Supervised Learning – An Introduction

Michael Biehl

University of Groningen, Groningen, The Netherlands

Bernoulli Institute for Mathematics, Computer Science
and Artificial Intelligence

Department of Computer Science, Intelligent Systems Group

Based on a set of lectures given at the

30^{th} Canary Islands Winter School of Astrophysics

Big Data Analysis in Astronomy

La Laguna, Tenerife, Spain, 11/2018

To a large extent, the material is taken from an MSc level course

Neural Networks and Computational Intelligence

Computing Science Programme, University of Groningen

These notes present a selection of topics in the area of supervised machine learning. The focus is on the discussion of methods and algorithms for classification tasks. Regression by neural networks is discussed only very briefly as it is in the center of complementary lectures [1]. The same applies to concepts and methods of unsupervised learning [2].

The selection and presentation of the material is clearly influenced by personal biasses and preferences. Nevertheless, the lectures and notes should provide a useful, albeit incomplete, overview and serve as a starting point for further exploration of the fascinating area of machine learning.

Version: 12-03-2019

Contact: `m.biehl@rug.nl` `http://www.cs.rug.nl/~biehl`

Contents

List of illustrations

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 of all times. This is particularly true for its plasticity, i.e. the ability to learn from experience and to adapt to (and survive in) ever-changing environments.

Ultimately, the performance of the brain must rely on its *hardware* (or *wetware*, rather). Apparently, all of its functionality emerges from the cooperative behavior of the 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 systems are inspired by natural systems without copying all their properties in detail. Due to biological constraints, nature (i.e. evolution) might have produced highly complex solutions to certain problems that can be dealt with in a simpler fashion in a technical realization. A somewhat over-used analogy in this context is the construction of efficient aircraft, which by no means required the use of moving wings in order to imitate bird flight.

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 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 meant as a very brief overview, only, 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

[1] including the capability of being fascinated

all aspects.

Detailed citations concerning specific topics are not provided in this chapter. Instead, the following list points to some sources which range from brief and superficial to very comprehensive and detailed reviews. The same is true for the discussion of the different conceptual levels on which biological systems can be modelled.

[3] **K. Guerney** *(Neural Networks)* gives a very basic overview and provides a glossary of biological or biologically inspired terms.

[4] The first sections of **S. Haykin's** *Neural Networks and Learning Machines* cover the relevant topics in slightly greater depth.

[5] 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 the most thorough analysis of the Hopfield model from a statistical physics perspective.

[6] **H. Horner and R. Kühn** give a brief general overview of *Neural Networks*, including a basic discussion of their biological background.

[7] 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.* It discusses the different modelling approaches and relates them to experimental data obtained from real world neurons.

[8] **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.*

[9] **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.*

[10] **M. van Rossum's** lecture notes on *Neural Computation* provide a overview of biological information processing and models of neural activity, synaptic interaction and plasticity. Moreover, modelling approaches are discussed in some detail.

Here, numbers refer to full citation information in the bibliography. Note that the selection is certainly incomplete and clearly biased by personal preferences.

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 faithful 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 controls the concentrations and, thus, the membrane potential. The original Hodgkin-Huxley models describes its temporal evaluation 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

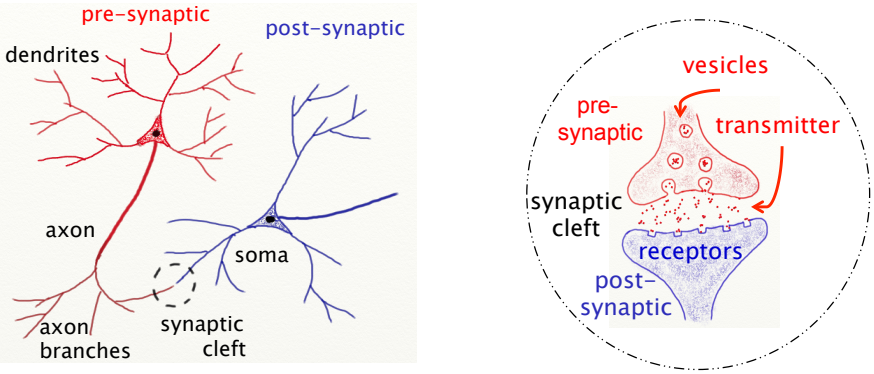


Figure 1.1: Schematic illustration of neurons (pyramidal cells) and their connections. **Left:** Pre-synaptic and post-synaptic neurons with soma, dendritic tree, axon, and axonic branches. **Right:** The synaptic cleft with vesicles releasing neuro-transmitters and corresponding receptors on the post-synaptic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

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 this 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 decreases.

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

[2]However, so-called gap junctions can play the role of bi-directional *electrical synapses*.

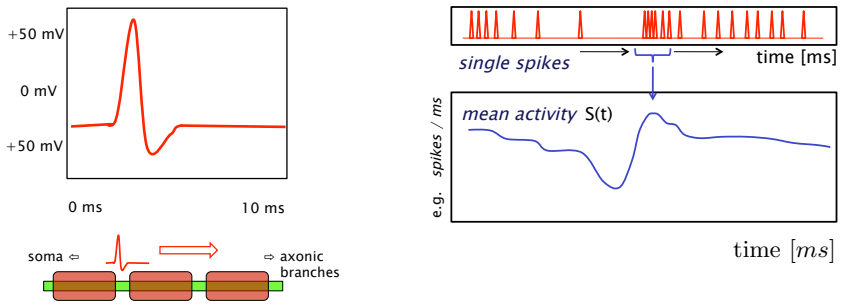


Figure 1.2: **Left (upper):** Schematic illustration of an action potential, i.e. a short pulse on mV- and ms-scale. **Left (lower):** Spikes travel along the axon through saltatory conduction via gaps in the insulating myelin sheath. **Right:** Schematic illustration of how mean firing rates are derived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

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 unknown, 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 the different ion concentrations as well as leak currents through the membrane. Similarly, the precise form of spikes, 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. Spikes are represented by structureless Dirac delta functions in time. 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.

1.2 Firing rate models

In another step of abstraction, the description of neural activity is reduced to 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). Hence, the precise timing of individual action potentials is completely disregarded. 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. 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 J of neurons $j \neq i$. 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

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

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

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

Here, the quantities $w_{ij} \in \mathbb{R}$ represent the strength of the synapse connecting neuron 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

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):

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

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

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 a later chapter. A very important class of plausible activations is given by so-called sigmoidal functions. Just one prominent[4] example being

$$h(x) \;=\; \frac{1}{2}\Big(1 + \tanh\big[\gamma(x-\theta)\big]\Big) \tag{1.2}$$

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.

1.2.3 Symmetrized representation of activity

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

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

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

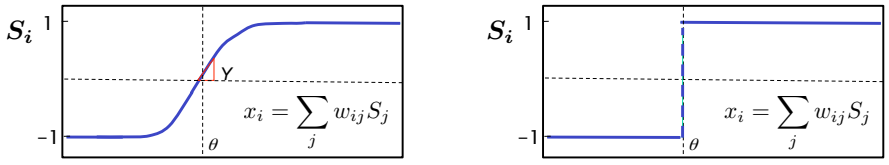


Figure 1.3: Schematic illustration of example (symmetrized) activation functions. **Left:** A sigmoidal transfer function with gain γ and threshold θ in the symmetrized representation, cf. Eq. (1.2). **Right:** The binary McCulloch Pitts activation as obtained in the limit $\gamma \to \infty$.

An example activation analogous to Eq. (1.2) is

$$g(x) \;=\; \tanh\big[\gamma(x-\theta)\big]. \tag{1.3}$$

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

$$S_i \;=\; g(x_i) \quad \text{with} \quad x_i = \sum_{j\in J} w_{ij}\, S_j. \tag{1.4}$$

implies that, now, the absence of activity ($S_j = -1$) in neuron j can 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 will 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, for instance, to the fruitful popularization of neural networks in the statistical physics community in the 1980's and 1990's.

1.2.4 McCulloch Pitts neurons

Quite frequently, an even more drastic modification is considered: For infinite gain $\theta \to \infty$ the sigmoidal activations become step functions and, for instance, Eq. (1.3) yields in this limit

$$g(x) \;= \text{sign}(x-\theta) = \begin{cases} +1 & \text{if } \; x \geq \theta \\ -1 & \text{if } \; x < \theta, \end{cases} \tag{1.5}$$

see Fig. 1.3 (right panel) for an illustration. 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, originally denoting the quiescent state by $S = 0$. The persisting popularity of the model is due to its simplicity and similarity to boolean concepts in conventional computing. In the following, we will frequently resort to binary model neurons in the symmetrized version (1.5). 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*.

1.2.5 Hebbian learning

Probably the most intriguing property of biological neural networks is their ability to learn. Instead of realizing *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.

Arguably the most plausible and most frequent process 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. The original hypothesis is 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 re-phrased 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.

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:

$$\Delta w_{AB} \propto S_A S_B. \tag{1.6}$$

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. (1.6) if both neurons are quiescent at the same time since $S_A S_B > 0$ in this case. 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 (1.6) 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. (1.6).

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 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: fully connected recurrent networks and feed-forward layered networks. The possibilities for modifications, hybrid and intermediate types are nearly endless. Some more specific architectures will be introduced in a later chapter addressing *shallow* and *deep* architectures.

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_i on obtains an *update* of the form

$$S_i(t+1) \;=\; g\left(\sum_j w_{ij}\, S_j(t)\right) \tag{1.7}$$

where the sum is over all units that neuron i receives input through a synapse $w_{ij} \neq 0$. Eq. (1.7) 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 can only refer the reader to the literature.

From an initial configuration which comprises the individual activity $\mathbf{S}(0) = (S_1(0), S_2(0), \ldots, S_N(0))^\top$ at time $t = 0$, 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 architectures is the fully connected Hopfield or Little-Hopfield model. It is, in a sense, extreme and very clear-cut: The Hopfield network comprises N neurons of the McCulloch Pitts type which are fully connected by bi-directional synapses

$$w_{ij} = w_{ji} \in I\!R \quad (i, j = 1, 2, \ldots N) \;\text{ with } w_{ii} = 0 \;\text{ for all } i. \tag{1.8}$$

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

$$S_i(t+1) = \text{sign}\left(\sum_{\substack{j=1 \\ j\neq i}}^{N} w_{ij} S_j(t).\right) \tag{1.9}$$

John Hopfield realized that the corresponding random sequential update can be seen as a *zero temperature* Metropolis Monte Carlo dynamics which is governed by an energy function of the form

$$H(\mathbf{S}(t)) = -\sum_{\substack{i,j=1 \\ i<j}}^{N} w_{ij}\, S_i(t)\, S_j(t). \tag{1.10}$$

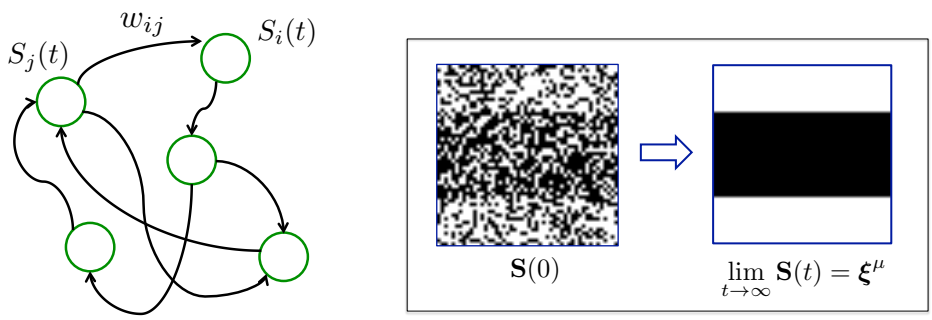


Figure 1.4: Recurrent neural networks. **Left:** A network of $N = 5$ neurons with partial connectivity and uni-directional synapses. **Right:** Illustration of the retrieval of a stored activity pattern from a noisy initial configuration in the Hopfield network.

The mathematical structure is analogous to the so-called Ising model in Statistical Physics. There, the degrees of freedom $S_i = \pm 1$ are typically termed *spins* and they are interpreted as to represent microscopic magnetic moments, originally. 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*.

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

$$w_{ij} = w_{ji} = \frac{1}{P} \sum_{\mu=1}^{P} \xi_i^\mu \, \xi_j^\mu \quad \text{for } i \neq j \quad \text{and } w_{ii} = 0 \ \text{ for all } i, \tag{1.11}$$

where the constant pre-factor $1/P$ follows the convention in the literature. It allows to interpret the weights as empirical averages over the data set, but is otherwise irrelevant. 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 $\boldsymbol{\xi}^\nu \in I\!P$ only in a limited fraction of neurons with $S_i(0) = -\xi_i^\nu$, the dynamics can retrieve the corrupted or noisy information. Ideally, the temporal evolution under the updates (1.9) 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).

Successful retrieval 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 (1.11) can also be interpreted as Hebbian Learning: Starting from a tabula rasa state of the synaptic strengths with $w_{ij}(0) = 0$, 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 the construction of (1.11) could be written as an iteration

$$w_{ij}(\mu) = w_{ij}(\mu - 1) + \frac{1}{P}\xi_i^\mu \xi_j^\mu \tag{1.12}$$

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 has served 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

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.

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 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

$$S_i^{(k)} = g\left(\sum_j w_{ij}^{(k)} S_j^{(k-1)}\right). \tag{1.13}$$

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 networks output, i.e. its response to the input or stimulus in the first layer. The illustration displays a single 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 to parametrize 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 feed-back is present. Inter-layer synapses or backward

[5] In the literature this critical value is often denoted as α_c, but it should not be confused with the storage capacity that we discuss later for feed-forward networks.

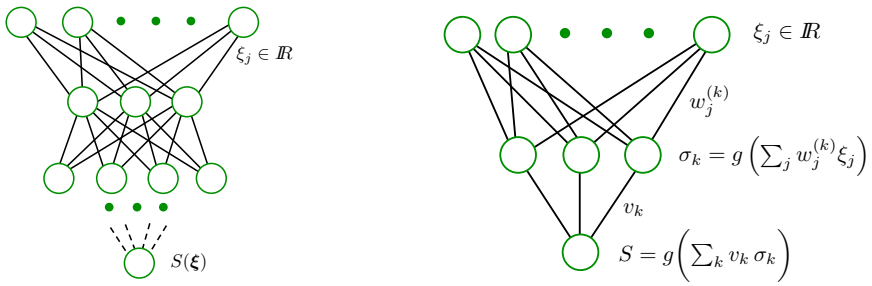


Figure 1.5: Feed-forward neural networks.[6] **Left:** A multilayered architecture with varying layer size and a single output unit. **Right:** A feed-forward network with a layer of input neurons, one hidden layer and a single output unit.

connections feeding information into previous ("higher") layers introduce feed-back 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) forming hidden layers in the feedforward architecture.

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

$$S(\boldsymbol{\xi}) = g\left(\sum_{k=1}^{K} v_k\ \sigma_k\right) = g\left(\sum_{k=1}^{K} v_k\ g\Big(\sum_{j=1}^{N} w_j^k \xi_j\Big)\right). \tag{1.14}$$

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. (1.14) the quantities w_j^k denote the weights connecting the k-th hidden unit to the input layer ($k = 1, 2, \ldots K$ with $K = 3$ in the example). They can be combined into vectors $\mathbf{w}^k \in I\!R^N$, while the hidden-to-output weights are denoted as $v_k \in I\!R$.

6

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 $\boldsymbol{\xi} \in I\!R^N \to S(\boldsymbol{\xi}) \in I\!R$ 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 choice of 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 constitutes an example of supervised learning as discussed in the next section.

Hence, 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

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

suitable architecture. This 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 an output activation with

$$S(\boldsymbol{\xi}) \in \{1, 2, \ldots C\}\,, \tag{1.15}$$

the system performs the assignment of all possible inputs $\boldsymbol{\xi}$ to one of C categories 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 most important example system in the course. The perceptron is defined as a linear threshold classifier with response

$$S(\boldsymbol{\xi}) = \text{ sign } \left(\sum_{j=1}^{N} w_j \xi_j - \theta \right) \tag{1.16}$$

to any possible input $\boldsymbol{\xi} \in I\!R^N$, corresponding to an assignment to one of two classes $S = \pm 1$. Comparison with Eq. (1.5) 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 great detail in the next main chapter and provides valuable insights into the basic concepts of machine learning.

1.3.3 Other architectures

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

Many prototype 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. The attractive framework of prototype based learning will be discussed in Chapter 5 in the context of, both, supervised and unsupervised learning.

Combinations of feed-forward structures with, for instance, layers of highly interconnected units are employed n the context of *Reservoir Computing*, see e.g. [11] for an overview and references. The basic idea is to represent input data as the initial configuration of a dynamical system. Eventually, the state of the system is mapped to a regression or classification target by one or several read-out layers.

Recently, the use of feed-forward architectures has re-gained significant popularity in the context of *Deep Learning*. Specific designs and architectures of *Deep Networks*, including e.g. so-called *convolutional* or *pooling* layers are discussed in the designated lectures by Marc Huertas-Company [1].

Chapter 2

Learning from examples

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

– Douglas Adams

Different forms of machine learning were briefly discussed in Section 1, already. Here we focus on the most clear-cut scenarios: supervised learning and unsupervised learning. The main chapters will deal with supervised learning, with emphasis on classification and regression. Several of the introduced concepts and methods, however, can also be transferred in unsupervised settings. This is the case, for instance, for the prototype-based methods discussed in Chapter 5.

2.1 Unsupervised learning

Unsupervised learning is an umbrella term which comprises the analysis of data sets which 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, comparing its response with a given ground truth or approximate representation thereof.

For more about the background of unsupervised learning, specific algorithms and applications, the reader is referred to the lectures given by Dalya Baron [2]. Here we only briefly discuss the framework of unsupervised data analysis in contrast to supervised learning.

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. An important framework is that of *Vector Quantization* which will be discussed in some detail.

- **compression:**
  Another form of unsupervised learning aims at replacing original data by lower-dimensional representations without reducing the actual number of data points. The representatives 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, for instance, 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.

- **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.

- **pre-processing:**
  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 pre-processing 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.

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 *user* defines the goal of the unsupervised analysis her- or himself. 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 implicit goal of the analysis.

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.

2.2 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 – on the one hand – the target task is difficult or impossible to define in terms of a set of simple rules, while – on the other hand – example data is available and 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.

- **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. An currently somewhat overstrained example is the discrimination of cats and dogs based on photographic images.

A variety of other 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 6. 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 work flow 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 set-ups. In fact, validation constitutes a key step in supervised learning.

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

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 can be phrased as

Learning 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 as agreement with the data set is measured by a continuous error measure, in general. However, the main idea of supervised learning remains the same: additional data provides evidence for some hypotheses while others become less likely.

2.3 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** [12]
  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** [13]
  In various practical contexts, feedback on the performance of a learning systems becomes only 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 path search problem in robotics.

- **transfer learning** [14]
  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 data which was obtained by using similar yet not identical technical platforms.

- **lifelong learning or continual learning** [15]
  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** [16]
  Mostly, regression systems and classifiers reflect correlations they have inferred from the data and which allow to make some form of prediction based on future observations. In general, this does not take *causal relations* into account explicitly. 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. Mostly, we will assume that training data is representative of the problem at hand and that it is complete and reliable to a certain extent.

2.4 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 the 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 etc.

2.4.1 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 very likely comes across are (without providing the exact reference or source):

- *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 reflects 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 a couple of illustrative examples.

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** [17]. A view on many important methods, including density estimation and *Expectation Maximization* algorithms is provided in *Neural Networks for Pattern Recognition* [18] and the more recent *Pattern Recognition and Machine Learning* [19] by **C. Bishop**.

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.

Very often, machine learning and statistical models have very similar or identical structures and, frequently the same mathematical tools or algorithms are used.

The main differences usually lie in the emphasis that is 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 take into account explicit assumptions about statistical properties of the observations,

usually. 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 [2] The goal is not so much the faithful description or interpretation of the data, it is the application of the derived hypothesis 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 obviously 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.

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 to illustrate the relation between machine learning and statistical modelling approaches and their underlying concepts.

2.4.2 Linear regression as a learning problem

Linear regression constitutes one of the earliest, most important and clearest examples of inference or learning. 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 [20]:

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

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. [21] for a quick start.

Two major lessons can be learnt from this example: (a) simple linear regression is 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 $H_o \approx 73.5 \frac{km/s}{Mpc}$ [21].

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 possible the application of 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 at greater detail in a forthcoming chapter.

[2] 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.

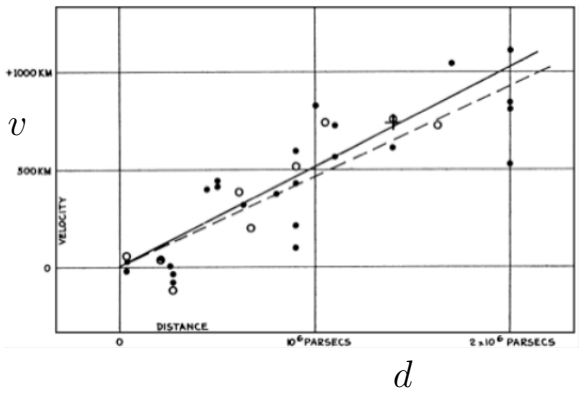


Figure 2.1: The velocity v of galaxies as a function of their distance d, from [20].

A heuristic machine learning approach

Equation (2.1) represents a linear dependence of a target function $v(d)$ on a single variable $d \in I\!R$. 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 $\boldsymbol{\xi} \in I\!R^N$.

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

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

is given. A hypothesis of the form

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

is assumed to represent or approximate the dependence $y(\boldsymbol{\xi})$ underlying the observed data set $I\!D$. 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 $\mathbf{w} \in I\!R^N$.

