Chapter 3

Neural Modeling

We observe the behavior of complex systems such as in distributed communication & transportation Grids, biologic federations of species taking roles as predictor and prey, or automated cascaded processes in industrial plants. These systems have one thing in common: they generate huge amounts of data without a prevalent comprehensive model synthesized bottom-up from the underlying principles. Hypothesis-based models as arise from system theory suffer from an explosion in complexity when many principles from various disciplines have to be coherently combined, while statistical modeling is too coarse. Both do not provide satisfactory overall models for their behavior to bring out patterns that are easily identified by a human. Fortunately we can resort to so-called computational intelligent methods such as artificial neural networks (ANNs). ANNs can capture dynamical patterns from data, making them potential candidates for novelty detection. They are also adaptive and hence can be capable of spotting harmful trends in a non-stationary system. Unfortunately some neural features, e.g. inherent redundancy and distributed information storage, are not only potentially interesting for fault tolerance, but also complicate neural design. Before utilizing neural learning we pay detailed attention to these complications.

Section 3.1 provides a coarse history of neural modeling, listing the common types of neural network and their typical applications. Then we follow the same line of discourse as in the previous chapter. First neural architectures for modeling dynamic data are treated in section 3.2 (refining 2.3 for the neural model). Then a discussion on neural learning in section 3.3 specifies in more detail how to train Multi-Layer Perceptron based dynamic neural networks (thus refining 2.4). In section 3.4 we discuss particular features of neural modeling, showing the difference with physical and statistical modeling and estimation.

3.1 Background

This section has been written for those readers entirely unfamiliar to the subject of neural modeling. Some general background is required to understand how the Multi-Layer Perceptron and error back-propagation are positioned in the larger class of architectures and learning algorithms. A historic overview in the development and evolution of neural modeling (subsection 3.1.1) explains how several areas have been combined to arrive at connectionist learning architectures. Section 3.1.2. briefly discusses the most common types of neural networks. We conclude this section with a short overview of typical applications for these common types of neural networks.
3.1.1 Developments and evolution

In this thesis we do not advertise the use of neural networks as general models for any solution. In fact at the end of this chapter we will discuss the limitations and applicability of neural networks to indicate that neural networks may have particular interesting properties but they are no snake oil for any problem. Yet, neural networks can hardly be discussed without mentioning some of the trends in the past 50 years. Much of the fierce criticism that neural networks had to endure from conventional research areas is still expressed today. We direct the interested reader to some of the paradigmatic literature on neural networks for a full background [Haykin 1994, Bishop 1995]. In this thesis we stick to the historic background that is relevant within the scope of this thesis.

Neural network theory finds it’s root in various areas including electrical engineering, biology and artificial intelligence. The field of research emerged in 1940s when the mathematician Pitts and the psychiatrist McCulloch hooked up. The mixture of neurology and mathematics has characterized neural networks ever since. Von Neuman got involved, and Hebb formulated the first learning theory. Computer simulation models soon followed [Rochester, Holland, Haibt, Duda, 1956]. Minsky worked out the basics of the neural pattern learning [Minsky, 1954, 1961, 1967]. Meanwhile related developments took place in the area of adaptive non-linear filtering [Gabor, 1960]. The Perceptron and a supervised learning rule was introduced by Rosenblat [Rosenblat, 1958], Widrow and Hoff introduced the least mean square algorithm for ADALINE [Widrow and Hoff, 1960], and the ingredients were available for the first multi-layered “neural network” MADALINE [Widrow, 1962]. Just a bit later, Amari developed the stochastic gradient method [Amari, 1967].

There are some interesting analogies between artificial and biological neural networks. A key building block of the brain is a synapse that merely connects one processing element to another; connectionism is the basis of neural networks theory. Neurons are cells that perform a mostly non-linear transformation from the incoming synapses to output dendrites. In the brain the transformations can be complex, in neural networks they are additions followed by a non-linear “squashing function”. In very large groups of neurons “knowledge” can be captured or learned from an initially blank “brain”. There is no presumed model of reality; hence we speak of non-parametric models. Another feature is Adaptivity; learning from stimuli, that is either evidence for hypotheses to learn a mapping or simply learning to distinguish context, is a key feature of biologic and artificial neural networks. Distributedness of information is another crucial feature of neural networks. The neural network is built from uniform elements which represent no particular function. Generic uniform elements can be combined to represent any mapping. The actual information in a huge combination of elements that hardly manifest in input-output behaviour must be highly distributed and redundant. This makes a brain fault-tolerant. Even if the elements are not flawless or break-down, it keeps behaving within a desirable range. The design of artificial structures representing mapping from unreliable elements is a problem that has been addressed by von Neuman [von Neuman, 1956] using redundancy. It was shown that a large number of elements could collectively represent a concept [Winograd, Cowan, 1963]. Then robustness and parallelism emerge as natural properties.

The possibility of composing of a complex function using simpler functions with less parameters is a generalization of the 13th problem that was stated by the famous mathematician Hilbert. It is claimed [Hecht-Nielsen, 1987] Kolmogorov reformulated this into a Mapping Neural Networks Existence Theory. This mapping problem was addressed by various people. They
have shown that Multi-Layer Perceptrons are universal approximators [Cybenko, 1988; Hornik, 1989], i.e. they can be used to express any function with arbitrary precision. However that does not mean that every mapping can be “learned” from data.

Fundamental limitations of single layer Perceptrons have been stated mathematically [Minsky, Papert 1969]. One of them is the credit/penalty assignment problem (C.A.P), i.e. in a redundant system who’s to blame for the error? In the 1970s the idea is spread that, since this C.A.P could not be solved for a single Perceptron, there was not hope for Multi-Layer Perceptrons. This idea and the lack of computational power has stalled the further development of neural networks during the 1970s, with the exception of Self-Organizing Feature Maps [von der Malsburg, 1973; Kohonen 1982] and behavioural continuous-time models, such as ART [Grossberg, 1976; Hopfield, 1982]. The plasticity-stability problem in learning stated by Grossberg, is still applicable; we will speak of it later on. A crucial breakthrough was the discovery of the Error Back-Propagation algorithm that was claimed by Rumelhart, Hinton and Williams [Rumelhart, Hinton and Williams, 1986] but was first described by Werbos in his PhD Thesis [Werbos, 1974]. Despite proof of universal approximation by neural networks, there are several reasons why learning may fail. Typically a connectionist model may be unviable, merely due to it’s structure, even if in principle a set of parameters exist that give a sufficiently small error. The learning process may not converge. Mapping with arbitrary precision is only achieved when there is no restriction on the number of neurons to deploy. In practice finding the right number of neurons, and layers, is not a trivial problem. Several statistics-based attempts have been made to estimate the “right” model dimensions.

The key property that distinguishes neural networks from most physical and statistical models, is that a solution to a problem is mostly not uniquely determined by the data. In mathematical terms this is called ill-conditioned, which is often also mistaken to be under-determined. A consequence of this is that the design of a neural network either requires quite some expertise and gut-feeling or computational intensive coverage of several design parameters. The intensive use of computing power to explore the neural design space is refered to as the frequentist approach. Though frequentist and experimentalists have ruled the application of neural networks for almost a decade, mathematicians and statisticians have called for a mathematical basis and a statistically sound design approach.

Sun-Ichi Amari has developed a stochastic macroscopic model of learning behaviour. A physical analogy was found in the spin-glass model, providing a probabilistic framework where the behavior of neural networks is explained on a macroscopic level based on the laws of thermodynamics and statistical mechanics. Sigmoid belief networks and the mean-field approximation from this framework are based on stochastical physics. Bayesian and probabilistic methods apply a random-field approach to parameter estimation instead of a feedback learning along the neural pathways. They advocate batch learning which is indeed a better approach for several types of problems. The Bayesians have addressed ill-posedness and weak convergence of back-propagation learning by introducing priors and probabilistic learning rules. In the Bayesian approach the connectivity of the neurons no longer poses a limitation on the assignment of errors or credits, since they are assumed to be independent stochastic variables. The academically attractive mathematical foundations of Bayesian and probabilistic learning methods depend on regularity conditions (e.g. Gaussianity). Such properties can conveniently be designed into toy problems, but the conditions are not met in real-world problems.
3.1.2 Neural networks overview

Neural networks cover a very large class of models. Good background readings providing a survey of the several neural networks types are found in [Lippmann, 1987], detailed descriptions and analysis are found in [Haykin, 1994], [Bishop, 1995]. A short overview is provided here for completeness to the readers unfamiliar to neural modeling. We characterize each type of model by it’s architectures and learning.

- **Multi-Layer Perceptrons (MLP)**
  Multi-layer Perceptrons are feed-forward input/output models. They usually have a number of hidden neurons interconnecting input and output variables. The free parameters are multiplicative weights and biases. The typical learning algorithm is gradient descent. This model and type of learning will be extensively discussed in the remainder of this chapter. The learning can be iterative per observation (pattern-learning) or using the full-training set to determine mean error before updating weights (batch-learning).

- **Recurrent and continuous-time neural network (RNN)**
  Some networks assume a continuous signal flow rather than discrete-time pattern with error propagation, such as the Hopfield and Cohonen-Grossberg ART model. Differential equations are used to describe these models as well as the learning in these models. Signal and chaos theory are used to analyse their behavior.

- **Self-Organizing Feature Map (SOFM)**
  Self-Organizing Features Maps have typically a Kohonen or von der Malzburg architecture. The vectors connecting a node to it’s inputs represent a prototype vector for a cluster. The prototype vectors are iteratively updated such that the clusters cover data, while the prototypes correspond to kernels representing data densities.

- **Radial Basis Function Network (RBFN)**
  Radial Basis Function Networks are also input/output models, with the difference that inputs are not used directly. Instead a membership to an input-cluster is used as input-value to the feed-forward structure. The inputs-clusters are so-called receptive-fields described by kernels representing probability distribution per cluster. The kernels are iteratively optimized using the full training-set to determine kernels parameters such as mean and variance for Gaussian kernel. The feed-forward structure is similar to a MLP.

- **Support Vector Machines (SVM)**
  A Support Vector Machine is a set of vectors taken from the data vectors, that characterize the boundaries between clusters in the data. The target is to find a minimal number of vectors that are orthogonal to the cluster boundaries. These are called the support vectors.

- **Probabilistic Neural Networks (PNN)**
  The differences with the other structures is that these aim to estimate probability density rather than finding a input-output mapping, or in the use of non-gradient based learning.

The scope of this research is limited to neural networks for modeling dynamic behaviour, assuming discrete-time data. Self-organizing feature maps are considered to classify dynamic patterns and Multi-Layer Perceptrons are applied for dynamic input-output relations.
3.1.3 Applications for neural networks

The various neural model types are applied for different tasks. A short discussion of particular applications with the typical neural model gives an indication why we choose to focus on the Multi-layer Perceptron. We provide some pointers to reference papers for neural applications.

