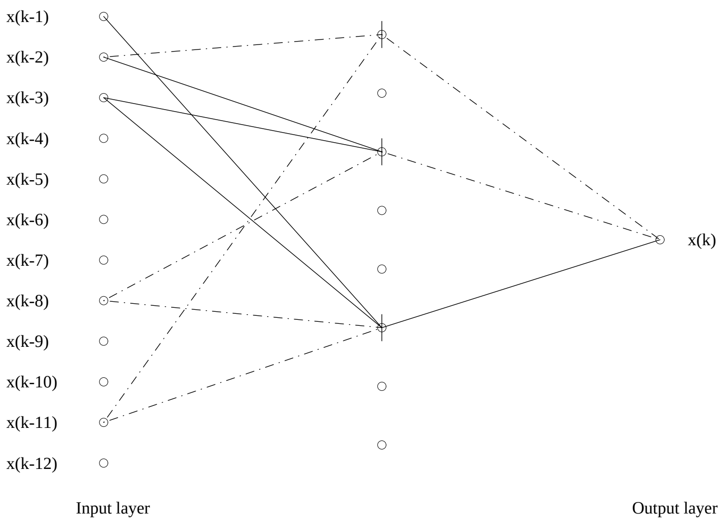
Neural Networks for Signal Processing



Ph.D. Thesis
by

Claus Svarer
ph.d. nr. 91-0112-134

CONNECT, Electronics Institute
Technical University of Denmark
DK-2800 Lyngby
Denmark
Email: csvarer@ei.dtu.dk

rev. May 17, 1995
December 30, 1994

Abstract

In this thesis, methods for optimization of neural network architectures are examined in order to achieve better generalization ability from the neural networks at tasks within signal processing. The feed-forward networks described have one hidden layer of units with tanh activation functions and linear output units. The major topics described in the thesis are:

- Reducing the number of free parameters in the network architecture by pruning of parameters. Pruning is based on estimates (Optimal Brain Damage) of which parameters induce the least increase in the network performance criterion (the cost-function) when they are removed from the network.
- Finding methods for estimation of the generalization ability of the network from the learning data set. A generalization error estimate (Akaike's Final Prediction Error estimate) is used for choosing the optimal network architecture among different pruned network configurations.
- Using methods for on-line tuning of the different parameters in the network optimization algorithms. The gradient-descent parameter is set using a second order Gauss-Newton method, while the weight-decay parameters are set using a method based on an optimization of the estimate of the generalization ability of the network.

The methods proposed for network optimization are all examined by experiments. Some of the experiments are done on data sets from the literature, so it has been possible to compare the obtained results to what has been reported in previous work. The examples are selected from the areas of classification, time-series prediction and model building for non-linear systems.

Synopsis

I denne afhandling er metoder til optimering af arkitekturen af neurale netværk undersøgt med det formål at opnå bedre generalisationsevne for de neurale netværk anvendt på signal behandlingsproblemer. De beskrevne neurale netværk er af "feed-forward" typen med et lag af skjulte enheder med tanh aktiveringsfunktioner og lineære output enheder. De væsentligste emner, der er behandlet i afhandlingen, er:

- Reducering af antallet af parametre i netværks arkitekturen ved at fjerne parametre fra netværket. Metoden (Optimal Brain Damage), der er anvendt til at fjerne parametre fra netværket, baserer sig på estimater af hvilken parameter, der forøger netværkets "performance" kriterium (den optimerede funktionen) mindst, når parameteren fjernes.
- Finde metoder til at estimere generalisationsevnen for netværkene ud fra indlæringsdatasættet. Et estimat af generalisationsevnen (Akaike's Final Prediction Error estimat) anvendes til at udvælge den optimale netværks arkitektur imellem forskellige reducerede netværks arkitekturer.
- Anvendelse af metoder til løbende at justere de forskellige parametre, der indgår i algoritmerne anvendt til optimering af netværkene. "Gradient-descent" parameteren justeres ved at anvende en anden ordens Gauss-Newton metode, mens "weight-decay" parameteren justeres ved at anvende en metode, der er baseret på en minimering af estimatet af netværkets generalisationsevne.

De foreslåede metoder til optimering af netværkene undersøges alle på eksempler. Nogle af eksperimenterne er udført på datasæt, hvor resultater er beskrevet i litteraturen. Dette har muliggjort en sammenligning af de opnåede resultater med, hvad der tidligere er rapporteret. Eksemplerne er udvalgt imellem emner som klassifikation, tidsserie-estimering og estimering af modeller for ikke lineære systemer.

Preface

The work presented in this thesis was done as a part of a Ph.D study at the Technical University of Denmark, started in September 1991 and finished in December 1994. The work took place at Electronics Institute, with Lars Kai Hansen, Peter Koefoed Møller and John Aasted Sørensen as supervisors. Electronics Institute is a part of the Computational Neural Network Center, CONNECT.

I wish to thank my advisors for their support throughout the study. In particular I wish to thank Lars Kai Hansen for his always kind and helpful support and for providing comments and suggestions which improved the quality of the thesis. I also wish to thank all the co-authors of the articles appended (Lars Kai Hansen, Jan Larsen, Anders Krogh, Carl Edward Rasmussen, Peter Salamon, Ole Winther, Jan Gorodkin, Torben Fog, Lars Hupfeldt Nielsen, Søren Holm, Ian Law, Olaf Paulson, Christian Linneberg, and Jan Meyrowitsch), Niels Mørch, Electronics Institute and Benny Lautrup, Niels Bohr Institute, CONNECT for their participation in many profitable discussions. Finally, I will like to thank Gina Vianelli who have had a hard job trying to improve the English language used throughout this thesis.

The work was granted by the Danish Natural Science and Technical Research Councils through the Computational Neural Network Center, CONNECT. I would like to thank The Otto Mønsted Foundation for financial support in connection with conference participations.

Lyngby,
December 30, 1994.

Claus Svarer

Contents

Chapter 1

Neural Networks

This chapter contains an introduction to the area of artificial neural networks, with emphasis on their use within different signal processing tasks. The different components in neural networks are introduced, and a discussion of the way these components can be combined is presented. A definition of the specific neural networks that are used for signal processing tasks within this thesis is given. It is also attempted to give a concise notation for the networks used throughout the work. In the last section a description of the structure of the report is given.

Throughout this report **neural networks** stands for **artificial neural networks** for simplification. This abbreviation is also used in most literature.

1.1 Introduction to Neural Networks

The idea of neural networks can be traced back to the 1940s where [McCulloch and Pitts, 1943] explained how simple calculation units, *neurons*, in the brain work together in parallel, exchange information, and in that way combine information and make decisions. This concept of having a lot of simple units doing similar calculations is the basic idea of artificial neural networks. Clearly it is desirable to imitate the human brain by a computer program. The human brain has a high degree of fault tolerance built-in; if a computing unit in the brain dies (this takes place all the time) the brain is still working well, it is just adopting to the new configuration without any need for modifications. This quality would be very helpful if it was possible to build it into a computer program. Further the brain is very flexible and fast in learning new tasks. Just think of the brain's capability to process the input from the eyes; it can be used for processing the input when reading a book, or driving a car, without any need for modification. It is possible to drive the car to places the person never has seen, and the brain automatically knows how to process these new images to instructions for the legs, arms, etc. Also the processing speed and handling of noisy and inconsistent information in the brain is superior to any known artificial image processing system. Humans just need a quick glimpse of a situation to recognize what is going on, whereas artificial image processing systems of today are slow and need the images to be taken and preprocessed in a way so they are very similar to what the systems have seen before in order to be able to recognize objects in the images.

Hence, the human brain is superior to artificial neural networks. One of the reasons for this of course is that it in no way has been possible to perfectly imitate the way the human brain works. The human brain consists of about 10^{11} simple computing units called **neurons**. These are working in parallel, and exchanging information through their connections **synapses**. The human neurons are summing up all the information that is going into them and if this sum is high enough, over a given potential called the **action potential**, they send a pulse through their connections to other neurons. After a pulse the activated neurons have a sleeping period, called the **refractory period**, where they are inactive. In the paper [McCulloch and Pitts, 1943] the way the human brain works was described, and this work has formed the basis for the people that began working in artificial neural networks.

Artificial neural networks describe a way of doing nonlinear modeling of a set of inputs into outputs. It consists of a set of simple calculation **units** (in the human brain the **neurons**) and connections (**weights** or **parameters**) between these units (from the human brain the **synapses**). Each of the units are connected to other units by a weight. The units then calculate a weighted sum of the inputs and translate it to an output through a function called a **squashing function** or **transfer function**. In figure 1.1 a simple artificial neural network model with recurrent connections (there are loops in the network structure) is sketched.