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

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

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

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:

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

where 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. The pre-factor $1/2$ conveniently cancels when taking derivatives, for instance in the gradient with respect to the weight vector. The necessary first order condition for $\mathbf{w}^*$ to minimize E^{SSE} reads

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

Note that the SSE is a very popular objective function in the context of non-linear regression in multi-layered networks, see e.g. [5, 17–19, 22]. With the convenient matrix and vector notation

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

we can re-write Eq. (2.6) and solve it formally:

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

where X^+ is the (Moore-Penrose) Pseudoinverse of the rectangular matrix X [23]. Note that the solution can be written in precisely this form only if the $(N \times N)$ matrix $[X^\top X]$ is non-singular and, thus, $[X^\top X]^{-1}$ 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 [23], which is a highly recommended source of information in the context of matrix manipulations.

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

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

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

Since the symmetric $[X^\top X]$ has only non-negative eigenvalues, $[X^\top X + \gamma\, I_N]$ is guaranteed to be non-singular for any $\gamma > 0$.

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 $|\,\mathbf{w}\,|^2$. Note that nearly singular matrices $[X^\top\, X]$ would lead to large magnitude weights according to Eq. (2.8).

This is our first encounter of *regularization*, i.e. the restriction of the solution space in a learning problem with the goal of improving the outcome of the training process. In fact, the concept of weight decay is applied in a variety of problems and is by no means restricted to linear regression. Other methods of regularization will be discussed in the context of overfitting in neural networks [1].

We will also see in a later chapter that the special case of linear regression can also be re-formulated as the minimization of $\mathbf{w}^2$ under suitable constraints. This approach solves the problem of having to chose 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

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

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

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

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 adaptive degrees of freedom are well possible but not essential for the sake of the illustration.

In the simplest case we assume that example data have been generated independently, implying that

$$p(\mathbb{D} \mid \mathbf{w}) = \prod_{\mu=1}^{P} p(y^\mu \mid \boldsymbol{\xi}^\mu, \mathbf{w}) \tag{2.12}$$

is the likelihood of observing the concrete set of target values in $\mathbb{D}$ for a given set of input vectors. The corresponding log-likelihood reads

$$\log p(\mathbb{D}\,|\,\mathbf{w}) = \sum_{\mu=1}^{P} \log p(y^\mu\,|\,\boldsymbol{\xi}^\mu, \mathbf{w}) = -\frac{P}{2}\log(2\pi\sigma^2) - \frac{1}{\sigma^2}\,\frac{1}{2}\sum_{\mu=1}^{P}\left(y^\mu - \mathbf{w}^\top \boldsymbol{\xi}^\mu\right)^2 \tag{2.13}$$

where we inserted the Gaussian model (2.11) already. 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, for instance, popular techniques like Principal Component Analysis, mixture models for clustering or Linear Discriminant Analysis.

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 knowing any data, we could express this in terms of an appropriate prior density, for instance

$$p_o(\mathbf{w}) \propto \exp\left[-\frac{1}{2\tau_o^2}\mathbf{w}^2\right]. \tag{2.14}$$

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

$$p(\mathbf{w}|\mathbb{D}) \propto p(\mathbb{D}|\mathbf{w})\, p_o(\mathbf{w}). \tag{2.15}$$

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

Inserting the particularly convenient Gaussian prior (2.14) we can write the logarithm of the posterior as

$$\log\left[p(\mathbf{w}|\mathbb{D})\right] \;\propto\; -E^{SSE} \;-\; \frac{1}{2}\gamma\,\mathbf{w}^2 \;+\; \text{const.} \tag{2.16}$$

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. (2.15).

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 Maximum Likelihood and SSE estimates 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 [17–19, 24].

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 to go even further: In the framework of *Bayesian Inference* we can consider all possible model settings at a time, yielding the *posterior predictive distribution*

$$p(y|\boldsymbol{\xi}, \mathbb{D}) \;\propto \int\; p(y|\boldsymbol{\xi}, \mathbf{w}) \quad \underbrace{p(\mathbf{w}|\mathbb{D})}_{\propto\, p(\mathbb{D}|\mathbf{w})\, p_o(\mathbf{w})} \quad d^N w. \tag{2.17}$$

Properly normalized, this defines the probability of response $y(\boldsymbol{\xi})$ to an 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, 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 the model and prior densities and/or derives easy to handle (e.g. Gaussian) approximations of the posterior predictive distribution (2.17).

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 [25]

3.1 History and literature

The term perceptron is used in a variety of meanings. Throughout these lecture notes, however, it will exclusively refer to a system representing inputs $\boldsymbol{\xi} \in I\!R^N$ in a layer of units, which are connected to a single binary output unit of the McCulloch Pitts type with $S(\boldsymbol{\xi}) \in \{-1, +1\}$ representing two categories of input data[1].

Hence, the term perceptron 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.

In the literature, more general architectures with several layers and/or continuous output are often referred to as (multilayer, soft, ...) perceptrons, as well. We will later consider layered networks which are constructed from perceptron-like units, but the term perceptron itself refers to the single layer, binary classifier in the following.

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

- Pioneered by Frank Rosenblatt [26, 27], 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 provided 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 popularity and success of the field, initially. It will be presented and proven in Sec. 3.3.3.

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

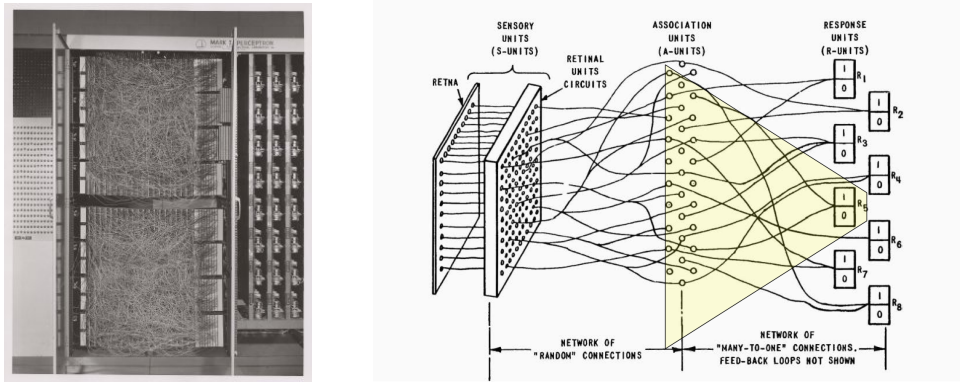


Figure 3.1: The Mark I Perceptron. **Left:** Hardware realization, foto c/o Cornell Library, url: `https://digital.library.cornell.edu/catalog/ss:550351`. The input of the Mark I was realized via a *retina* of 400 photosensors, adaptive weights were realized by potentiometers that could be tuned by electric motors.
Right: Schematic outline of the Mark I architecture, based on a figure taken from [34]. The shaded region marks a subset of units that is commonly referred to as *the* perceptron in this text.

- 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 [28] put it: "The perceptron is the hydrogen atom of neural network research."

- In its modern, conceptually extended re-incarnation, the Support Vector Machine (SVM) [29–32], 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 [33].

Several original texts from the early days of the perceptron are available in the public domain. This includes an original article from 1958 [26], the highly interesting official *Manual of the Perceptron Mark I* hardware [34] and Rosenblatt's monograph *Principles of Neurodynamics* [27]. An interesting TV documentation is available at [35].

The so-called *Perceptron Controversy* and its perception and long-lasting impact on the machine learning community is analysed in a paper entitled *A Sociological Study of the Official History of the Perceptron Controversy* by **M. Ozaran** [33].

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** [5] and by **S. Haykin** in [4], for instance.

The counting argument for the number of linearly separable functions is presented in [5]. In this context, it should be useful to consult the original publications by **R. Winder** [36], **T.M. Cover** [37] and **G.J. Mitchison** and **R.M. Durbin** [38]. The latter work also presents the extension of the counting argument to two-layered networks *(machines)* with K hidden units.

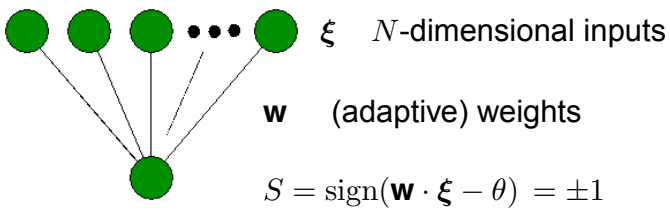


Figure 3.2: Illustration of the single layer perceptron with N-dim. inputs and a binary output of the McCulloch Pitts type.

3.2 Linearly separable functions

The perceptron can be viewed as the simplest feed-forward neural network. It responds to real-valued inputs $\boldsymbol{\xi} \in I\!R^N$ 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

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

corresponds to a (hyper-)plane orthogonal to to $\mathbf{w}$ with an off-set $\theta\,|\mathbf{w}|$ from the origin[2]. 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 $\mathbf{w}, \boldsymbol{\xi} \in I\!R^N$. 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:

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

We observe that

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

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

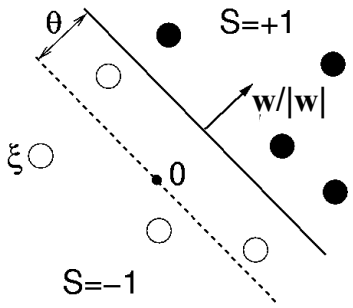


Figure 3.3: Geometrical interpretation of the perceptron. The hyperplane orthogonal to $\mathbf{w}$ with off-set θ from the origin separates feature vectors with output $S = +1$ and $S = -1$, respectively.

As a consequence, a non-zero threshold θ in N-dimensions can always be re-written 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.

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

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

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

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

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

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

(Q1) When is a given data set of the form (3.5) linearly separable? Can we reformulate the condition (3.6) as a set of equations or inequalities which we can solve (numerically) in weight space?

(Q2) Given a data set which is indeed linearly separable, (how) can we find a perceptron weight vector $\mathbf{w}$ that satisfies (3.6)?

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

(Q4) If, for a given $I\!D$, 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?

(Q5) If $I\!D$ comprises examples of a linearly separable function, which can be applied to any input vector, how does the realization or "storage" of the given labels in $I\!D$ relate to the learning of the unknown rule?

(Q6) What can we do if a given data set is not linearly separable? Can we still approximate the classification by means of a perceptron? Which alternatives or extensions exist?

The first five questions will be addressed in the forthcoming sections, while Section 4 deals with the realization or approximation of classification schemes beyond linear separability.

3.3 The Perceptron Storage Problem

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 already [26, 27]. In the following, we first define precisely 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 Formulation of the problem

Here we address question (Q1) of the list given in the previous section. First we consider the task of reproducing the labels given in 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 and look up the correct S_T^μ when needed.

In order to re-write 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 positive constant can be constructed. This is a consequence of the fact that the function $S_\mathbf{w}(\boldsymbol{\xi}) = \mathrm{sign}(\mathbf{w} \cdot \boldsymbol{\xi})$ only depends on the direction of $\mathbf{w}$ in N-dim. feature space while it is invariant under changes of the norm $|\mathbf{w}|$ only.

3.3.2 Iterative training algorithms

In order to address question (Q2) 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.[3]

$$t \in \mathbb{Z}^+ \quad \text{with } \nu(t) = 1, 2, 3, \ldots, P, 1, 2, 3, \ldots \tag{3.11}$$

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 UPDATE (WEIGHTS) (3.12)

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

po

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 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:

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

GENERIC ITERATIVE PERCEPTRON UPDATE (EMBEDDING STRENGTHS) (3.14)

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

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

(all other embedding strengths remain unchanged at time t)

At the end of the training process, the actual weight vector can be constructed according to Eq. (3.13).

Several algorithms considered in the following can be run in the weights directly or, alternatively, in terms of the embedding strengths, along the lines of (3.12) or (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, with only one of them modified per training step. Thus, the computational efficiency of training and the corresponding storage needs will depend on the ration 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. For instance, in the AdaTron algorithm [39–41] for the perceptron of optimal stability, constraints of the from $x^\mu \geq 0$ are imposed, see Sec. 3.5. 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, $\{x^\mu = 0\}_{\mu=1}^P$
- deterministic, cyclic presentation of the examples in $\mathbb{D}$ according to (3.11)
- the modulation function

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

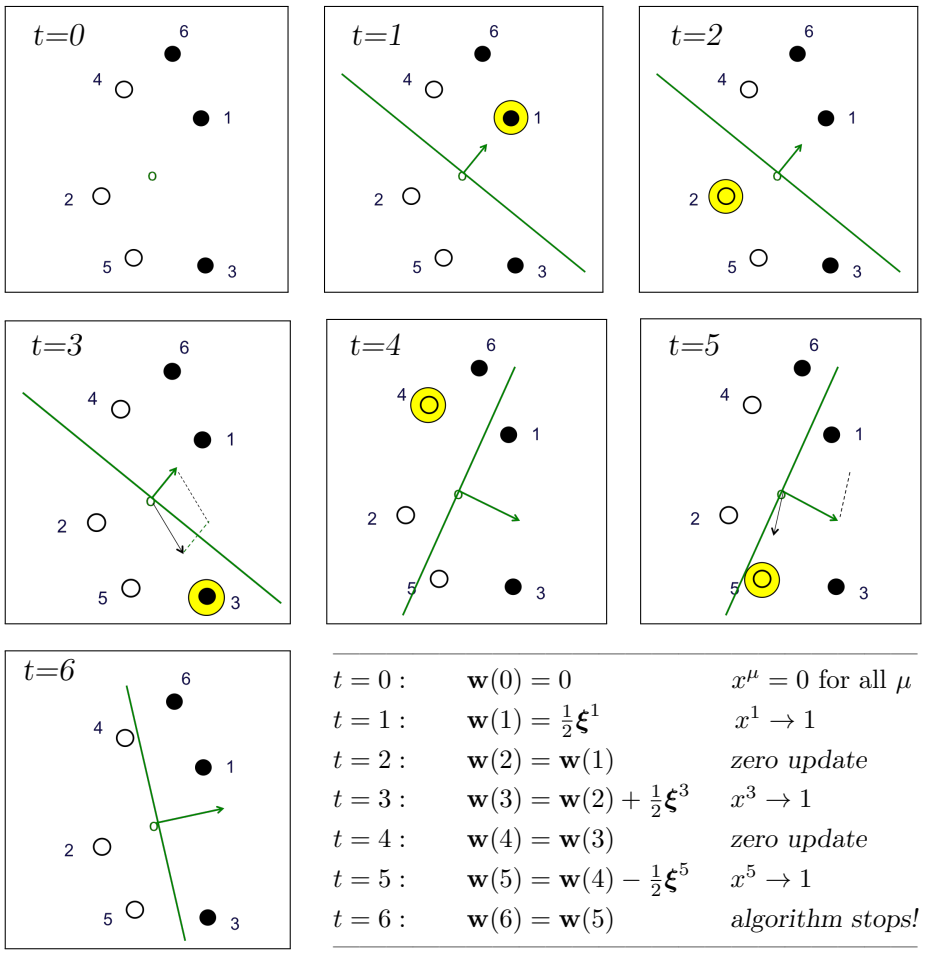
with the Heaviside function $\Theta[x] = \begin{cases} 1 \text{ for } x > 0 \\ 0 \text{ else.} \end{cases}$

The update (3.15) modifies $\mathbf{w}(t)$ only if the example input is misclassified by the current weight vector with a *margin* c as in (3.9). In that case, a Hebbian term is added. This is often referred to as "learning from mistakes", a principle which is the basis of several other training algorithms discussed in forthcoming sections.

Most frequently, the updates are performed with the simple setting $c = 0$, and the resulting algorithm is referred to as the Rosenblatt perceptron algorithm, in the literature.

Note that the modulation function (3.15), the x^μ remain unchanged or increase by 1 in every update step. Consequently, the resulting embedding strengths are non-negative integers.

The ($c = 0$) algorithms 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.



$t=0:$	$\mathbf{w}(0)=0$	$x^\mu=0$ for all μ
$t=1:$	$\mathbf{w}(1)=\frac{1}{2}\boldsymbol{\xi}^1$	$x^1\to 1$
$t=2:$	$\mathbf{w}(2)=\mathbf{w}(1)$	*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*
$t=5:$	$\mathbf{w}(5)=\mathbf{w}(4)-\frac{1}{2}\boldsymbol{\xi}^5$	$x^5\to 1$
$t=6:$	$\mathbf{w}(6)=\mathbf{w}(5)$	*algorithm stops!*

Figure 3.4: Rosenblatt perceptron algorithm.
Illustration of the training scheme with $c=0$ in (3.15) in an $(N=2)$-dimensional feature space. A set of six examples is presented sequentially, empty circles represent feature vectors labelled 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 displayed as 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. In 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 stops.

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

First we illustrate the algorithm in terms of a simple two-dimensional feature space and a data set $I\!D$ comprising 6 labelled inputs, see Fig. 3.4. In this specific case, the perceptron classifies all feature vectors in $I\!D$ correctly, and the algorithm stops after one epoch of training, already.

3.3.4 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 (3.16)

For linearly separable data $I\!D = \{\boldsymbol{\xi}^\mu, S_T^\mu\}_{\mu=1}^P$, the Rosenblatt Perceptron Algorithm (3.15) stops after a finite number of update steps (3.15) and yields a weight vector $\mathbf{w}$ with $\mathbf{w}\cdot\boldsymbol{\xi}^\mu S_T^\mu > 0$ for all $\mu = 1, 2, \dots P$.

A few remarks:

- The required number of training steps is *finite* according to the theorem. However, their actual number depends on the detailed properties of $I\!D$ and can be very large.
- It is difficult to decide whether a given $I\!D$ is linearly separable: 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 epochs or that, indeed, a solution does not exist.
- For the convergence proof, we have to assume the existence of a solution. The Perceptron Convergence Theorem does not provide any insight into the algorithm's performance if $I\!D$ is not linearly separable.

In the following we sketch the proof of convergence. We consider a linearly separable $I\!D = \{\xi^\mu, S_T^\mu\}_{\mu=1}^P$, which implies that at least one solution $\mathbf{w}^*$ of the (PSP) exists with

$$\left\{\mathrm{sign}\,(\mathbf{w}^*\cdot\boldsymbol{\xi}^\mu) = \mathrm{S}_\mathrm{T}^\mu\right\}_{\mu=1}^\mathrm{P} \quad \text{or, equivalently,} \quad \left\{\mathrm{E}^{\mu *} = \mathbf{w}^*\cdot\boldsymbol{\xi}^\mu\,\mathrm{S}_\mathrm{T}^\mu \geq \mathrm{c} > 0\right\}_{\mu=1}^\mathrm{P} \quad (3.17)$$

for some non-negative constant c.
We do not have to further specify $\mathbf{w}^*$ here. In fact, for a given $I\!D$ there could be many solutions of the form (3.17), but here it is sufficient to assume the existence of at least one. We will furthermore denote its squared norm as

$$Q^* \equiv \mathbf{w}^*\cdot\mathbf{w}^* = |\mathbf{w}^*|^2. \quad (3.18)$$

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

$$0 \leq \frac{(\mathbf{w}\cdot\mathbf{w}^*)^2}{|\mathbf{w}^*|^2\,|\mathbf{w}|^2} = \cos^2\angle\{\mathbf{w}, \mathbf{w}^*\} \leq 1. \quad (3.19)$$

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

$$\mathbf{w}(t) \;=\; \frac{1}{N}\sum_{\mu=1}^{P} x^{\mu}(t)\;\boldsymbol{\xi}^{\mu} S_T^{\mu} \quad \text{for} \;\; \mathbf{w}(0)=0. \tag{3.20}$$

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

$$M(t) = \sum_{\mu=1}^{P} x^{\mu}(t). \tag{3.21}$$

Now let us consider the projection $R(t) = \mathbf{w}(t)\cdot\mathbf{w}^*$. Inserting (3.20) and exploiting the condition (3.17) we obtain the following lower bound:

$$R(t) \;=\; \frac{1}{N}\sum_{\mu=1}^{P} x^{\mu}(t)\;\left[\mathbf{w}^*\cdot\boldsymbol{\xi}^{\mu}S_t^{\mu}\right] \;=\; \frac{1}{N}\sum_{\mu=1}^{P} x^{\mu}(t)\;\underbrace{E^{\mu *}}_{\geq c} \geq\; \frac{1}{N}\,c\,M(t). \tag{3.22}$$

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

$$\begin{aligned} Q(t+1) &= \left(\mathbf{w}(t) + \frac{1}{N}\Theta\left[c - E^{\nu(t)}\right]\boldsymbol{\xi}^{\nu(t)}\,S_T^{\nu(t)}\right)^2 \\ &= Q(t) + \frac{2}{N}\Theta\left[c-E^{\nu(t)}\right]E^{\nu(t)} \;+\; \frac{1}{N^2}\Theta^2\left[c-E^{\nu(t)}\right]\left|\boldsymbol{\xi}^{\nu(t)}\right|^2 \end{aligned} \tag{3.23}$$

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

$$\Gamma \;\equiv\; \frac{1}{N}\,\max_{\mu}\left\{|\boldsymbol{\xi}^{\mu}|^2\right\}_{\mu=1}^{P}, \tag{3.24}$$

where the scaling with dimension N is convenient in the following[4]. Next, we observe that $\Theta^2(x) = \Theta(x)$ for all x. Furthermore we note that

$\Theta[c - E^{\nu(t)}] = 0$ 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. (3.23) to obtain the upper bound

$$Q(t+1) \leq Q(t) \;+\; \frac{2}{N}\,c\,\Theta[c-E^{\nu(t)}] + \frac{1}{N}\,\Gamma\,\Theta[c-E^{\nu(t)}] \tag{3.25}$$

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

$$Q(t) \leq \frac{1}{N}\;(2\,c\;+\;\Gamma)\;\;M(t), \tag{3.26}$$

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

$$R(t) \geq \frac{1}{N}\,c\,M(t) \quad \text{and} \quad Q(t) \leq \frac{1}{N}\,(2c+\Gamma)\,M(t), \tag{3.27}$$

respectively. Exploiting Eq. (3.19) we can write

$$1 \;\geq\; \frac{(\mathbf{w}(t)\cdot\mathbf{w}^*)^2}{(|\mathbf{w}(t)||\mathbf{w}^*|)^2} = \frac{R^2(t)}{Q^*\,Q(t)} \geq \frac{\frac{1}{N^2}\,c^2\,M^2(t)}{Q^*\,\frac{1}{N}(2c+\Gamma)M(t)} \;= \frac{c^2}{Q^*\,N\,(2c+\Gamma)}\,M(t). \tag{3.28}$$

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

We conclude that

$$M(t) \leq M^* = \frac{(2c+\Gamma)\, N\, Q^*}{c^2}, \tag{3.29}$$

where the right hand side involves only constants: N and Γ can be 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. (3.29) 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 was also discussed in the context of Eq. (3.10). In the limit $c \to 0$ with $Q^* \propto c^2$ the upper bound becomes

$$\lim_{c\to 0} M^* \approx \Gamma N \frac{Q^*}{c^2} \tag{3.30}$$

Hence, we can express our finding without reference to a specific value of c in (3.16).