- **Pattern recognition and classification**
  Pattern recognition and classification is the task of associating a symbol representing a class to a particular input pattern. Speech recognition is one of the first applications of neural networks to the modeling of dynamic patterns [Elman, 1990]. Multi-Layer Perceptrons are also used for classification, vision is another key application area.

- **Process identification and control**
  Neural networks are used also for process identification and control. In control and accommodation tasks the models interaction with the environment is a crucial element: control is performed by steering the actuators of a process to achieve a small error as observed from sensors. The most common structure is the single layer recurrent Perceptron [Narendra and Pasaranthy, 1990], but also Radial Basis Function networks are applied. Neural applications have added value to conventional control theory when the non-linearity is critical for achieving a sufficiently accurate model. Iterative updates of the neural networks are considered usefull for sliding-mode systems.

- **Regression, function approximation, time-series modeling**
  Neural networks are considered also as extension to existing statistical models. They offer a powerful non-linear alternative for regression and time-series modeling, e.g. for prediction, when the non-linearities cannot be easily identified or interact. Note that in case the non-linear behavior can be isolated into independent variables that have a linear interaction, a standard least-squares or QR-decomposition will generally provide a better solution than a neural network, as typical neural design problems (section 3.4) can be avoided. The Multi-Layer Perceptron is most commonly used among neural regression. A comparison between statistical and neural networks [Sarle, 1994] shows the similarity and inherent problems of any non-linear estimator in a statistical sense.

- **Biologically plausible models**
  The pursuit of mathematical and/or intuitive insight into computationally intelligent behavior has been one of the major forces driving the neural network community in the decades before the 1980s when neural models could not yet compete with conventional modeling techniques. The goal of imitating the behavior of biological neural processes using neural networks has given rise to several dynamic neural network architectures such as associative memory [Hopfield, 1982], Adaptive Resonance Theory (ART) of Grossberg [Grossberg, 1976]. These networks are self-organizing continuous-time networks with the goal of storing and reproducing input activations. Though much insights for dynamical structure have been gained from the analysis of these networks [Amari 1990; Perreto, 1986; Grossberg, 1976], these architectures are designed for pattern recognition tasks rather than for the modeling of dynamics in a series of measurements.

Among neural applications the Multi-Layer Perceptron is mostly used for practical classification and regression problems. Classification and biological processes are outside the scope of this research. Our scope is limited to Multi-Layer Perceptrons with a some attention for self-organising features maps, since clustering is widely applied for abnormality detection.
3.2 MLP-based dynamic models

Starting from the basic building block of standard Multi-Layer Perceptrons, in section 3.2.1, we provide a foundation to understand the basic Multi-Layer Perceptron architecture in section 3.2.2. The basic architecture can be extended to model dynamic data, similar to the extensions discussed in 2.3.

3.2.1 The Perceptron and alternative kernels

The Perceptron [Rosenblatt, 1962] is a projection of a weighted sum over all the neuron inputs, see equation 3.1. The \( \text{neuron activation} \) \( v_j \) is common to all types of neurons, while the \( \text{transfer function} \) which projects the activation determines the type of neuron. The adaptive filtering capabilities of interconnected linear Perceptrons, called \textit{Madalines}, has been extensively studied by Widrow [Widrow, 1962]. The original Rosenblatt Perceptron has a hard-limiting transfer function, but the transfer function needs to be differentiable for learning non-linear mappings. A linear transfer function will result in an architecture that can be reduced to a single layer model i.e. each output is a linear weighted sum over the outputs which can be optimized with a least mean square (LMS) algorithm, or a so-called Wiener filter. Most Perceptrons have an adaptive offset or bias level which is obtained by adding a constant input \( x_0 = -1 \); the bias is denoted \( \theta_j = w_{j0} \). The powerful modeling capacities as well as the intriguing learning behavior result from the use of a non-linear transfer function. A common non-linearity is the logistic function as in equation 3.2, where \( a \) is called the slope parameter often set to a fixed value of \( a = 1 \).

\[
y_j^{(l)}(x) = \Phi(v_j(x)) \quad v_j(x) = \sum_i w_{ji} x_i
\]

\[
\Phi_a(v) = \frac{1}{1 - e^{-av}} \quad \text{or alternatively} \quad \Phi_a(v) = \frac{2}{\pi} \tan^{-1}(av)
\]

The generalized kernel-based building block of equation 3.3 can be used with several kinds of kernels. Using specific types of kernels combined with batch-like estimation extends the model for probability estimation and radial-basis function like architecture.

\[
y[n] = \Phi \left( \sum_{j=1}^{J} w_j(g_j([n] \cdot x[n])) \right)
\]

It has been suggested that linear transfer functions are as effective as sigmoid transfer functions, [Weigend, 1996]. Linear transfer functions facilitate the modeling of non-smooth functions; for function approximation and prediction a linear transfer function is preferred over a sigmoid transfer function as a sigmoid output neuron only tends to make the modeling problem more complex. The use of linear output neurons provides a decreased stability of the learning process but yields a better convergence. Stability in the learning process is improved by a non-linear \textit{squashing function} as it limits the error back-propagating through the network learning best in the linear region of the sigmoid. EBP is assumed to work faster for asymmetric transfer functions, i.e. \( \Phi(-x) = -\Phi(x) \); hence a popular alternative [Haykin, 1994] for the logistic transfer is the hyperbolic tangent function.
3.2.2 The Multi-Layer Perceptron

The Multi-Layer Perceptron has been the dominating architecture in the neural networks community for classification, prediction and function approximation from 1986 till halfway the 1990s. Though some variants of the basic Multi-Layer Perceptron have been introduced for non-stationary data and dynamical modeling, few truly different basic architectures can meet the MLPs popularity. This architecture has proven successful in many different applications but is still received with much sceptis mainly due to the lack of understanding, theoretical background and non-unique data representations of this non-linear model. MLPs are used in this thesis for two reasons:

- They can be used to model any functional relation from data.
  Multi-Layer Perceptrons are universal approximators [Cybenko, 1988; Hornik, 1987], i.e. they can be used to express any function with arbitrary precision. However, crucial to the success of the architecture is the learning algorithm through which arbitrary mappings can be learning from data.

- Their learning process reveals interesting dynamics which characterizes the data
  Finding a good configuration depends on the learning algorithm, discussed in section 3.4. It is the learning behavior and the learning process which is studied to characterize learning behavior. Much unmodelled dynamics is reflected in the learning behavior of the Multi-Layer Perceptron in combination with EBP.

The Multi-Layer Perceptron is dealt with by many authors in the field of neural networks and pattern recognition, for a full coverage see [Haykin, 1994] or [Bishop, 1995]. As we will be using the basic architecture throughout this thesis, we will briefly describe it’s architecture here. Perceptrons are organized in layers, where each MLP has at one input layer and one output layer. The input layer simply holds the last offered input, while the following layers are sets of Perceptrons taking the output of the previous layer as their input, figure 3.1. One speaks of feed-forward networks because of the signal flows from input to output without feedback.

![Figure 3.1](image)

**Figure 3.1** : The architecture of a Multi-Layer Perceptron. The input layer holds the last offered pattern, following layers are sets of Perceptrons which take the output of the previous layer as their input.

The Multi-Layer Perceptron can be expressed as a superposition of sigmoid functions. The connectivity of the MLP is given by *connections* between layers of Perceptrons. Usually the
connectivity is expressed by numbering the neurons and denoting the connection weights \(w_{ji}\) with resp. output neuron \(j\) and input neuron \(i\) of the connection \((j, i)\) in the subscript. The connectivity in feed-forward networks is thus expressed by a weight-matrix. The output of a layer is \(y^{(l)} = (y^{(l)}_1, y^{(l)}_2, ..., y^{(l)}_n)\) where \(l\) denotes the layer and \(y^{(l)}_j\) denotes the \(j\)th neuron in the layer and the \(j\)th neuron in the network. For a homogeneous choice of transfer functions the mapping of a neural network can be expressed by equation 3.4. The biases are defined \(y_0 = -1\) and \(y^{(0)} = x\) the input.

\[
o_j = y^{(l)}_j = \varphi \left( \sum_i w_{ji} y^{(l-1)}_i \right) = \varphi \left( \sum_i w_{ji} \varphi \left( \sum_k w_{ik} y^{(l-2)}_k \right) \right) = ... (3.4)
\]

The connectivity of a neural network together with the transfer functions, inputs and outputs is called the architecture of the neural network. A configuration of a neural network is a particular choice for it’s adaptive parameters \(w\).

### 3.2.3 Dynamic extensions of the Multi-Layer Perceptron

Our prime interest is in dynamical phenomena. We seek to capture as many dynamical structure present in the data as possible. The modeling of dynamical phenomena with a neural network has a long history dating back to before the rediscovery of the EBP algorithm.

The Multi-Layer Perceptron offers a generic approximation structure, but it is static. A static function approximator with a learning algorithm can be extended in several ways to incorporating dynamics. Two of the most common approaches are: 1) through data processing techniques; 2) through architectural changes to the basic MLP.

Subsequently, architectural modifications can be applied to the static structure as a whole or to the building block of the model: the Perceptron. The basic ways to incorporate dynamics in the neural network process are listed below. Though all these techniques can be used on their own, it is not uncommon that a combination is used to realize a suitable architecture.

- **Explicit notion of time, assuming a time-variant dependency in \(f_j\)**
  The most direct way to incorporate time into a static model is by giving it an explicit representation [Haykin, 1994; Kindermann and Trappenberg, 1999]. This can be achieved by adding one or more input signals derived from counters or periodic signals. The idea is that an existing trend or cyclic behavior (presumably present in the data set) can be found by generating such a signal but with tunable parameters so that it can be amplified, scaled and stretched. Introducing an explicit notion of time may lead to sufficient time-series models. However it is not a truly dynamical model as it has no internal state or memory. Explicit notion of time is still used but almost always as an extension to a dynamic model [Suykens, 1996; Venema, 1998]. The necessity for architectural extensions to capture dynamics in time-series models is emphasized by Elman [Elman, 1990].

- **Finite time-window and memory**
  A very direct way to capture dynamics is to explicitly model the relation between \(v[n+k]\) and a finite number \(q\) of past observations \(x[n] = Z^{(q)} v[n] = (v[n], v[n-1], ..., v[n-q])\). The static model is explicitly given a short-term memory \(Z^{(q)}\). The static model does not require structural modifications, as the delayed input patterns can be created before training and evaluating the model. A finite time-window can also be realized through internal memory, i.e. weighted tapped delays of the hidden neuron or synaptic outputs.
Feedback and internal state
Long-term dependencies usually result from the presence of internal state in the information process that generates the time-series. Though it is possible to capture this with finite time-windows, such will often require too many delay variables. The long-term effects are best modelled by incorporating feedback. Feedback of predicted variables \( v_i \) is referred to as auto-regression, while the use of independent feedback variables enable an internal state in the model. It is a structural modification which cannot be realized by manipulating input and output variables.