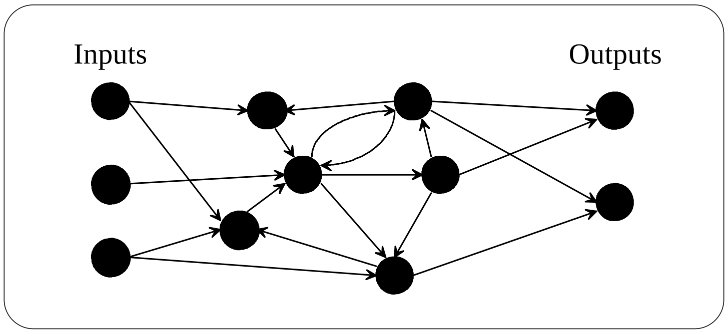


Figure 1.1: Recurrent neural network with feed-forward and feed-back connections (recurrent means that there are loops in the network structure).

This way of modeling an input-output relation was proposed in [Rosenblatt, 1958; Rosenblatt, 1962], and shortly after in [Widrow and Hoff, 1960; Widrow, 1962]. Rosenblatt called his model a **perceptron** and Widrow called his model an **adaline** model. They both define a nonlinear relation between the inputs and outputs from the model, but only [Rosenblatt, 1962] defined the multi-layer network (the layered structure of neural networks is defined in the next section). They both found algorithms for tuning the weights from examples, but it was only for networks with one layer of units (no hidden units).

Rosenblatt's algorithm tunes the weights with the non-linear perceptrons in place, while Widrow's algorithm tunes the weights with linear units. In the algorithm the non-linear units then are inserted after tuning the weights! It was first when [Rumelhart et al., 1986b] found the **Back-Propagation algorithm**, which can tune weights in multi-layer networks from examples of input-outputs, that this way of doing non-linear models has shown all the capabilities that it has. This method for learning the connections between inputs and outputs is called **supervised learning** or **learning by examples**. An alternative learning technique is called **unsupervised learning**, where the learning is done by a statistical expectation of how the output varies over the inputs. This learning technique uses one or more of the following features in the input space: familiarity, clustering, feature mapping, principal component analysis, etc. [Hertz et al., 1991].

The first examples which Widrow used his single layer network for, were for recognition of objects (simple shapes) in images. In the papers [Widrow and Hoff, 1960; Widrow, 1962] the networks have been tested within recognition and separation of different shapes, and especially the networks' generalization ability has been examined. Because of the limited capacity of single layer networks, it was first when [Rumelhart et al., 1986b] found the learning algorithms for multi-layer networks that the network modeling technique was spread out into a broad field of applications. Some of the first examples of using multi-layer networks are also found within the area of image processing, for example recognition of hand-written digits and letters [Le Cun et al., 1989]. But many applications have also emerged within the field of non-linear signal processing, e.g. prediction of time-series and estimation of non-linear system models [Weigend et al., 1990; Weigend and Gershenfeld, 1994].

1.2 Feed-forward Neural Networks.

In this thesis the work will be concentrated on networks that are especially suitable for tasks within the area of signal processing. The work is further concentrated on networks that can be optimized and analyzed in a relatively simple way.

Supervised learning is very well suited for signal processing tasks, both classification, time-series estimation and others. An input is processed by the network and the output is then compared to a desired output. The error between the correct answer and the answer of the network is then used to change the weights/parameters in the network, with the purpose of minimizing the difference between the network output and the desired output. The learning strategy is explained in more detail in chapter 2.2.

1.2.1 Connectivity in the Neural Network

In figure 1.1 a neural network with recurrent connections is shown. This means that there are loops of connections in the network. From linear filter/model theory [Oppenheim and Schafer, 1975; Ljung, 1987] it is known that recurrent connections in models make it possible, with fewer parameters, to model systems with a more complex impulse response. In linear theory the filters/models with recurrent connections are called **infinite impulse**

response filters (IIR-filters), where filters with only forward connections are called **finite impulse response** filters (FIR-filters). As for linear models, recurrent connections in neural network models make it possible to have infinite impulse response from the model. This would give the model a possibility to implement a kind of memory in the network itself; the new outputs from the network are dependent on earlier inputs/outputs given to the network model. However, it is more difficult to analyze and optimize recurrent neural networks, and the algorithms which have been proposed for tuning the weights, for example reinforcement learning and RTRL learning [Hertz et al., 1991], have not been very successful. In particular, the learning rate for the algorithms seems to be very low.

In the rest of this work we will concentrate on feed-forward neural networks, because they are especially simple to optimize and analyze. Further it can be shown that they have a general function approximation capability [Hornik et al., 1989; Cybenko, 1989; Hornik, 1991] if enough connections and units are incorporated in the model. That is, any continuous function can be approximated by this kind of neural network. In figure 1.2 a multilayer feed-forward neural network is sketched. The characteristic of the feed-forward network is that there are no loops of connections in the networks, and each layer in the network is connected only to the next layer in the network. The layers of neurons in the middle of these networks are called hidden layers because there is no direct connection to inputs or outputs. The learning method derived in section 2.2 only has to be extended a little, when weights in multilayer networks have to be trained. This kind of neural network was named **multilayer perceptron**, when first studied.

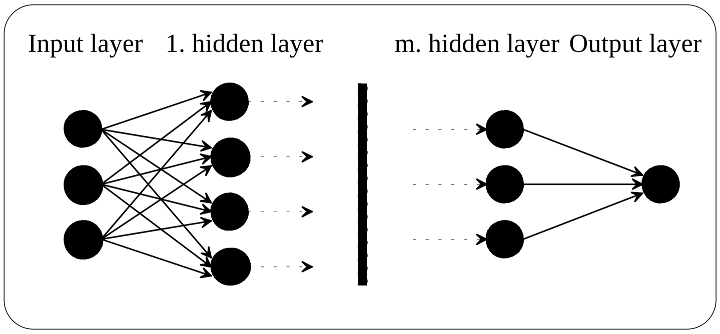


Figure 1.2: Multilayer feed-forward neural network.

1.2.2 Different types of Neurons

In the brain models developed by [McCulloch and Pitts, 1943], each neuron has a transfer function with the characteristic that it sums up the weighted inputs. If this sum exceeds a threshold value, the neuron fires (gives an output). After a neuron has fired, it has a refractory period before it is possible for the neuron to fire again. This can be generalized to a simpler neuron model where the updates of the output from the neurons are done asynchronously, in random order. The delay is then not explicitly specified. The model can then be sketched as shown in figure 1.3 (inspired by [Hertz et al., 1991]).

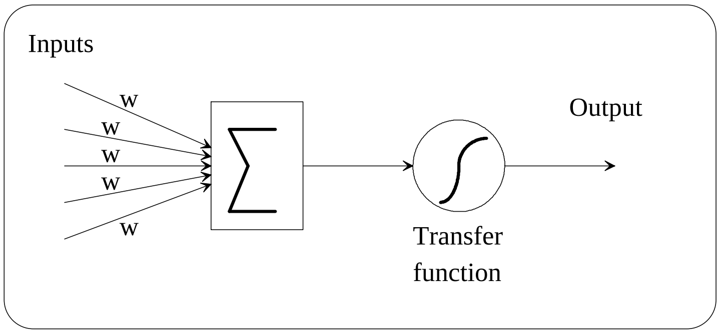


Figure 1.3: Model of a brain neuron. The transfer function is sketched as a tanh function, but this can be a step function as well as used in the original neuron model by [McCulloch and Pitts, 1943].

The equation for this model of a neuron can be written as

$$O_i^\mu = g(\sum_j w_{ij} \cdot \xi_j^\mu - w_{0j}) \tag{1.1}$$

where $g(\cdot)$ is a step function which fires if the argument is larger than 0. The weights in the neuron model are w_{ij}, while the term w_{0j}, which is subtracted from the weighted sum, is the threshold value that the sum should exceed to make the neuron fire. For pattern μ the j'th input pattern to the neuron is ξ_j^μ, while O_i^μ is output from the i'th neuron.

This model of brain neurons is the inspiration for the units in the artificial neural networks, which are used throughout the thesis. They are summing up the weighted input and using a function to transfer this to an output. In the model unit there is normally not a refractory period built in. Several different functions have been proposed as transfer functions for the units. Some of them are shown in figure 1.4. The **tanh function** or **sigmoidal function** with a threshold value (varying with the length of the refractory period and in general with the time) are the transfer functions that best model the brain neurons. These functions also have the property that they are differentiable, which is a most desirable property of the transfer function. In section 2.2 where the learning algorithms for the weights is derived, this is shown. When the **linear function** is used as the transfer function for the units, the network will only have a linear capability to fit functions. In many cases this will not be enough to solve problems, and to show the full usefulness of the neural network way of modeling. Therefore the most common transfer functions are either the tanh function or sigmoidal function, which give the possibility of building non-linear models. The only difference between these two transfer functions is the scaling and the symmetry around 0. The same functions can be fitted with both functions, but the learning algorithms have a better numerical stability, with the tanh because of the symmetry around 0. Another possibility is to use a **step function**, but this function is not differentiable. In chapter 2.6.1 it is discussed, why it is important to have zero mean value signals in the network.