3.4 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.

In the forthcoming sections, we will 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

$$S_T^\mu = S_R^\mu \quad \text{for all} \quad \mu = 1, 2, \ldots, P. \tag{3.31}$$

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 mislabelled 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: Few examples for a non-separable function or examples corrupted by noise can be very well linearly separable

3.4.1 Student-teacher scenarios

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.

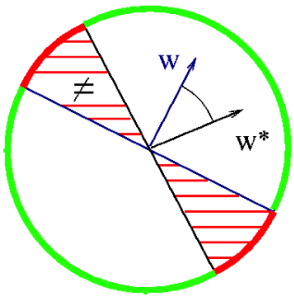


Figure 3.5: Generalization error of the perceptron in a student/teacher scenario. For N-dim. random input vectors generated according to an isotropic density, the probability of disagreement between student vector $\mathbf{w}$ and teacher $\mathbf{w}^*$ is proportional to the red shaded area, i.e. to the angle $\angle(\mathbf{w}, \mathbf{w}^*)$.

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

$$S_R(\boldsymbol{\xi}) = \text{sign}\,(\mathbf{w}^* \cdot \boldsymbol{\xi}) \tag{3.32}$$

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 for instance assume $|\mathbf{w}^*| = 1$ implicitly, without loss of generality.

A perceptron with weights $\mathbf{w}^*$ is always correct[5]. It is therefore referred to as the teacher perceptron (or teacher, for short) and can be thought of providing the example data from which to learn. Likewise, the trained perceptron with adaptive weights $\mathbf{w}$ will be termed the student.

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

$$S_{\mathbf{w}}(\boldsymbol{\xi}) \;=\; \text{sign}\,(\mathbf{w} \cdot \boldsymbol{\xi}) \quad \text{for all } \xi \in I\!R^N. \tag{3.33}$$

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 $I\!D = \{\boldsymbol{\xi}^\mu, S_R^\mu\}$ is to achieve perfect agreement with the unknown teacher in terms of the given examples. However, the agreement should not only concern data in $I\!D$ 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 3.5 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.

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

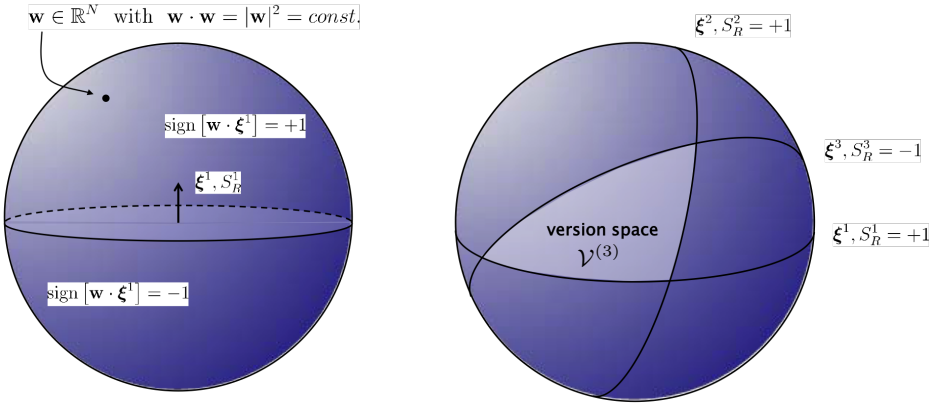


Figure 3.6: Dual geometrical interpretation of linear separability. Illustration in terms of input vectors $\boldsymbol{\xi} \in I\!R^3$ and normalized weight vectors with $|\mathbf{w}|^2 = const.$ **Left:** A single, labelled input $\boldsymbol{\xi}^1$ separates all weight vectors with $S^1_{\mathbf{w}} = +1$ from those with $S^1_{\mathbf{w}} = -1$. **Right:** A set of P labelled input vectors (here: $P = 3$) defines the (lighter) 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.

We assume that a test input $\boldsymbol{\xi}$ is generated according to an isotropic, unstructured density anywhere in $I\!R^N$. 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 red shaded segments, i.e. to the angle $\angle(\mathbf{w}, \mathbf{w}^*)$:

$$\epsilon_g \;=\; \frac{1}{\pi} \arccos\left(\frac{\mathbf{w} \cdot \mathbf{w}^*}{|\mathbf{w}||\mathbf{w}^*|}\right). \tag{3.34}$$

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.

3.4.2 Learning in version space

In the classical set-up 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 $I\!D$.

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 $I\!D$?

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. 3.6 (left panel), we note that every vector $\boldsymbol{\xi}$ also defines a hyperplane through the origin of the N-dim. feature space. This plane separates weights $\mathbf{w} \in I\!R^N$ 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

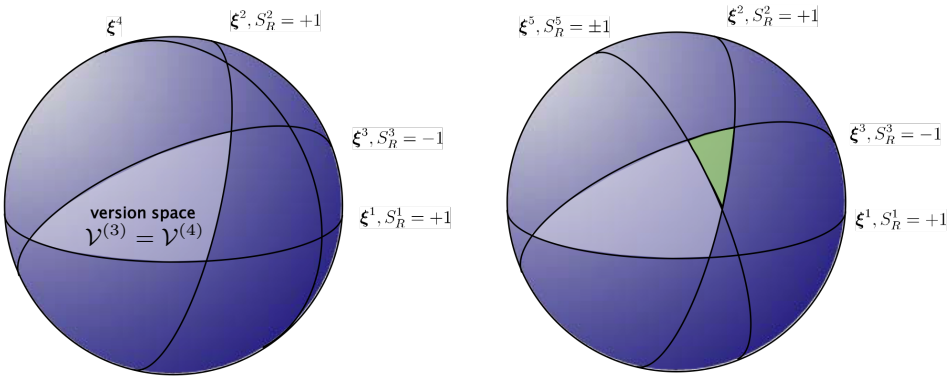


Figure 3.7: Illustration of perceptron learning in version space.
Left: The hyperplane associated with the additional example $\boldsymbol{\xi}^4$ does not intersect the version space $\mathcal{V}^{(3)}$. Consequently, all weight vectors in $\mathcal{V}^{(3)}$ already classify $\boldsymbol{\xi}^4$ correctly 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 or the remaining light area, depending on the actual target S_R^5.

label $S_R(\boldsymbol{\xi})$, the plane orthogonal to $\boldsymbol{\xi}$ separates correct from wrong students in N-dimensional weight space.

Consequently, a set $I\!D$ of P labelled 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 illustrated in Fig. 3.6 (right panel).

The set of all perceptron weight vectors which are consistent with the P examples in $I\!D$, 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

$$\mathcal{V} = \left\{ \mathbf{w} \in I\!R^N,\, \mathbf{w}^2 = 1 \;\middle|\; \mathrm{sign}(\mathbf{w} \cdot \boldsymbol{\xi}^\mu) = S_R^\mu \;\text{ for all }\; \{\boldsymbol{\xi}^\mu, S_R^\mu\} \in I\!D \right\}. \tag{3.35}$$

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

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 omitted.

The version space is non-empty, $\mathcal{V} \neq \emptyset$, if and only if $I\!D$ is linearly separable: If a (normalized) teacher vector $\mathbf{w}^*$ defines the linearly separable rule represented by the examples in $I\!D$, 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 $I\!D$, 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 $I\!D$ 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 (3.16) this is always possible, e.g. by means of the Rosenblatt perceptron algorithm[6]. In our low-dimensional illustration, Fig. 3.6 (right panel), this means that we can always place a student vector $\mathbf{w}$ somewhere in the lighter region representing $\mathcal{V}^{(3)}$ for $P = 3$ training examples.

[6] ... with subsequent normalization in order to match the definition (3.35) precisely.

Now, consider a fourth labelled example $\{\boldsymbol{\xi}^4, S_R^4\}$ as displayed in Fig. 3.7 (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 $\text{sign}(\mathbf{w} \cdot \boldsymbol{\xi}^4)$.

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 $S_R^4 = \text{sign}(\mathbf{w} \cdot \boldsymbol{\xi}^4)$ 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 to modify the hypothesis or select 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. 3.7. 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 rather mild assumptions on the properties of N-dimensional feature vectors.

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 (3.34), it will achieve perfect generalization $\epsilon_g \to 0$ for $P \to \infty$.

Our elementary, illustrative considerations do not enable us to obtain more concrete mathematical relations which quantify the generalization error as a function of the training set size P. Here, we can only refer the reader to the literature. One can show, for instance, that under rather mild assumptions on the input data, the generic asymptotic generalization behavior is given by

$$\epsilon_g \propto \left(\frac{P}{N}\right)^{-1} \quad \text{as} \quad \left(\frac{P}{N}\right) \to \infty \tag{3.36}$$

for linearly separable rules learnt from P examples in N-dimensional feature space.

This result has been obtained by so-called counting arguments which determine the number of linearly separable functions of P feature vectors in N dimensions under quite general assumptions [36–38]. Alternatively, 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 (3.36) of the generalization error on (P/N), see [5, 22, 42] and references therein.

3.4.3 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. 3.5. 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

$$\mathbf{w}_{cm} = \int_{\mathcal{V}} \mathbf{w} \; d^N w. \tag{3.37}$$

By definition, it has the smallest average 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.

In principle, the definition (3.37) 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 [43]

$$\mathbf{w}_{cm}^{(est)} \; = \; \frac{1}{M} \sum_{i=1}^{M} \mathbf{w}^{(i)}. \tag{3.38}$$

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., [43–45].

3.5 The perceptron of optimal stability

Here we approach the question of how to choose (and define) a *good* or even optimal weight vector in version space from a different perspective. It avoids the explicit computation of the center of mass of $\mathcal{V}$ and leads to a well-defined problem of quadratic optimization.

3.5.1 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

$$\kappa^\mu \; = \; \frac{E^\mu}{|\mathbf{w}|} \; = \; \frac{\mathbf{w} \cdot \boldsymbol{\xi}^\mu \, S_T^\mu}{|\mathbf{w}|}. \tag{3.39}$$

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 distance of the input vector from the separating hyperplane, see Fig. 3.8. 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 $\mathbb{D}$:

$$\kappa(\mathbf{w}) \; = \; \min \left\{ \kappa^\mu \right\}_{\mu=1}^{P}. \tag{3.40}$$

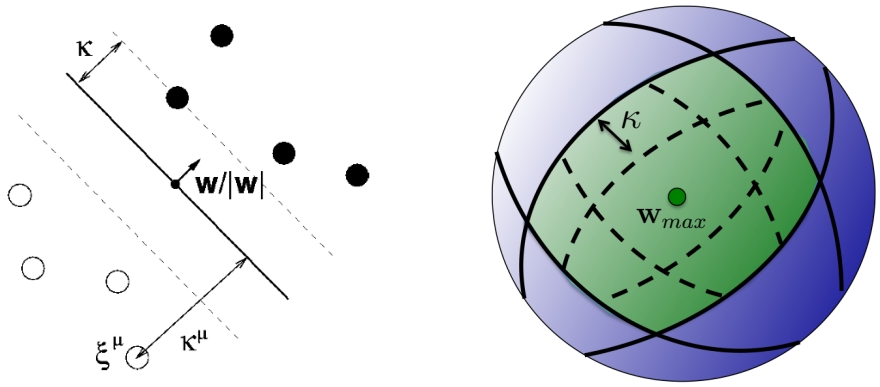


Figure 3.8: Stability of the perceptron. **Left:** The stability κ^μ, defined in Eq. (3.39), corresponds to the oriented distance of $\boldsymbol{\xi}^\mu$ from the plane orthogonal to $\mathbf{w}$. The stability of the perceptron $\kappa(\mathbf{w})$ is defined as the smallest κ^μ in the set of examples, 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 $\mathbf{w}_{max}$.

Note that if the perceptron does not separate the classes correctly (yet), $\kappa(\mathbf{w}) < 0$ 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 $I\!D$. 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. (3.39). Therefore, we can also define the perceptron of optimal stability without requiring linear separability of the data set $I\!D = \{\boldsymbol{\xi}^\mu, S_T^\mu\}$ with more general targets $S_T(\boldsymbol{\xi}^\mu) = \pm 1$.

Hence, the perceptron of optimal stability[7] is the target of the following problem:

PERCEPTRON OF OPTIMAL STABILITY (3.41)

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

with $\mathbf{w}_{max} = \underset{\mathbf{w}\in I\!R^N}{\text{argmax}}\ \kappa(\mathbf{w})$ 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 [45] 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. 3.4.3, 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,

[7] Not very precisely termed the *optimal perceptron* in the literature, occasionally.

symmetric shape. However, one can expect that $\mathbf{w}_{cm}$ is quite close to the true center of mass, generically, 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,

3.5.2 The MinOver algorithm

An intuitive algorithm that can be shown to achieve optimal stability for a given lin. sep. data set $I\!D$ has been suggested in [46]. The so-called MinOver algorithm performs Hebbian updates for the currently least stable example in $I\!D$. We assume here *tabula rasa* initialization, i.e. $\mathbf{w}(0) = 0$, but more general initial states could be considered. The update is given as follows:

MinOver algorithm (3.42)

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 $I\!D$
- 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

$$\mathbf{w}(t+1) \;=\; \mathbf{w}(t) + \frac{1}{N} \, \boldsymbol{\xi}^{\widehat{\mu}} \, S_T^{\widehat{\mu}} \tag{3.43}$$

$$\left[\text{ or, equivalently, increment the corresponding embedding strength } x^{\widehat{\mu}}(t+1) = x^{\widehat{\mu}}(t) + 1. \right] \tag{3.44}$$

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 $\kappa^\mu(t) = E^\mu(t)/|\mathbf{w}(t)|$.

A few remarks:

- According to the original presentation of the algorithm [46], 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 [46], optimal stability is only achieved in the limit $c \to \infty$, which is equivalent to (3.43).

- In the above formulation (3.43), which corresponds to the limit $c \to \infty$ in the previous remark, MinOver updates the weights (or embedding strengths) even if all examples in $I\!D$ 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 the norm $|\mathbf{w}(t)|$ only are disregarded as they do not affect the classification or its stability.

- From the definition of the MinOver algorithm (3.43) 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 [46], the MinOver algorithm converges and yields the perceptron weights of optimal stability, if $I\!D$ is linearly separable. The proof of convergence is similar in spirit to that of the Perceptron Convergence Theorem, cf. (3.16). We refrain from reproducing it here. Instead, we show only that the perceptron of optimal stability can always be written in the form

$$\mathbf{w}_{max} = \frac{1}{N}\sum_{\mu=1}^{P} x_{max}^{\mu}\,\boldsymbol{\xi}^{\mu}\,S_T^{\mu} \quad \text{with embedding strengths } \{x_{\max}^{\mu} \in I\!R\}_{\mu=1}^{P}\,. \tag{3.45}$$

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

$$\mathbf{w}_1 = \sum_{\mu=1}^{P} x_1^{\mu}\boldsymbol{\xi}^{\mu}\,S_T^{\mu} \quad \text{with embedding strengths } \{x_1^{\mu}\}_{\mu=1}^{P}\,, \tag{3.46}$$

while for the second weight vector we assume that

$$\mathbf{w}_2 = \mathbf{w}_1 + \boldsymbol{\delta} \;\text{ with }\; |\boldsymbol{\delta}| > 0 \;\text{ and }\; \boldsymbol{\delta}\cdot\boldsymbol{\xi}^{\mu} = 0 \;\text{ for all } \mu = 1, 2, \ldots, P. \tag{3.47}$$

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

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

$$E_2^{\mu} = \mathbf{w}_2\cdot\boldsymbol{\xi}^{\mu}S_T^{\mu} = \mathbf{w}_1\cdot\boldsymbol{\xi}^{\mu}S_T^{\mu} + \underbrace{\boldsymbol{\delta}\cdot\boldsymbol{\xi}^{\mu}S_T^{\mu}}_{=0} = E_1^{\mu}. \tag{3.48}$$

On the other hand we have

$$|\mathbf{w}_2|^2 = |\mathbf{w}_1+\delta|^2 = |\mathbf{w}_1|^2 + 2\,\underbrace{\mathbf{w}_1\cdot\boldsymbol{\delta}}_{=0} + \underbrace{|\boldsymbol{\delta}|^2}_{>0} \quad\Rightarrow\quad |\mathbf{w}_2| > |\mathbf{w}_1|, \tag{3.49}$$

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

$$\kappa_2^{\mu} = \frac{E_2^{\mu}}{|\mathbf{w}_2|} = \frac{E_1^{\mu}}{|\mathbf{w}_2|} < \frac{E_1^{\mu}}{|\mathbf{w}_1|} = \kappa_1^{\mu} \;\text{ for all } \mu, \text{ and therefore } \kappa_2 < \kappa_1. \tag{3.50}$$

We conclude that any contribution orthogonal to all $\boldsymbol{\xi}^{\mu}$ inevitably reduces the stability of the weight vector $\mathbf{w}_1$. This implies that maximum stability is indeed achieved by weights of the form (3.45). 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. (3.47) cannot exist for $P > N$, in general. Obviously, if $span\left(\{\boldsymbol{\xi}^{\mu}\}_{\mu=1}^{P}\right) = I\!R^N$, any N-dimensional vector including $\mathbf{w}_{max}$ can be written as a linear combination. In this case, Eq. (3.45) holds true, trivially.

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 $x_{max} \geq 0$. We will recover this result more formally in the next section.

3.6 Perceptron training by quadratic optimization

Here we will exploit the fact that the problem of 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 [47]. 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 [39–41] Before deriving and discussing this training scheme we re-visit another prototypical training scheme from the early days of neural network models: B. Widrow and M.E. Hoff's *Adaptive Linear Neuron* or Adaline [48, 49].

3.6.1 Optimal stability re-formulated

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:

$$\underset{\mathbf{w} \in \mathbb{R}^N}{\text{maximize}} \quad \kappa(\mathbf{w}) \text{ where } \kappa(\mathbf{w}) = \min \left\{ \kappa^\mu = \frac{E^\mu}{|\mathbf{w}|} = \frac{\mathbf{w}^\top \boldsymbol{\xi}^\mu S_T^\mu}{|\mathbf{w}|} \right\}_{\mu=1}^P \tag{3.51}$$

which is just a more compact version of (3.41).

Obviously, the stability κ can be made larger by increasing the 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:

$$\underset{\mathbf{w} \in \mathbb{R}^N}{\text{minimize}} \quad \frac{N}{2} \mathbf{w}^2 \text{ subject to inequality constraints } \{E^\mu \geq 1\}_{\mu=1}^P \tag{3.52}$$

Hence, we have re-written 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 (3.52) is irrelevant for the definition of the problem but is kept for convenience and consistency of notation.

3.6.2 The Adaptive Linear Neuron - Adaline

The problem of optimal stability in the formulation (3.52) involves a system of inequalities. Before we address its solution in the following sections, we resort to the more familiar case of equality constraints, i.e. we consider the simpler optimization problem

$$\underset{\mathbf{w} \in \mathbb{R}^N}{\text{minimize}} \quad \frac{N}{2} \mathbf{w}^2 \text{ subject to constraints } \{E^\mu = 1\}_{\mu=1}^P . \tag{3.53}$$

Historically, this relates to the the so-called *Adaptive Linear Neuron* or Adaline model which was introduced by B. Widrow and M.E. Hoff in 1960 [48]. Like the Rosenblatt Perceptron, it constitutes one of the earliest artificial neural network models and truly groundbreaking work in the area of machine learning.[8]

[8] A series of videos about B. Widrow's pioneering work is presented in the youtube channel "widrowlms" [50].

Widrow realized learning Adaline systems in hardware as "resistors with memory" and introduced the term *Memistor*.[9] The concept was also extended to layered Madaline (Many Adaline) networks [49] consisting of several linear units which were combined in a majority vote for classification.

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. 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 [47]. 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

$$\mathcal{L}\Big(\mathbf{w}, \{\lambda^\mu\}_{\mu=1}^P\Big) = \frac{N}{2}\mathbf{w}^2 - \sum_{\mu=1}^P \lambda^\mu \left(\mathbf{w}^\top \boldsymbol{\xi}^\mu S_T^\mu - 1\right). \tag{3.54}$$

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 (3.53) read

$$\nabla_w \mathcal{L}|_* = 0 \quad \Rightarrow \quad \mathbf{w}^* = \frac{1}{N}\sum_{\mu=1}^P \lambda^{*\mu}\, \boldsymbol{\xi}^\mu S_T^\mu \tag{3.55}$$

$$\text{and} \quad \left.\frac{\partial \mathcal{L}}{\lambda^\mu}\right|_* = 0 \quad \Rightarrow \quad E^{*\mu} = \mathbf{w}^{*\top} \boldsymbol{\xi}^\mu S_T^\mu = 1 \ \text{ for all } \mu \tag{3.56}$$

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 according to the theory of Lagrange parameters, the function $\mathcal{L}$ is minimized in $\mathbf{w}$ but maximized with respect to the λ^μ [47].