Adaptation
It is a small step from using state variables and feedback to learning. While feedback variables reflect the acquired information in the network, so do the weights! In contrast with finite time-windows the weight in the neural network are also referred to as long-term memory. Learning is also a form of feedback, so naturally the neural network can be learned “on-line” to adapt to new situations.

Figure 3.2 provides an overview of neural networks for modeling dynamic phenomena.

![Temporal Neural Networks](image)

**Figure 3.2** : An overview of different types of neural networks for modeling dynamic phenomena:

A static network is made dynamic by modifying the neuron either through feedback, as in the Hopfield network, or by adding a linear filter (see figure 3.3). In the first variant, figure 3.3a, the synapses are modelled by linear time-invariant filters [Shamma, 1989].

\[
x_1(t) \rightarrow h_{j1} \\
\vdots \\
x_p(t) \rightarrow h_{j2}
\]

\[
x_1(t) \rightarrow w_{j1} \\
\vdots \\
x_p(t) \rightarrow w_{j2}
\]

**Figure 3.3** : Neuron extensions to include dynamics (a) Dynamical extension through synapse modification; (b) RC-neuron model, neuron activation is extended with memory.
The second variant is the biologically motivated RC-neuron [Scott, 1977; Rall, 1989] shown in figure 3.3b. The Perceptron of figure 3.3a is called a Finite Impulse Response (FIR) Perceptron when all the filters at the synapses are causal and have a finite time-window. Implementations of the FIR Perceptron are discrete-time linear filters with activation defined in equation 3.5, where $M$ is the memory depth.

$$v_j[n] = \sum_{i=1}^{p} \sum_{m=0}^{M} w_{jim}x_i[n-m] + w_{j0}$$

The FIR-type neural networks are based on the FIR dynamic model of a Perceptron. This model can be used internally, in which case a Time Delay Neural Network (TDNN) is constructed, or only following the input nodes, in which case an Tapped Delay Line (TDL) neural network is obtained. In TDNNs [Waibel et al. 1989; Lang and Hinton, 1988], linear time-invariant causal finite impulse response filters are used as synapses anywhere in the network figure 3.4a. There are several drawbacks to this type of neural network. First there is training of this model: in order to use an error back-propagation algorithm the network has to be unfolded [Haykin, 1994]. The unfolded network is much larger than the original one, resulting in higher computational complexity and problems in the loss of coupling of related synapses. Unfolding is in fact not an analytically sound technique as the unfolded network is not equivalent to the original one. Another drawback is the poor heuristics on the estimation of filter orders in the different layers in the network and the interpretation of internal state in the model. Though some successes have been achieved in time-series modeling, e.g. speech recognition, they lost attention already in the last years of the 20th century.

**Figure 3.4**: Finite-memory neural networks; (a) Time delay MLP; (b) Tapped delay-line MLP.

In the Tapped Delay Line neural network (figure 3.4b) the linear time-invariant finite impulse response filters are implemented by simply adding a tapped delay-line to all or some of the network inputs. Instead of connecting all the delay-line outputs to the neural network, the
delay-line can also be partially connected [Diepenhorst et al., 1996]. TDL networks are commonly applied in neural forecasting as they are easy to implement, are guaranteed to be stable and require no significant alterations to the EBP learning algorithm discussed in section 3.2. A drawback of FIR approaches in general is that in real-world applications the required filter order (or delay-line length) will be large for time-series containing long-term dependencies. The large number of weights slows down the learning process.

Many processes have an internal state as the total effect of many interacting factors. Therefore a state-space model is to be preferred over a finite-memory model. Feedback in a process causes dynamics of the manifest behavior and is best captured in a model through the use of feedback or internal state. The IIR type of neural networks use feedback connections. Feedback connections can be inserted in many different ways in the neural network. One of the earliest recurrent network architecture is the Hopfield network: a network which is typically used for pattern storage and error-correction. The Hopfield network is viewed as a non-linear associative memory or CAM (content addressable memory) [Hopfield, 1982]. The Hopfield network is evaluated asynchronously; as a result the network always converges to a steady state, if it exists. A variant which uses synchronous (parallel) evaluation [Shaw and Little, 1975] does not necessarily converge to a steady state. In practice, continuous-time architectures are implemented by Discrete-Time (DT) networks with a high frequency internal clock. A network trained with an RTRL algorithm, and containing feedback, is called a real-time recurrent network [Williams and Zipser, 1989]. This type of network can be transformed into a partially recurrent network if the exogenous outputs of the network are not connected to the network inputs. An example of such neural networks are those using contextual inputs [Robson and Fallside, 1991; Elman, 1990].

A common dynamic extension of the MLP which preserves all the properties that allow the use of the standard error back-propagation algorithm is the Nonlinear Auto Regressive model with eXogenous inputs or NARX [Narendra and Parthasarathy, 1990]. This network uses a TDL-type network as described with a feedback connection from the output of the network to one of the inputs, not using the instantaneous output during learning. The advantage of this type of model is that no modification to the default evaluation and learning algorithm has to be made to capture temporal information using feedback.

Particular design issues for IIR networks make them less reliable. First of all, any structure that uses some kind of feedback potentially suffers from instability; moreover the adaptive parameters in such a structure have non-convex behavior. The result of these potential features may be that the model does not converge during learning or that the adaptive parameters diverge. The global feedback makes working with IIR networks problematic. However pure FIR implementations can only model large memory depth systems at the price of many time-delays. Thus feedback is often desired.

### 3.2.4 Focused time-lagged architectures and gamma networks

The attention of the neural network community has clearly shifted from recurrent networks and TDNNs to focussed time-lagged neural networks, see for example the first [Haykin, 1994] and second edition [Haykin, 1999] of Simon Haykin's introduction to neural networks. Several authors have contributed both theoretical as empirical evidence, that favor focussed time-lagged neural networks (FTLNNs) over networks with delay elements or feedback within the neural network [Mozer, 1994; Sandberg and Xu, 1997a].
Chapter 3

MLP-based dynamic models

A justification for the use of FTLNNs, apart from the design problems with recurrent networks and TDNNs encountered in practice, is Sandberg and Xu’s Myopic Mapping Theorem [Sandberg and Xu, 1997b], that states: *Any shift-invariant myopic dynamic map can be uniformly approximated arbitrarily well by a structure consisting of two functional blocks: a bank of linear filters feeding a static neural network.*

The mapping \( g \) is shift invariant, if the mapping gives as output at time \( \alpha - \beta \) the same value as it would have at time \( \alpha \) applied to the variable delayed \( \beta \), i.e. it is invariant time, formally if and only if eq. 3.6 holds.

\[
\forall \beta : (Gx)(\alpha - \beta) = (Gz^{-\beta}x)(\alpha)
\]  

(3.6)

In figure 3.5 Focussed Time-lagged Neural Nets are shown. FTLNN supporters claim the only remaining problem, considering MLPs are universal approximators [Hornik, 1989], is the estimation of a good linear filter bank. This linear filter bank is the only part of the architecture capable of storing temporal information, which relieves both network and learning algorithm of all the expensive dynamic extensions. A promising type of network belonging to this class is the Gamma Network [deVries and Principe, 1992] which trades memory resolution for memory depth by applying a generic delay kernel using limited feedback to guarantee stability while allowing for large time scopes [Sandberg and Xu, 1997a; Principe et al., 1992].
The difference between the Gamma Network and a TDL MLP is that each tap in the delay-line has a feedback connection, figure 3.6a. The forward and the feedback connection weights are defined by a single adaptive parameter \( \mu \), which is equal to the memory resolution of the gamma TDL. The output of each tap is computed according to equation 3.7. Similar to the generic non-linear kernel model, equation 3.3, equation 3.32 is the kernel function of the gamma neuron. The response of the delay kernels for gamma neurons is shown in figure 3.7.

\[
x_k[n] = (1 - \mu)x_k[n-1] + \mu x_{k-1}[n]
\] (3.7)

![Gamma delay kernels: obtained through a step-response with \( k=4 \) and \( \mu = 0.2 \)](figure)

\[
g_j[n] = \left( \frac{n-1}{j-1} \right) \mu^j (1 - \mu)_{n-j} 1_{j \leq n} \Rightarrow G_j(z) = Z(g_j[n]) = \left( \frac{\mu}{z - (1 - \mu)} \right)^k
\] (3.8)

\[
n_k = \sum_{n=0}^{\infty} ng_k[n] = Z\{ng_k\}[z = 1] = -z \frac{d}{dz} G_k(z) = -z \frac{d}{dz} \left( \frac{\mu}{z - (1 - \mu)} \right)^k = \frac{k}{\mu}
\] (3.9)

The sampling time for one tap of the gamma memory is computed according to equation 3.9. Memory depth is the largest sampling time minus the smallest sampling time. Hence for a gamma memory, this is \( n_k \) for the “oldest” tap minus \( n_0 = 0 \) for the newest output giving \( n_k - n_0 = \frac{k}{\mu} \). The memory resolution is the reciprocal of the sampling period (equation 3.10)

\[
\frac{1}{\Delta n_k} = \frac{1}{k + \frac{1}{\mu} - \frac{k}{\mu}} = \mu
\] (3.10)

Hence memory depth (D) and memory resolution (R) are coupled through the relation \( D \times R = \) constant. Gamma networks have been successfully applied to speech recognition [Lawrence et al., 1997], outperforming the ordinary TDL MLP and TDNN. Similar results are reported in a comparison of FGN, TDL and TDNN to time-series prediction and system identification [Principe et al., 1992].
Table 3.1 gives an overview of the properties of dynamical extended neural networks based on an MLP. While the locality of the feedback helps to improve the stability, due to the non-robust learning configuration, this gamma filter is no snake oil to dynamical modeling.

Table 3.1: Properties of dynamical extensions of neural networks

<table>
<thead>
<tr>
<th></th>
<th>FIR</th>
<th>GAMMA</th>
<th>IIR</th>
</tr>
</thead>
<tbody>
<tr>
<td>Stability</td>
<td>always</td>
<td>for $0 &lt; \mu &lt; 2$</td>
<td>non-trivial</td>
</tr>
<tr>
<td>Depth vs. order</td>
<td>coupled: $k$</td>
<td>semi-coupled $\frac{k}{\mu}$</td>
<td>decoupled</td>
</tr>
<tr>
<td>Learning complexity</td>
<td>$O(k)$</td>
<td>$O(k)$</td>
<td>$O(k^2)$</td>
</tr>
</tbody>
</table>

3.3 Neural estimation

Iterative learning procedures in neural networks behave different from parameter estimation in linear models, as discussed in chapter 2. The neural estimation algorithms are summarized in subsection 3.2.1. In subsection 3.2.2 we discuss in detail the error back-propagation algorithm, since this procedure is an essential ingredient to understanding the typical aspects of neural estimation. Some extensions of the error back-propagation algorithm, required to facilitate the learning in dynamic MLPs, are presented in subsection 3.2.3.

