis of performance measures for classification tasks. *Inf. Process. Manage.*, 45(4):427–437, 2009. Online: https://www.researchgate.net/publication/222674734.

[SLF88] S.A. Solla, E. Levin, and M. Fleisher. Accelerated learning in layered neural networks. *Complex Systems*, 2:625–640, 1988. Online: https://www.researchgate.net/publication/239033363.

[SM15] S. Sonoda and N. Murata. Neural network with unbounded activation functions is universal approximator. *Appl. and Computational Harmonic Analysis*, 43(2):233–268, 2015. Preprint: https://arxiv.org/abs/1505.03654.

[SNW11] S. Sra, S. Nowozin, and S. Wright, editors. *Optimization for machine learning*. MIT Press, Cambridge, MA, 2011. Online: https://mitpress.mit.edu/books/optimization-machine-learning.

[SO03] S. Seo and K. Obermayer. Soft Learning Vector Quantization. *Neural Computation*, 15:1589–1604, 2003. Online: https://www.researchgate.net/publication/10698895_Soft_Learning_Vector_Quantization.

[SR98] D. Saad and M. Rattray. Learning with regularizers in multilayer neural networks. *Physical Review E*, 57(2):2170–2176, 1998. Online: https://proceedings.neurips.cc/paper/1996/file/e1d5be1c7f2f456670de3d53c7b54f4a-Paper.pdf.

[SS02] B. Schölkopf and AJ. Smola. *Learning with Kernels: Support Vector Machines, Regularization, Optimization, and Beyond*. Adaptive Computation and Machine Learning. MIT Press, Cambridge, MA, USA, 2002. Online: https://mitpress.mit.edu/books/learning-kernels.

[SSM98] B. Schölkopf, A. Smola, and K.-R. Müller. Nonlinear Component Analysis as a Kernel Eigenvalue Problem. *Neural Computation*, 10(5):1299–1319, 1998. Online: https://citeseerx.ist.psu.edu/viewdoc/summary?doi=10.1.1.100.3636.

[STC04] J. Shawe-Taylor and N. Cristianini. *Kernel Methods for Pattern Analysis*. Cambridge University Press, Cambridge, UK, 2004. Online: https://doi.org/10.1017/CBO9780511809682.

[Sto04] J.V. Stone. *Independent Component Analysis : a tutorial introduction*. MIT Press, 2004. Online: https://direct.mit.edu/books/book/2325/Independent-Component-AnalysisA-Tutorial.

[Str19] G. Strang. *Linear Algebra and Learning from Data*. Wellesley-Cambridge Press, Wellesley, MA, 2019. Online: https://math.mit.edu/~gs/learningfromdata/.

[STW11] Y. Shao, G.N. Taff, and S.J. Walsh. Comparison of early stopping criteria for neural-network-based subpixel classification. *IEEE Geoscience and Remote Sensing Letters*, 8(1):113–117, 2011. Online: https://ieeexplore.ieee.org/document/5530351.

[SVC07] B. Schrauwen, D. Verstraeten, and J. Van Campenhout. An overview of reservoir computing: theory, applications, and implementations. In M. Verleysen, editor, *Proceedings of the European Symposium on Artificial Neural Networks ESANN 2007*, pages 471–482. d-side publishing, 2007. Online: https://www.researchgate.net/publication/221166209.

[SVG+18] S. Sardi, R. Vardi, A. Goldental, Y. Tugendhaft, H. Uzan, and I. Kanter. Dendritic learning as a paradigm shift in brain learning. *ACS Chemical Neuroscience*, 9(6):1230–1232, 2018. Online: https://doi.org/10.1021/acschemneuro.8b00204.

[svm99] svm.org. Support Vector Machines, 1999. Online repository: https://www.svms.org.

[SY95] A.S. Sato and K. Yamada. Generalized Learning Vector Quantization. In G. Tesauro, D. Touretzky, and T. Leen, editors, *Advances in Neural Information Processing Systems*, volume 7, pages 423–429, 1995. Online: https://proceedings.neurips.cc/paper/1995/file/9c3b1830513cc3b8fc4b76635d32e692-Paper.pdf.

[SY98] A.S. Sato and K. Yamada. An analysis of convergence in Generalized LVQ. In L. Niklasson, M. Bodén, and T. Ziemke, editors, *International Conference on Artificial Neural Networks, ICANN'98*, pages 172–176. Springer, Berlin, 1998. Online: https://link.springer.com/chapter/10.1007/978-1-4471-1599-1_22.