The second condition (3.56) merely reproduces the original equality constraints, which have to be satisfied by any candidate solution, obviously. The first, more interesting stationarity condition (3.55) implies that the formally introduced Lagrange parameters can be identified as the embedding strengths of the weights and re-named accordingly. Hence, we can eliminate the weights from $\mathcal{L}$ to obtain

$$\begin{aligned}\mathcal{L}\Big(\{\mathbf{x}^\mu\}_{\mu=1}^P\Big) &= \frac{1}{2N}\sum_{\mu,\nu} x^\mu S_T^\mu \boldsymbol{\xi}^{\mu\top} \boldsymbol{\xi}^\nu S_T^\nu x^\nu - \sum_\mu x^\mu \left[\frac{1}{N}\sum_\nu x^\nu S_T^\nu \boldsymbol{\xi}^\nu\right]^\top \boldsymbol{\xi}^\mu S_T^\mu + \sum_\mu x^\mu \\ &= -\frac{1}{2N}\sum_{\nu,\mu=1}^P x^\mu S_T^\mu \boldsymbol{\xi}^{\mu\top}\, \boldsymbol{\xi}^\nu S_T^\nu x^\nu + \sum_{\mu=1}^P x^\mu. \end{aligned} \tag{3.57}$$

It turns out useful to resort to a compact notation which also 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

$$C = C^\top \in I\!R^{P\times P} \ \text{ with elements } \ C^{\mu\nu} = \frac{1}{N} S_T^\mu S_T^\nu\, \boldsymbol{\xi}^{\mu\top} \boldsymbol{\xi}^\nu \tag{3.58}$$

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

We will also write $\vec{a} > \vec{b}$ in order to indicate that $a^\mu > b^\mu$ for all $\mu = 1, 2, ...P$ and use analogous notations for the relations "$<$", "$\geq$" and "$\leq$".

[9] not to be confused with the more recent concept of *Memristor* elements [51]

In the convenient matrix-vector notation, we have furthermore

$$E^\nu = \frac{1}{N}\sum_{\mu=1}^{P} x^\mu \underbrace{S_T^\mu S_T^\nu \, \boldsymbol{\xi}^{\mu\top}\boldsymbol{\xi}^\nu}_{NC^{\mu\nu}} = [C\vec{x}]^\nu \quad \text{i.e} \quad \vec{E} = C\vec{x} \tag{3.59}$$

$$\mathbf{w}^2 = \frac{1}{N^2}\sum_{\mu,\nu} x^\mu x^\nu \overbrace{S_T^\mu S_T^\nu \boldsymbol{\xi}^{\mu\top}\boldsymbol{\xi}^\nu} = \frac{1}{N}\vec{x}^\top C\,\vec{x} = \frac{1}{N}\vec{x}^\top\vec{E}. \tag{3.60}$$

In this compact notation, the Lagrange function (3.57) 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 $\vec{x}$. Consequently we can re-formulate the optimization problem (3.61) after having exploited the stationarity conditions as the following unconstrained maximization:

$$\underset{\vec{x}\,\in\,\mathbb{R}^P}{\text{maximize}} \quad f(\vec{x}) = -\frac{1}{2}\,\vec{x}^\top C\vec{x} + \vec{x}^\top\vec{1}. \tag{3.61}$$

Compared to (3.53), the cost function appears slightly more complicated. In turn, however, the constraints are eliminated. While completely equivalent with (3.53), the re-written problem is given in terms of the embedding strengths $\vec{x} \in \mathbb{R}^P$.

In absence of constraints, it is straightforward to maximize $f(\vec{x})$ and we could resort to a variety of methods. Here, we discuss simple gradient ascent 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

$$\vec{x}(t+1) = \vec{x}(t) + \eta\nabla_x f = \vec{x}(t) + \eta\left(\vec{1} - \vec{E}(t)\right) \tag{3.62}$$

$$\left[\ \text{or} \quad \mathbf{w}(t+1) = \mathbf{w}(t) + \frac{\eta}{N}\sum_{\mu=1}^{P}\left(1 - E^\mu(t)\right)\boldsymbol{\xi}^\mu S_T^\mu \ \right] \tag{3.63}$$

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.

As an important alternative to the parallel algorithm, sequential gradient-based methods can be devised, which present the example data repeatedly in, for instance, deterministic sequential order and update only the corresponding embedding strength in each step with learning rate $\widetilde{\eta}$:

ADALINE algorithm, sequential updates (repeated presentation of $\mathbb{D}$)
– at time step t, present example[10] $\mu = 1, 2, 3, ..., P, 1, 2, 3, ...$
– perform the update

$$x^\mu(t) = x^\mu(t) + \widetilde{\eta}\,(1 - E^\mu(t)) \tag{3.64}$$

$$\left[\ \text{or} \quad \mathbf{w}(t+1) = \mathbf{w}(t) + \frac{\widetilde{\eta}}{N}\left(1 - E^\mu(t)\right)\boldsymbol{\xi}^\mu S_T^\mu. \ \right] \tag{3.65}$$

[10] More precisely we should use the index $\mu(t)$ for the current example. For the sake of brevity we omit the explicit time-dependence.

Both versions of the Adaline algorithm can be shown to converge for suitable choices of the learning rate. In the parallel version, η has to be small enough enough to ensure convergence. If $E^\mu = 1$ can be achieved for all μ, the sequential version finds the solution for $0 < \widetilde{\eta} < 2$ with the canonical choice $\widetilde{\eta} = 1$. The convergence properties and conditions will be discussed in greater detail when we re-visit gradient-based learning in the context of more general regression problems.

Remark: relation to the SSE in linear regression

It is interesting to note that (3.63) cannot be interpreted as a gradient ascent in weight space along $\nabla_w f$ of the cost function (3.61). It is the equivalent of (3.62) in weight space, but the projection does not preserve the gradient property, in general. In fact, (3.63, 3.65) correspond to gradient based methods for a different, yet related cost function: The Sum of Squared Errors (SSE), cf. Eq. (2.5), for linear regression with the special target values $S_T^\mu = \pm 1$

$$E^{SSE} = \frac{1}{2}\sum_{\mu=1}^{P}(1-E^\mu)^2 = \frac{1}{2}\sum_{\mu=1}^{P}\left(S_T^\mu - \mathbf{w}^\top \boldsymbol{\xi}^\mu\right)^2 \quad \text{with } \nabla_w E^{SSE} = \sum_{\mu=1}^{P}(1-E^\mu)\boldsymbol{\xi}^\mu S_T^\mu. \tag{3.66}$$

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

The sequential algorithm (3.65) in weight space is equivalent to Widrow and Hoff's original LMS (Least Means Square) method for the Adaline [48, 50]. The LMS, also referred to as the *delta-rule* or the *Widrow-Hoff algorithm* in the literature, can be seen as one of the most important ancestors of gradient-based training methods such as the prominent *backpropagation of error* for multilayered neural networks. A review of the history and conceptual relations between these classical algorithms is given in [49].

3.6.3 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 facilitates negative Hebbian updates 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-productive feature of the Adaline is corrected for in the Adaptive Perceptron (AdaTron) algorithm [39–41]. It retains the concept of adaptivity but takes into account the inequality constraints in problem (3.52), 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 [47]. 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 (3.52), which we repeat here for clarity:

PERCEPTRON OF OPTIMAL STABILITY (weight space)

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

Formally, we have to consider the same Lagrange function as for equality constraints, already given in Eq. (3.54):

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

The Kuhn-Tucker Theorem of optimization theory [47] provides the first order necessary stationarity conditions for general, non-linear optimization problems with inequality and/or equality conditions. In our special case, these so-called Kuhn-Tucker (KT) conditions, sometimes referred to as Karush-Kuhn-Tucker conditions, read

$$\mathbf{w}^* = \frac{1}{N}\sum_{\mu=1}^{P} \lambda^{*\mu}\,\boldsymbol{\xi}^\mu S_T^\mu \qquad \text{embedding strengths } \lambda^\mu \qquad (3.67)$$

$$E^{*\mu} = \mathbf{w}^{*\top}\boldsymbol{\xi}^\nu S_T^\mu \geq 1 \;\text{ for all } \mu \qquad \text{linear separability} \qquad (3.68)$$

$$\lambda^{*\mu} \geq 0 \;\; (\text{not all } \lambda^{*\mu} = 0) \qquad \text{non-negative multipliers} \qquad (3.69)$$

$$\lambda^{*\mu}\;(1 - E^{*\mu}) = 0 \;\text{ for all } \mu \qquad \text{complementarity.} \qquad (3.70)$$

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 (3.45). 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 (3.68) merely reflects the original constraint, i.e. linear separability.

Intuitively, the non-negativity of the multipliers, KT-condition (3.69), 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 (3.67), we formally recover the insight from Sec. 3.5.2 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 (3.52) as a KT-point. Using our convenient matrix-vector notation, the KT conditions conditions read

$$\vec{E}^* = C\vec{x}^* \geq \vec{1} \qquad \text{linear separability} \qquad (3.71)$$

$$\vec{x}^* \geq 0 \quad (\vec{x}^* \neq 0) \qquad \text{non-negative embeddings}^7 \qquad (3.72)$$

$$x^{*\mu}\,(E^{*\mu} - 1) = 0 \;\text{ for all } \mu \qquad \text{complementarity.} \qquad (3.73)$$

The most interesting condition is that of complementarity (3.70) and (3.73). It states that at optimal stability, any example with non-zero embedding will have $E^{*\mu} = 1$ while input/output pairs with $E^{*\mu} > 1$ do not contribute to the linear

[10] Obviously $\vec{x} = 0$, $\mathbf{w} = 0$ cannot be a solution of the problem.

combination (3.45). We will discuss this property in greater detail later. Complementarity also implies that $x^{*\mu}E^{*\mu} = x^{*\mu}$ for all μ and, therefore

$$\vec{x}^{*\top} C \vec{x}^* = \vec{x}^{*\top} \vec{E}^* = \sum_{\mu=1}^{P} x^{*\mu} E^{*\mu} = \sum_{\mu=1}^{P} x^{*\mu} = \vec{x}^{*\top} \vec{1}. \tag{3.74}$$

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:

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

Setting $\vec{u} = \vec{x}_1 - \vec{x}_2$ we obtain with (3.74):

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

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:

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

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 for singular matrices C despite $C\vec{x}_1 = C\vec{x}_2$, the solution is unique in terms of the weights. Moreover, this finding implies that any local solution (KT-point) of the problem (3.52) 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 [47].

The stationarity conditions also facilitate a re-formulation of the optimization problem. It amounts to the elimination of the weights in terms of the Lagrange multipliers, as in the Adaline. One arrives at a special case of what is known as the Wolfe Dual in optimization theory, see [47] for the general definition and proof of the corresponding Duality Theorem. It is frequently applied in order to rewrite a given problem in terms of a modified cost function with simplified constraints.

Without going into detail we only note that the following formulation is fully equivalent to problem (3.52):

PERCEPTRON OF OPTIMAL STABILITY (embedding strengths, dual problem)

$$\underset{\vec{x}}{\text{maximize}} \;\; f(\vec{x}) \; = \; -\frac{1}{2} \vec{x}^{\top} C \vec{x} + \vec{x}^{\top} \vec{1} + \quad \text{subject to } \vec{x} \geq 0. \tag{3.75}$$

Hence, the resulting dual problem still comprises constraints, albeit simpler ones, which can be taken care of by restricting the search to the hyperoctant $\vec{x} \geq 0$ of non-negative embedding strengths.

It is straightforward to construct a corresponding *projected gradient* ascent [47]. Again, we can perform sequential or parallel updates. In the former case, we obtain the following prescription:

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

– at time step t, present example $\mu = 1, 2, 3, ..., P, 1, 2, 3, ...$

– perform the update

$$x^\mu(t+1) \quad = \quad \max\left\{ 0, \quad x^\mu(t) + \widehat{\eta}\Big(1 - [C\vec{x}(t)]^\mu\Big) \right\} \tag{3.76}$$

with learning rate $\widehat{\eta}$.

The name AdaTron has been coined for this Adaptive Perceptron algorithm [39–41]. By comparison with the sequential Adaline algorithm (3.64) 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 $+\widehat{\eta}\,(1 - E^\mu)$ 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. The sequential AdaTron algorithm satisfies the constraint $\vec{x} \geq 0$ by simply *clipping* each x^μ to *zero* whenever the gradient ascent based update would lead into the excluded region of $x^\mu < 0$.

The sequential AdaTron algorithm can be shown to converge and yield optimal stability for $0 < \widehat{\eta} < 2$ if the data set $\mathbb{D}$ is linearly separable [39–41]. The proof is based on the following key insights:

a) For lin. sep. $\mathbb{D}$, the cost function $f(\vec{x})$ is bounded from above in $\vec{x} \geq 0$. This is a consequence of Farkas' lemma for inhomogeneous systems, see [47] for details.

b) It is straightforward to show that the function $f(\vec{x})$ increases monotonically under non-zero, sequential AdaTron updates with $0 < \widehat{\eta} < 2$ [39–41].

c) Obviously, any KT-point $\vec{x}^*$ of the problem, cf. Eq. (3.71–3.73), is a fixed point (with *zero update*) of the AdaTron and *vice versa.*

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

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 $\widehat{\eta}$, the cost function will also decrease monotonically [39–41]. We refrain from giving a more explicit mathematical formulation and discussion of the parallel updates, here.

In contrast to the Rosenblatt or Adaline algorithm, it is not straightforward to re-write 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.

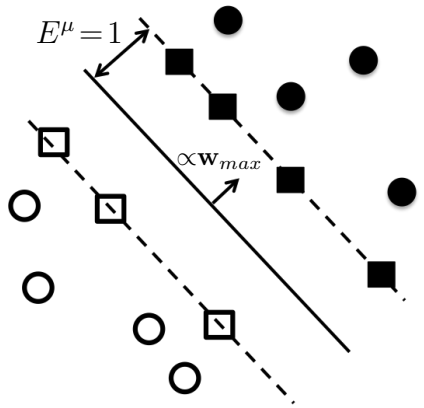


Figure 3.9: Support Vectors in the perceptron of optimal stability.
The arrow represents the normalized weight vector $\mathbf{w}_{max}/|\mathbf{w}_{max}|$. Class membership is indicated by filled ($S_T = +1$) and open symbols ($S_T = -1$), respectively. Support vectors, marked as squares, fall into the two hyperplanes with $E^\mu = 1, i.e.\ \kappa^\mu = \kappa_{\max}$. All other examples (circles) display greater stability without being embedded explicitly.

Remarks

Non-separable data
The outcome of the AdaTron training (3.76) for data that is not linearly separable is not obvious. Modifications of the SSE (3.66) 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. We refer to Sec. 4.1 for a more detailed discussion.

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 support vectors.

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 J.C. Platt's *Sequential Minimal Optimization (SMO)* approach [52]. It is the basis of many implementations, see e.g. [53] for links and the SVM literature for further discussions and references [29–32].

3.6.4 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 [54] for just one 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 [29–32], see Sec. 4.3.

One of the most important observations with respect to optimal stability is a consequence of the complementarity condition (3.73). It implies that in a KT-point we have

$$\text{either} \quad \left\{ \begin{array}{c} E^\mu = 1 \\ x^\mu \geq 0 \end{array} \right\} \quad \text{or} \quad \left\{ \begin{array}{c} E^\mu > 1 \\ x^\mu = 0 \end{array} \right\}. \tag{3.77}$$

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 of outputs, see Fig. 3.9. Only these examples, marked in red in the figure, contribute to the linear combination (3.45) 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 (3.45).

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 (3.61) 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, implicitly. 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.

3.7 Concluding 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 will therefore consider a number of ways to deal with non-separable data sets and address classification problems beyond linear separability.

Interestingly, the perceptron still ranks among the most frequently applied machine learning tools in practice. 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

Nothing is more practical than a good theory.

– Vladimir Vapnik

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 mislabelled 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 mislabelled examples, for instance in the presence of noise. Depending on the number P of examples and the degree of the corruption the following may occur:

   i.1) In particular small data sets $\mathbb{D}$ with few mislabelled 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) In particular small data sets may be linearly separable and one can expect/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.

In practice, it is a difficult task 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. Nabutovsky and Domany present a perceptron-like algorithm which either finds a lin. sep. solution or establishes non-separability [55].

In the following we discuss several strategies for coping with non-separable data sets, addressing the following basic ideas:

- 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. For the latter, 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 [56, 57]. So-called *Random Forests*, i.e. ensembles of decision trees [58], constitute one of the most prominent examples in the literature, currently. We refrain from a detailed discussion of ensembles and refer the point to the literature [56,57]. 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.

3.6. For a detailed discussion in the context of multilayered networks we refer to the lectures on deep learning [1].

- Prototype-based systems [59] offer another, very powerful framework for classification beyond linear separability. In chapter 5 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 $I\!D = \{\boldsymbol{\xi}^\mu, S_T^\mu\}$ with $\boldsymbol{\xi}^\mu \in I\!R^N$ 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 $\mathbf{w} \in I\!R^N$ in a simple perceptron student. Formally, the corresponding optimization problem reads

MINIMAL NUMBER OF ERRORS (PERCEPTRON) (4.1)

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

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

While the idea of minimizing H^{err} appears plausible at a glance, we should realize that the outcome should be very sensitive to individual, misclassified examples in $I\!D$. The 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 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 and can be seen as a modification of the Rosenblatt algorithm (3.15) has been introduced by S.I. Gallant [60]. 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 $\widehat{H} = 0$
- at time step t, select a single feature vector $\boldsymbol{\xi}^\mu$ with class label S_T^μ randomly from the data set $I\!D$ 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)

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

- 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

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

$$\widehat{H}(t+1) \;=\; H^{err}\big(\widehat{\mathbf{w}}(t+1)\big)$$

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* [60, 61].

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 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. 3.6.2, the gradient based minimization of several cost functions similar to Eq. (3.66) has been considered in the literature. For instance [62] discusses objective functions which correspond to

$$H^{[k]}(\mathbf{w}) = \sum_{\mu=1}^{P} (c - E^\mu)^k\, \Theta[c - E^\mu] \;\text{ with } E^\mu = \mathbf{w} \cdot \boldsymbol{\xi}^\mu S_T^\mu$$

in our notation. Note that the limiting case $c = 0, k \to 0$ would recover the non-differentiable H^{err}, Eq. (4.1). In [62] 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 $H^{[2]}(\mathbf{w}) = 0$, 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. [29] 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 (3.52) 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 $\beta^\mu \in I\!R$ 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 (3.52) if $\beta^\mu = 0$ 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 misclassifications ($E^\mu < 0$), potentially. In (4.3), the Lagrange multiplier $\gamma \in I\!R$ controls to which extent 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 re-write the problem in terms of embedding strengths:

SOFT MARGIN PERCEPTRON (embedding strengths)

$$\underset{\vec{x},\vec{\beta}}{\text{minimize}} \quad \frac{1}{2}\,\vec{x}^\top C \vec{x} \,+\, \gamma \vec{\beta}^\top \vec{1} \qquad \text{subject to} \quad \vec{E} \geq \vec{1} - \vec{\beta} \quad \text{and} \quad \vec{\beta} \geq 0 \tag{4.4}$$

Here, the derivation of the Wolfe Dual [47] amounts to the elimination of the slack variables $\vec{\beta}$. Similar to the error-free case, Sec. 3.6, 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 (3.52) 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.

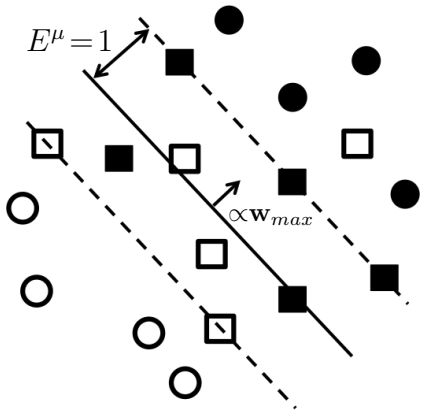


Figure 4.1: Support vectors in the soft margin perceptron.
The arrow represents the normalized weight vector $\mathbf{w}_{max}/|\mathbf{w}_{max}|$. Filled and open symbols correspond to the classes $S_T^\mu = \pm 1$, respectively. Support vectors, displayed as squares, fall onto the two hyperplanes with $E^\mu = 1$, 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.