3.3.1 Procedures for fitting data

Neural learning algorithms are designed for a particular neural architecture and use. There are two different types of learning: supervised and unsupervised. Supervised learning is based on the availability of examples that are input-target pairs, such that an error can be computed given an input-output response of a neural network. The error is subsequently used for adjusting weights; this is also called “learning by a teacher”. Unsupervised learning only requires data-patterns that are used to fit for example probability densities against the data distribution. Supervised learning is usually also based on minimization of squared error. The derivative of weights to error $J_w e[n]$ is the basis for weight-updates is called steepest descent.

Step-wise improvement of the model configuration through an update function is a procedure we call learning. Parameter estimation is a technical procedure rather than a mystical process. Nonetheless the resulting behavior remains obscure and intriguing in that it seems to be erratic and chaotic. There is yet no comprehensive model of the global learning behavior with steepest descent in non-linear neural networks. The original steepest descent algorithm has been extended with several update functions for several reasons, among which are the mathematical soundness of the algorithm and the convergence rate of the algorithm. Table 3.2 lists the most prominent descendants of the EBP update function [Saarinen et al., 1991].

Table 3.2: Overview of learning algorithm [Saarinen et al., 1991].

<table>
<thead>
<tr>
<th>Learning Algorithm</th>
<th>Search direction</th>
</tr>
</thead>
<tbody>
<tr>
<td>SD - Steepest Descent</td>
<td>$\Delta w[n] = -J_w^T e[n]$</td>
</tr>
<tr>
<td>CG - Conjugate Gradient</td>
<td>$\Delta w[n] = -J_w^T e[n] + \beta \Delta w[n-1]$</td>
</tr>
<tr>
<td>N - Newton</td>
<td>$\Delta w[n] = -J_w^T (J_w^T J_w + \sum_{i=1}^m e_i [\nabla_w^2])^{-1} e[n]$</td>
</tr>
</tbody>
</table>
For conjugate gradient and steepest descent the convergence is worse than for many other second-order learning methods and the learning method is not exact. Nonetheless, where quality is an issue, bad configurations can be discarded in the model selection independent of data, while learning problems are highly characteristic for the behavior of the data; hence learning problems can serve to characterize the normal (learning) behavior.

### 3.3.2 Error back-propagation

Despite the quality of many “better” learning algorithms, error back-propagation remains a popular and intuitively comprehensive learning algorithm. The basic idea is to walk in the direction which will decrease the error most, also the direction of steepest descent on the error-surface. Moreover the error back-propagation algorithm reveals more interesting dynamics than “so-called” theoretically sound or optimal learning algorithm. As EBP plays such a elementary role for our signature computation we will dedicate a few pages to a summary on it’s background and derivation. Error back-propagation is a learning technique presumed to be discovered by several researchers in parallel [Rumelhart and McClelland, 1985; Parker, 1986; LeCun, 1985] while the basic idea behind the error back-propagation is first described by Werbos in his PhD. thesis [Werbos, 1974]. Through the rediscovery of error back-propagation in the mid 1980s neural networks revived from some hard strokes of criticism uttered by Minsky and Papert in their book Perceptrons [Minsky and Papert, 1969].

The task we consider here is the modeling of some unknown mapping between an input and an output. The mapping to be estimated is only available through samples of patterns. The pattern error expresses the distance between actual and desired output, as expressed in equation 3.11, with the pattern evaluated at time rather than the pattern from the sample.

The estimation task is the minimization of a cost-function w.r.t. the free model parameters , usually SSE (Sum of Squared Errors) over all targets in the pattern , as expressed in equation 3.12. The pattern error is a vector while the sum of squared errors is a scalar.

1. The use of the common notation in place of is avoided here because has a special meaning in the context of detection, which may easily cause confusion in the next chapters.
\[ E_{SSE}[n](w) = \frac{1}{2} e_w[n] e_w^T[n] = \frac{1}{2} \sum_j e_j(w)^2[n] \]

\[ E_{SSE}(w, v) = \frac{1}{2} \sum_j e_j^2(w, v) \]  \hspace{1cm} (3.12)

The ASE (average squared error) is the error over an entire set of patterns, equation 3.13, which is the total cost-function to be minimized for a complete database of samples \( D = (\xi_i) \) containing a total of \( N = \sum_\xi \) patterns. This empirical cost-function reflects the data-driven approach we are taking. A probabilistic function approximation formulation will rather consider the entire data space \( \mathbb{R}^p \) and introduce the a priory probability \( p_\theta(v) \) of a pattern \( v \) to be drawn given information source \( I_\theta \), as in equation 3.13.

\[ E_{ASE}(D) = \langle E_{SSE}(v) \rangle_D = \frac{1}{N} \sum_\xi \sum_v E_{SSE}(v) \]  \hspace{1cm} (3.13)

The idea of any steepest descent algorithm is to find the direction which will decrease the error most. This direction is given by the derivative of each parameter to the observed error, as in equation 3.14 based on the instantaneous error \( E_{SSE}[n] \). Based on the average squared error \( E_{ASE}(D) \) it should be multiplied by \( \frac{1}{N} \).

\[ \frac{\partial E[n]}{\partial w_{ji}[n]} = \frac{\partial E[n]}{\partial e_j[n]} \frac{\partial e_j[n]}{\partial y_j[n]} \frac{\partial y_j[n]}{\partial v_j[n]} \frac{\partial v_j[n]}{\partial w_{ji}[n]} = e_j[n] \cdot -1 \cdot \varphi'_j(v_j[n]) \cdot y_j[n] \]  \hspace{1cm} (3.14)

The new weight \( w_{ji}[n+1] = w_{ji}[n] + \Delta w_{ji} \), using an adaptation \( \Delta w_{ji} \) without the use of a stabilizing momentum, is then obtained straightforward from equation 3.15 with some positive learning rate constant \( \eta \), which determines how fast we will be running down-hill, multiplied by the gradient \( \delta_j[n] = \frac{\partial E[n]}{\partial w_{ji}[n]} \), also the local gradient \( \delta_j[n] \) times the connection input \( y_j[n] \).

\[ \Delta w_{ji}[n] = \eta \frac{\partial E[n]}{\partial w_{ji}[n]} = \eta \delta_j[n] y_j[n] \]  \hspace{1cm} (3.15)

The computation of the local gradient depends on the transfer function and connectivity of the neuron, i.e. being a hidden or a output neuron. For output neurons the local gradient is expressed by equation 3.16. No error propagation is required as the local error is provided at the output.

\[ \delta_j[n] = \frac{\partial E[n]}{\partial e_j[n]} \frac{\partial e_j[n]}{\partial y_j[n]} \frac{\partial y_j[n]}{\partial v_j[n]} = e_j[n] \varphi'_j(v_j[n]) \]  \hspace{1cm} (3.16)

The local error of hidden units does not come from a target, but needs propagation of the provided error signal back through the following layer, figure 3.8. Hence the term error backpropagation. With the local gradients now available in the following layer, the local error can be computed as the derivative w.r.t. to the output error of the neuron output \( y_j \). The output error of the hidden neuron \( j \) is given by the back-propagated errors, i.e the local gradients in the preceding layer times correction weight \( \delta_k[n] w_{jk}[n] \). The resulting local gradient is then obtained similarly by the error signal \( e_j[n] \), times the derivative of the transfer function w.r.t. it’s activation, as expressed in equation 3.17.
For the common sigmoid transfer function with constant slope \( a = 1 \), the gradient has the nice feature that \( \phi'(x) = \phi(x)(1 - \phi(x)) \). Hence the local gradient computation reduces to equation 3.18. The process of back-propagation is then continued to compute all the weight gradients from the last hidden layer through all preceding layers till the input neurons are reached which do not have any adaptive parameters. Going back to equation 3.15, shows that with all the weight gradients available, the weights can be adapted and we are done. This is what error back-propagation is all about.

\[
\delta_j[n] = \phi'(v_j[n]) \sum_k \delta_k[n] w_{kj}[n] = \phi'(v_j[n]) \delta_j[n] w_{kj}[n] = y_j[n] (1 - y_j[n]) \sum_k \delta_k[n] w_{kj}[n] \tag{3.18}
\]

A full discussion on the pros and cons of EBP is beyond our scope; however some characteristic features deserve our attention. The adaptation of a parameter only requires availability of fan-in, fan-out and back-propagated error. Hence EBP is a local adaptation method, which solves the credit assignment problem posed by Minsky and Papert. This property of locality, which is mostly biologically inspired and a general feature of connectionist models, makes it especially suitable for parallel computation. However in the output error the internal model dependencies are obscured. These dependencies are the conglomerate effect of all the chosen weights. The learning rate parameter is much criticized as it seems a rather tricked way to make the idea of steepest descent work. The EBP algorithm is not guaranteed to converge. It suffers from various problems such as instability, local minima and premature saturation [Lee et al., 1990] as discussed in the next section. The convergence of the EBP algorithm can be improved by the use of a stabilizing momentum [Rumelhart et al., 1986; Qian, 1999]. The heuristic is that contradicting gradients should damp the speed of adaptation, while persistent adaptation is amplified. This heuristic is realized by the momentum term \( \alpha \Delta w_{ji}[n-1] \), resulting in the learning rule of equation 3.19.

\[
\Delta w_{ji}[n] = \alpha \Delta w_{ji}[n-1] + \eta \delta_j[n] y_j[n] \Delta w_{ji}[n] = \eta \sum_{t=0}^{n} \alpha^{n-1} \delta_j[t] y_j[t] \tag{3.19}
\]

The learning rule (equation 3.19) is not entirely correct to optimize the cost-function of the batch error \( \xi_{\text{ASE}} \). The batch or exact learning rule, also called delta rule, minimizes the average squared error by adapting the weights in the direction of the average gradient. This learning approach is called batch learning.

\[
\Delta w_{ij} = -\eta \frac{\partial \xi}{\partial w_{ij}[n]} = -\frac{\eta}{N} \sum_{n=1}^{N} e_j(n) \frac{\partial e_j[n]}{\partial w_{ij}[n]} \tag{3.20}
\]

The instantaneous learning approach, i.e. backpropagating after each evaluation, expressed in equations 3.18 and 3.19 is called pattern learning (or on-line learning). In batch learning the stability of the weights is much less an issue and does not require a momentum term. Theoretical analysis reveals that pattern pattern learning closely approaches the batch learning for small values of the learning rate \( \eta \) [Heskes and Wiegerinck, 1996]. Due to the arbitrary ordering of patterns, pattern learning is a stochastic learning process while batch-learning is a
deterministic procedure, i.e. the effect of ordering is removed through the averaging process in equation 3.20. We discuss the randomness of learning process in section 3.4.