[TdSL00] J. B. Tenenbaum, V. de Silva, and J. C. Langford. A global geometric framework for nonlinear dimensionality reduction. *Science*, 290:2319–2323, 2000. https://www.science.org/doi/10.1126/science.290.5500.2319.

[Tes21] H. Tessier. Neural Network Pruning 101 – All you need to know to get lost, 2021. Online: https://towardsdatascience.com/neural-network-pruning-101-af816aaea61.

[Tho53] Robert L. Thorndike. Who belongs in the family? *Psychometrika*, 18(4), 1953. Online: https://link.springer.com/article/10.1007/bf02289263.

[Tou12] M. Toussaint. Lecture Notes: some notes on gradient descent, 2012. Online: https://www.user.tu-berlin.de/mtoussai/notes/gradientDescent.pdf.

[TWH02] R. Tibshirani, G. Walther, and T. Hastie. Estimating the number of clusters in a data set via the gap statistics. *J. Royal Statistical Society, Statistical Methodology, Series B*, 63(2), 2002. Online: https://rss.onlinelibrary.wiley.com/doi/10.1111/1467-9868.00293.

[UMW17] K. Ullrich, E. Meeds, and M. Welling. Soft weight-sharing for neural network compression. In *5th International Conference on Learning Representations (ICLR)*, 2017. Preprint: https://arxiv.org/abs/1702.04008.

[Urb97] R. Urbanczik. Storage capacity of the fully-connected committee machine. *Journal of Physics A: Math. and Gen.*, 30(11):L387–L392, 1997. Online: https://iopscience.iop.org/article/10.1088/0305-4470/30/11/007.

[Urb00] R. Urbanczik. Online Learning with Ensembles. *Physical Review E*, 62(1):1448–1451, 2000. Preprint: https://arxiv.org/pdf/cond-mat/9907487.pdf.

[VBVS17] T. Villmann, M. Biehl, A. Villmann, and S. Sarajalew. Fusion of deep learning architectures, multilayer feedforward networks and Learning Vector Quantizers for deep classification learning. In *Proc. 12th Workshop on Self-Organizing Maps and Learning Vector Quantization (WSOM+)*, pages 248–255. IEEE Press, 2017. Online: https://pure.rug.nl/ws/files/47382763/Fusion_of_Deep_Learning_Architectures_Multilayer.pdf.

[Vie23] T. Viering. *On Safety in Machine Learning*. PhD thesis, Technical University Delft, 2023. Online: https://research.tudelft.nl/en/publications/on-safety-in-machine-learning.

[VKHB16] T. Villmann, M. Kaden, W. Herrmann, and M. Biehl. Learning Vector Quantization classifiers for ROC-optimization. *Computational Statistics*, 2016. Online: https://research.rug.nl/en/publications/learning-vector-quantization-classifiers-for-roc-optimization.

[VKNR12] T. Villmann, M. Kästner, D. Nebel, and M. Riedel. ICMLA face recognition challenge – results of the team 'Computational Intelligence Mittweida'. In *Proc. of the International Conference on Machine Learning*

Applications (ICMLA), pages 7–10. IEEE, New York, NY, 2012. Online: https://ieeexplore.ieee.org/document/6406802.

[VL63] V.N. Vapnik and A.Y. Lerner. Recognition of patterns with help of generalized portraits. *Avtomat. i Telemekh.*, 24(6):774–780, 1963. Online: https://www.mathnet.ru/php/archive.phtml?wshow=paper&jrnid=at&paperid=11885&option_lang=eng.

[VMC16] H. de Vries, R. Memisevic, and A. Courville. Deep Learning Vector Quantization. In M. Verleysen, editor, *Proc. Europ. Symp. on Artificial Neural Networks (ESANN)*, pages 503–508. i6doc.com, 2016. Online: https://www.esann.org/sites/default/files/proceedings/legacy/es2016-112.pdf.

[VMGL12] A. Vellido, J.D. Martín-Guerro, and P. Lisboa. Making machine learning models interpretable. In M. Verleysen, editor, *Proc. of the European Symposium on Artificial Neural Networks (ESANN 2012)*, pages 163–172. d-side, 2012. Online: https://citeseerx.ist.psu.edu/viewdoc/download?doi=10.1.1.431.5382&rep=rep1&type=pdf.