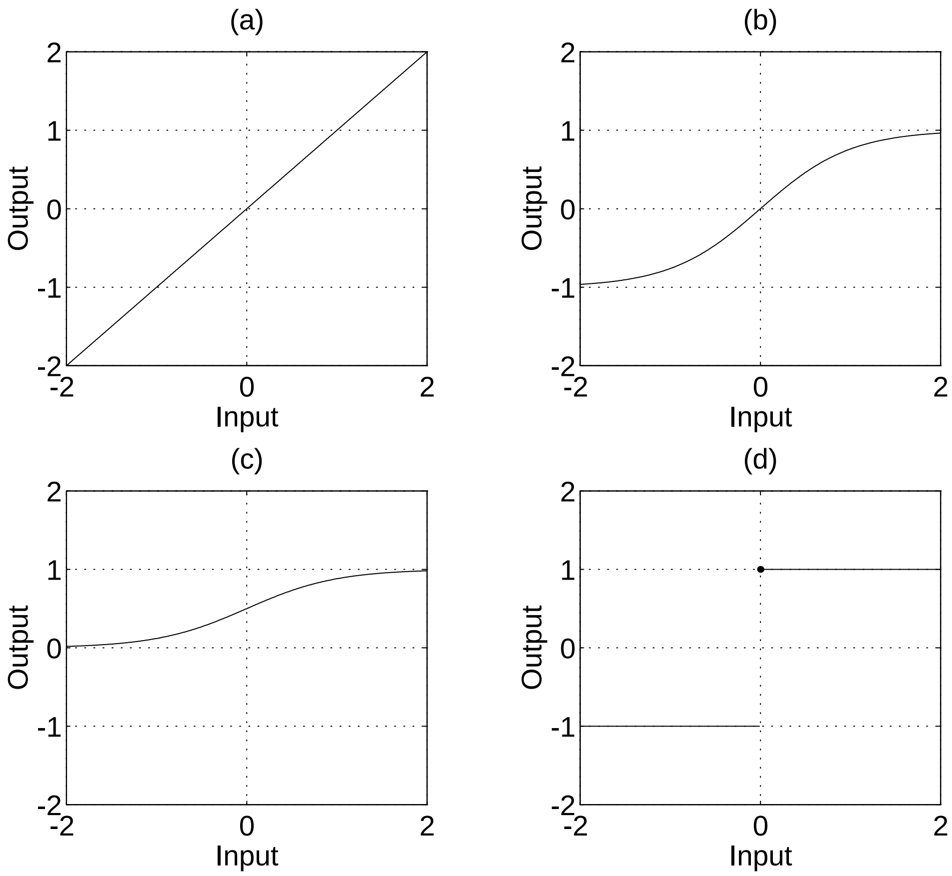


Figure 1.4: Different transfer functions for model neurons, (a) linear function, (b) tanh function, (c) sigmoidal function, (d) step function.

Another type of unit which can be used is the **radial basis function** unit [Moody and Darken, 1988; Moody and Darken, 1989]. This can be described as shown in the following equation

$$\phi_i^\mu = g(\|\overline{\xi_j^\mu} - \overline{C_j}\|) \tag{1.2}$$

where $\|\cdot\|$ is a distance measure between the center of the unit $\overline{C_j}$ and the $\mu'th$ input's to the unit $\overline{\xi_j^\mu}$. The function $g(\cdot)$ weights the distance. It is common to use a Gaussian type of function.

This unit is especially suitable for problems where the input can be clustered around certain centers and where these centers can be used to classify the output.

The different kinds of units and transfer functions can be combined and in that way more advanced neural networks can be built. In [Moody and Darken, 1988; Moody and Darken, 1989] some examples of combining the different neuron types (especially with neurons of the radial basis type used as hidden units) are shown. It is claimed that a better performance is obtained than could be obtained by using only one type of units. Though this is true, we will only use the simple perceptron type of units with tanh or linear transfer functions throughout the work done in this Ph.D. study. No automatic method for choosing the neuron types has been proposed and it is in general very difficult to decide what type of unit to use.

1.3 Two layer feed-forward Neural Networks

As pointed out by [Hornik et al., 1989; Hornik, 1991; Cybenko, 1989] the feed-forward type of neural network can be used as universal function approximators under very general conditions for the activation functions (transfer functions). To have a general function approximator, the neural network only needs to include one hidden layer and an output layer. The conditions on the hidden layer activation functions are that they are continuous, bounded and non-constant. The activation functions on the output layer can be chosen under the same conditions. The only condition for learning a given, bounded input set is then that the hidden layer has sufficient units.

For simplicity only feed-forward neural networks with one-hidden layer, tanh activation functions in the hidden layer and linear output unit will be used. This network architecture is especially easy to optimize as will be shown in section 2.2 and 3.1. In figure 1.5 this network architecture is sketched and some possibilities of different activation function of the hidden units and the output units are given in figure 1.4.

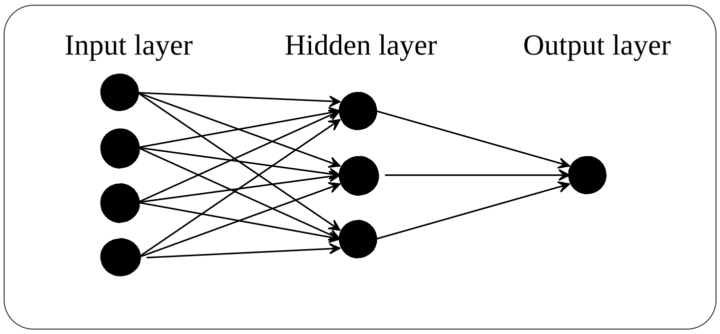


Figure 1.5: Two layer feed-forward neural network.

1.4 Problems discussed in this thesis

The main theme discussed in this thesis is

> **Optimization of neural network architectures for achieving better performance of the networks, as regards generalization ability, learning speed and computational burden.**

To obtain a good generalization performance, it is necessary to build the structure of the data into the neural network structure. The generalization ability expresses the network's performance on an independent data set that has not been used when training/optimizing the network. Further an optimized structure for the network will include fewer weights and therefore give less computational burden, both when optimizing the weights and when used on new data sets.

Optimization of the neural network architecture is tested on different signal processing tasks such as simple classification and time-series prediction.

1.4.1 Optimization of Neural Network architectures

Many different criterion for an optimal neural network architecture can be formulated; some of them are:

- Possible to learn the training set *sufficiently* well.
- No *overfitting* of data set.
- Better generalization.
- Faster adaption to change in the environment, the model, etc.
- Faster learning of neural network weights.
- Less computational burden.
- Hardware restrictions when implementing the network in custom hardware.

Optimization of neural network architectures can be achieved by **pruning** and/or **growth** of neural network architectures. Within the approach of pruning neural networks, the network first trained should have too many resources. This can include too many layers of units, too many units in each layer and/or too many connections between the units. It is standard to start with a fully connected network, which means that all units in one layer ares connected to all the units in the previous layer. For the growth algorithms it is customary to start with a fully connected network with only one hidden layer and only a few units.

Many different algorithms for **pruning** of neural networks have been proposed. The first algorithms proposed were simple weight-decay algorithms, where the cost-function (the function that is minimized while learning the weight values) was changed to give the weights a tendency to decrease towards zero, so the network itself removes non-useful connections. Within this scheme of pruning many different approaches have been proposed. They punish the weights in different ways [Hertz et al., 1991], and are tested at different examples with more or less success. The weight-decay way of pruning does not remove the weights totally from the network, but just fixes some of the parameters so the **effective** number of parameters is less than the real number. Other algorithms remove the weights totally from the network. If all the input or output connections to a unit are removed, they remove the unit from the network. For the pruning algorithms there are a variety of different approaches. The first algorithm proposed was called **magnitude** based pruning. This algorithm simply removes the weights with the smallest magnitudes from the networks, but it was discovered that the performance of the networks was not always due to the *big* weights alone, but also depended on the weights that were removed. For this reason some new algorithms, **Optimal Brain Damage** [Le Cun et al., 1990] and **Optimal Brain Surgeon** [Hassibi et al., 1993], were proposed. These algorithms try to estimate the importance of the weights for the cost-function which is optimized during training of the weights, instead of just using the magnitude of the weights.

Algorithms for network **growth** have also been proposed. The initial network structure is simple, with few layers and only a few units in each layer. Then more and more connections and layers are added. One of the first algorithms proposed is the so-called **upstart algorithm** by [Frean, 1990]. The first network structure has only one hidden layer, and only a few units. Extra units are then added to this layer to separate out the patterns that aren't already learned. Instead of keeping the one layer structure, a hierarchical structure for the network is built. [Fahlman and Lebiere, 1990] have proposed another algorithm, the so-called **cascade-correlation algorithm**, where new units are added and trained on the error that isn't done by the original network. All the methods have problems in getting a low generalization error, because it is necessary to add a lot of units to solve the problems. This is due to the fact that it is very difficult to get the new units to cooperate with the already existing units, so the problem is learned in an optimal way, by the fewest resources. Due to this it is difficult to define a stopping criterion of when to stop adding new resources (units and weights) into the network.

1.4.2 Discussion of the work described in this thesis

It is attempted in this thesis to make a common description of the problems which appear when neural networks are optimized (trained and pruned). The only network structure used is feed-forward neural networks with one hidden layer. The performance of the training and pruning algorithms is tested by examples within the fields of signal processing and simple classification. The major goal is to

achieve the best possible generalization error

by optimizing the network architecture and the algorithms which tunes the network parameters. The generalization error is defined as the error on a data set, which has not been used when tuning the weights.