3.3.3 Learning in dynamic neural networks

The a priori unknown amount of variables (state and delay variables in particular) makes dynamical modeling a different and harder class of problems than function approximation and classification. The analysis and feature selection as well as the model selection need a more extensive approach, as a result of the unknown and higher dimensionality. The configuration of the learning process requires some additional issues to be resolved. A good overview of design issues in dynamical neural modeling is found in [Maier and Dandy, 2000]. The differences compared to static MLP design w.r.t. learning in dynamically extended MLPs are:

- **initialization time**
The dynamical model relies on past inputs and internal state. Previous inputs are not available during the first evaluations. Hence the error computed after evaluation is an overestimation of the actual error. To resolve this, in absence of internal feedback, transform the data to a static mapping, i.e. a training pattern explicitly stores the required amount of past values of some variables. Apart from the wasted amount of memory and disk space, this approach will not work for feedback models. Two more common approaches are: a) skip several learning cycles, i.e. evaluate only before learning. The memory depth of a model can be estimated using the techniques described above. This provides a reasonable estimate for the initialization time; b) rely on the batch error rather than back-propagate the instantaneous errors.

- **instantaneous error invalid**
The instantaneous back-propagation algorithm is not analytically correct for dynamical neural networks. The problem is that the current output of the model is determined by
past values of the weights and internal state rather than by their value at the time of back-
propagation [Haykin, 1994; Benvenuto et al., 1994]. Hence the instantaneous gradients
do not apply to current weights put to their past value. Alternative algorithms take care of
this problem by using a recursive learning procedure such as the RTRL or unfolding the
architecture (TDNN) such that the previous values of weights are explicitly stored. The
latter solution is expensive in terms of the number of additional required parameters.

• **randomness not be implemented by permutation**
Randomness is a crucial element in neural learning, as discussed in section 3.3.4. Incorpor-
ating randomness in the learning process through permutation of the patterns, which
is a common approach in both pattern and batch learning [Heskes and Wiegerinck, 1996],
cannot be used in dynamical modeling as the observations need to be evaluated in order.
Several solutions to this problem exists. One is to randomize samples rather than observ-
ations. Alternatively observations may be evaluated by the subsequent EBP step and
may be skipped randomly.

• **Dynamic vs. spatial problem**
Parameter estimation for dynamical models consists of two intertwined processes. The
current state and input of a model needs to be mapped to the desired output; this is a spa-
tial problem similar to that of function approximation. Meanwhile also suitable feedback
and delay settings have to be selected; this is a dynamic problem which affects the behav-
or of the models memory. These two processes interact in complex ways during learn-
ing, which makes the learning process of dynamical models less robust compared to
classification and function approximation. Learning parameters need to be chosen with
more care. Often the momentum term is dropped and batch learning is preferred over pat-
tern learning. In our experience pattern learning and a momentum term can be used in
dynamical models, but the momentum term is best annealed to zero. Due to increased
dimensionality of the problem, a large memory/internal state is introduced, hence over-
parameterization is more common in dynamic modeling than static modeling.

It is clear that dynamic neural networks require special training algorithms. There are several
extensions for recurrent neural networks, as applied for process identification, such as the
RLS (Recursive Least Squares) and BRLS (Block Recursive Least Squares) [Parisi et al.,
1996], but als the BPT (Backpropagation through time), FNBP (Folding in Time), IBP
(instantaneous backpropation) and CBPT (Causal BP) [Benvenuto et al., 1994]. Real-time
recurrent learning (RTRL) algorithm is used for learning in recurrent neural networks [Will-
iams and Zipser, 1989] tailored to small dynamic neural networks. The focused time-delay
learning algorithms do not require specific changes to the basic error back-propagation algo-

If \( \mu \) is adaptive, the learning algorithm called Focused Back-propagation [deVries and Princ-
ipe, 1992] trades memory depth for memory resolution, focusing on the frequencies actually
present in the signal. The adaptation of the weights of the gamma neuron is given by the equa-
tion 3.21 which is the standard weight adaptation without momentum derived earlier.

\[
\Delta w_k[n] = \eta e_j[n] x_k[n] 
\]

The adaptation of the memory resolution, which is shared among the taps in the gamma
tapped delay-line, is derived by the derivative of error to parameter \( \frac{\xi[n]}{\mu[n]} \). Up till the deriva-
tion $e_j[n] = \frac{\epsilon[j][n]}{y_j[n]}$ of the local error at the gamma filter output $y_j$, there is no difference. The derivative for the local gradient $\frac{\partial e_j}{\partial \mu}$ is then given in equation 3.22.

$$
\frac{\partial e_j}{\partial \mu} = \sum_{k=1}^{K} \frac{\partial e_j}{\partial x_k} \frac{\partial x_k}{\partial \mu} = \sum_{k=1}^{K} e_j w_k \frac{\partial x_k}{\partial \mu}
$$

(3.22)

Then the problem that needs to be solved is taking the derivative $\frac{\partial}{\partial \mu} x_k[n]$. When the computation of $x_k[n]$ as given in equation 3.7 is used to expand this expression one directly obtains:

$$
\frac{\partial}{\partial \mu} x_k[n] = \frac{\partial}{\partial \mu} (1 - \mu) x_k[n - 1] + \frac{\partial}{\partial \mu} \mu x_{k-1}[n]
$$

(3.23)

The derivative in equation 3.24 is obtained in which one recognizes the recurrence, applying the product-rule. This is explicitly stated by equation 3.25, where $\alpha_k[n] = \frac{\partial}{\partial \mu} x_k[n]$. Thus obtaining the gamma learning rule, with learning rate parameter $\eta$, of equation 3.26. The algorithm is initialized with $\alpha_k[0] = 0.000$ from which all further $\alpha$ can be derived.

$$
\alpha_k[n] = (1 - \mu) \alpha_k[n - 1] + \mu \alpha_{k-1}[n] - x_{k-1}[n] + x_k[n - 1]
$$

(3.25)

$$
\Delta \mu[n] = \eta \sum_{k=1}^{K} e[n] w_k[n] \alpha_k[n] \text{ with } \alpha_k[n] = \frac{\partial}{\partial \mu} x_k[n]
$$

(3.26)

The adaptation of the temporal focus of the network has a severe impact on the requirements for the mapping on the static network, i.e. the mapping required of the static neural net can severely change as the time-window and resolution at the input are altered. Vice versa the time-window to optimize a mapping expressed by the neural network can cause the gamma filters to change. Instability during learning requires some kind of regularization. The learning problem is particularly hard as the behavior in $\mu$-space is non-convex; the bifurcation at $\mu = 1$ is particularly hazardous. Three approaches appear to be helpful in resolving this problem: a) start with a small learning rate for the feedback parameter $\mu$; b) constrain $\mu$ to a small range [Veelen et al., 1999]; and, c) use batch learning rather than pattern learning. Ordinary local filters, such as context units, have uncoupled feedback parameters. Further a gradient descent algorithm will greatly suffer from non-convex behavior and instability, and the coupling of the $\mu$ parameter in the gamma filter reduces the complexity by restricting the degrees of freedom (hence it is considered a form of regularization).

### 3.3.4 Convergence and stopping criteria

The training error should not be used for the stopping criterion, as the training error is biased toward the training set: the score on a testing set better reflects the score on the available data. A pitfall is the introduced dependency between model and testing set if the testing set is used to determine the stopping criteria. To estimate the performance of the final model a third validation set should be drawn from the data separately from the training and testing set.

An equilibrium in the learning process is characterized by the observation that model parameters cannot be steered independently with the available data, i.e. either parameter adaptations...
Neural design and learning issues

Chapter 3

Neural modeling, consisting of model design and learning, is known to have particular problems with the consequence that particular expertise is required to come up with a good neural solution. In this section we analyze why neural modeling and estimation is difficult and summarize attempts to solve the problems. In subsection 3.4.1 we start out with a discussion on neural model features that are distinct compared to conventional models. The typical problems with modeling and estimation are summarized in subsection 3.4.2. The problems are analyzed particularly to relate them to the typical features of neural modeling in section 3.4.3. In the past 30 years of research into neural networks much energy was spent on solving these problems. This has provided several design heuristics and modifications to the learning procedure; these are summarized in subsection 3.4.4.

3.4 Neural design and learning issues

Several features of neural networks and learning processes distinguish neural modeling from conventional models and estimation. These features are related to the neural design problems and learning that are encountered in practice. Before we analyze these problems, here are the distinctive features:

- Non-linearity
- Stochastic Nature of the Learning Process
- Lack of Semantic Interpretation (data-driven/black-box)
- Diffuse and redundant information storage
- Fault-tolerance

are coupled statically, or adaptation in the “right” direction for one pattern is cancelled by (the combined) effect of other learning samples within a learning epoch [Veelen et al., 2000].

Convergence cannot be guaranteed in general; hence there are no well defined stopping criteria. In many cases it will be unfeasible to distinguish local and global minima. At a minimum, the first-order derivative of the error-function is required to be zero w.r.t. to it’s weights:

$$\nabla_w \mathcal{E}[n] = 0$$ implemented as $$\|\delta(w)\| < \varepsilon_\delta$$ (3.27)

A stopping criterion based thereon is a stable sufficiently small gradient vector [Kramer and Sangiovanni-Vincentelli, 1989]. However it will take many epochs before this stop criterion is reached. The stationary behavior of the error in the equilibrium can also be used as a stopping criterion. The learning process has converged when the error is sufficiently small. Unfortunately ‘sufficiently small’ can hardly be defined.

$$ w = w^* + \delta(\xi, w^*) \text{ and } \mathcal{E}(\xi, w) = \mathcal{E}(\xi, w^*)$$ (3.28)

The control problem can be observed in the dependency of the gradients: i.e. there is no mean gradient improving the model, while individual patterns cause non-zero and structural gradients: expected value of error-surface is influenced by noise level and model capacity. The variance in the error can be estimated from data and reflects the uncertainty while gradient dynamics reveal typical dynamics of learning due to specific dynamics in the data.
• Interpolation is good (generalizing)
• Extrapolation is poor, worse than linear models

Functional complexity is usually described by the class of mappings that can be realized by the model. Generic non-linear kernels have much to offer as they allow for more complex mappings. The modeling capability of a neural network is practically only limited by the number of non-linear nodes and connections. Therefore the neural model is easily made redundant. Redundancy is measured through SVD or PCA [Emmerson and Damper, 1993]. The ratio of (significant) basis factors and the number of weights is an indicator for the redundancy in the network. The redundancy of neural networks is associated with fault-tolerance, since the information is distributed. Part of the model can be lost before the model fails.

The neural model deviates from conventional models as it is black-box, particularly due to (a) distributed information storage, (b) non-minimal number of degrees of freedom, (c) the connectionistic aspects that couples model architecture directly to the learning procedure, and (d) the iterative learning that requires some randomness causing a stochastic behavior.

### 3.4.2 Observed neural design and learning problems

The key design issue is determining the right coding of the problem to be solved. The effectiveness of a neural solution is highly dependent on the way data is presented to the neural network. Some preprocessing techniques generally applied for time-series modeling discussed in section 2.23 should also be applied for neural models. Disappointing results of a neural network application often comes from skipping the problem analysis and transformation one would ordinary pay attention to. When a neural model and it’s learning algorithm are drawn from a magic hat it is not clear what is the actual problem to solve. It is always a good practice to consider first which cost-function is to be optimized.