In analogy to the derivation of the AdaTron algorithm (3.76) we can devise a similar, sequential projected-gradient descent algorithm:

AdaTron with errors, sequential updates (repeated presentation of $I\!D$)

– at time step t, present the example $\mu = 1, 2, ..., P, 1, 2, ...$
– perform the update

$$\begin{aligned} \widetilde{x}^\mu(t+1) &= x^\mu(t) + \widehat{\eta}\,(1 - [C\vec{x}(t)]^\mu) && \text{gradient step} \\ \widehat{x}^\mu(t+1) &= \max\{0, \widetilde{x}^\mu(t+1)\} && \text{non-negative embeddings} \\ x^\mu(t+1) &= \min\{\gamma, \widehat{x}^\mu(t+1)\} && \text{with limited magnitude} \end{aligned} \quad (4.6)$$

Compared to the original AdaTron for separable data, the only difference is the restriction of the search to the region

$$0 \leq \vec{x} \leq \gamma\,\vec{1}.$$

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 $E^\mu < 1$ or is even misclassified with $E^\mu < 0$. If some of the errors cannot be corrected because $I\!D$ 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 (1.2) 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 -

[1] the violation of $E^\mu \geq 1$, to be more precise

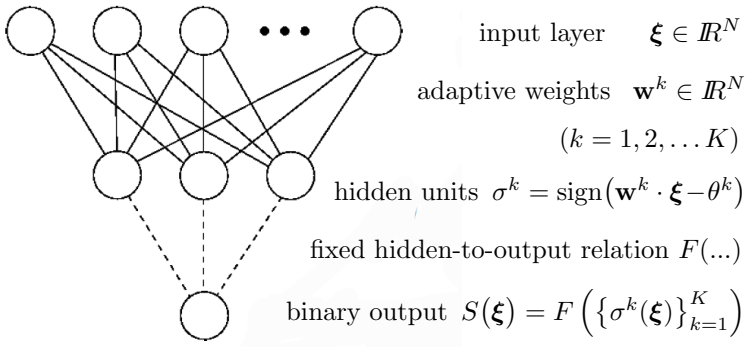


Figure 4.2: The architecture of a "machine" as introduced in Sec. 4.2. A number K of hidden units of the perceptron type are connected by adaptive weights with the N-dim. input layer, $K = 3$ in the illustration. The binary response $S(\boldsymbol{\xi})$ is determined by a pre-defined, fixed functional dependence $F\left(\sigma^1, \sigma^2, ...\sigma^K\right)$.

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. 3.6.4 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.

4.2 Multi-layer 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 [22, 38, 42, 63–66] and references therein.

As illustrated in Fig. 4.2) the architecture of a *machine* is outlined. It comprises

- an input layer representing feature vectors $\boldsymbol{\xi} \in I\!R^N$
- 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 $\mathbf{w}^k \in I\!R^N$ and local thresholds[2] $\theta^k \in I\!R^N$
- 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 adaptive $\mathbf{w}^k$ connecting input and hidden layer.

[2]Thresholds could be replaced by a formal weight fro a clamped input as outline in (3.4).

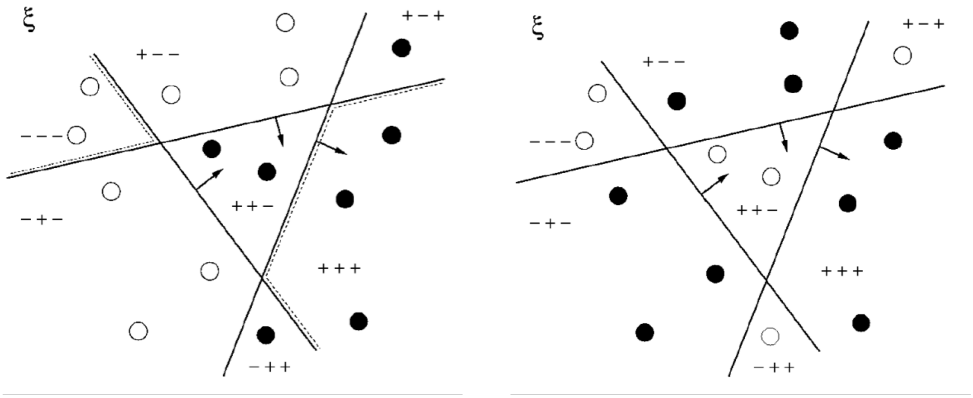


Figure 4.3: Illustration of the output functions of Committee machine and the Parity machine, applied to identical sets of feature vectors $\boldsymbol{\xi}$. In both machines, $K = 3$ oriented hyperplanes tesselate the feature space. Arrows mark the respective half-space of $\sigma^k = +1$. The networks' 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. Dotted lines mark pieces of the hyperplanes which do not separate outputs $S = +1$ from $S = -1$. In **Right panel:** In the PM, the total output is $S = \prod_k \sigma^k$. Every hyperplane separates total outputs $S = \pm 1$ locally, everywhere.

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. 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 [66].

PM: In the parity machine the output is computed as a product over the hidden unit states σ^k [22, 42, 65]:

$$F^{PM}\left(\{\sigma^k\}_{k=1}^K\right) = \prod_{k=1}^K \sigma^k \Rightarrow S^{PM}(\boldsymbol{\xi}) = \prod_{k=1}^K \text{sign}\left[\mathbf{w}^k \cdot \boldsymbol{\xi}\right] \tag{4.8}$$

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.

[3]Interestingly, the gist of Minsky and Papert's book [25] is often reduced to this single insight, which is not only a gross injustice but also largely irrelevant.

On the contrary, the committee machine can be interpreted as a conventional two-layered feedforward neural network with $N-K-1$ architecture as discussed in Sec. 1.3.2 and illustrated in Fig. 1.5 (right panel). However, compared with the general form of the output, Eq. (1.14), we have to use 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 [38, 66, 67] for examples and [22, 42, 63] for further references.

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 $I\!D = \{\boldsymbol{\xi}^\mu, S_T^\mu\}_{\mu=1}^P$ 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 $I\!D$ has been implement. Mezard and Parisi coined the term tiling algorithm for a particular procedure [68], which adds neurons one by one. Several other similar growth schemes have been suggested, see [69, 70] for examples and references.

A particular tiling-like algorithm for the PM was introduced and analysed in [65]. 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 $I\!D_1 = I\!D$, aiming at a low number of errors 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 misclassified examples in $I\!D$.

Define the new training set $I\!D_{m+1} = \{\boldsymbol{\xi}^\mu, [S_m(\boldsymbol{\xi}^\mu)\, S_T^\mu]\}_{\mu=1}^P$

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 with $\sigma_{m+1}(\boldsymbol{\xi}) = \text{sign}\,[\mathbf{w}_{m+1} \cdot \boldsymbol{\xi} - \theta_{m+1}]$ as to achieve a low number of errors Q_{m+1} with respect to data set $I\!D_{m+1}$

[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.

Note that if a solution with *zero error* $Q_M = 0$ is found in step (III) for the M-th hidden unit σ_M, the total output of the PM is

$$\prod_{m=1}^{M} \sigma_m(\boldsymbol{\xi}^\mu) = S_T^\mu \quad \text{for all examples in } I\!D,$$

i.e. the data set is perfectly reproduced by the PM of M units.

It is surprisingly straightforward to show that the number of errors can be decreased 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 pre-processing step. The second condition (4.12) is trivially satisfied and δ can be determined in any given data set by computing all pairwise scalar products.

GRANDMOTHER NEURON (4.10)

Consider a set of feature vectors $\{\boldsymbol{\xi}^\mu\}_{\mu=1}^P$ with

$$|\boldsymbol{\xi}^\mu|^2 = \Gamma \qquad \text{for all } \mu = 1, 2, ..., P \tag{4.11}$$

$$0 < \delta < \Gamma - \boldsymbol{\xi}^\mu \cdot \boldsymbol{\xi}^\nu \quad \text{for all } \mu, \nu \ (\mu \neq \nu). \tag{4.12}$$

Compute a perceptron weight vector and threshold as

$$\mathbf{w} = -\boldsymbol{\xi}^P \ \text{ and } \ \theta = \delta - \Gamma. \tag{4.13}$$

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 $I\!D$ are reproduced correctly.

[5] An idea which is possibly not quite as unrealistic as it may seem, see for instance [71] for a discussion of grandmother neurons and "Jennifer Aniston cells".

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,
- 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 set $I\!D$:

UNIVERSAL CLASSIFIER (parity machine) (4.15)

For a given data set $I\!D = \{\boldsymbol{\xi}^\mu, S_T^\mu\}_{\mu=1}^P$ 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 $\mathbf{w}^k \in I\!R^N$ and thresholds $\theta^k \in I\!R$ 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]

This finding is 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.

These results parallel the findings of Cybenko [72] and others, that a single hidden layer with sufficiently many continuous non-linear units constitute universal approximators, see also [17] for a discussion and further references.

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 studied in [73, 74].

[6] Strictly speaking the PM does not fall into this class, as F^{PM} cannot be realized by a single perceptron-like unit.

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, [29–32]. A comprehensive repository of materials, including a short history of the approach is provided at `www.svms.org` [53].

The concept of applying a kernel function in the context of pattern recognition dates back to at least 1964, see Aizerman et *al.* [75]. Probably the first practical version of Support Vector Machines, close to their current form, was introduced by Boser, Guyon and Vapnik in 1992 [76] and relates to early algorithms developed by Vladimir Vapnik in the 1960s [77]. According to Isabelle Guyon [78], the MinOver algorithm [46] 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 [79].

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

$$\boldsymbol{\xi} \in I\!R^N \;\rightarrow\; \underline{\Psi}(\boldsymbol{\xi}) \in I\!R^M \quad \text{with components } \Psi_j(\boldsymbol{\xi}) \tag{4.16}$$

where M can be different from N, in general. A function of the form

$$S(\boldsymbol{\xi}) = \text{sign}\,[\underline{W} \cdot \underline{\Psi}(\boldsymbol{\xi})] \quad \text{with weights } \underline{W} \in I\!R^M \tag{4.17}$$

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, originally: 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 [26, 34], see also [25]. In the hardware realization, for instance, 512 association units were connected to a subset of the 400 photosensor units and performed a threshold operation on an effectively randomized weighted sum of the incoming voltages, see [34] for details.

In the Support Vector Machine, the non-linear mapping is – in general – 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 [80]: 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. 4.4 (left panel). We apply the explicit transformation

$$\underline{\Psi}(\xi_1, \xi_2) \;=\; (\xi_1^2,\, \sqrt{2}\,\xi_1\,\xi_2,\, \xi_2)^\top \in I\!R^3$$

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. 4.4.

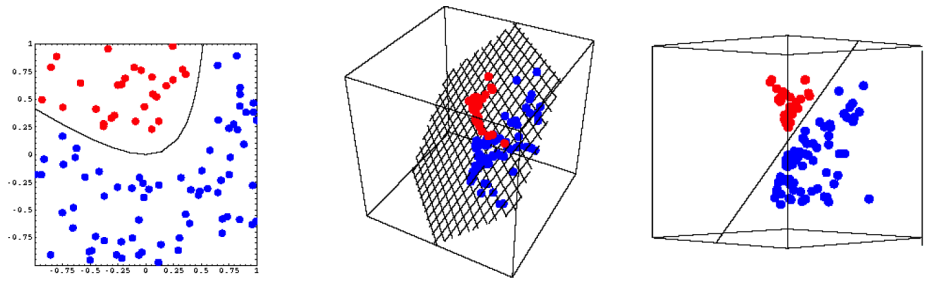


Figure 4.4: Illustration courtesy of Rainer Dietrich [80]. A two-dimensional data set with two classes that are not linearly separable (left panel) can become linearly separable after applying an appropriate non-linear transformation to a higher-dimensional space (center and right panel, two different viewpoints).

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 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 $\boldsymbol{\xi} \rightarrow \underline{\Psi}$ 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 $I\!D_N = \left\{\boldsymbol{\xi}^\mu \in I\!R^N, S_T^\mu\right\}_{\mu=1}^P$ we have found a suitable transformation such that

$$I\!D_M = \left\{\underline{\Psi}^\mu \in I\!R^M, S_T^\mu\right\}_{\mu=1}^P$$

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 (3.16) we are even guaranteed to find a solution.

However, in general, we do not have an 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. 3.4.1, 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. 3.4.2 – we can expect good generalization ability.

The mathematical structure of the corresponding problem is fully analogous to the original (3.41). The M-dim. counterpart reads

PERCEPTRON OF OPTIMAL STABILITY (M-dim. feature space) (4.18)

For a given data set $I\!D_M = \{\underline{\Psi}^\mu, S_T^\mu\}_{\mu=1}^P$, find the vector $\underline{W}_{max} \in I\!R^M$

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. 3.5 to the transformed space.

So far we have assumed that the transformation $\boldsymbol{\xi} \to \underline{\Psi}$ exists and is explicitly known. We could for instance formulate and apply an M-dimensional version of the MinOver algorithm (3.43) or (3.44). Moreover, we can apply the optimization theoretical concepts and methods presented in Sec. 3.6 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. 3.6, we can introduce the embedding strengths $\vec{X} = (X^1, X^2, ..., X^P)^\top \in I\!R^P$. With the shorthand $\underline{\Psi}^\mu = \underline{\Psi}(\boldsymbol{\xi}^\mu)$ we also define the correlation matrix

$$\Gamma \text{ with elements } \Gamma^{\mu\nu} = \frac{1}{M} S_T^\mu \underline{\Psi}^\mu \cdot \underline{\Psi}^\nu S_T^\nu \tag{4.19}$$

and analogous to Eqs. (3.60) we obtain

$$\underline{W} = \frac{1}{M}\sum_{\mu=1}^P X^\mu\, \underline{\Psi}^\mu\, S^\mu \quad \text{and } \underline{W}^2 = \frac{1}{M}\,\vec{X}^\top \Gamma\, \vec{X}. \tag{4.20}$$

Eventually, we can re-formulate the problem (4.18) as

PERCEPTRON OF OPTIMAL STABILITY (M-dim. feature space)

$$\underset{\vec{X}}{\text{minimize}}\ \ \frac{1}{2}\,\vec{X}^\top \Gamma \vec{X} \ \text{ subject to inequality constraints } \ \Gamma\vec{X} \geq \vec{1}. \tag{4.21}$$

and proceed along the lines of Sec. 3.6 to derive, for instance, the corresponding AdaTron algorithm, see below.

The output of the M-dim. perceptron can be written as

$$S(\boldsymbol{\xi}) = \text{sign}\Big[\underline{W}\cdot\underline{\Psi}(\boldsymbol{\xi})\Big] = \text{sign}\left[\frac{1}{M}\sum_{\mu=1}^P X^\mu\, S^\mu \underline{\Psi}^\mu \cdot \underline{\Psi}(\boldsymbol{\xi})\right]. \tag{4.22}$$

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

$$K : I\!R^N \times I\!R^N \to I\!R \ \text{ with } K(\boldsymbol{\xi}, \widehat{\boldsymbol{\xi}}) = \frac{1}{M}\,\underline{\Psi}(\boldsymbol{\xi})\cdot\underline{\Psi}(\widehat{\boldsymbol{\xi}}) = \frac{1}{M}\sum_{j=1}^M \Psi_j(\boldsymbol{\xi})\,\Psi_j(\widehat{\boldsymbol{\xi}}) \tag{4.23}$$

which represents the scalar product in $I\!R^M$. We observe that

$$S(\boldsymbol{\xi}) = \text{sign}\left[\sum_{\mu=1}^P X^\mu\, S^\mu K(\boldsymbol{\xi}^\mu, \boldsymbol{\xi})\right] \tag{4.24}$$

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

$$E^\mu \;=\; \left[\Gamma \vec{X}\right]^\mu \;=\; S_T^\mu \sum_{\nu=1}^{P} S_T^\nu \, X^\nu \, K(\boldsymbol{\xi}^\mu, \boldsymbol{\xi}^\nu). \tag{4.25}$$

One can also formulate the AdaTron algorithm for optimal stability in the M-dimensional space. The Kernel AdaTron was introduced and discussed in [81] and has been applied in a variety of practical problems. In analogy to (3.76) it is given as

KERNEL ADATRON algorithm, sequential updates (repeated presentation of $\mathbb{D}$)
– at time step t, present example $\mu = 1, 2, 3, ..., P, 1, 2, 3, ...$
– perform the update

$$X^\mu(t+1) \;=\; \max\left\{0,\; X^\mu(t) + \widehat{\eta}\left(1 \;-\; S_T^\mu \sum_{\nu=1}^{P} S_T^\nu \, X^\nu \, K(\boldsymbol{\xi}^\mu, \boldsymbol{\xi}^\nu)\right)\right\} \tag{4.26}$$

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:
$K^{\mu\nu} = K(\xi^\mu, \xi^\nu) = {}^1\!/_M \, \underline{\Psi^\mu} \cdot \Psi^\nu$.

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 (4.26).

d) Classify an arbitrary $\boldsymbol{\xi} \in \mathbb{R}^N$ according to

$$S(\boldsymbol{\xi}) \;=\; \text{sign}\left[\sum_{\mu=1}^{P} X_{max}^\mu \, S^\mu K(\boldsymbol{\xi}^\mu, \boldsymbol{\xi})\right]$$

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 explicitly updated in in the training (c), nor is it used for the classification in working phase (d).

Ultimately, this suggests to by-pass the explicit transformation in the first place replace step (a) by

a') For a given, non-separable $\mathbb{D}_N$, identify a suitable kernel function $K(\xi, \widehat{\xi})$ 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.

Interestingly, sufficient conditions for a kernel to be valid can be provided according to Mercer's Theorem [82], see also [29–32]. 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) (4.27)

A given kernel function K can be written as $K(\boldsymbol{\xi}, \widehat{\boldsymbol{\xi}}) = \frac{1}{M}\, \underline{\Psi}(\boldsymbol{\xi}) \cdot \underline{\Psi}(\widehat{\boldsymbol{\xi}})$,

with a transformation $\boldsymbol{\xi} \in I\!R^N \to \underline{\Psi} \in I\!R^M$ of the form (4.16), 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 $\int g(\boldsymbol{\xi})^2 d^N\xi < \infty$.

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

$$K(\boldsymbol{\xi}^\mu, \boldsymbol{\xi}) = (1 + \boldsymbol{\xi}^\mu \cdot \boldsymbol{\xi}\,)^q \quad \text{yielding} \quad S(\boldsymbol{\xi}) = \text{sign}\left[\sum_{\mu=1}^{P} X^\mu\, S_T^\mu\; (1 + \boldsymbol{\xi}^\mu \cdot \boldsymbol{\xi}\,)^q\right] \tag{4.28}$$

as the input-output relation of the classifier.

As a special case, let us consider the simplest polynomial kernel:

Linear kernel ($q = 1$)

$$K(\boldsymbol{\xi}^\mu, \boldsymbol{\xi}) = (1 + \boldsymbol{\xi}^\mu \cdot \boldsymbol{\xi}\,) \quad \text{with } S(\boldsymbol{\xi}) = \text{sign}\left[\sum_{\mu=1}^{P} X^\mu\, S_T^\mu\; (1 + \boldsymbol{\xi}^\mu \cdot \boldsymbol{\xi}\,)\right] \tag{4.29}$$

$$= \text{sign}\Bigg[\underbrace{\sum_{\mu=1}^{P} X^\mu\, S_T^\mu}_{\equiv M\,\Theta} + \underbrace{\sum_{\mu=1}^{P} X^\mu\, S_T^\mu\, \boldsymbol{\xi}^\mu}_{\equiv M\underline{W}} \cdot \boldsymbol{\xi}\Bigg].$$

In this case, we can provide an immediate, almost trivial interpretation of the kernel: It corresponds to the realization of a linear separable function in the original feature space ($M = N$) with

$$\text{weights} \quad \underline{W} \mathrel{\widehat{=}} \mathbf{w} = \sum_{\mu=1}^{P} X^\mu\, S_T^\mu\, \boldsymbol{\xi}^\mu \qquad \text{and off-set} \quad \Theta = \sum_{\mu=1}^{P} X^\mu S_T^\mu.$$

The *SVM with linear kernel* is applied very frequently in practice. There is an unfortunate trend to refer to it as *"the SVM"*, even. 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). [7]

[7] It should be referred to as such, even if "SVM..." may sound more sophisticated.

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 $q = 2$ in (4.28):

Quadratic kernel ($q = 2$)

$$K(\boldsymbol{\xi}^\mu, \boldsymbol{\xi}) = (1 + \boldsymbol{\xi}^\mu \cdot \boldsymbol{\xi})^2 \;=\; 1 + 2\sum_{j=1}^{N} \xi_j^\mu \, [\xi_j] \;+ \sum_{j,k=1}^{N} \xi_j^\mu \, \xi_k^\mu \, [\xi_j \, \xi_k] \tag{4.30}$$

Hence, the output $S()$ in Eq. (1.2) corresponds to an inhomogeneously linearly separable function in terms of the N original features ξ_j augmented by $1/2 N(N+1)$ 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 non-linear mapping from N-dimensional feature space to in total $M = 1/2 N(N+3)$ transformed features (original, squares and mixed products). An explicit formulation is reminiscent of Quadratic Discriminant Analysis (QDA) [17], albeit aiming at a different objective function in the training.

Similarly, for the general polynomial kernel (4.28) 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

$$K(\boldsymbol{\xi}^\mu, \boldsymbol{\xi}) \;=\; \exp\left[-\frac{1}{2\,\sigma^2} \left(\boldsymbol{\xi}^\mu - \boldsymbol{\xi} \right)^2 \right] \tag{4.31}$$

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 $M \to \infty$ 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 Control parameters and soft-margin SVM

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. (4.31). The data-driven adaptation of kernel parameters as part of the training process has also been discussed in the literature, see [83] for just one example.

In addition to the choice of the kernel and potential parameters thereof, one often resorts to a soft margin version of the SVM [79], see also [29–32]. 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. (4.20) and (4.19), 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.

In practice, 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.

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.

4.3.5 Efficient implementations of SVM training

The remark at the end of Sec. 3.6.3 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)* [52] 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 [53].

Chapter 5

Prototype-based systems

There is nothing objective about objective functions.

– James L. McClelland

Prototype-based models constitute a very successful family of machine learning approaches. They are appealing for a number of reasons: The extraction of information from previously observed data in terms of typical representatives, so-called 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 for instance [17,18,24]. 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 ([8, 17, 24]).

In the following 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, moreover, to the use of adaptive measures in so-called relevance learning schemes [59].

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 learning. It provides, at best, first insights into supervised schemes and can serve as a starting point for the interested reader.