To reach this goal the following things have to be taken into consideration:

- an effective optimization (learning) method for the parameters (weights) in the neural network has to be found.
- a method for removing superfluous parameters from the network has to be found if tuning of network parameters is started at a network with too many parameters.
- a criterion for selecting the best network architecture among a set of networks.
- a method or rule for tuning the different parameters in the learning algorithm will be of great importance.

The first problem, to optimize the network parameters to get as low error as possible on the training set, is in this thesis solved by using a second order Gauss-Newton based method when optimizing the network parameters (weights). When starting to train the network there are superfluous resources in the network model, and these resources have to be removed by pruning algorithms. The method used is the so-called Optimal Brain Damage method.

It is not enough to know that resources can be removed; it is also essential to have a criterion for stopping the pruning. For classification problems a method which simply uses the classification error on the training set is proposed, while for problems within the field of time-series estimation a method for estimation of the optimal number of parameters in the network is proposed. This method is based on Akaike's Final Prediction Error estimate of the generalization error [Akaike, 1968; Ljung, 1987].

Further, a method for tuning the weight-decay parameters in the learning algorithm is proposed. This method uses the overall goal of minimizing the generalization error. It is based on Akaike's theory and the derivation in [Hansen and Rasmussen, 1994].

1.4.3 Demarcation of the work

None of the algorithms described and tested within this thesis are based on growth algorithms for neural networks. The cascade-correlation method was tested on a few examples in the start of the project, but the results showed that it is difficult to add new units to the network and achieve substantial improvements of the network performance. Output from new units has a tendency to correlate with output from units already in the network.

Further, none of the recursive network structures have been tested even though they have the capability of implementing very long memory of the past, with only few parameters. Especially within signal processing tasks this can be shown to be an important feature, as seen by linear filters (infinite impulse response filters). These techniques are not used because the training methods suggested for these networks have a very low convergence rate. Further, no method for analyzing their generalization capability has yet been proposed. Some recursive networks are though described in an overview for the signal processing task of prediction the future, where the recursion isn't used for units in the hidden layer, but only in the form of using old output from the network as inputs. These networks can be trained as ordinary feed-forward neural networks.

1.5 Structure of the thesis

This thesis attempts to give a concise description of artificial neural networks, especially feed-forward neural networks, within the field of architectural optimization of the networks. Experiments on the architectural optimization algorithms are within the field of signal processing (classification and time-series processing tasks).

In chapter 2 a description of the mathematical notation for the neural network and the learning algorithm is given and in chapter 3 different approaches for architectural optimization algorithms are described. In appendix A the methods for estimating the optimal network structure are derived. In particular, a method for optimization of the weight-decay parameter in the network learning algorithm is derived.

In chapter 4 and 5 examples of the use of the theory of neural networks and especially pruning of the networks, within the fields of classification (cell shape recognition and the contiguity problem) and time-series processing (time-series prediction and model estimation) are given.

Conclusions about the obtained results and the methods are drawn in chapter 6.

Finally the articles that have been written during this Ph.D study are appended in appendix B, C, D, E, F and G.

Chapter 2

Neural Network Modeling

When modeling physical processes, classification problems, etc. there are two philosophies as described in [Weigend and Gershenfeld, 1994]. The idea of **strong modeling** takes the form of expressing the expectation of the connections between input and output in a few equations. An example of this could be physical processes where the basal physical laws could be used. The other modeling approach is called **weak modeling**, where some general equations (function approximators) with no built in knowledge about the system are used to fit data. Of course the strong modeling technique is superior to weak modeling when enough and correct knowledge about the system is present, but this is often not the case. Often the situation is that there is only poor theoretical knowledge about the system, instead it is possible to get a lot of data generated by the system or problem. This has for example been the case where it has been standard to use data acquisition systems to follow and measure parameters in processes (physical or human). In these situations the easiest and often best solution will be to use the acquired data to fit a general model for the problem, and use that model for predicting the future, classifying a pattern, etc.

2.1 Weak modeling

In the weak modeling domain there are many different approaches. From traditional statistics there are numerous modeling techniques available, as an example linear models [Ljung, 1987] or polynomial regression models [Conradsen, 1984a]. Further, new modeling techniques such as neural networks can be used. When building models using these methods, one must choose what size of models should be used, ie. the number of parameters has to be selected. Generally, when more parameters are put into the model, it has more degrees of freedom (more free parameters), and the model can fit more complex data sets. The degree of freedom in the model is also called the **model order**. A major problem when building weak models is to choose the model order so the model fits the input data sufficiently well but does not overfit the data (does not fit the noise in the data). In the next two sections some problems where the correct model order has to chosen are sketched.

2.1.1 Underfitting

Underfitting is the problem that arises when too few resources are incorporated into the model so the model can't fit the data sufficiently well. When the model is trained to learn a data set, the error ceases to decrease at too high a level. A simple example of this is, if data which are generated by a third order polynomial model, are fitted with a second order polynomial model. This second order model will not be able to fit the data well. In figure 2.1 this situation is sketched. The data is generated by a third order polynomial $y = a_0 + a_1 \cdot x + a_2 \cdot x^2 + a_3 \cdot x^3 + v$, where the constants are set to the values mentioned in the figure label, and the noise v is Gaussian distributed noise with zero mean and variance 1. The data is fitted by a second order polynomial model (squared error minimized by second order method in the parameters) and as can be seen, the model is underfitting the data. There are not enough resources in the model to fit the data; the output from the second order model almost looks like a straight line. At the figure it can be seen that the data from the third order model has both an up and down curvature (the inflexion point in the third order model is in the data set). The second order model hasn't the degrees of freedom to model this inflexion point. The solution that minimizes the squared error most for the parameters in the second order model could be seen to be a model which is almost a straight line. It must be due to the symmetry of the data point around the inflexion point of the third order polynomial that has generated the data points.

2.1.2 Overfitting

Overfitting is the opposite of underfitting, when too many resources are put into the model. The problem is then that the model fits the data too well, so the noise in the data is fitted too. This can be seen in figure 2.1 for the 14th order polynomial model. This model is fitting the noisy data so well that it almost passes through all the data points. But if the fitted model is compared to the correct model, it is seen that the data generated by the model is coming from a much too complex model; it is not even possible for the model to interpolate. For data points outside the interval from the learning data, the estimates from the model are totally wrong; the estimates are exploding.

From the figure it can be seen that fitting a third order model (the correct model order) to the noisy learning data, gives a model which both can interpolate and extrapolate "quite" well. From this it is seen that it is important to estimate the correct number of parameters in the model, when it is fitted to the data set. This can of course be done when the model order is known, but often this is not the case. In a later section methods for regularization of the model (putting bounds on the model parameters, by weight-decay) will be discussed. A method for estimating the model order is also discussed in chapter 3.3.

2.1.3 Neural networks

In this thesis the only weak modeling technique which is discussed in detail is the feed-forward neural network modeling technique. The technique can be seen as a part of the broader field of **machine learning**, which covers the general technique of modeling from

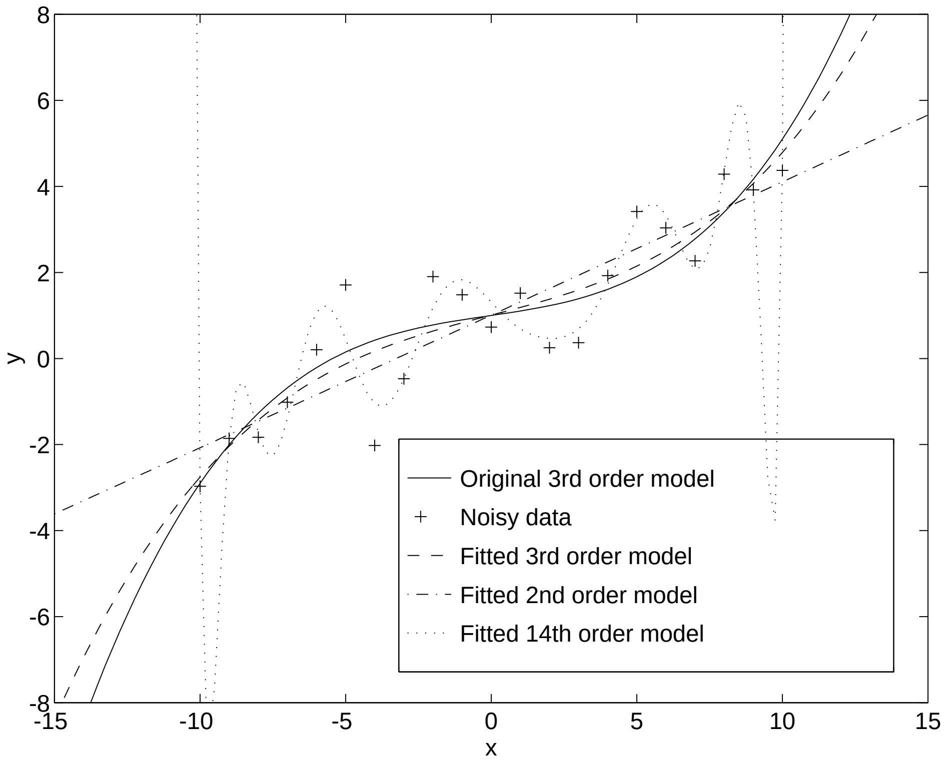


Figure 2.1: Data generated by a third order polynomial model $y = 1.0 + 0.1 \cdot x + 0.001 \cdot x^2 + 0.003 \cdot x^3 + v$, where v is random noise with distribution N(0,1). Data is fitted by a least-square method with a second, a third and a 14th order polynomial model. It is seen that the third order model (dashed line) fits the data generating model very well, while the second order model (dashed-dot line) does not at all have the resources to follow the data generating model (underfitting), and the 14th order model (dotted line) almost fits the noisy data points totally (overfitting). The second and 14th order model show especially bad performance in extrapolation outside the data they have been fitted to.

examples. This holds a lot of methods including genetic algorithms, Boltzmann machines, fuzzy logic, and others.