The first symptom in neural learning is the unpredictable and stochastic nature of the learning process. There are sudden jumps in the behaviour during adaptation, it is apparent chaotic behaviour, and there can be an apparent instability of the learning process.

A second related symptom, despite the universal approximation capability, is that the asymptotic convergence to a global optimum cannot be guaranteed. There are three typical behaviors: a) premature saturation: (local) a neuron freezes but the network still learns; b) stagnation: (global) the entire network freezes but a better solution is known to exist; c) dynamic equilibrium: there is no convergence but also no improvements, the adaptation effects seem to cancel each other out.

The third and fourth symptoms relate to the plasticity-stability problem, discussed extensively by Grossberg [Grossberg, 1976]. In academic examples the neural models have been shown to memorize and forget. Memorizing means the input-output pairs are stored but the relation between input and output is not represented by the model, i.e. it does not generalize. When the neural model is capable of learning in the dependencies in a dataset but it fits one general pattern at the cost of earlier information, then it is unstable and forgetting.

The fifth symptom, observed when considering sets of neural solutions, concerns the robustness of neural modeling and neural models. Among a set of fitted neural networks the quality of the solution varies, some learning processes may have fail entirely, there is no way to verify if the "best" solution is not found. Considering the architecture of a model one can
observe that sometimes one neuron can be crucial while in other cases all neurons appear equally important. Neural models can be robust, but the internal information distribution is unpredictable.

A frequently heard complaint about neural networks is the accuracy of the solution is not acceptable, as a better solution is known to exist. In cases where a physically or logically plausible model of a complex function or process exists, it can hardly be expected that any universal statistical method can fit an equally good solution from input-output examples. Consider a chaotic Lorentz system, without a model it may be impossible in finite time to identify it from data.

The seventh symptom in neural learning and evaluation is that it can be extremely sensitive to non-stationary inputs and variance in the data. This is observed as erratic behavior. Related this symptom is the poor extrapolation beyond the domain of example data.

The eight symptom, similar but not the same as the second, is that learning can be very slow, hence time-consuming, with sudded improvements.

These symptoms indicate the complicated neural architecture design and learning process configuration. There are many hyper-parameters to be determined: i) model architecture: delays, feedback, layers, neurons, transfer function; ii) learning process: initialization, stopping-criterion, rates, momentum; iii) data: pre-processing, coding of targets, sampling, randomization. The quality is highly sensitive to these hyper-parameters. In the next subsection we analyze the possible causes of these observed symptoms.

### 3.4.3 Problem analysis: typical features causing problems

There are several means to analyse the learning behaviour and design problems of neural networks. We consider here neural measures and neural analysis techniques. There are exogeneous measures and endogeneous measures w.r.t the model. Exogeneous measures are input-output responses and learning-curves (error through time). The endogeneous measures can be local or microscopic or macroscopic. Microscopic measures consider the behavior of individual weights independent of the rest of the model [Amari, 1990]. Typical microscopic measures are neuron activations or outputs, and neuron weights and biases through time, The behavior of individual weights, even when analyzed w.r.t. to inputs and outputs, will reveal more than seemingly random movements in the equilibria, as the interactions of data and weights are often very complex and not independent. Macroscopic measures take the overall behavior of the model into account, e.g. consider the global states in the learning process [Amari, 1990]. Macroscopic measures are the Jacobian and Hessian and estimates of the error-surface that can be obtained through weight-space sampling or perturbation of weights.

The Jacobian is the derivative of the weights w.r.t. all the pattern errors. Hence the matrix contains all the feedback signals of a sample. The Jacobian is expressed in equation 3.29, where \( e_j = e(x_j) \) or \( e_i = e(x_i, s_i) \) denotes the internal state and \( x_i \) is the input of the \( i \)th pattern of \( \xi \). For analysis the Jacobian is sometimes computed from the model output rather than from the output error. This exception is denoted \( J^{(M)} \) hence \( J^{(M)}_{ij} = \nabla_w M(x_j) = \frac{\partial}{\partial w} M(x_j) \). The Hessian matrix is the second derivative of the cost-function \( \mathcal{E}(w) \) with respect to \( w \), i.e. \( H = \frac{\partial^2 \mathcal{E}(w)}{\partial w^2} \). Though computationally expensive some of the proposed algorithms rely on estimation of the Hessian and it’s inverse, as both are very characteristic of the error-surface.
In case of stochastical estimation problems the rank of a matrix as an exact measure is not that useful. As an alternative one can consider the “importance” of the Eigen-vectors of the matrix, perturbation analysis is used to locally estimate the error surface by perturbing either weights [Minai and Williams, 1994] or input-output data. The error-surface in the neighborhood of an equilibrium can be estimated using the second-order expansion, equation 3.30 (note that the gradient is zero in the equilibrium)

$$\varepsilon(w) = \varepsilon(w^*) + \frac{1}{2}(w-w)^TH(w^*)(w-w)$$ (3.30)

The Eigen Analysis or rank-determining methods are well-founded on linear estimation theory. A matrix needs to have a full-rank to be invertible. If this is not the case then the matrix is a diagonal matrix with diagonal, is rank deficient if $$\sigma_{r+1} = \sigma_{r+2} = \ldots = \sigma_q = 0$$ and $$\sigma_r \neq 0$$. The Degree of rank-deficiency is then $$q-r$$.

In case of stochastical estimation problems the rank of a matrix as an exact measure is not that useful. As an alternative one can consider the “importance” of the Eigen-vectors of the matrix, by comparing their Eigen-values. The rank-deficiency of a matrix, usually of the Jacobian or the Hessian, can be expressed by the condition number. The condition number can be estimated by comparing the ordered Eigen values of the matrix. The condition number is than given by equation 3.31 where $$\sigma_r$$ is the smallest non-zero Eigen value. For a large value of $$\kappa$$ the matrix is ill-conditioned. If this holds for either the Jacobian or the Hessian of the error function, learning problems will arise

$$\kappa(A) = \frac{\sigma_1}{\sigma_r}$$ (3.31)

The model’s capability does not match the complexity of the problem. The issue here is balancing generalization and memorization. Generalization is the capability to extract an underlying concept from a limited number of manifestations. Fitting parameters of a generic approximator to closely resemble the functional relation between observed variables may be enforced by physical principles but obscured by irrelevant interferences and noise. Memorization is the incorporation of information contained in the training data not specific for the
underlying information source. Over-sized models tend to memorize the data-patterns from the test set rather than extract the underlying function.

- **If the complexity of the model is too high (a)** the sampling of weight space is too fine-grained. Over-parameterized models will have parameters that are under-determined, or parameters that are dependent. Memorization will occur if the model is too large. A model is overfitting if the degrees of freedom in the model are larger than the degrees of freedom in the data. The ratio $\frac{n_y}{\dim(w)}$ has been proposed [Amari, 1997] to estimate the overfitting of a neural model. The generalization can be estimated [Ponnapalli et al., 1999] according to equation 3.32.

- **If the complexity of the model is too low (b),** a solution does not exist at all. If the complexity is too low even if it’s good, then there can still be learning problems, as the models architecture may be unviable.

\[
R^{(GF)} = \frac{\hat{C}^{(\text{test})}}{\hat{C}^{(\text{train})}}
\]

(3.32)

**Dependencies.** The configuration of the learning process can be seen as a control design problem. The control perspective (problem of controllability section 2.4.4) sheds new light on the training problem, as many learning problems are related to the limitation of pushing the weights in the right direction given the model architecture and the available data.

- **Correlated Inputs (a)** cause a learning convergence problem. In iterative LMS large Eigen values of the auto-correlation-matrix indicate strong dependence, in which case convergence is not guaranteed [Widrow, 1976]. LMS and linear gradient methods are sensitive to non-stationary and correlated inputs. This sensitivity can be expressed by the condition number of the auto-correlation matrix.: $\chi(R_x) = \frac{\lambda_{\text{max}}}{\lambda_{\text{min}}}$. Intuitively, the relations will be harder to find when there is more intrinsic dependency in the data, and the controllability of the weights is small when the connectivity of the neural network dictates dependencies in weight adaptations.

- **Instantaneous (static) dependencies between weights (b).** The connectivity of the network causes bottlenecks in the back-propagation of errors from one layer to the previous. Consequently dependencies will appear. Learning problems are caused by the dependency of the steering vectors as contained in the Jacobian [Zhou, 1998; Aires et al., 1999]. If learning fails every time, these networks are called non-viable architectures [Nablan and Zomaya, 1994].

- **Dynamic interactions between weights, or weights and training data (c).** Dependencies between inputs and between neuron outputs cause large Eigen-values in the Hessian matrix. The Jacobian tends to have a large number of dependent columns (nearly linear dependent) [Saarinen et al., 1991; Wilson et al., 1997] causing rank-deficiency. Hence if a learning process does not converge, this does not mean the neural network is not redundant.

- **Non-unique solutions (d).** There are many local minima of the error function, each of them occurring $N!$ times because of symmetry (all the permutations of $N$ hidden units). Finding the global minimum is unlikely [Wilson et al., 1997].
• **Locality of the gradient descent approach (e).** The gradient descent approach is spatially local as it disregards its neighbors adaptation. The approach is also local in time as previous adaptations are not considered. Temporal and spatial dependencies can result from this, causing cancellation effects.

**Problematic Error Surface.** When we zoom into the learning problems we end up studying the error surface, i.e. the error as a function of weights. This function is determined by model architecture and data together. The EBP algorithm fails in case of **non-differential error** w.r.t. weights or **non-smoothness (erratic) error-surface**, The Eigen vectors of the Hessian reflect the smoothness of the error-surface in the point of operation. If the error surface is non-smooth, local instantaneous gradients do not represent the local mean gradient of the error-surface. This kind of chaotic behaviour (small changes give large impact) occurs particularly in feedback models where the parameters have poles.

**Coding of the problem.** The representation of the information is of crucial importance to neural modeling. In practice over 80% of the design trajectory consists of selecting and implementing a solvable problem representation, i.e. problem analysis and feature selection. It is not always possible to solve an estimation or identification problem by coding. There are two key attributes that should be resolved by a properly chosen coding:

• **Ill-conditioned problem (a).** Large Eigen-values of the Hessian result from wide variations in the second-order derivatives of the cost-function to individual weights; particularly they are much smaller for synapses in the first layer compared to the last layer.

• **Biased inputs or neurons (b).** Non-zero mean input signals or non-zero mean neuron outputs cause large Eigen-values in the Hessian matrix. If, in the Eigen-values of the Hessian matrix one usually find a relatively large number of medium-sized Eigenvalues, learning will be hard.