Emphasis will be on Teuvo Kohonen's Learning Vector Quantization and its extensions. 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 liter-

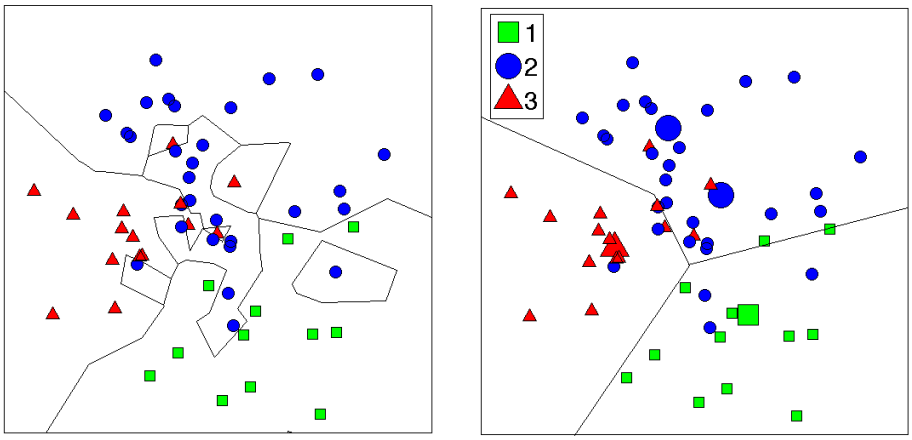


Figure 5.1: **Left panel:** Illustration of the Nearest Neighbor (NN) Classifier for an artificial data set containing three different classes. **Right panel:** A corresponding NPC scheme for the same data. Prototypes are represented by larger symbols. Both schemes are based on Euclidean distance and yield piece-wise linear decision boundaries.

ature, 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 [84, 85].

5.1 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 ([59]).

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 [8, 9, 86], 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 [8]. As a starting point for the discussion, we briefly revisit the well-known k-Nearest-Neighbor (kNN) approach to classification, see [24, 87].

5.1.1 Nearest Neighbor and Nearest Prototype Classifiers

Nearest Neighbor classifiers [24, 87] 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 we do not have to restrict the presentation to binary labels, here. We therefore

denote the (possibly multi-class) labels by $y^\mu \in \{1, 2, \ldots C\}$ where C is the number of classes.

An arbitrary novel feature vector or *query* $\boldsymbol{\xi} \in I\!R^N$ 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 $\boldsymbol{\xi}^\mu \in I\!D$ 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 $I\!D$. In the more general kNN classifier, the assignment is determined by means of a voting scheme that considers the k closest reference vectors [87].

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 [17, 24, 87]. Consequently, the method serves, to date, as an important baseline algorithm and is frequently used as a benchmark to compare performances with.

Fig. 5.1 (left panel) illustrates the NN classifier and displays how the system implements piece-wise 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. 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 $I\!D$. While methods for efficient sorting can reduce the computational costs to 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* $I\!D$ was already suggested in [88]. An alternative, essentially *bottom-to-top* approach is considered in the following sections.

5.1.2 Learning Vector Quantization

This successful and particularly intuitive approach to classification was introduced and put forward by Teuvo Kohonen [8, 9, 59, 89–91]. 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 [8]. 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

$$\left\{ \mathbf{w}^j,\, c^j \right\}_{j=1}^{M} \text{ with } \mathbf{w}^j \in I\!R^N \text{ and } c^j \in \{1, 2, \ldots C\}. \tag{5.1}$$

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

$$\mathbf{w}^*(\boldsymbol{\xi}) \text{ with } d\Big(\mathbf{w}^*(\boldsymbol{\xi}), \boldsymbol{\xi}\Big) = \min \Big\{ d(\mathbf{w}^j, \boldsymbol{\xi}) \Big\}_{j=1}^{M} \tag{5.2}$$

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 5.1 (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 competitive in a variety of practical classification problems [92]. In addition, their flexibility and interpretability constitute important advantages of prototype-based classifiers: Prototypes are obtained and can be interpreted within the space of observed data, directly. This feature facilitates the discussion with domain experts and is in contrast to many other, less transparent machine learning frameworks.

5.1.3 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 [8, 90, 91, 93–96]. The first, original scheme suggested by Kohonen [8] is known as LVQ1. Essentially, it includes already 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 $d(\mathbf{w}^*_\mu, \boldsymbol{\xi}^\mu) = \min\left\{d(\mathbf{w}^j, \boldsymbol{\xi}^\mu)\right\}_{j=1}^{M}$. (5.3)
  with class label $c^*_\mu = c(\mathbf{w}^*_\mu)$.
- perform a *Winner-Takes-All* (WTA) update: (5.4)
  $$\mathbf{w}^*_\mu(t{+}1) = \mathbf{w}^*_\mu(t) + \eta_w\, \Psi(c^*_\mu, y^\mu)\left(\boldsymbol{\xi}^\mu - \mathbf{w}^*_\mu\right) \text{ with } \Psi(c,y) = \begin{cases} +1 & \text{if } c{=}y \\ -1 & \text{else.} \end{cases}$$

The magnitude of the update is controlled by the learning rate η_w. The actual update step (5.4) 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 even 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.

$$\mathbf{w}^j(0) = \sum_{\mu=1}^{P} \delta[y^\mu, c^j]\ \boldsymbol{\xi}^\mu \Big/ \sum_{\mu=1}^{P} \delta[y^\mu, c]$$

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 [91, 93–96] 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 [97]. The popular Generalized LVQ (GLVQ) [98, 99] is guided by an objective function that relates to the concept of large margin classification [100]:

$$E^{GLVQ} = \sum_{\mu=1}^{P} \Phi(e^\mu) \quad \text{with} \quad e^\mu = \frac{d(\mathbf{w}_\mu^J, \boldsymbol{\xi}^\mu) - d(\mathbf{w}_\mu^K, \boldsymbol{\xi}^\mu)}{d(\mathbf{w}_\mu^J, \boldsymbol{\xi}^\mu) + d(\mathbf{w}_\mu^K, \boldsymbol{\xi}^\mu)}. \tag{5.5}$$

Here, the vector $\mathbf{w}_\mu^J$ denotes the closest of all prototypes which carry the same label as the example $\boldsymbol{\xi}^\mu$, i.e. $c_\mu^J = y^\mu$. Similarly, $\mathbf{w}_\mu^K$ denotes the closest prototype with a label different from y^μ. For short, we will frequently refer to these vectors as the *correct winner* $\mathbf{w}_\mu^J$ and the *incorrect winner* $\mathbf{w}_\mu^K$, respectively.

The cost function (5.5) comprises a, in general, non-linear and monotonically increasing function $\Phi(e)$. A particularly simple choice is the identity $\Phi(e) = e$, while the authors of [98,99] 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 $d(\mathbf{w}_\mu^J, \boldsymbol{\xi}^\mu) < d(\mathbf{w}_\mu^J, \boldsymbol{\xi}^\mu)$.[1] For large values of the steepness γ the costs approximate the number of misclassified 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 [101–103]. In contrast to LVQ1, two prototypes are updated in each step of the GLVQ procedure:

[1] note that the argument of Φ obeys $-1 \leq e^\mu \leq 1$

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 $I\!D$ with uniform probability $1/P$
- identify the *correct* and *incorrect winners*, i.e. the prototypes

$$\mathbf{w}_\mu^J \text{ with } d(\mathbf{w}_\mu^J, \boldsymbol{\xi}^\mu) = \min \left\{ d(\mathbf{w}^j, \boldsymbol{\xi}^\mu) \;\middle|\; c_\mu^j = y^\mu \right\}_{j=1}^M .$$
$$\mathbf{w}_\mu^K \text{ with } d(\mathbf{w}_\mu^K, \boldsymbol{\xi}^\mu) = \min \left\{ d(\mathbf{w}^j, \boldsymbol{\xi}^\mu) \;\middle|\; c_\mu^j \neq y^\mu \right\}_{j=1}^M . \tag{5.6}$$

  with class labels $c_\mu^J = y^\mu$ and $c_\mu^K \neq y^\mu$, respectively.
- update both winning prototypes according to:

$$\mathbf{w}_\mu^J(t+1) = \mathbf{w}_\mu^J(t) - \eta_w \frac{\partial \Phi(e^\mu)}{\partial \mathbf{w}_\mu^J}$$
$$\mathbf{w}_\mu^K(t+1) = \mathbf{w}_\mu^K(t) - \eta_w \frac{\partial \Phi(e^\mu)}{\partial \mathbf{w}_\mu^K}, \tag{5.7}$$

  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 [98, 99]. Note that – if Euclidean distance is used - the chain rule implies that the updates are along the gradients

$$\frac{\partial d(\mathbf{w}_\mu^L, \boldsymbol{\xi}^\mu)}{\partial \mathbf{w}_\mu^L} \propto \left(\mathbf{w}_\mu^L - \boldsymbol{\xi}^\mu\right) \quad \text{for} \quad L = J, K. \tag{5.8}$$

Moreover, the signs of the pre-factors in (5.7) are given by $\Psi(c^L, y^\mu) = \pm 1$ as in (5.1.3). 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 algorithms presented above, LVQ1 and GLVQ, very often a decreasing learning rate η_w is used to ensure convergence of the prototype positions in practice [101]. Alternatively, schemes for automated learning rate adaptation or more sophisticated optimization methods can be applied, see e.g. [104], which we will not discuss here.

5.2 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.

5.2.1 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. (5.8). 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. (5.5), but it is also possible for the more heuristic LVQ1, which will serve as an example here. As a generalization of Eq. (5.4) we obtain the analogous WTA update from example μ at time t:

$$\mathbf{w}^*_\mu(t+1) = \mathbf{w}^*_\mu(t) \; - \; \eta_w \; \Psi(c^*_\mu, y^\mu) \; \tfrac{1}{2} \, \tfrac{\partial \delta\left(\mathbf{w}^*_\mu, \boldsymbol{\xi}^\mu\right)}{\partial \mathbf{w}^*_\mu}. \tag{5.9}$$

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 [104]. Alternatively, differentiable approximations of non-differentiable δ can be used, see [105] 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., [59, 105, 106] 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* [107]. This classical measure is a popular tool in the analysis of data sets. Duda et *al.* present a detailed discussion and several application examples [24].

Standard *Minkowski distances* satisfy metric properties for values of $p \geq 1$ in

$$d_p(\boldsymbol{\xi}, \widehat{\boldsymbol{\xi}}) = \left[\sum_{j=1}^N \left| \xi_j - \widehat{\xi}_j \right|^p \right]^{1/p} \quad \text{for} \;\; \boldsymbol{\xi}, \widehat{\boldsymbol{\xi}} \in I\!R^N, \tag{5.10}$$

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 [65, 108] for specific examples.

The squared Euclidean distance can be rewritten in terms of scalar products:

$$d(\mathbf{w}, \boldsymbol{\xi})^2 = (\mathbf{w} \cdot \mathbf{w} - 2\mathbf{w} \cdot \boldsymbol{\xi} + \boldsymbol{\xi} \cdot \boldsymbol{\xi}) \, . \tag{5.11}$$

So-called *kernelized distances* [109] replace all inner products in (5.11) by a kernel function κ:

$$d_\kappa(\mathbf{w}, \boldsymbol{\xi})^2 = \kappa(\mathbf{w}, \mathbf{w}) - 2\kappa(\mathbf{w}, \boldsymbol{\xi}) + \kappa(\boldsymbol{\xi}, \boldsymbol{\xi}). \tag{5.12}$$

As in the SVM formalism, the function κ can be associated with a non-linear transformation from $I\!R^N$ 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 [110] 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 Kullbach-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 [111, 112]. 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.

5.2.2 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 [113–118], for instance. Relevance learning has proven useful in a variety of applications, including biomedical problems and image processing tasks, see for instance ([119]).

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 [114].

Similar to several other schemes (e.g. [120–122]), Matrix Relevance LVQ employs a generalized quadratic distance of the form

$$\delta_\Lambda(\mathbf{w}, \boldsymbol{\xi}) \;=\; (\mathbf{w}-\boldsymbol{\xi})^\top \;\Lambda\; (\mathbf{w}-\boldsymbol{\xi}) \;=\; \sum_{i,j=1}^{N} (w_i - \xi_i)\, \Lambda_{ij}\, (w_j - \xi_j). \tag{5.13}$$

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.

In order to fulfil the minimal requirement of non-negativity, $\delta_\Lambda \geq 0$, a convenient re-pameterization is introduced in terms of an auxiliary matrix $\Omega \in I\!R^{N\times N}$:

$$\Lambda = \Omega^\top \Omega, \quad \text{i.e.} \quad \delta_\Lambda(\mathbf{w}, \boldsymbol{\xi}) \;=\; \left[\Omega\; (\mathbf{w}-\boldsymbol{\xi})\right]^2. \tag{5.14}$$

Hence, δ_Λ can be interpreted as the conventional squared Euclidean distance but after a linear transformation of feature space. Note that Eqs. (5.13, 5.14) define only a *pseudo-metric* in $I\!R^N$: Λ can be singular with $\mathrm{rank}(\Lambda) < N$ 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 (5.13) 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 $\Lambda \in I\!R^{N\times N}$ 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 [113], which assigns a single, non-negative weighting factor to each dimension in feature space. Rectangular $(N \times M)$-matrices matrices Ω with $M < N$ can be used to parameterize a low-rank relevance matrix [118]. The corresponding low-dimensional intrinsic representation of data facilitates, for instance, the class-discriminative visualization of complex data [118]. 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 [114, 118].

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 [59] and its convergence behavior is analysed in [123].

Gradient based updates for the simultaneous adaptation of prototypes and relevance matrix can be derived from a suitable cost function: We observe that

$$\frac{\partial \delta_\Lambda(\mathbf{w}, \boldsymbol{\xi})}{\partial \mathbf{w}} = \Omega^\top \Omega \left(\boldsymbol{\xi} - \mathbf{w}\right) \quad \text{and} \quad \frac{\partial \delta_\Lambda(\mathbf{w}, \boldsymbol{\xi})}{\partial \Omega} = \Omega \left(\mathbf{w} - \boldsymbol{\xi}\right) \left(\mathbf{w} - \boldsymbol{\xi}\right)^\top . \tag{5.15}$$

The full forms of the gradients with respect to the terms e^μ in the GLVQ cost function are presented in [114], 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 $I\!D$ with uniform probability $1/P$
- with respect to the distance δ_Λ (5.13) with $\Lambda = \Omega(t)^\top \Omega(t)$, identify the *correct* and *incorrect winners*, i.e. the prototypes

$$\mathbf{w}_\mu^J \quad \text{with} \quad \delta_\Lambda(\mathbf{w}_\mu^J, \boldsymbol{\xi}^\mu) = \min \left\{ \delta_\lambda(\mathbf{w}^j, \boldsymbol{\xi}^\mu) \;\middle|\; c_\mu^j = y^\mu \right\}_{j=1}^M .$$

$$\mathbf{w}_\mu^K \quad \text{with} \quad \delta_\Lambda(\mathbf{w}_\mu^K, \boldsymbol{\xi}^\mu) = \min \left\{ \delta_\lambda(\mathbf{w}^j, \boldsymbol{\xi}^\mu) \;\middle|\; c_\mu^j \neq y^\mu \right\}_{j=1}^M . \tag{5.16}$$

  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:

$$\mathbf{w}_\mu^J(t+1) = \mathbf{w}_\mu^J(t) - \eta_w \frac{\partial \Phi(e^\mu)}{\partial \mathbf{w}_\mu^J}$$

$$\mathbf{w}_\mu^K(t+1) = \mathbf{w}_\mu^K(t) - \eta_w \frac{\partial \Phi(e^\mu)}{\partial \mathbf{w}_\mu^K},$$

$$\Omega(t+1) \quad = \quad \Omega(t) \; - \; \eta_\Omega \frac{\partial \Phi(e^\mu)}{\partial \Omega}. \tag{5.17}$$

  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.

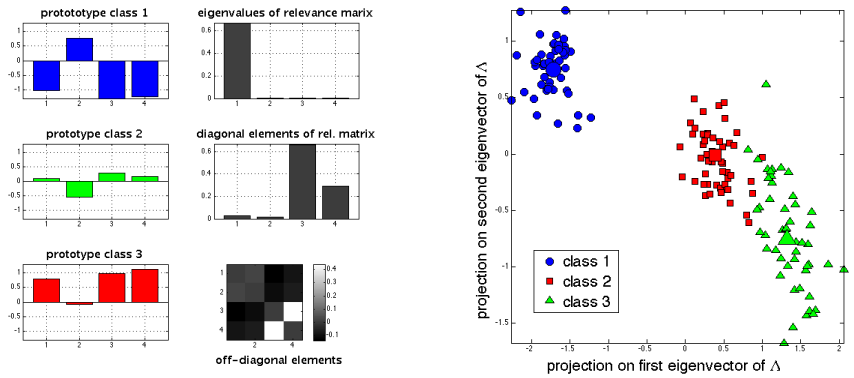


Figure 5.2: Visualization of the Generalized Matrix Relevance LVQ system as obtained from the z-score transformed Iris flower data set, see Sec. 5.2.2 for details. **Left panel:** Class prototypes are shown as bar plots with respect to the four feature space components in the left column. The right column 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 e.g. [124].

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 [114].

In the following we illustrate Matrix Relevance LVQ in terms of a classical benchmark data set: In the famous Iris data [125], 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 [126] and applied a simple GMLVQ system with one prototype per class and global relevance matrix $\Lambda \in I\!R^{4\times 4}$. For the training, we employed the freely available *beginner's toolbox for GMLVQ* ([124]) with default parameter setting. An additional z-score transformation was applied, resulting in re-scaled features with zero mean and unit variance in the data set.

Figure 5.2 visualizes the obtained classifier. The resulting LVQ system achieves almost perfect, error-free classification of the training data and very good generalization behavior with respect to test set performance.

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 most 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 [119] for a variety of applications. A recent application in the context of galaxy classification based on astronomical catalogue data is presented in [127, 128].

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 [123]. 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. 5.2 displays the Iris flower data set.

5.3 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 [59, 91] 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 [1]. 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, [84, 129] and references therein.

Chapter 6

Evaluation and validation

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 vital to know or at least have some estimate of the performance that can be expected in the working phase.

In this chapter we will discuss several aspects related to the evaluation and validation of supervised learning. We will take a rather general perspective on overfitting and underfitting effects without necessarily addressing a particular classifier or regression framework. The specific scenarios considered in the following should be seen as as illustrative examples, only.

In addition, practical aspects will be taken into account, such as the problem of unbalanced data sets in classification and the appropriate use of performance measures beyond the simple *overall accuracy*. Finally, we address the use of interpretable models in machine learning.

6.1 Bias and variance

Different sources of error can influence the performance of supervised learning systems. Here, we will decompose the expected prediction error into two contributions, see e.g. [17, 19]: The following, the so-called *bias* corresponds to systematic deviations of a trained model from the true target, while the term *variance* refers to variations of the model performance when trained from different realizations of the training data[1].

Frequently, a so-called *irreproducible error* is considered as a third, independent contribution [17]. It could refer to, for instance, 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, here.

[1]Note that the terms bias and variance are used in many different scientific contexts with possibly different, area specific meanings.

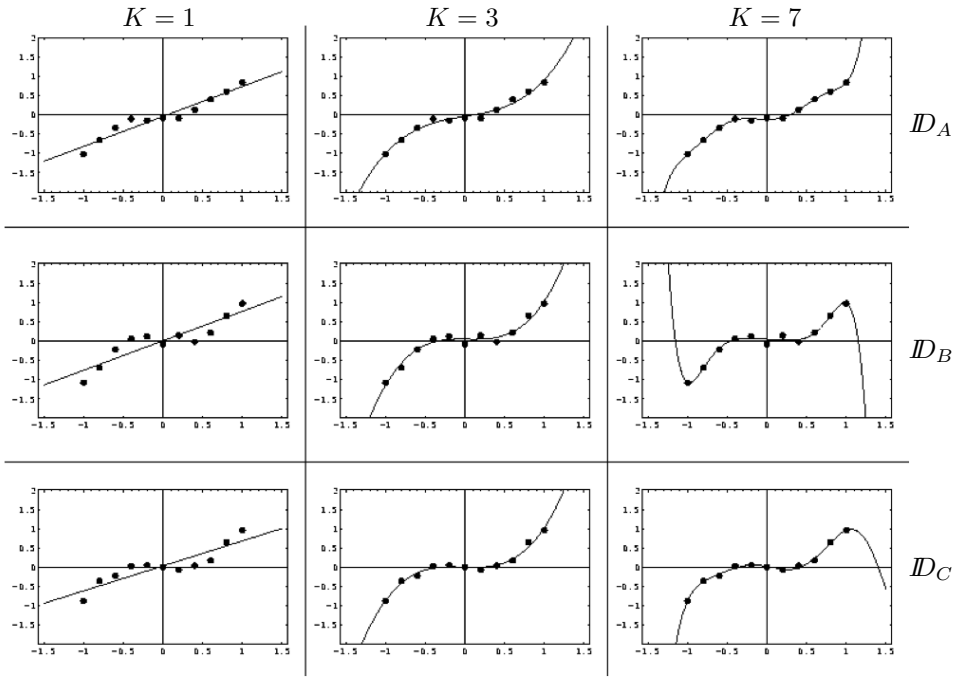


Figure 6.1: Illustration of the bias-variance dilemma in regression. In each row of graphs, a particular set of 10 points $\{x_i, y_i\}_{i=1}^{10}$ is approximated by least square linear regression ($K = 1$), by a cubic fit ($K = 3$), and by fitting a polynomial of degree seven ($K = 7$). Rows correspond to three randomized, independently generated data sets $\mathbb{D}_{A,B,C}$.