Compared to conventional statistical modeling, as the example of polynomial regression, the neural network technique has shown its superior quality as pointed out in [Weigend and Gershenfeld, 1994]. Especially the generalization ability of neural networks has been emphasized as its strength. In the polynomial regression method of modeling, the input consists of the original inputs $(x_1, x_2, x_3, \ldots)$, the pairs $(x_1x_2, x_1x_3, \ldots)$, the triplets $(x_1x_2x_3, x_1x_2x_4, \ldots)$, etc. If enough terms are included, any function can be approximated [Conradsen, 1984a]. For both methods the "capacity" of the models is proportional to the number of parameters [Cover, 1965; Mitchison and Durbin, 1989], so this is not where the difference is. The problem with the polynomial regression method is instead the way that the input and combinations of them are brought into the model. If the number of "original" inputs are n and the pairs are also used as inputs to the model, the number of parameters is already increased to $(n + n(n-1)/2)$. This could be an incredibly large number of parameters, and it would for many modeling purposes be necessary to limit the complexity of inputs and choose which combination of inputs should be used to avoid overfitting in the models. No method for choosing between the combinations of inputs is given before data already is fitted by the model, so this could be rather difficult.

For layered neural networks the number of parameters is only increasing linearly with the number of inputs. The network chooses itself which combination of inputs is needed, and the capabilities of the network can be tuned by increasing the number of hidden units. This automatically increases the number of parameters in the network, but only linearly with the number of connections to the hidden units.

The neural networks used throughout this thesis will further be specialized to only incorporate two layers, which means that they only hold one hidden layer of units. They will also be limited to only hold tanh activation functions in the hidden layer, and linear activation functions in the output layer. By limiting the networks to this structure it is especially simple to tune the parameters and analyze the performance of the network, as should be seen in the next chapter. The structure of the networks is then as sketched in figure 2.2. It is shown by [Cybenko, 1989; Hornik et al., 1989] that this network structure can approximate any continuous function, if enough hidden units are incorporated[1]. This type of network can be described by the following equations

$$h_j^\mu = \sum_k w_{jk}\xi_k^\mu, \tag{2.1}$$

$$V_j^\mu = g_h(h_j^\mu) = \tanh\left(\sum_k w_{jk}\xi_k^\mu\right), \tag{2.2}$$

$$h_i^\mu = \sum_j W_{ij}V_j^\mu = \sum_j W_{ij}g_h(h_j^\mu) = \sum_j W_{ij}\tanh\left(\sum_k w_{jk}\xi_k^\mu\right), \tag{2.3}$$

$$O_i^\mu = g_o(h_i^\mu) = \sum_j W_{ij}V_j^\mu = \sum_j W_{ij}\tanh\left(\sum_k w_{jk}\xi_k^\mu\right), \tag{2.4}$$

where

[1]In the articles no method is given for estimating how many hidden units are enough.

μ	-	index of the input-output data pattern.
i, j, k	-	indices of output from the output units, output from the hidden units and input to the network, respectively.
ξ_k^μ	-	input patterns to the network (output from input units).
h_j^μ	-	input pattern to the hidden units.
V_j^μ	-	output patterns from the hidden units.
h_i^μ	-	input pattern to the output units.
O_j^μ	-	output patterns from the output units (output from the network).
$g_h(\cdot)$	-	activation function of hidden units ($\tanh(\cdot)$ function).
$g_o(\cdot)$	-	activation function of output units (linear function).

The notation is taken from [Hertz et al., 1991] and is also used in the articles appended in appendix B, C and D. In the notation for the basic equations for the network units, there is no term describing the threshold values of each unit, as in equation (1.1). The reason is that the threshold can be treated as an ordinary weight in the network. The input for this weight is kept constant at one. In the networks described throughout this thesis an input which always is 1 is incorporated. In the figures this threshold is depicted as a vertical bar through a unit as shown in figure 2.2.

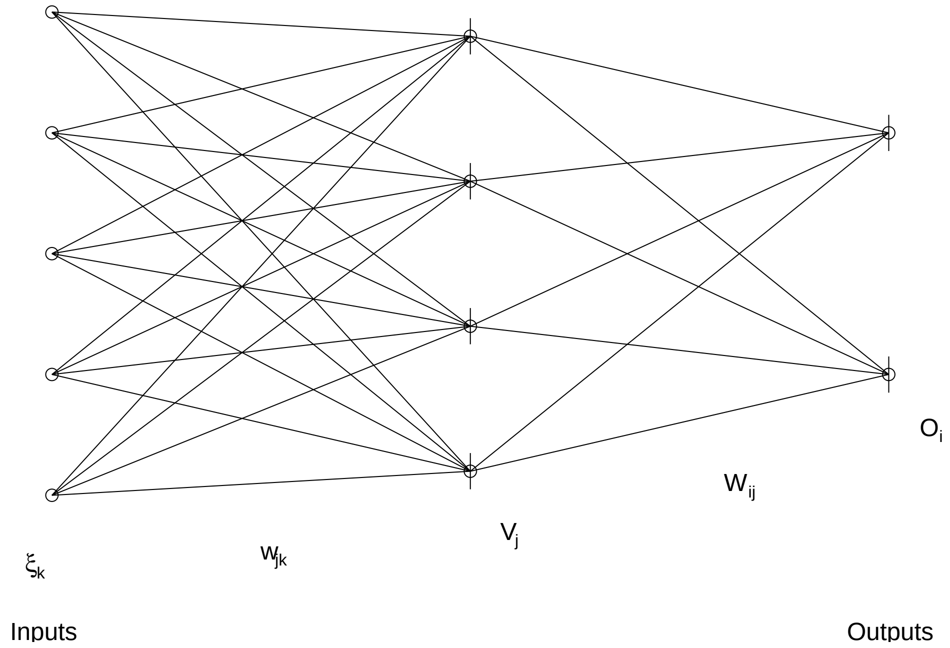


Figure 2.2: Neural network with 5 input units, 4 hidden units and 2 output units. A vertical bar through all the units indicates that the units have an active threshold incorporated.

2.2 Learning Algorithms

In neural network terminology, **learning** is the phase of the modeling process where the neural network is fitted to data. In the literature there are two kinds of learning depending on the form of data that is available in the learning process [Hertz et al., 1991]. The two methods are:

- **Supervised learning** is the learning method where the data set available for the network is a combination of inputs and outputs. The learning process is then to tune the parameters in the model to have the output from the model that best fits the output in the learning set for a given input. This learning technique is also called **learning with a teacher**. This technique is especially well suited to tasks within time-series prediction and modeling, and simple classification tasks. Supervised learning also includes **reinforcement learning**, where the only feedback to the learning process is whether the answer is correct.
- **Unsupervised learning** is the technique for learning patterns where no output is specified for a given set of input data. Instead the learning process should try to find correlation in the inputs, and from this specify output classes by itself. It is especially used within the field of image processing for segmentation or filtering.

In this thesis only supervised learning is dealt with, in the form of learning with a teacher (the learning set is specified in pairs of inputs and outputs). The main purpose of learning is to tune the parameters of the model, so the output from the model for a given input set fits the desired output in the "best way". In section 2.3 some different methods for measuring how well the output from the model fits the desired output are described. These methods are based on **cost-functions**, which measure the distance between the desired output and the output from the model over all the patterns in the training set.

2.3 Cost-Functions

The cost-function gives a functional description of how well the output from the model fits the desired output. The cost-function therefore is a "distance" measure between the desired output and the model output. The learning algorithm tunes the parameters in the model so the distance measure (the cost-function) is minimized. Different ways of measuring the distance have been proposed. The quadratic cost-function and the entropic cost-function are shown in the following equations

$$E_{\text{quadratic}} = \frac{1}{p} \sum_{i\mu} (\zeta_i^\mu - O_i^\mu)^2 \tag{2.5}$$

$$E_{\text{entropic}} = \frac{1}{p} \sum_{i\mu} \left(\frac{1}{2}(1+\zeta_i^\mu) \log \frac{1+\zeta_i^\mu}{1+O_i^\mu} + \frac{1}{2}(1-\zeta_i^\mu) \log \frac{1-\zeta_i^\mu}{1-O_i^\mu} \right) \tag{2.6}$$

where

$E_{\text{quadratic}}$	-	quadratic cost-function.
E_{entropic}	-	entropic cost-function.
ζ_i^μ	-	specified output (goal).
O_i^μ	-	output from neural network.
p	-	number of examples in learning set.
i	-	i'th output (for networks with more than one output).
μ	-	μ'th learning pattern.