**Numerical precision and error computation accuracies.** This issue is often overlooked. Numerical precision limitations are a hazard in neural networks. Erosion of the feedback signal is a serious problem since there are mostly multiple layers and many error feedback computations. Finite-precision coding, e.g. for FPGA or ASIC implementation of neural networks, indicate it is not a trivial problem [Diepenhorst et al., 2001]; in some cases truncation may have the same effect as randomization which can speed up learning at the cost of the accuracy of the end result. Numerical precision is not merely an issue limited to finite-precision coding. Non-linear models cascading many layers, through which the error must be fed back, require a high dynamic range for coding the weights.

**Sampling issues with the data: sample size and resolution.** There are two potential data problems, apart from coding related aspects. The first is an **Insufficient amount of examples and good coverage.** The statistical issue of data amount is related to model complexity given the data to be fitted as discussed in section 2.4.5. A second potential problem is a **non-uniform sampling.** It is not a typical neural problem. Expectations on black-box methods are often too high to solve problems that are not solved by classical methods. If the problem coding is poor black-box models should not be expected to outperform classical methods that fail.

The **stopping criterion can be hard to decide.** A rule-of-thumb is that stationary behaviour of the error is a sign of equilibrium, but an equilibrium occurs in local optima too.
The wrong choice of learning parameters can lead to instable and chaotic behavior [van der Maas et al, 1990]. A fixed learning rate and momentum have the disadvantage that weights on slopes of the error-surface are adapted as fast or slow as weights on rough flats of the error surface. The choice of learning parameters should at least depend on the neural architecture and dimensions. Use of multiple hidden layers and dynamic extensions tend to decrease stability of the learning process and induce learning stagnation. Learning too fast causes instability and chaotic behaviour, whereas learning too slow is time-consuming and comes at the risk of getting stuck in a local minima.

Limited trajectories are followed through the weight space. A deterministic learning process can get stuck in a local minima or a saddle point causing cancellation and indecisiveness [Barakova, 1999]. Symmetry and even non-controllability due to overconstrained architectures are instances of learning processes being too deterministic. This is what the Bayesians address with randomness. The limitations of the weight-space sampling can also come from using too few initial random initialized weight.

Neurons or parameters become irrelevant. In the extremes of the sigmoid the derivative is always small, which causes saturation. A large redundancy of the model as a whole makes some weights nearly useless, i.e. each individual weight does not hold much information. Number of weights can be traded for numerical precision. Due to the architecture of a model some connection weights cannot be estimated from data.

Neural modeling is tedious as consequence of several factors that have been described in this subsection. The Bias-Variance problem related to model complexity and the controllability limitations in relation to the plasticity-stability problem, are the most dominant causes of observed design and learning problems. Model complexity and the controllability problem are also related as a wrong choice of architecture can limit the learning process.

3.4.4 Neural design heuristics and architectural modifications

The general idea is to manipulate the three primary influences DATA, MODEL and UPDATE to realize a projection of the real Jacobian to a Jacobian with a reasonable condition number. The common approaches to mold the learning process into a better performing estimation procedure are based on automating hyperparameter selection and adaptation, aimed at: a) a uniform distribution of information in the model, and b) a learning process that is controlled and asymptotically converges. The procedures discussed below are supporting neural design automation, as shown in the figure 3.9.
Complexity penalty. The uniform approximation capabilities seem to provide unlimited capabilities for neural modeling. Learning and design problems, discussed above, reveal that not any architecture works as long as sufficient degrees of freedom are added. Due to non-linearity and redundancy in a Multi-Layer Perceptron it is not at all clear whether each weight in a neural network counts as a single parameter so that the degree of freedom is the number of weights -1 as is the case for linear models. Statistical analysis of the problems has triggered a search for models of optimal size. This has resulted in including the complexity of a model as part of the estimator risk, such that the variance of the model or unreliability of the method can be reduced. A typical cost-function including complexity penalty \( \lambda \) is shown in equation 3.33.

\[
R(w) = \mathcal{E}_d(w) + \lambda \mathcal{E}_c(w)
\]  

(3.33)

Note that these complexity penalties never occur in physically plausible models. The optimization methods discussed in this subsection are about finding the appropriate penalty for complexity, such that learning is good and results in a “good” model. There are three stages in the learning where regularization can be applied:

- **Initialization of learning process.** The complexity of a mapping has been analyzed from the scaling of the inputs, the distribution from which the initial weights are chosen, and the number of hidden neurons [Atiya and Ji, 1997].

- **Regularization of model or learning rate.** Weight space regularization can help to solve the controllability problem by reducing the effective number of columns in the Jacobian [Zhou, 1998; Aires et al., 1999], thereby improving the condition number. The Bayesians have proposed weight-priors for regularization [Castellano, 1997; Ponnapalli et al., 1999].

- **Termination to preventing specialization or increasing fault-tolerance.** The model architecture can cause premature saturation. This can be prevented by using a small
value range, and a small number of hiddens, to keep the neurons in their linear region, [Lee et al., 1991]. The number of effective parameters $p_{\text{eff}}$ [Moody, 1992] can only be determined if the noise/variance of the data is known a priori. Table 3.3 gives an overview of complexity penalty terms.

<table>
<thead>
<tr>
<th>[Moody, 1992]</th>
<th>$&lt;\mathcal{E}(\lambda) &gt;<em>{\mathcal{L}</em>{\text{train}}, \mathcal{L}<em>{\text{test}}} = &lt;\mathcal{E}(\lambda) &gt;</em>{\mathcal{L}<em>{\text{train}}} + 2\sigma</em>{\text{eff}}^{2} \frac{p_{\text{eff}}(\lambda)}{n}$</th>
</tr>
</thead>
<tbody>
<tr>
<td>[Cherkassky et al., 1999]</td>
<td>$\mathcal{E}_{c}(\lambda, w) = \lambda|w|^{2}$</td>
</tr>
<tr>
<td>[Haykin, 1994]</td>
<td>$\mathcal{E}_{c}(w, k) = \frac{1}{2} \int \left| \frac{\partial}{\partial x} M(w, x) \right|^{2} \mu(x) dx$</td>
</tr>
<tr>
<td>Penalty used in SARPOP [Treadgold and Gedeon, 1999]</td>
<td>$\mathcal{E}<em>{c}(\lambda, w) = \frac{\lambda}{2} \sum</em>{ij} \ln(1 + w_{ij}^{2})$</td>
</tr>
</tbody>
</table>

**A priori model selection.** Model selection is the term for choosing an appropriate architecture for the model. In white-box approaches this is always a priori. In neural networks model selection takes place a priori, a posteriori and dynamically w.r.t. the learning. Coding is one aspect of model selection concerning targets and cost-functions, data analysis and pre-processing; this was covered in section 2.2.3. A well-chosen internal neural architecture can prevent learning problems or even speed up learning by reducing the complexity. There are many examples of a priori model selection: a) use modular or hierarchical models if regimes or states and state conditions are known; b) start with a very large model that will surely be capable of fitting the data; c) select the proper transfer function; d) a non-zero mean transfer function in the neurons can cause systematic bias which harms the learning; and, e) building in dependencies (linked weights, as with the gamma filter) to prevent instability.

**Dynamic and a posteriori model selection with metrics.** The dynamic or a posteriori selection or rather modification of a neural architecture depends on metric-based heuristics. Several heuristics exist for selecting the appropriate structure. Through structural monitoring non-viable structures can be detected [Hecht-Nielsen, 1987]. Such an approach is based on the type of architecture [Nabhan and Zomaya, 1994]. There are specific methods for increasing and decreasing methods as may appear necessary during the learning process:

- **Increasing model complexity to meet needs or improve learning.** Bottlenecks in error back-propagation can result from the input-target coding. A solution is to widen the channel, i.e. change the problem representation. This is achieved by increasing or diversifying inputs and targets through some transformations. Some applicable preprocessing and output-coding transformations are discussed in chapter 2. In cluster networks, a more common remedy is to increase the number of neurons, such as in the Resource Allocation Network (RAN) [Roberts and Tarassenko, 1994] and in the ART network adding nodes [Grossberg, 1976]. Combinations of adding and deleting exist, like RAN & Pruning [Molina and Niranjan, 1996]. Adaptive regularization and pruning techniques are especially suitable for modeling non-stationary process [Kaishansen and Edwaradrasmussen, 1994; vandeLaar and Heskes, 1999]. In function approximation dynamic
growth is also applied, such as Addel [Ji, 1997], DNAL (dynamic node architecture learning). [Bartlett, 1994] takes an information-theoretic approach. Example of iterative estimates for model size are the constructive cascading network [Treadgold and Gedeon, 1999] that combines bootstrapping with early-stopping to determine the size of the model. Evolutionary algorithms find a natural application in the selection of model size, such as mixed-mode learning and PLAN [Teng and Wah, 1990]. Fluctuation of behaviour of a neuron indicates insufficient capability. In that case one may split the “mother” neuron into two neurons [Weng and Khorasani, 1996].

Table 3.4: An overview of sensitivity measures use for pruning neural networks

<table>
<thead>
<tr>
<th>Neuron Sensitivity</th>
<th>$S_i = \sum_j \left( \frac{\partial \varepsilon}{\partial w_{ij}} \right)^2$</th>
</tr>
</thead>
<tbody>
<tr>
<td>Local Relative Sensitivity Index</td>
<td>$S_{jk}^{LRRI} = \frac{\left</td>
</tr>
<tr>
<td>Neuron relevance</td>
<td>$\rho_j^+(x) = \sum_i \left</td>
</tr>
<tr>
<td>Saliency in OBD</td>
<td>$S_i = \frac{w_{ii}^2}{2[H^{-1}]_{ii}}$</td>
</tr>
<tr>
<td>Information Theoric Relevance</td>
<td>$I(y_p, x_j(i)) = U(x_j(i), y_p) - \sum_{m \neq i} R(x_j(i), x_m(i))$  [ U(x_j(i), y_p) = \frac{H(x_j(i)) + H(y_p) - H(x_j(i), y_p)}{H(x_j(i)) + H(y_p)} ]  [ R(x_j(i), x_j(i)) = \frac{H(x_j(i)) + H(x_j(i), y_p)}{H(x_j(i)) + H(y_p)} ]</td>
</tr>
<tr>
<td>Relevance used in Skeletonization</td>
<td>$\varepsilon^{(\text{without neuron } i)}(\xi) - \varepsilon^{(\text{with neuron } i)}(\xi)$</td>
</tr>
</tbody>
</table>