[VRV+20] T. Villmann, J. Ravichandran, A. Villmann, D. Nebel, and M. Kaden. Investigation of activation functions for generalized learning vector quantization. In A. Vellido, K. Gibert, C. Angulo, and J.D. Martín Guerrero, editors, *Advances in Self-Organizing Maps, Learning Vector Quantization, Clustering and Data Visualization*, pages 179–188, Cham, 2020. Springer International Publishing. Online: https://www.researchgate.net/publication/332719447.

[VWB21] R.J. Veen, F. Westerman, and M. Biehl. A no-nonsense beginner's toolbox for GMLVQ, Version v3.1, 2021. Online: https://www.cs.rug.nl/~biehl/gmlvq.

[Wat93] T.L.H. Watkin. Optimal Learning with a Neural Network. *Europhys. Lett.*, 21(8):871–876, 1993. Online: https://iopscience.iop.org/article/10.1209/0295-5075/21/8/013.

[WBJH18] M. van der Wilk, M. Bauer, S.T. John, and J. Hensman. Learning invariances using the marginal likelihood. In S. Bengio, H. Wallach, H. Larochelle, K. Grauman, N. Cesa-Bianchi, and R. Garnett, editors, *Advances in Neural Information Processing Systems*, volume 31. Curran Associates, Inc., 2018. Online: https://proceedings.neurips.cc/paper/2018/file/d465f14a648b3d0a1faa6f447e526c60-Paper.pdf.

[Wei99] E.W. Weisstein. Cake Number, 1999. From *MathWorld*– A Wolfram Web Resource: https://mathworld.wolfram.com/CakeNumber.html.

[Wer74] P. Werbos. *Beyond regression: New tools for prediction and analysis in the behavorial sciences*. PhD thesis, Harvard University, 1974. Online: https://www.researchgate.net/publication/35657389.

[Wet78] J.E. Wetzel. On the division of planes by lines. *The American Mathematical Monthly*, 85(8):647–656, 1978. Online: https://www.jstor.org/stable/2320333.

[WH60] B. Widrow and M.E. Hoff. Adaptive switching circuits. In *Proc. IRE WESCON Convention Rec.*, pages 96–104, 1960. Online: https://www-isl.stanford.edu/~widrow/papers/c1960adaptiveswitching.pdf.

[Wid60] B. Widrow. An Adaptive "Adaline" Neuron Using Chemical "Memistors", 1960. Technical Report No. 1553-2. Online: https://www-isl.stanford.edu/~widrow/papers/t1960anadaptive.pdf.

[Wid12] B. Widrow. Youtube channel widrowlms, 2012. Online: https://www.youtube.com/user/widrowlms.

[Wik22] Wikipedia. Receiver operating characteristic, 2022. Online: https://en.wikipedia.org/wiki/Receiver_operating_characteristic.

[Wil65] J.H. Wilkinson. *The Algebraic Eigenvalue Problem*. Oxford University Press, 1965. Online:https://global.oup.com/academic/product/the-algebraic-eigenvalue-problem-9780198534181.

[Win61] R.O. Winder. Single stage threshold logic. In *2nd Ann. Symposium on Switching Circuit Theory and Logical Design (SWCT 1961)*, pages 321–332, 1961. Online: https://doi.org/10.1109/FOCS.1961.29.

[Wit10] A.W. Witoelar. *Statistical physics of Learning Vector Quantization*. PhD thesis, University of Groningen, 2010. Online: https://www.rug.nl/research/portal/files/14692629/11complete.pdf.

[Wit20] D. Witten. The bias-variance-tradeoff & double descent, 2020. Twitter: https://threadreaderapp.com/thread/1292293102103748609.html.

[WL90] B. Widrow and M.A. Lehr. 30 years of adaptive neural networks: perceptron, madaline, and backpropagation. *Proc. of the IEEE*, 78(9):1415–1442, 1990. Online: https://www-isl.stanford.edu/~widrow/papers/j199030years.pdf.

[WM05] C.J. Willmott and K. Matsuura. Advantages of the mean absolute error (MAE) over the root mean square error (RMSE) in assessing average model performance. *Climate Research*, 30(1):79–82, 2005. Online: https://www.jstor.org/stable/24869236.