The two cost-functions are shown in figure 2.3.

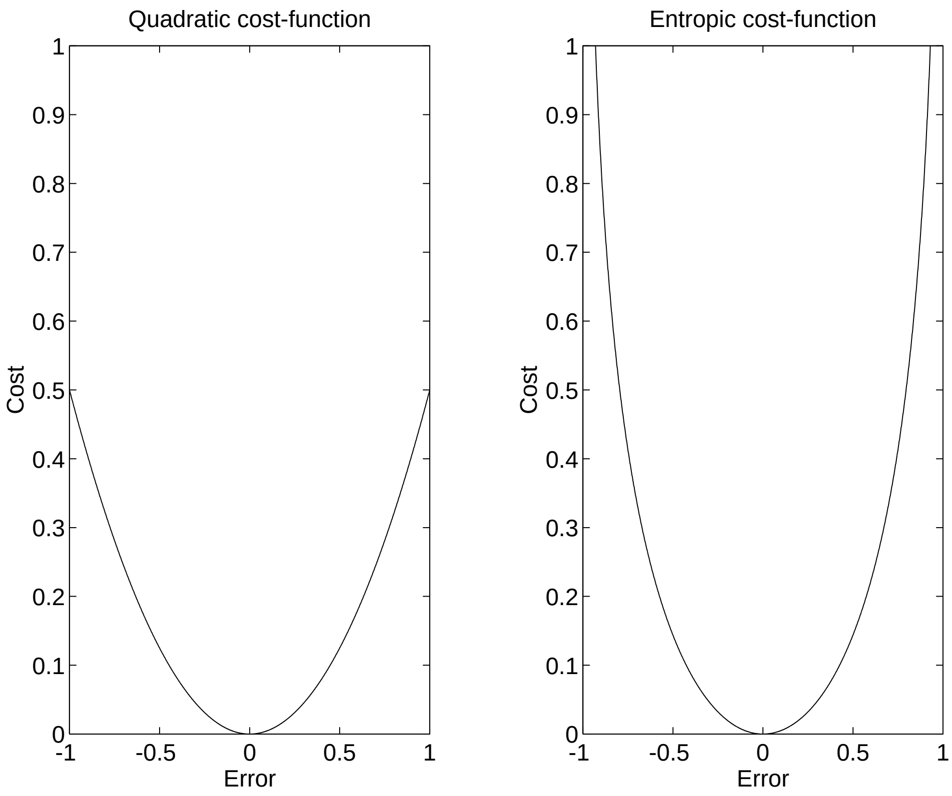


Figure 2.3: The Quadratic and Entropic cost-functions. The cost-functions are defined in equation (2.5) and (2.6). Both of them are plotted for a desired output of 0, and an output from the model varied between -1 and 1. Further there is only one output and one pattern in the learning set ($\mu = 1$). For the Entropic cost-function the cost is going to $+\infty$ in the limit of ± 1.

Both cost-functions are differentiable. This makes them well suited for minimization with gradient based methods as described in section 2.4. In the same section the difference between the two cost-functions will be explained, and it will be discussed when they are appropriate.

2.3.1 Regularization

Before discussing the minimization techniques, **regularization methods** will be described. What is the purpose of regularizing the solution, when minimizing the cost-function? For a problem where the information in the data is insufficient to optimize all the parameters in the model (called **ill-posed** problems), it is necessary to have some constraints on the parameters to transform the problem into a **well-posed** problem, where all the parameters in the model can be determined. An example could be to fit a curve to a couple of noisy points and then constrain the solution by a demand of smoothness e.g. for the examples shown at figure 2.1. The theory of establishing these constraints on parameters is called regularization theory.

In this section a method for forcing the parameters in the direction of zero during training is discussed. If the parameters have exactly the value zero it corresponds to having a model with fewer parameters. This is not the case using the described method, but using the method the parameters are bounded, so the resulting model has fewer free parameters. The method changes the cost-function by including a term which penalizes large parameters.

The standard method adds a term to the cost-function, which penalizes weights by summing up the squares of the weights. How the original cost-function equation (2.5) or (2.6) should be weighted against the regularization is controlled by the parameter α, called the **weight-decay** parameter. The new cost-function can be written as

$$E = E_{\text{train}} + \frac{\alpha}{p} \sum_{l}^{N_u} u_l^2 \tag{2.7}$$

where

E	-	value of the regularized cost-function to be minimized.
E_{train}	-	value of the original cost-function, could be the quadratic or the entropic cost-function.
u_l	-	the l'th parameter in the network.
N_u	-	number of parameters.
α	-	regularization parameter (weight-decay parameter).
p	-	number of patterns in the training set.

This regularization scheme is called **ridge regression** in conventional statistics [Conradsen, 1984a]. Many other schemes for regularizing the solution (removing unnecessary parameters) to the minimization problem have been proposed [Hertz et al., 1991]. The cost-function (2.7) penalizes large weights very much, and [Weigend et al., 1990] has proposed another scheme where large weights compared to a parameter in the algorithm are not given much more punishment than parameters which are just around this parameter. In equation (3.3) this weight-decay scheme is described. In the paper [Weigend et al., 1990] it is claimed that better results could be reached by using this method. In some experiments done in this Ph.D. study it is attempted to demonstrate these qualities for regression problems, but it has been very difficult to tune these extra parameters in the algorithm, to obtain better results. [MacKay, 1992a; MacKay, 1992b] has proposed another method where an individual weight decay has to be set for each parameter. These methods estimate the weight-decay parameters using Bayes statistical methods. In Bayes methods a prior is used to regularize the superfluous parameters. It has in experiments by [Hansen and Rasmussen, 1994] been shown, that many problems arise when this method is used with individual weight-decay tuning for all parameters. It has especially been seen that the method has a tendency to find the null solution, ie. regularize all the parameters to zero, which must be due to the prior used in the Bayes method.

In this thesis where all experiments are done with two layers networks, the regularization scheme used has been slightly modified compared to the simple one, as an individual weighting of the parameters for each of the layers has been used. The cost-function is then described as

$$E = E_{\text{train}} + \frac{\alpha_w}{p} \sum_{jk}^{N_w} w_{jk}^2 + \frac{\alpha_W}{p} \sum_{ij}^{N_W} W_{ij}^2 \tag{2.8}$$

where

w_{jk}	-	parameters from input units to hidden units.
α_w	-	weight of regularization term from input units to hidden units.
N_w	-	number of parameters from input units to hidden units.
W_{ij}	-	parameters from hidden units to output units.
α_W	-	weight of regularization term from hidden units to output units.
N_W	-	number of parameters from hidden units to output units.

The rest of the terms used in the equation are the same as in equation (2.7).

In some of the experiments in prediction of time-series, it is attempted to use an automatic tuning algorithm for the weight-decay parameters. In appendix A Akaike's estimate of the generalization error is derived together with a method for tuning the weight-decay parameters. Experiments are described in the article in appendix E. Some new experiments done at the Mackey-Glass time-series and at the channel equalizer problem are described in section 5.2.3 and 5.3.

As described, the regularizing term forces the parameters in the direction of zero and in that way transforms the problem to a well-posed problem. In the methods for optimization of the parameters described in chapter 2.4, and the methods of pruning the network described in chapter 3.1, where the parameters are removed explicitly instead of just bounding them together, some local approximations are used. If the parameters are close to zero the approximations used will be more precise, and therefore the use of a weight-decay term on the cost-function also helps in getting better estimates in these methods.

2.4 Optimization Techniques

The previous section defines the cost-function, which has to be minimized when the optimal set of parameters is found. There is no exact solution to the minimization of the cost-function when using non-linear functions in the network units, instead iterative methods that optimize the parameters are developed. None of these methods guarantee that the global minimum of the cost-function is reached. The cost-function could have more than one minimum (due to symmetries) and further it can have local minima in which the parameters get stuck during the optimization process.

2.4.1 Back-propagation learning

The basic idea is to calculate a change in each parameter that would decrease the cost-function. As the cost-function is differentiable with regard to each parameter it is possible to calculate the gradient vector. Moving along the negative gradient for each parameter it is possible to find the optimal parameter values, the parameter values which locally minimize the cost-function. However, it is not known how big the update step should be. In figure 2.4 it is shown how each parameter could be updated using the negative gradient of the cost-function in each parameter. This learning strategy is called **gradient descent** learning.