6.1.1 The decomposition of the error

For the purpose of illustration, we consider a simple one-dimensional regression problem as an example, see Fig. 6.1. The obtained insights, however, carry over to much more complex systems.

As a simple learning scenario we consider least squares regression based on data sets $\mathbb{D} = \{x^\mu, y^\mu\}_{\mu=1}^P$. The sets comprise sample arguments x^μ , e.g. equidistant $x^\mu \in [-1, 1]$, and their corresponding target values $y^\mu \in \mathbb{R}$.

We assume that the latter represent a target function $f(x)$ which is of course unknown to the learning system. The provided training labels are considered to be randomized, *noisy* versions of the target values:

$$y^\mu = f(x^\mu) + r^\mu \quad \text{with} \quad \langle r^\mu \rangle = 0 \quad \text{and} \quad \langle r^\mu \, r^\nu \rangle = \rho^2 \delta_{\mu\nu} \tag{6.1}$$

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

$$f_K(x) = \sum_{j=0}^K a_j x^j \quad \text{with coefficients } a_j \in \mathbb{R} \tag{6.2}$$

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

$$E^{SSE} = \tfrac{1}{2} \sum_{\mu=1}^P \left[f_H(x^\mu) - y^\mu \right]^2. \tag{6.3}$$

Fig. 6.1 displays example data sets $\mathbb{D}_{A,B,C}$ with $n = 11$ for 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 *Gedankenexperiment* of performing the same training/fitting processes for a 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 training labels in $\mathbb{D}$. In this sense, the expected total quadratic deviation of a hypothesis function f_H from the true target f in a point $x \in \mathbb{R}$ is given by

$$\left\langle \left[f_H(x) - f(x) \right]^2 \right\rangle_{\mathbb{D}} , \tag{6.4}$$

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. (6.4) we obtain

$$\left\langle f_H^2(x) \right\rangle_{\mathbb{D}} - 2 \left\langle f_H(x) \right\rangle_{\mathbb{D}} f(x) + f^2(x). \tag{6.5}$$

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 re-write (6.5) as

$$\underbrace{\left\langle f_H \right\rangle_{\mathbb{D}}^2}_{*} - 2 \left\langle f_H(x) \right\rangle_{\mathbb{D}} f + f^2 + \left\langle f_H^2 \right\rangle_{\mathbb{D}} \underbrace{-2 \left\langle f_H \right\rangle_{\mathbb{D}}^2 + \left\langle f_H \right\rangle_{\mathbb{D}}^2}_{*} \tag{6.6}$$

and obtain a decomposition of the expected quadratic deviation in x:

$$\left\langle \left[f_H - f \right]^2 \right\rangle_{\mathbb{D}} = \underbrace{\left(f - \left\langle f_H \right\rangle_{\mathbb{D}} \right)^2}_{\text{bias}^2} + \underbrace{\left\langle \left(f_H - \left\langle f_H \right\rangle_{\mathbb{D}} \right)^2 \right\rangle_{\mathbb{D}}}_{\text{variance}} . \tag{6.7}$$

The equality with (6.6) is seen easily by expanding the squares and exploiting that $\left\langle f_H \left\langle f_H \right\rangle_{\mathbb{D}} \right\rangle_{\mathbb{D}} = \left\langle f_H \right\rangle_{\mathbb{D}}^2 = \left\langle \left\langle f_H \right\rangle_{\mathbb{D}}^2 \right\rangle_{\mathbb{D}}$.

Hence we can identify two contributions to the total expected error:

- **Bias (squared):** $\left(f - \left\langle f_H \right\rangle_{\mathbb{D}} \right)^2$

  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 would indicate that there is very little systematic deviation of the hypotheses from the unknown target rule.

- **Variance:** $\left\langle \left(f_H - \left\langle f_H \right\rangle_{\mathbb{D}} \right)^2 \right\rangle_{\mathbb{D}}$

  The variance indicates how much the individual predictions, obtained after training on a given $\mathbb{D}$, will differ from the mean prediction, typically.
  The observation of a small variance implies that the outcome of the learning is robust with respect to details of the training data.

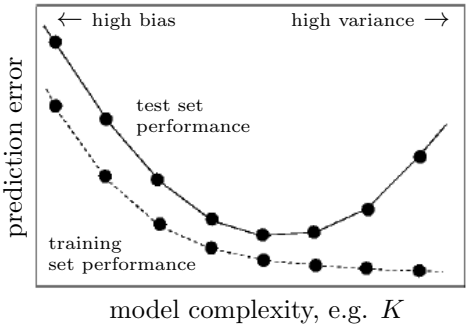


Figure 6.2: Schematic illustration of underfitting and overfitting (after [17]): Expected error with respect to a test set (generalization error) and training set performance as a function of the model complexity.

Similar considerations apply to more general learning problems, including classification schemes [130].

6.1.2 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 [17–19, 130]. It is closely related to the problem of overfitting in unnecessarily complex systems and its counterpart, the so-called underfitting in simplistic models.

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. (6.1), the averaged fit result will also be in exact agreement with the target in the x^μ. In fact, since the objective function (6.3) 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. 6.1, (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. (6.1). 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. 6.1 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 (6.2). This should prevent the fits from being

overly specific to the individual data set. As illustrated in the left column of Fig. 6.1, we achieve nearly identical linear models from the different data sets.

However, a price has to be paid for this 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 small 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.

The trade-off

The above considerations suggest that there is a trade-off between the goals of small variance and bias [17, 130, 131].

Indeed, in many machine learning scenarios one observes such a trade-off which is illustrated in Fig. 6.2. 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 [5, 18, 19].

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 test set performance 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. 6.2. 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 (6.7) itself does not imply the existence of a trade-off, strictly speaking.

As argued and demonstrated in e.g. [132] and [131], a given practical problem does not necessarily display the U-shaped dependence of the generalization error schematically shown in Fig. 6.2. There is also no general guarantee that measures which reduce the variance in complex models will really improve the performance of the system [132]. 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 estimation of the expected generalization ability would be highly desirable in a given supervised learning scenario. It could be used, for instance, to select the suitable complexity in a situation as sketched in Fig. 6.2.

Similarly, it would be very useful to be able to compare and evaluate the use of different approaches, say SVM and LVQ, in a given practical problem. In the next section we discuss the basic idea of how to obtain estimates of the generalization performance.

6.2 Validation procedures

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 harmful 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.

On the negative side, 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 on the training data does not provide us with such an insight.

Validation procedures can be employed which allow us to estimate the expected generalization performance [17, 19]. 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 applied to the test set eventually. This way, we can simulate working phase behavior while using only available data.

Obviously, the data is assumed to be representative for the task at hand, a crucial assumption which we already discussed in Sec. 2.2 in the context of the generic workflow of supervised learning.

6.2.1 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,[2] we cannot afford to disregard the information contained 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 optimistic or pessimistic, 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. 6.1.1.

All these issues are addressed in a very popular standard approach known as n-fold cross-validation, see e.g. [17, 19, 133]. The idea is to split the available data randomly into a number n of disjoint subsets of (nearly) equal size:

$$\mathbb{D} = \cup_{i=1}^{n} \mathbb{D}_i \quad \text{with} \quad \cap (\mathbb{D}_i, \mathbb{D}_j) = \emptyset \quad \text{for } i \neq j \quad \text{and } |\mathbb{D}_i| = P/n,$$

where the last condition may be satisfied only approximately in practice.
Now we can construct n splits of the data into

$$\text{training sets} \quad \mathbb{D}_i^{train} = \mathbb{D} \setminus \mathbb{D}_i \text{ and validation sets } \mathbb{D}_i^{val} = \mathbb{D}_i \text{ for } 1 \leq i \leq n.$$

For each of the splits we can train the classifier or regression system on $(1 - 1/n)\, P$ examples and evaluate its performance with respect to the P/n left out data 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 SSE 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, [17, 133]. In the literature, a canonical value of $n = 10$ has become a standard choice, apparently.

[2] Image classification tasks like "cats vs. dogs" have become one of the few notorious exceptions.

Variations
Many variations of the basic idea of n-fold cross-validation have been considered in the literature [133].

The split into n disjoint subsets of data may also suffer from lucky/unlucky set composition. So-called *Leave-p-out* schemes realize all possible splits of the data into $(P-p)$ training and p validation examples. Obviously their number $\binom{P}{p}$ grows rapidly with P and the computational costs can become unrealistically high. Therefore, one often resorts to a few repetitions of the n-fold scheme, performed with randomized splits and an additional average over the realizations.

Alternatively, Monte Carlo validation schemes generate a limited number of independent binary splits into $P-p$ and p examples and perform averages accordingly.

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, [134] and to the more general machine learning literature, e.g. [17–19].

Leave-one-out cross-validation
As an important extreme case, for very small data sets one often resorts to the so-called Leave-One-Out validation [17–19,133]. 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 and even bears the risk of yielding misleading results, systematically: In 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 [133, 135]. A modification that hinders the effect is known by the self-explanatory name *Leave-one-out from each class*, generating validation sets that represent all classes [135].

6.2.2 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 [133]. Obviously we could use, for instance, n-fold cross-validation to compare the expected performance of different classifiers or regression systems. We can also employ it to fix 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. 6.2, for instance, we would select the model complexity that corresponds to the minimum of the U-shaped generalization error curve.

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 LVQ-training based on a given data set $\mathbb{D}$. Assume that, on the basis of n-fold cross-validation, we conclude that systems with, say, $K = 3$ prototypes per class yield the best generalization ability[3].

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 specific 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

[3]For the difficulty to even define "the best" see the next sections.

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 for every independent parameter in an additional *layer* of validation, separately. Obviously practical limitations apply, in particular when only very small data sets are available.

6.3 Performance measures for classification

Despite the conceptual clarity of supervised learning, even the choice of an appropriate measure of success can constitute a problem in practice.

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 of cross-validation we obtain the estimates for the generalization ability in terms of the overall error as, say,

$$\epsilon_g^A = 0.05 \quad \text{and} \quad \epsilon_g^B = 0.30.$$

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 to prefer, obviously.

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 in the training of a classifier?
- Can we evaluate classifiers beyond their *overall accuracies* in order to obtain better insight into the performance?

6.3.1 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) [17–19,24, 136]. 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

$$S = \text{sign}\left[g(\boldsymbol{\xi})\right] \text{ with } g(\boldsymbol{\xi}) = \sum_{j=1}^{N} w_j \xi_j \tag{6.8}$$

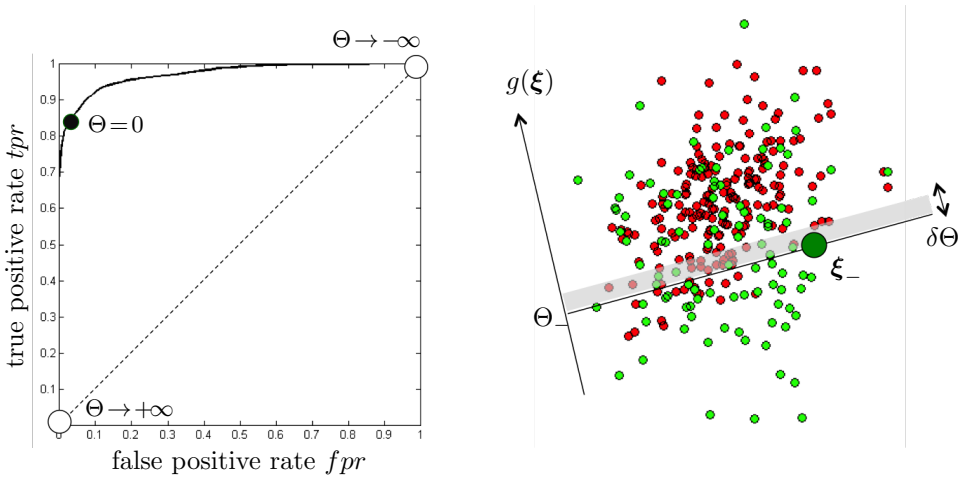


Figure 6.3: **Left panel:** Schematic illustration of Receiver Operating Characteristics. The extreme working points with $\Theta \to \pm\infty$ are marked by empty circles. A filled circle corresponds to a (hypothetical) classifier with $\Theta = 0$, while the dashed line represents random, biased guesses. **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 (red) 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. 6.3.2 to obtain the statistical interpretation of the *AUC*.

as discussed in Chapter 3 in great detail. Having trained the perceptron as to implement the homogeneous lin. sep. function (6.8), we can introduce a threshold Θ after training and consider the modified classification

$$S_\Theta = \text{sign}\left[g(\boldsymbol{\xi}) - \Theta\right]. \tag{6.9}$$

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 manually after training. Furthermore, the concept could be applied to any discriminatory function for binary classification.

By tuning Θ in Eq. 6.9 we can realize and control a bias towards one of the two classes. 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 two-class setup it makes sense 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 signal a truly positive case.[4] 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. (5.2). Identifying $\mathbf{w}^*_{(-1)}$, the closest one among all prototypes representing class -1 and the

[4] In the literature, other terms like *type I/II* errors are used frequently, but are avoided here for clarity.

closest class-(+1)-prototype $\mathbf{w}^*_{(+1)}$, we can assign an arbitrary feature vector $\boldsymbol{\xi}$ to class +1 if

$$d\Big(\mathbf{w}^*_{(+1)}, \boldsymbol{\xi}\Big) < d(\mathbf{w}^*_{(-1)}, \boldsymbol{\xi}) - \Theta \tag{6.10}$$

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 feedforward 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 absolute number of false positive classifications FP, the observed number of false negatives FP, and the number of true positive TP and true negatives assignments TN. The corresponding rates are defined as

$$fpr = \frac{FP}{FP+TN},\ tpr = \frac{TP}{FN+TP},\ fnr = \frac{FN}{FN+TP},\ \text{and}\ tnr = \frac{TN}{FP+TN}. \tag{6.11}$$

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* refers to $SPEC = tnr$.

The quantities in Eq. (6.11) 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[5].

Figure 6.3 (left panel) displays an example ROC curve for illustration. The lower left corner, as marked by an empty circle, would correspond to the extreme 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 (6.9) or (6.10): 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 6.3 (left panel). It could correspond, for instance, to the NPC in LVQ or the homogeneous, unbiased perceptron, Eq. (6.8). 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. 6.3 (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

[5] For efficient implementation ideas see [24, 136].

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. 6.3. 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.

6.3.2 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 AUC or $AUROC$ [136]. 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 one quality measure for the evaluation of classifiers.[6]

The AUC with respect to novel data can be estimated, for instance, in the course of cross-validation along the lines of Sec. 6.2. 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. 6.3 (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 frequency at which a pair of $\{\boldsymbol{\xi}_-, \boldsymbol{\xi}_+\}$ is ordered correctly according to 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.

[6] In detail, the precise shape of the ROC should be taken into account as well. In practice, individual ROC can differ significantly from the idealized shape displayed in Fig. 6.3.

true class membership	predicted class membership: 1	2	3	4	5	sum
1	63.8	10.8	19.3	5.5	0.7	100
2	3.1	90.7	0.1	1.3	4.8	100
3	18.4	1.7	66.1	13.5	0.3	100
4	2.0	6.8	9.3	73.7	8.2	100
5	1.1	13.1	0.1	14.6	71.1	100

Figure 6.4: An example confusion matrix of a 5-class classification problem, taken from [127]. Matrix elements $c(i, j)$ correspond to the percentage of samples from (truly) class i which are assigned to class j.

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 [137, 138].

6.3.3 Alternative quality measures and multi-class problems

A variety of quality measures for binary classification schemes have been suggested in the literature. As an alternative to the ROC, schemes like the Precision-Recall (PR) formalism or other frameworks can be considered [139]. In the two-class setting, they are also based on the quantities defined in Eq. (6.11) and provide, therefore, similar information. For a discussion of commonalities and supposed advantages or disadvantages see, for instance [139] and references therein.

Simpler *point estimates* can be computed at a single, specific working point of a classifier, the over-all accuracy being just the most obvious example.

The so-called balanced accuracy $BAC = (tpr + tnr)/2$ is supposed to be more suitable for unbalanced classification problems or data sets, respectively. Similar claims have been made for versions of the so-called F-measure, the Matthews-Correlation- Coefficient (MCC) and several other quality measures, see e.g. [140] for an overview and further references. In fact, a large variety of measures is in use which can lead to considerable confusion. We refrain from defining and discussing them here for the sake of clarity. In [141], also a discussion of LVQ-based training methods is provided, which can aim at a direct optimization of the quality measures.

Most commonly, the so-called confusion matrix c is provided in order summarize the class-specific performance of a multi-class system with respect to a given data set, e.g. in a validation set of examples not used for training. Illustration 6.4 displays an actual confusion matrix of a particular 5-class classification problem, taken from [127]. Here, each element $c(i, j)$ corresponds to the percentage of feature vectors which belong to class i, truly, and are assigned to class j [7]. Diagonal $c(i, i)$ correspond to the class-wise accuracies, while the off-diagonal elements provide insight into which classes are relatively easy or difficult to separate, respectively.

Note that, while the confusion matrix provides detailed information about class-wise performances, it corresponds to a single working point of the classifier, only. The generalization of the ROC formalism to multi-class problems is non-trivial [136, 142], while many of the above mentioned point measures can be extended to multi-class problems in a straightforward fashion, see [140].

[7] Alternatively, absolute numbers can be provided.

6.4 Interpretable systems

The above discussed quality measures and validation procedures focus on the performance of a trained system in terms of classification or regression accuracy. This is also true for the consideration of more sophisticated measures beyond overall or class-wise accuracies in supervised classification.

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. Some of these may not be expressed in terms of simple quantitative measures.

As an illustration, we discuss an entertaining *machine learning urban legend* [143]. As all urban legends it is not true, strictly speaking. However, it illustrates and summarizes an important issue in machine learning along the lines of the opening quotation of this chapter: "Accuracy is not enough."

The story is that the US military supposedly designed a classifier, aiming at the discrimination of US tanks from Russian tanks. The system was trained from a given set of labeled still images. The performance was nearly perfect: both training and validation accuracies were close to 100%. However, in practice[8], the classifier failed miserably.

At last, somebody checked the image data base and noted that all American tanks had been photographed on sunny days in the midst of lush vegetation, while the photos of Russian tanks were all taken in some Siberian winter landscape. The classifier had "learned" to distinguish snow covered scenes from the green fields of Kansas.

As usual with urban legends, it is told in great diversity, see [143] for an interesting account of several versions. However, the legend's core message is most relevant:

Un-noticed 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 tank urban legend. As a particularly important example, medical data set are frequently prone to selection biases that facilitate a seemingly successful discrimination of diseased from healthy subjects. For example, the age or gender distribution in the different 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 the nature of machine learning systems that they are excellent [**?**].

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 urban legend example: Is it the shape of the tanks 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.

[8] The use of the term *practice* is somewhat worrisome in this context.

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 genuine machine learning discussed in Sec. 2.4.

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) in recent years [144–146]. The overview articles and session contributions should provide a useful starting point for further explorations of the topic.

Chapter 7

Concluding remarks

In any case, the number of chapters should be prime.

– Michael Biehl

These lecture notes are supposed to serve as a first introduction to supervised machine learning. They cannot provide a comprehensive, complete overview of this highly dynamical field of research.

Basic terms and the relation of artificial neural networks with their biological counterparts are reviewed in Chapter 1. While the correspondence is, generally speaking, rather weak, biological systems have inspired the area of machine learning to a large extent and continue to do so.

The second chapter addresses different machine learning frameworks and puts them into perspective. In particular, common practices in machine learning are compared and discussed in relation to statistical modelling approaches.

The perceptron model, cf. Chapter 3, serves as a key example framework in which to discuss a number of essential issues in supervised learning. Extensions of the concept beyond linear separability are discussed in Chapter 4. This includes layered neural networks of perceptron-like units for classification and the very successful framework of the Support Vector Machine.

Chapter 5 is devoted to prototype-based approaches in supervised learning. Learning Vector Quantization serves as the key example and illustrates the concept of distance based classification. As an example of a data-driven weighting of features, relevance learning is presented and discussed in the context of LVQ.

The important question of how to evaluate the performance of supervised systems is addressed in Chapter 6. In particular, the estimation of working phase performance based on available data is discussed. In addition, specific quality measures for classification schemes are presented. Finally, we briefly discuss the importance of interpretable systems in supervised learning.

Quite a few important topics could not be addressed in the lecture notes or have been mentioned only very briefly:

- Continuous, multi-layered neural networks
  - The popular realization of classification by means of thresholded regression was only discussed briefly in terms of the Adaline system as a historical example. To a certain extent, classification by use of multilayered continuous neural networks is also presented in [1].

  - We have covered regression only in terms of very simple examples, i.e. linear regression and the Adaline algorithm. A more detailed presentation of regression was left to the lectures on deep learning [1].
  - For the discussion of gradient-based training algorithms for layered neural networks of continuous units, the reader is also referred to the set of lectures by M. Huertas-Company [1]. This includes deep neural networks and deep learning in general. An introduction and discussion of this currently very successful area within machine learning is provided in [1].

- Alternative network architectures
  A large variety of alternative architectures and frameworks for regression and classification could only be mentioned briefly or were not treated at all in the lectures. Among others, this includes Extreme Learning Machines [147], Radial Basis Function Networks [4, 18, 19], and other shallow architectures. Moreover, the areas of echo state machines, liquid state machines and related frameworks of reservoir computing could not be covered, see [11] for an overview and references.

- Ensemble based methods
  Methods employing ensembles of many classifiers or regression systems have only been mentioned without going into detail. For an introduction to this area, see [17, 56, 57]. Related techniques of bagging and boosting are also introduced in [17].