[WO12] M. Wiering and M. van Otterlo, editors. *Reinforcement Learning, State-of-the-Art*. Springer, Berlin, 2012. Online: https://link.springer.com/book/10.1007/978-3-642-27645-3.

[Wol61] P. Wolfe. A duality theorem for nonlinear programming. *Quarterly of Applied Mathematics*, 19(3):239–244, 1961. Online: https://www.jstor.org/stable/43635235.

[WRB93] T.L.H. Watkin, A. Rau, and M. Biehl. The statistical mechanics of learning a rule. *Reviews of Modern Physics*, 65:499–556, 1993. Online: https://research.rug.nl/en/publications/the-statistical-mechanics-of-learning-a-rule.

[Wri15] S.J. Wright. Coordinate descent algorithms. *Math. Programming*, 151(1):3–34, 2015. Online: https://doi.org/10.1007/s10107-015-0892-3.

[WS09] K. Weinberger and L. Saul. Distance metric learning for Large Margin Nearest Neighbor classification. *J. of Machine Learning Research*, 10:207–244, 2009. Online: https://jmlr.csail.mit.edu/papers/volume10/weinberger09a/weinberger09a.pdf.

[WZZ+13] L. Wan, M. Zeiler, S. Zhang, Y. Le Cun, and R. Fergus. Regularization of neural networks using dropconnect. In S. Dasgupta and

D. McAllester, editors, *Proceedings of the 30th International Conference on Machine Learning*, volume 28 of *Proceedings of Machine Learning Research*, pages 1058–1066, Atlanta, Georgia, USA, 2013. PMLR. Online: https://proceedings.mlr.press/v28/wan13.pdf.

[YCC18] M. Yamada, J. Chen, and Y. Chang. *Transfer Learning, Algorithms and Applications*. Morgan Kaufmann, 2018. Google books: https://books.google.nl/books/about/Transfer_Learning.html?id=AOeMCgAAQBAJ.

[You07] YouTube channel Pseudo1ntellectual. Perceptron Research from the 50's & 60's, video clip, 2007. Online: https://www.youtube.com/watch?v=cNxadbrN_aI.

[You21] A. Young. Every Data Scientist Should Know: The Bias-Variance Trade-off Generalization is Wrong, 2021. Towards Data Science. Online: https://tinyurl.com/mr2f5p36.

[Zad19] A.M. Zador. A critique of pure learning and what artificial neural networks can learn from animal brains. *Nature Communications*, 10:3770, 2019. Online: https://doi.org/10.1038/s41467-019-11786-6.

[Zas75] T. Zaslavsky. *Facing Up to Arrangements: Face-Count Formulas for Partitions of Space by Hyperplanes*, volume 154 of *Memoirs of the American Mathematical Society*. AMS, 1975. Online: https://bookstore.ams.org/memo-1-154/.

The Shallow and the Deep is a collection of lecture notes that offers an accessible introduction to neural networks and machine learning in general. However, it was clear from the beginning that these notes would not be able to cover this rapidly changing and growing field in its entirety. The focus lies on classical machine learning techniques, with a bias towards classification and regression. Other learning paradigms and many recent developments in, for instance, Deep Learning are not addressed or only briefly touched upon.
Biehl argues that having a solid knowledge of the foundations of the field is essential, especially for anyone who wants to explore the world of machine learning with an ambition that goes beyond the application of *some software package to some data set*. Therefore, *The Shallow and the Deep* places emphasis on fundamental concepts and theoretical background. This also involves delving into the history and *pre-history* of neural networks, where the foundations for most of the recent developments were laid.
These notes aim to demystify machine learning and neural networks without losing the appreciation for their impressive power and versatility.

Michael Biehl is Associate Professor of Computer Science at the Bernoulli Institute for Mathematics, Computer Science and Artificial Intelligence of the University of Groningen, where he joined the Intelligent Systems group in 2003. He also holds an honorary Professorship of Machine Learning at the Center for Systems Modelling and Quantitative Biomedicine of the University of Birmingham, UK.
His research focuses on the modelling and theoretical understanding of neural networks and machine learning in general. The development of efficient training algorithms for interpretable, transparent systems is a topic of particular interest. A variety of interdisciplinary collaborations concern practical applications of machine learning in the biomedical domain, in astronomy and other areas.

University of Groningen Press

ISBN 978-94-034-3028-7