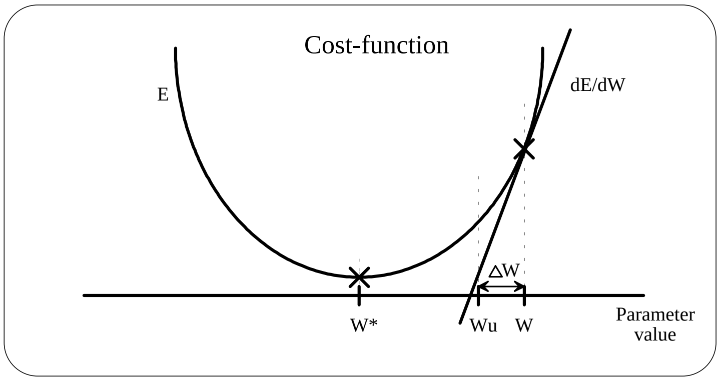


Figure 2.4: Update of each parameter in a neural network using the gradients. W^* is the parameter solution that is searched for. W is the actual parameter value, ΔW is the update step for the parameter and, W_u is the updated parameter value.

For a two layer feed-forward neural network with linear output units and tanh activation functions on hidden units and weight-decay as described in (2.8), update of the parameters using the gradient descent learning strategy can be written as shown in (2.9) for the hidden to output layer weights, and as shown in (2.10) for the input to hidden layer.

$$\Delta W_{ij} = -\eta \frac{\partial E}{\partial W_{ij}} = -\eta \left(\frac{\partial E_{\text{train}}}{\partial W_{ij}} + \frac{2}{p} \alpha_W W_{ij} \right) \tag{2.9}$$

$$\Delta w_{jk} = -\eta \frac{\partial E}{\partial w_{jk}} = -\eta \left(\frac{\partial E_{\text{train}}}{\partial w_{jk}} + \frac{2}{p} \alpha_w w_{jk} \right) \tag{2.10}$$

where η is the parameter which determines the update step size. This parameter is called the **gradient parameter**.

If the **quadratic** learning cost-function (2.5) is used the derivative of E_{train} can be written as

$$\begin{aligned}\frac{\partial E_{\text{train}}}{\partial W_{ij}} &= -\frac{2}{p}\left(\sum_{\mu}(\zeta_i^{\mu} - O_i^{\mu})\frac{\partial O_i^{\mu}}{\partial W_{ij}}\right) \\ &= -\frac{2}{p}\left(\sum_{\mu}(\zeta_i^{\mu} - O_i^{\mu})V_j^{\mu}\right) \qquad (2.11)\end{aligned}$$

and

$$\begin{aligned}\frac{\partial E_{\text{train}}}{\partial w_{jk}} &= -\frac{2}{p}\left(\sum_{i\mu}(\zeta_i^{\mu} - O_i^{\mu})\frac{\partial O_i^{\mu}}{\partial w_{jk}}\right) \\ &= -\frac{2}{p}\left(\sum_{i\mu}(\zeta_i^{\mu} - O_i^{\mu})W_{ij}\frac{\partial V_j^{\mu}}{\partial h_j^{\mu}}\xi_k^{\mu}\right). \qquad (2.12)\end{aligned}$$

These equations can then be rewritten as

$$\Delta W_{ij} = \eta\frac{2}{p}\left(\sum_{\mu}\delta_i^{\mu}V_j^{\mu} + \alpha_W W_{ij}\right), \qquad (2.13)$$

with

$$\delta_i^{\mu} = \zeta_i^{\mu} - O_i^{\mu} \qquad (2.14)$$

and

$$\Delta w_{jk} = \eta\frac{2}{p}\left(\sum_{\mu}\delta_j^{\mu}\xi_k^{\mu} + \alpha_w w_{jk}\right), \qquad (2.15)$$

with

$$\delta_j^{\mu} = \frac{\partial V_j^{\mu}}{\partial h_j^{\mu}}\sum_i W_{ij}\delta_i^{\mu} \qquad (2.16)$$

where V_j^{μ} can be calculated as shown in (2.2). $\partial V_j^{\mu}/\partial w_{jk}$ can therefore be found as

$$\frac{\partial V_j^\mu}{\partial h_j^\mu} = \frac{\tanh(h_j^\mu)}{\partial h_j^\mu} = \left(1 - \tanh^2\left(\sum_k w_{jk}\xi_k^\mu\right)\right) \qquad (2.17)$$

This scheme for the gradient-descent algorithm is called the **back-propagation** method [Rumelhart et al., 1986a; Rumelhart et al., 1986b] because the update of weights can be written as a multiplication of the input signals propagated forward through the preceding layers (ξ_k^μ and V_j^μ), and a term of the output error signals propagated backwards through the weights of the succeeding layers (δ_i^μ and δ_j^μ) multiplied by the input signals propagated forward through the differentiated activation functions of the neurons.

The method can be implemented in two different ways. It can be implemented as written directly in (2.13) and (2.15) where the updates of the parameters are summed over all the learning patterns, before the parameters are changed. This is called **batch-updating**. According to a short notice in [Hertz et al., 1991] it seems to give faster convergence, fewer problems with local minimum, just to calculate the update of the parameters based on one of the learning patterns, update the parameters, and then calculate the update of the parameters based on the next pattern. This is called **online-updating**. The problem with batch-updating is that the algorithm easily gets stuck in a local minimum or a "flat valley" in the cost-function space. The online-updating method seems to have the possibility to solve these problems, especially when the order in which the learning patterns are used is random for each epoch[2].

The problem with the back-propagation method is, as written before, that it is not known what the size of each update step should be, the size of η is not known. If the parameter is too small, it will take too many steps before the minimum is reached; if it is too big, there is a risk that the algorithm is leaving the minimum (valley) that the gradient is calculated for. It is therefore necessary to find a method so the parameter η can be calculated. A method for this is described in section 2.4.2.

If the **entropic** cost-function (2.6) is used instead of the quadratic, the back-propagation learning algorithm can be derived in the same way. For the output layer units a tanh activation function is used. It is described by

$$O_i^\mu = g_o(h_i^\mu) = \tanh\left(\sum_j W_{ij}V_j^\mu\right) = \tanh\left(\sum_j W_{ij}\tanh\left(\sum_k w_{jk}\xi_k^\mu\right)\right). \qquad (2.18)$$

Using the entropic cost-function, it can be shown that the **back-propagation** updates of the parameters takes a simple form, where they can be described by the same equations (2.13) through (2.16).

If the quadratic learning cost-function is used instead in combination with a tanh output unit equation (2.14) is changed to

[2] An epoch is defined as one pass through all the patterns in the learning set.

$$
\begin{aligned}
\delta_i^\mu &= (\zeta_i^\mu - O_i^\mu)\frac{\partial \tanh(h_i^\mu)}{\partial h_i^\mu} \\
&= (\zeta_i^\mu - O_i^\mu)(1 - \tanh^2(h_i^\mu))
\end{aligned}
\tag{2.19}
$$

It has been shown in some experiments done in classification tasks during this Ph.D. study, where the performance of the learning algorithm using the entropic and the quadratic algorithm is compared, that if a tanh activation function is used in the output layer, it is a very good idea to use the entropic learning cost-function to accelerate the learning speed. This is also mentioned in a notice in [Hertz et al., 1991]. The problem is that if the quadratic cost-function and a tanh output unit is used, the cost-function becomes almost "flat", when the input to the output units h_i^μ is large compared to ± 1. In equation (2.19) it can be seen that the term $(1 - \tanh^2(h_i^\mu))$ and therefore δ_i^μ will approach zero. The equations for back-propagation learning then show that the updates of the parameters approaches zero. This means that a large error is not punished very much by this cost-function and the convergence rate for the learning algorithm will therefore be low.

2.4.2 Gauss-Newton learning algorithms

The problem of finding an optimal update step to reach the minimum for the cost-function, can be solved by using a second order method e.g. a Gauss-Newton based method [Ljung, 1987; Hertz et al., 1991; Buntime and Weigend, 1993]. This method approximates the cost-function by a second order Taylor series in the parameters. The cost-function can then be written as

$$
\begin{aligned}
E &= E_0 + \left(\frac{\partial E}{\partial \mathbf{u}}\right)^T \delta\mathbf{u} + \frac{1}{2}\delta\mathbf{u}^T \frac{\partial^2 E}{\partial \mathbf{u}^2}\delta\mathbf{u} + o(\|\delta\mathbf{u}\|^3) \\
&\approx E_0 + \left(\frac{\partial E}{\partial \mathbf{u}}\right)^T \delta\mathbf{u} + \frac{1}{2}\delta\mathbf{u}^T \mathbf{H}\delta\mathbf{u}
\end{aligned}
\tag{2.20}
$$

where

E	-	Cost-function.
E_0	-	Value of cost-function in the point, where the Taylor approximation is done.
$\mathbf{u}$	-	Parameter vector with all parameters from input to hidden and hidden to output layer (all the parameters w_{jk} and W_{ij}).
$\delta\mathbf{u}$	-	Variation of parameters $\mathbf{u}$.
$\mathbf{H}$	-	Hessian matrix (2. order derivative of cost-function with respect to all the parameters).