- The lectures focussed on clear-cut scenarios of supervised learning. Important other forms like semi-supervised learning, reinforcement learning, causal learning could only be mentioned in Sec. 2.3, where references are also provided. This includes

- All issues related to the pre-processing of data have been excluded from the presentation, essentially. The influence of normalization, transformations, imputation of missing data, integration of multimodal, heterogeneous data, correlation analysis and many more processing techniques deserve significant attention. Several of these aspects are treated in great detail in the standard literature, e.g. [5, 17–19, 22, 24, 148].

- The problems of model selection and overfitting have been addressed in Chapter 6, to a very limited extent. While the interplay of model complexity and generalization performance was discussed in principle, the important aspect of regularization was left out, essentially. Techniques like weight decay or drop-out can improve the performance of supervised systems significantly, see among others [17, 19, 148].

- Feature selection plays an important role in practical applications of supervised learning. The related framework of relevance learning was briefly discussed in Chapter 5. A variety of techniques for the selection and evaluation of feature importance is presented in [17, 19] and other textbooks, the review article [149] provides an overview and further references.

Clearly, this set of lectures is far from being complete, which also applies to the above list of missing topics. A large number of highly relevant aspects of supervised learning have not been taken into account at all. Nevertheless, the author hopes that the lecture notes provide a useful starting point for further explorations into the exciting world of machine learning.

Bibliography

[1] M. Huertas-Company. Deep Learning. Canary Islands Winter School of Astrophysics: Big Data Analysis in Astronomy. accessed 02/2019, `http://www.iac.es/winterschool/2018/pages/about-the-school/syllabus.php`.

[2] D. Baron. Unsupervised Learning. Canary Islands Winter School of Astrophysics: Big Data Analysis in Astronomy. accessed 02/2019, `http://www.iac.es/winterschool/2018/pages/about-the-school/syllabus.php`.

[3] K. Guerney. *An Introduction to Neural Networks*. UCL Press, London, 1997.

[4] S. Haykin. *Neural Networks and Learning Machines*. Pearson Education, Upper Saddle River, NJ, USA, third edition, 2009.

[5] J.A. Hertz, A.S. Krogh, and R.G. Palmer. *Introduction To The Theory Of Neural Computation*. Addison-Wesley, Reading, MA, USA, 1991.

[6] H. Horner and R. Kühn. Neural Networks. In U. Ratsch, M. Richter, and I.O. Stametescu, editors, *Intelligence and Artificial Intelligence, an Interdisciplinary Debate*. Springer, Heidelberg, Germany, 1998.

[7] C. Koch. *Biophysics of Computation: Information Processing in Single Neurons*. Oxford University Press, New York, NY, USA, 1998.

[8] T. Kohonen. *Self-Organizing Maps*. Springer, Berlin, Germany, 1997.

[9] H. Ritter, T. Martinetz, and K. Schulten. *Neural Computation and Self-Organizing Maps*. Addison-Wesley, New York, NY, USA, 1992.

[10] M. van Rossum. *Neural Computation*. Univ. of Edinburgh, Edinburgh, UK, 2007. http://homepages.inf.ed.ac.uk/mvanross/ln_all.pdf, accessed: 12/2018.

[11] 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.

[12] O. Chapelle, B. Schölkopf, and A. Zien. *Semi-supervised learning*. MIT Press, Cambridge, MA, 2005.

[13] M. Wiering and M. van Otterlo, editors. *Reinforcement Learning, State-of-the-Art*. Springer, Berlin, 2012.

[14] M. Yamada, J. Chen, and Y. Chang. *Transfer Learning, Algorithms and Applications*. Morgan Kaufmann, 2018.

[15] G. Ditzler, M. Roveri, C. Alippi, and R. Polikar. Learning in nonstationary environment: A survey. *Comput. Intell. Mag.*, 10:12–25, 2015.

[16] J. Peters, D. Janzing, and B. Schölkopf. *Elements of Causal Inference: Foundations and Learning Algorithms*. MIT Press, Cambridge, MA, 2017.

[17] T. Hastie, R. Tibshirani, and J. Friedman. *The Elements of Statistical Learning*. Springer Series in Statistics. Springer, New York, NY, USA, 2001.

[18] C.M. Bishop. *Neural Networks for Pattern Recognition*. Oxford University Press, Inc., New York, NY, USA, 1995.

[19] C.M. Bishop. *Pattern Recognition and Machine Learning (Information Science and Statistics)*. Springer, Heidelberg, Germany, 2006.

[20] E. Hubble. A relation between distance and radial velocity among extragalactic nebulae. *Proceedings of the National Academy of Sciences (PNAS)*, 15(3):168–173, 1929.

[21] J. Huchra. home page. `http://www.cfa.harvard.edu/~dfabricant/huchra`, accessed 12/2018.

[22] A. Engel and C. van den Broeck. *The Statistical Mechanics of Learning*. Cambridge University Press, Cambridge, UK, 2001.

[23] K.B. Petersen and M.S. Pedersen. The Matrix Cookbook, 2012. Version 20121115, `http://www2.imm.dtu.dk/pubdb/p.php?3274`, accessed 12/2018.

[24] R.O. Duda, P.E. Hart, and D.G. Stork. *Pattern Classification*. Wiley, New York, 2000.

[25] M. Minsky and S. Papert. *Perceptrons: An Introduction to Computational Geometry*. MIT Press, Cambridge, MA, USA, 1969.

[26] F. Rosenblatt. The Perceptron: A Probabilistic Model for Information Storage and Organization in The Brain. *Psychological Review*, pages 65–386, 1958.

[27] F. Rosenblatt. *Principles of Neurodynamics. Perceptrons and the Theory of Brain Mechanisms*. Cornell Aeronautical Laboratory Inc., Buffalo, NY, USA, 1961. `http://apps.dtic.mil/dtic/tr/fulltext/u2/256582.pdf`, accessed 01/2019.

[28] M. Opper, 1990. private communication.

[29] 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.

[30] N. Cristianini and J. Shawe-Taylor. *An Introduction to Support Vector Machines and Other Kernel- based Learning Methods*. Cambridge University Press, Cambridge, UK, 2000.

[31] J. Shawe-Taylor and N. Cristianini. *Kernel Methods for Pattern Analysis*. Cambridge University Press, Cambridge, UK, 2004.

[32] R. Herbrich. *Learning Kernel Classifiers, Theory and Algorithms*. MIT Press, Cambridge, MA, 2002.

[33] M. Olazaran. A sociological study of the official history of the perceptron controversy. *Social Studies of Science*, 26:611–659, 1996.

[34] Rosenblatt, F. et *al*. *Mark I Perceptron Operator's Manual, Report No. VG-1196-G-5*. Cornell Aeronautical Laboratory Inc., Buffalo, NY, USA, 1960. `http://apps.dtic.mil/dtic/tr/fulltext/u2/236965.pdf`, accessed 01/2019.

[35] Unknown. A short clip about perceptron research done in the 1950's and 1960's. avaialble at `https://www.youtube.com/watch?v=cNxadbrN_aI`, accessed 03-2019.

[36] R.O. Winder. Adaptive switching circuits. In *2nd Ann. Symposium on Switching Circuit Theory and Logical Design (SWCT 1961)*, pages 321–332, 1961.

[37] 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.

[38] G.J. Mitchison and R.M. Durbin. Bounds on the learning capacity of some multi-layer networks. *Biological Cybernetics*, 60:345–356, 1989.

[39] J.K. Anlauf and M. Biehl. The AdaTron: An Adaptive Perceptron Algorithm. *Europhys. Lett.*, 10(7):687–692, 1989. See [40] for an Erratum.

[40] J.K. Anlauf and M. Biehl. The AdaTron: An Adaptive Perceptron Algorithm (Erratum). *Europhys. Lett.*, 11(4):387, 1990. Erratum for [39].

[41] 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.

[42] T.L.H. Watkin, A. Rau, and M. Biehl. The statistical mechanics of learning a rule. *Reviews of Modern Physics*, 65:499–556, 1993.

[43] T.L.H. Watkin. Optimal Learning with a Neural Network. *Europhys. Lett.*, 21(8):871–876, 1993.

[44] M. Opper and D. Haussler. Generalization performance of Bayes optimal classification algorithm for learning a perceptron. *Phys. Rev. Lett.*, 66:2677–2680, 1991.

[45] P. Ruján. Playing billiards in version space. *Neural Computation*, 9(1):99–122, 1997. available online at `http://doi.org/10.1162/neco.1997.9.1.99`, accessed 03-2019.

[46] W. Krauth and M. Mezard. Learning algorithms with optimal stability in neural networks. *J. Phys. A: Math. Gen.*, 20(11):L745–L752, 1987.

[47] R. Fletcher. *Practical Methods of Optimization (2nd Edition)*. Wiley, 2000.

[48] B. Widrow and M.E. Hoff. Adaptive switching circuits. In *Proc. IRE WESCON Convention Rec.*, pages 96–104, 1960. available online at `http://www-isl.stanford.edu/~widrow/papers/c1960adaptiveswitching.pdf`, accessed 03-2019.

[49] 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. available at `http://www-isl.stanford.edu/~widrow/papers/j199030years.pdf`, accessed 03-2019.

[50] B. Widrow. Youtube channel widrowlms. http://www.youtube.com/channel/UCCt3FeRLD5ajFM1EoS6QOkQ, accessed 02/2019.

[51] L.O. Chua. Memristor – The Missing Circuit Element. *Transaction on Circuit Theory*, CT-18(5), 1971. {http://citeseerx.ist.psu.edu/viewdoc/summary?doi=10.1.1.189.3614}.

[52] J.C. Platt. Sequential minimal optimization: A fast algorithm for training support vector machines. Technical report, ADVANCES IN KERNEL METHODS - SUPPORT VECTOR LEARNING, 1998. available online at https://citeseerx.ist.psu.edu/viewdoc/summary?doi=10.1.1.43.4376, accessec 03-2019.

[53] Support vector machines. Online repository: http://www.svms.org, accessed 03/2019.

[54] P. Ruján. A fast method for calculating the perceptron with maximal stability. *J. de Physique I*, 3(2):277–290, 1993. available online at http://hal.archives-ouvertes.fr/file/index/docid/246719/filename/ajp-jp1v3p277.pdf, accessed 03-2019.

[55] D. Nabutovsky and E. Domany. Learning the unlearnable. *Neural Computation*, 3(4):604–616, 1991.

[56] D. Opitz and R. Maclin. Popular ensemble methods: an empirical study. *J. of Artificial Intelligence Research*, 11:169–198, 1999.

[57] R. Urbanczik. Online Learning with Ensembles. *Physical Review E*, 62(1):1448–1451, 2000.

[58] L. Breiman. Random Forests. *Machine Learning*, 45:5–32, 2001.

[59] M. Biehl, B. Hammer, and T. Villmann. Prototype-based models in machine learning. *Wiley Interdisciplinary Reviews: Cognitive Science*, 7(2):92–111, 2016.

[60] S.I. Gallant. Perceptron-based learning algorithms. *IEEE Transactions on Neural Networks*, 1(2):179–191, 1990.

[61] P. Rojas. *Neural Networks - A Systematic Introduction.* Springer, Berlin, Germany, 1996.

[62] M. Griniasty and H. Gutfreund. Learning and retrieval in attractor neural networks above saturation. *J. of Phys. A: Math. Gen.*, 24:715–734, 1991.

[63] B. Aubin, A. Maillard, J. Barbier, F. Krzakala, F. Macris, and 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, 2018.

[64] R. Monasson and R. Zecchina. Learning and generalization theories of large committee-machines. 9(30):1887–897, 1995.

[65] M. Biehl and M. Opper. Tilinglike learning in the parity machine. *Physical Review A*, 44(10):6888–6894, 1991.

[66] H. Schwarze and J. Hertz. Generalization in fully connected committee machines. *Europhysics Letters*, 21(7):785–790, 1993.

[67] 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. url: https://link.aps.org/doi/10.1103/PhysRevLett.72.2113.

[68] 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. url: https://doi.org/10.1088%2F0305-4470%2F22%2F12%2F019.

[69] M. Golea and M. Marchand. A growth algorithm for neural network decision trees. *Europhysics Letters*, 12(3):205–210, 1990. url =https://doi.org/10.1209%2F0295-5075%2F12%2F3%2F003.

[70] M. Frean. The upstart algorithm: A method for constructing and training feedforward neural networks. *Neural Computation*, 2(2):198–209, 1990. url: https://doi.org/10.1162/neco.1990.2.2.198.

[71] 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.

[72] G. Cybenko. Approximations by Superpositions of a Sigmoidal Function. *Math. Control Signals Systems*, 2:303–314, 1989.

[73] P. Rojas. Deepest Neural Networks. *arXiv:1707.02617v1*, 2017. available online: http://arxiv.org/abs/1707.02617, 9 pages, accessed 02/2019.

[74] 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. http://ieeexplore.ieee.org/document/1224071.

[75] 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.

[76] Guyon I.M. Vapnik V.N. Boser, B.E. A training algorithm for optimal margin classifiers. In *Proc. 5th Ann. Workshop on Computational Learning Theory (COLT '92), publisher=ACM Press, NY, pages=144-152,*, 1992.

[77] V.N. Vapnik and A.Y. Lerner. Recognition of patterns with help of generalized portraits. *Avtomat. i Telemekh.*, 24(6):774–780, 1963.

[78] I. Guyon. Data mining history: The invention of Support Vector Machines. text reprinted at http://www.kdnuggets.com/2016/07/guyon-data-mining-history-svm-support-vector-machines.html.

[79] C. Cortes, , and V. Vapnik. Support-vector networks. *Machine Learning*, 20(3):273–297, 1995. URL: http://doi.org/10.1023/A:102262741141.

[80] R. Dietrich. *Statistical Mechanics of Neural Networks: Enhancement by Weighting of Examples.* PhD thesis, 2000. available online: http://citeseerx.ist.psu.edu/viewdoc/summary?doi=10.1.1.15.4844, accessed:02/2019.

[81] 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.

[82] J. Mercer and A.R. Forsyth. 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. available at `http://royalsocietypublishing.org/doi/abs/10.1098/rsta.1909.0016`, accessed 03/2019.

[83] 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.

[84] 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.

[85] 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.

[86] B. Hammer, M. Strickert, and T. Villmann. Supervised neural gas with general similarity measure. *Neural Processing Letters*, 21(1):21–44, 2005.

[87] T. Cover and P. Hart. Nearest neighbor pattern classification. *IEEE Trans. Information Theory*, 13:21–27, 1967.

[88] P. Hart. The condensed nearest neighbor rule. *IEEE Trans. Information Theory*, 14:515–516, 1968.

[89] 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.

[90] P. Sommervuo and T. Kohonen. Self-organizing maps and learning vector quantization for feature sequences. *Neural Processing Letters*, 10(2):151–159, 1999.

[91] D. Nova and P.A. Estévez. A review of Learning Vector Quantization classifiers. *Neural Computing and Applications*, 25(3-4):511–524, 2014.

[92] Neural Networks Research Centre, Helsinki. Bibliography on the Self-Organizing Maps (SOM) and Learning Vector Quantization (LVQ). *Otaniemi: Helsinki Univ. of Technology. Available on-line: http://liinwww.ira.uka.de/bibliography/Neural/SOM.LVQ.html*, 2002.

[93] 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.

[94] M. Biehl, A. Ghosh, and B. Hammer. Dynamics and generalization ability of LVQ algorithms. *Journal of Machine Learning Research*, 8:323–360, 2007.

[95] A. Ghosh. *Robustness of shape descriptors and dynamics of Learning Vector Quantization.* PhD thesis, 2007. University of Groningen, available online: `http://hdl.handle.net/11370/c2778753-fdf1-4fac-949d-57fc5b784f9b`.

[96] A.W. Witoelar. *Statistical physics of Learning Vector Quantization.* PhD thesis, 2010. University of Groningen, available online: `http://www.rug.nl/research/portal/files/14692629/11complete.pdf`, accessed 03/2019.

[97] S. Seo and K. Obermayer. Soft Learning Vector Quantization. *Neural Computation*, 15:1589–1604, 2003.

[98] 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.

[99] 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.

[100] 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.

[101] H. Robbins and S. Monro. A stochastic approximation method. *The Ann. of Mathematical Statistics*, 22:405, 1951.

[102] L. Bottou. Stochastic gradient learning in neural networks. In *Proc. of Neuro-Nimes 91*. EC2 editions, 1991.

[103] 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.

[104] S. Sra, S. Nowozin, and S. Wright, editors. *Optimization for machine learning.* MIT Press, Cambridge, MA, 2011.

[105] 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.

[106] 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.

[107] P. Mahalanobis. On the generalised distance in statistics. *Proc. of the National Inst. of Sciences of India*, 2:49–55, 1936.

[108] O. Golubitsky and S. Watt. Distance-based classification of handwritten symbols. *Int. J. on Document Analysis and Recognition (IJDAR)*, 13:133–146, 2010.

[109] B. Schölkopf. The kernel trick for distances. In *Proc. Adv. in neural information processing systems*, pages 301–307, 2001.

[110] 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.

[111] 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.

[112] E. Mwebaze. *Divergences for prototype-based classification and causal structure discovery: Theory and application to natural datasets.* PhD thesis, 2014. University of Groningen, available online: `http://www.rug.nl/research/portal/files/14550974/kbunte_thesis.pdf`, accessed 03/2019.

[113] B. Hammer and T. Villmann. Generalized Relevance Learning Vector Quantization. *Neural Networks*, 15(8-9):1059–1068, 2002.

[114] P. Schneider, M. Biehl, and B. Hammer. Adaptive relevance matrices in Learning Vector Quantization. *Neural Computation*, 21:3532–3561, 2009.

[115] P. Schneider. *Advanced methods for prototype-based classification.* PhD thesis, 2010. University of Groningen, available online: `http://www.rug.nl/research/portal/files/14618216/thesis.pdf`, accessed 03/2019.

[116] K. Bunte. *Adaptive dissimilarity measures, dimension reduction and visualization.* PhD thesis, 2011. University of Groningen, available online: `http://www.rug.nl/research/portal/files/14550974/kbunte_thesis.pdf`, accessed 03/2019.

[117] 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.

[118] 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.

[119] 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.

[120] K. Weinberger and L. Saul. Distance metric learning for Large Margin Nearest Neighbor classification. *J. of Machine Learning Research*, 10:207–244.

[121] 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.

[122] 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.

[123] 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.

[124] M. Biehl. A no-nonsense beginner's toolbox for gmlvq, version v2.2, 2016. url: `http://www.cs.rug.nl/~biehl/gmlvq`, accessed 03-2019.

[125] R. Fisher. The use of multiple measurements in taxonomic problems. *Annual Eugenics*, 7:179–188, 1936.

[126] M. Lichman. UCI Machine Learning Repository, 2013. url: `http://archive.ics.uci.edu/ml`.

[127] 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.

[128] A. Nolte, L. Wang, M. Bilicki, B. Holwerda, and M. Biehl. Galaxy classification: A machine learning analysis of GAMA catalogue data. *Neurocomputing*, 2019. in press, available online: `http://www.sciencedirect.com/science/article/pii/S0925231219301353`.

[129] S. Saralajew, L. Holdijk, M. Rees, and T. Villmann. Prototype-based neural network layers: Incorporating vector quantization. *CoRR*, abs/1812.01214, 2018. available online `http://arxiv.org/abs/1812.01214`, accessed 03-2019.

[130] 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.

[131] 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.

[132] C. Schaffer. Overfitting avoidance as bias. *Mach. Learning*, 10:153–178, 1993.

[133] S. Raschka. Model evaluation, model selection, and algorithm selection in machine learning. *CoRR*, abs/1811.12808, 2018. available online at `http://arxiv.org/abs/1811.12808`, accessed 03-2019.

[134] P. Burman. A comparative study of ordinary cross-validation, v-fold cross validation and the repeated learing testing-model methods. *Biometrika*, 76:503–514, 1989.

[135] 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. available online at `http://proceedings.mlr.press/v8/airola10a.html`, accessed 03-2019.

[136] T. Fawcett. An introduction to ROC analysis. *Pattern Recognition Letters*, 27:861–874, 2006.

[137] A. Herschtal and B. Raskutti. Optimising area under the ROC curve using gradient descent. In *Proc. of the 21st International Conference on Machine Learning*, pages 49–, 2004.

[138] T. Villmann, M. Kaden, W. Herrmann, and M. Biehl. Learning Vector Quantization classifiers for ROC-optimization. *Computational Statistics*, 2016.

[139] 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. `http://doi.acm.org/10.1145/1143844.1143874`, accessed 03/2019.

[140] T. Kautz, B.M. Eskofier, and C.F. Pasluosta. Generic performance measure for multiclass classifiers. *Pattern Recognition*, 68:1125, 2017.

[141] 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.

[142] 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.

[143] `gwern.net`. The Neural Net Tank Urban Legend, 2009. url: `http://www.gwern.net/Tanks` accessed: March 2019.

[144] 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.

[145] V. Van Belle and P. Lisboa. Research directions in interpretable machine learning models. In M. Verleysen, editor, *Proc. of the European Symposium on Artificial Neural Networks (ESANN 2013)*, pages 533–431. d-side, 2013.

[146] 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.

[147] Guang-Bin Huang, Qin-Yu Zhu, and Chee-Kheong Siew. Extreme learning machine: Theory and applications. *Neurocomputing*, 70(1):489 – 501, 2006. available online at `http://www.sciencedirect.com/science/article/pii/S0925231206000385`, accessed 03-2019.

[148] I.J. Goodfellow, Y. Bengio, and A. Courville. *Deep Learning*. MIT Press, Cambridge, MA, USA, 2016. `http://www.deeplearningbook.org`.

[149] I. Guyon and A. Elisseeff. An Introduction to Variable and Feature Selection. *J. of Machine Learning Research*, pages 1157–1182, 2003.