- **Decrease complexity to remove variations and minimize model size**, or to make the model interpretable. Redundancy in the form of free-energy can have a negative effect on the dynamics of the learning process and prevent convergence. The first ideas about restructuring neural networks have been formulated in the late eighties [Hanson and Pratt, 1989; Mozer and Smolensky, 1991; Sietsma and Dow, 1989]. In [Reed, 1993] one finds an overview of pruning techniques. The optimal brain damage (OBD) approach [LeCun, 1990] removes neurons from the network, whereas algorithm and the optimal brain surgeon (OBS) [Hassabi et. al., 1992] removes only connections based on a Hessian. After pruning one should continue to learning process to adjust the weights [Castelanno, 1997]. Pruning may cause a degradation of learnability. The methods for pruning connections or nodes in a neural network are based on the importance or sensitivity of a single node relative to the function represented by the model, to determine the impact of
a single parameter on the functional behavior of the whole. A common approach in sensitivity analysis is to determine the change or degradation in performance when weights are slightly modified or perturbated. Such a sensitivity analysis for neural networks was first performed on Madaline [Stevenson, 1990]. The first perturbation analysis of Multi-layer Perceptrons was published two years later [Young Choi and Choi, 1992]. Table 3.4 lists the measures used in the OBD, OBS and other related methods.

\[
\Delta w_{ji}[n] = \alpha \Delta w_{ji}[n-1] + \eta[n] \delta[j][n] y[j][n]
\]

(3.34)

A priori configuration of learning process. There are a number of ways to constrain the learning process in a deterministic fashion, i.e. not data-dependent. Firstly in a constrained initialization a complexity penalty is derived that constrains the initialization of weights such that the neurons do not saturate early in the learning process [Atiya and Ji, 1997]. Secondly, complexity can be constrained through selecting the data samples. Thirdly, convergence can be forced through deterministic selection of time-dependent learning parameters, for example by reducing the learning rate and momentum using a monotonic decreasing function. This technique is called annealing. Though the learning process converges in the sense that at a certain moment improvements will no longer be made, the converged model is likely to be enforced in local minima. A more elaborate time-dependent learning rate is the STC (search-then-converge) schedule [Darken et al., 1992]. Before a certain time \( \tau \) the learning rate is approximately constant (the “search” stage) while after \( \tau \) the learning rate is annealed (the “converge” stage). A serious drawback is the introduction of another two design parameters, \( c \) and \( \alpha \) next to the initial learning rate \( \eta_0 \). Fourthly, one can linearize the learning process so that it resembles EKF. Preventing the non-linearity can significantly speed-up learning [Ruck et al., 1992] and theoretically provides asymptotical approximation of the non-linear model. The fifth and last approach is the use of stopping criteria. Static a priori stopping critiera are: i) to use a finite number of learning-cycles, or ii) to define an absolute error level, based on a priori known noise. The bias-variance problem actually states the uncertainty about the noise level. These stopping criteria do not require convergence of the learning process; hence a stable equilibrium is not always reached.

<table>
<thead>
<tr>
<th>Table 3.5: Deterministic time-dependent learning parameters</th>
</tr>
</thead>
</table>

<table>
<thead>
<tr>
<th>Annealing</th>
<th>Search-then-converge</th>
</tr>
</thead>
</table>
| \[
\begin{pmatrix}
\Delta \eta \\
\Delta \alpha
\end{pmatrix}
= \begin{pmatrix}
c^{(\eta)} & 0 \\
0 & c^{(\alpha)}
\end{pmatrix}
\begin{pmatrix}
\eta \\
\alpha
\end{pmatrix}
\]
with \( c^{(\eta)} < 0 \) and \( c^{(\alpha)} < 0 \) | \( \eta(t) = n_0 \frac{1 + \frac{c}{\eta_0 \tau} t}{1 + \frac{c}{\eta_0 \tau} + \frac{\tau t^2}{2}} \)

Dynamic configuration of the learning process. Instability of the learning process leads to stagnation which can be prevented by tuning the learning rate and by using different learning parameters and algorithms for different types adaptive parameters and weights. The choice of a suitable step-size influences whether the global optimum is reached. Individual learning parameters for different connections or neurons are generally better than global learning parameters.
• **Low-pass filtering and heuristical adaptation based on previous updates.** Dynamic and adaptive learning-rate algorithms usually taking the form of equation 3.34. This is in principle a low-pass filter of the gradients. The use of a momentum term is not always effective in reducing the weight oscillations [Ochiai et al., 1994].

• **Second-order learning rules.** In case of erratic error surfaces rather use microscopic metrics to improve the stability of the learning process [Wong, 1996]. Second-order learning rules make use of equation 3.30. Heuristical methods estimate the local error surface without explicitly computing the Hessian.

• **Heuristical measures to improve convergence and plasticity** of the learning process [Jacobs, 1988]. Most heuristics are derived from the following: **if the (local) error (surface) is erratic or conflicts in time, the learning rate should be decreased as there is much uncertainty on the search direction; if similar (local) errors occur, the learning rate can be increased as there is not much uncertainty on the search directory.** Table 3.6 provides an overview of heuristical adaptations for learning parameters. The factor $\beta$ is a normalization factor applied in the conjugate gradient method. The dynamics can be explained, as $\beta(k) = 0$ for large $k$ and for small values of $\angle(\nabla_w E(k), \nabla_w E(k-1)) < 0.2^1$, while stabilization and acceleration stops when subsequent gradients are uncorrelated $\nabla_w E(k), \nabla_w E(k-1) = 0$.

• **Randomization of data or weights.** The determinism in the learning process can be solved by breaking the symmetry in the architecture as well as in the data. Randomness can help to hop to another part of the weight-space if the learning process gets stuck due to indecisiveness or cancellation.

• **Stopping criteria.** In a minimum, the first-order derivative of the error-function is required to be zero w.r.t. to it’s weights. $\nabla_w E[n] = 0$. Stationary behavior of the error in the equilibrium, $w = w^* + \xi(w^*)$ and $E(\xi, w) = E(\xi, w^*)$, is a stopping criterion. However an equilibrium in the learning process as observed from a stable mean error can also result from uncontrollable model parameters or cancellation [Veelen et al., 2000]. Alternatively gradients $\|\delta(w)\| < \varepsilon_\delta$ can be a measure of an equilibrium [Kramer and Sangiovanni-Vincentelli, 1989], but equilibria can be dynamic and still be optimal, the optimum being a basin rather than a single point. The stopping-criteria comparing response on test and training data is known as cross-validation [Hecht-Nielsen, 1990].

Using early stopping can obstruct modeling of non-linearities [LeBaron and Weigend, 1998], which may not be desirable. The search for optimal stopping time has been criticized by [Masters, 1993; Ripley, 1994], claiming an optimum cannot be determined and it is hazardous to try since learning is stopped prematurely.

### Table 3.6: Heuristics for adapting learning parameters

| [Magoulas, 1997] | $\eta(t) = \frac{1}{2L_k} \text{ with } L_k = \frac{\|\nabla_w E(w(t)) - \nabla_w E(w(t-1))\|}{\|w(t) - w(t-1)\|}$ |

---

1. The intended edge between two vectors $a$ and $b$ here is $\angle(a, b) = \frac{a^T b}{\|a\|\|b\|}$
Optimal conditions for the convergence have been investigated from the perspective of coupled damped harmonic oscillators [Qian, 1999]. Both heuristics and the theoretically founded learning rate adaptations and rules improve the convergence of the learning process and the achieved performance of the model. Accelerations are typically in the order of 10 and 100 compared to standard EBP, while performance improvements vary from nil to factors 1000 [Xiao-Hu Yu and Guo-An Chen, 1997; LeCun et al., 1993; Haykin, 1999].

**Detecting learning problems.** There are three main causes of learning problems one would like to detect: a) dependencies in data are determined from a priori conflicts in the input-target pairs, while correlation between inputs is measured by the condition number of the auto-correlation matrix: \( \chi(R) = \lambda_{\text{max}} / \lambda_{\text{min}} \); b) cancelating and similar dynamic equilibria are determined from periodicities, e.g. using standard data analysis techniques of section 2.2.2; c) uncontrollability is associated with wildly varying weights, and non-adjusting and non-important weights. If the network does not contain enough neurons to represent and learn a specific map, then the weights tend to fluctuate and may even never converge [Weng and Khorasani, 1996].

Apart from all the heuristics discussed there are some straightforward solutions to the observed problems. First of all one may refrain from directed search like gradient descent, and use random field theory, since many of the problems come from iterative learning. Second, it is an option not to use the non-linear Perceptron, thus preventing the struggling with the hardship of non-linear modeling. There are also some extensions to neural design that have proven to be useful. The amount of models that is used for training or even in the final application can be increased to improve reliability, using bootstrapping and boosting resp.. There are also ways to improve the semantics such that the model is more easy to interpret and improve. Examples thereof are: a) rule-extraction, and b) built-in-semantics e.g. fuzzy-neural models or Bayesian decision rules.

### 3.4.5 Status-quo of neural design and learning issues

Higher-order approximation of the error surface yields learning rules, adaptive learning parameters schema’s and architecture selection and restructuring methods. While many successes of
such approaches have been claimed, the controllability problems of the neural learning process appear to extend to these higher-order extensions. Some crucial questions have been raised by Saarinen [Saarinen et al., 1993]: will higher-order methods speed up learning? is the mathematical formulation the correct one? is the difficulty (in neural learning) an intrinsic feature of the neural network architecture (rather than of the data or the training algorithm). After three decades of research we are still faced with issues as:

- Hyper-parameter selection remains a problem: the space is not continuous differentiable. Computing power allows us having a poor yield on experiments.
- Higher-order statistics from learning behavior can hardly ever be estimated reliably. Only for large amounts of data the statistics converge, but the "on average" does not explain sufficiently.
- Interpretability of neural networks remains poor. One accepts the black-box model as it is or alternatively one chooses to include much a priori (false?) knowledge.
- Most design time is still spent on data analysis to get the problem coding right. The black-box models hardly reduces the effort for problem coding compared to conventional modeling approaches.
- Capturing dependencies within high-dimensional high-volume of data is still limited by available computing power.
- Introducing heuristics often comes with new hyper-parameters.
- The learning process is still hard to understand and interpret.

Randomization, regulatization and pruning are attempts to overcome the modeling problems. Localization of redundancy, design and selection of viable architectures and learning constraints rely on neural metrics. The neural design process relies heavily on heuristics and metrics; this is still an active field of research. Furthermore we can conclude, there is no generic solution relating the cause of learning problems to adequate design remedies; hence the neural design relies heavily on expert knowledge.

3.5 Summary

We have provided the theoretical basis for modeling and estimation using dynamic neural networks. Analysis of design complications and attempts to overcome them make use of second-order estimations of the error surface, either in a mathematical, information-theoretical or heuristic manner. The toughest problems come from complexity estimation and the inherent redundancy of a neural model. Most learning problems are caused by a wrong choice of architecture. The discussion in section 3.4 shows that the merely adding of weights does not always improve the model; it is clear that redundancy becomes manifest in different, sometimes problematic ways. The skepticism on neural modeling is due to the ambiguous relation between observed design complications and their causes, as well as the ambiguous relation between the complications and the proposed remedies. Through the discussion of design complications we have introduced first and second-order metrics of the learning process. These metrics are considered for signature computation from neural modeling for detection, discussed in chapter 8.