For simplifying reasons the vector/matrix notation for the parameters and the derivatives of the cost-function is used. The format of the terms is

$$\mathbf{u} = [u_1, u_2, u_3, \ldots, u_n]^T, \tag{2.21}$$

$$\frac{\partial E}{\partial \mathbf{u}} = [\frac{\partial E}{\partial u_1}, \frac{\partial E}{\partial u_2}, \frac{\partial E}{\partial u_3}, \ldots, \frac{\partial E}{\partial u_n}]^T, \tag{2.22}$$

$$\mathbf{H} = \frac{\partial^2 E}{\partial \mathbf{u}^2} = \begin{bmatrix} \frac{\partial^2 E}{\partial u_1^2} & \frac{\partial^2 E}{\partial u_1 \partial u_2} & \frac{\partial^2 E}{\partial u_1 \partial u_3} & \cdots & \frac{\partial^2 E}{\partial u_1 \partial u_n} \\ \frac{\partial^2 E}{\partial u_2 \partial u_1} & \frac{\partial^2 E}{\partial u_2^2} & \frac{\partial^2 E}{\partial u_2 \partial u_3} & \cdots & \frac{\partial^2 E}{\partial u_2 \partial u_n} \\ \frac{\partial^2 E}{\partial u_3 \partial u_1} & \frac{\partial^2 E}{\partial u_3 \partial u_2} & \frac{\partial^2 E}{\partial u_3^2} & \cdots & \frac{\partial^2 E}{\partial u_3 \partial u_n} \\ \vdots & \vdots & \vdots & & \vdots \\ \frac{\partial^2 E}{\partial u_n \partial u_1} & \frac{\partial^2 E}{\partial u_n \partial u_2} & \frac{\partial^2 E}{\partial u_n \partial u_3} & \cdots & \frac{\partial^2 E}{\partial u_n^2} \end{bmatrix}, \tag{2.23}$$

where n is the number of parameters in the neural network.

Within the second order approximation to the cost-function, the minimum can be found by differentiating the Taylor series with respect to the parameter vector. Setting the derivative to zero, the optimal parameter vector $\mathbf{u}^*$ can be found as

$$\begin{aligned} \delta \mathbf{u} &= \mathbf{u}^* - \mathbf{u} = -\mathbf{H}^{-1} \frac{\partial E}{\partial \mathbf{u}} \\ &\Updownarrow \\ & \mathbf{u}^* = \mathbf{u} - \mathbf{H}^{-1} \frac{\partial E}{\partial \mathbf{u}}. \end{aligned} \tag{2.24}$$

The problem is to find the inverse of the Hessian matrix $\mathbf{H}$. The matrix has the dimension $(n \times n)$. When the Hessian matrix is estimated it should be inverted, which is very time consuming for big matrices (the number of calculations is increasing with n^3). Instead there are a couple of possibilities. An iterative method for estimating the inverse Hessian directly can be implemented as described in section 3.1.5 [Ljung, 1987; Hassibi et al., 1993; Buntime and Weigend, 1993].

Another possibility is to further approximate the cost-function. If only the diagonal terms of the Hessian matrix are used it is easy to invert the matrix. Then it is only necessary to invert each element in the diagonal. This method is called the **pseudo Newton** method [Hertz et al., 1991; Svarer et al., 1993a; Gorodkin et al., 1993]. Updates of the parameters can then be written as

$$\begin{aligned} u_i^* &= u_i - \frac{\partial E}{\partial u_i} \Big/ \frac{\partial^2 E}{\partial u_i^2} \\ &= u_i - \left(\frac{\partial E_{train}}{\partial u_i} + \frac{2\alpha_{u_i}}{p} u_i \right) \Big/ \left(\frac{\partial^2 E_{\text{train}}}{\partial u_i^2} + \frac{2\alpha_{u_i}}{p} \right) \end{aligned} \tag{2.25}$$

where u_i is the update of the i'th weight in the neural network. The second derivative of the cost-function can using the quadratic cost-function be found as

$$\begin{aligned}\frac{\partial^2 E_{\text{train}}}{\partial W_{ij}^2} &= \frac{2}{p}\sum_{\mu}\left((V_j^\mu)^2 - (\zeta_i^\mu - O_i^\mu)\frac{\partial V_j^\mu}{\partial W_{ij}}\right) \\ &= \frac{2}{p}\sum_{\mu}(V_j^\mu)^2 \end{aligned} \tag{2.26}$$

$$\begin{aligned}\frac{\partial^2 E_{\text{train}}}{\partial w_{jk}^2} &= \frac{2}{p}\sum_{i\mu}\left(\left(W_{ij}\frac{\partial V_j^\mu}{\partial h_j^\mu}\xi_k^\mu\right)^2 - (\zeta_i^\mu - O_i^\mu)W_{ij}\frac{\partial^2 V_j^\mu}{\partial (h_j^\mu)^2}(\xi_k^\mu)^2\right) \\ &\approx \frac{2}{p}\sum_{i\mu}\left(W_{ij}\frac{\partial V_j^\mu}{\partial h_j^\mu}\xi_k^\mu\right)^2 \end{aligned} \tag{2.27}$$

This approximation for calculating the second derivative of E_{train} with respect to w_{jk} is known as the Levenberg-Marquart approximation [Le Cun et al., 1990]. It assumes that the error $(\zeta_i^\mu - O_i^\mu)$ is uncorrelated with the input patterns ξ_k^μ. This is in most cases a very good approximation as shown in numerical examples in appendix D. The approximation has the property of giving guaranteed positive estimates of second derivatives of the cost-function, which is very convenient when using (2.25) to update the parameters, it insures updates in the correct direction. This can be seen by

$$\delta\mathbf{u} = -\mathbf{A}\frac{\partial E}{\partial \mathbf{u}}, \tag{2.28}$$

and

$$\begin{aligned}\delta E &= \frac{\partial E}{\partial \mathbf{u}}\delta\mathbf{u} + o(\delta u^2) \\ &\approx -\frac{\partial E}{\partial \mathbf{u}}\mathbf{A}\frac{\partial E}{\partial \mathbf{u}} < 0. \end{aligned} \tag{2.29}$$

Equation (2.29) is true if $\mathbf{A}$ is a positive definite matrix. When using the Levenberg-Marquart approximation for the diagonal terms of the Hessian matrix this is fulfilled. Therefore the update steps $\delta\mathbf{u}$ of the parameter vector will always give a decrease of the cost-function, $\delta E < 0$.

2.4.3 Optimization of parameters in linear models

If the neural network model has linear output units the method described in the previous section is an exact Newton method, and the optimal parameters in the output layer are found. As proposed by [Barton, 1991] this could more easily be written using only the first derivative from equation (2.11) with the weight-decay term added. Setting this to zero yields

$$
\begin{aligned}
\frac{2}{p}\left(-\sum_\mu(\zeta_i^\mu - O_i^\mu)V_j^\mu + \alpha_W W_{ij}\right) &= 0 \\
\Updownarrow \qquad & \\
-\sum_\mu\left(\zeta_i^\mu - \sum_{j'} W_{ij'}V_{j'}^\mu\right)V_j^\mu + \alpha_W W_{ij} &= 0 \\
\Updownarrow \qquad & \\
\sum_{j'}\left(W_{ij'}\sum_\mu V_{j'}^\mu V_j^\mu\right) + \alpha_W W_{ij} &= \sum_\mu \zeta_i^\mu V_j^\mu,
\end{aligned}
\tag{2.30}
$$

which can be solved for the output parameters W_{ij}. Using a matrix notation it can be written as

$$
\mathbf{AC} = \mathbf{Y} \tag{2.31}
$$

where

$$
Y_{ij} = \sum_\mu \zeta_i^\mu V_j^\mu \tag{2.32}
$$

$$
A_{jj'} = \begin{cases} \sum_\mu V_j^\mu V_{j'}^\mu & j \neq j' \\ \sum_\mu V_j^\mu V_{j'}^\mu + \alpha_W & j = j' \end{cases} \tag{2.33}
$$

$$
C_{ij} = W_{ij}. \tag{2.34}
$$

The optimal weights for the output layer of the neural network are then easily calculated as

$$
\mathbf{C} = \mathbf{A}^{-1}\mathbf{Y}, \tag{2.35}
$$

The weight-decay term in the cost-function, which in equation (2.33) gives the α_W term for $j = j'$, guarantees that the A-matrix is positive definite and therefore invertible. Therefore it is always possible to find a solution for the parameters, using equation (2.35).

2.5 Optimization in two layer Neural Networks

For a two layer feed-forward neural network with linear output units optimization of the parameters can, using the second order methods discussed in the previous sections, be done in a iterative way. The parameters in the input to hidden layer are first updated using (2.25) in batch mode, with hidden to output parameters kept constant. Then the input to hidden parameters are kept constant and the optimal values of the hidden to output parameters are calculated using (2.35).

By using this scheme iteratively it is possible to get a better and better solution in the weight space. Although the output parameters are calculated exact each time, the calculation of the input to hidden weights is based on a second order diagonal approximation, which in each iteration gives a better and better solution, and thus decreases the cost-function.

The simulator used throughout the experiments done in this thesis and the papers in the appendices uses the principles described in this chapter. The simulator is written as a C-program with some facilities to load network setup and dump results from/to MATLAB[3] data files. This makes it easy to change the network setup and to analyze and report the results from experiments.

2.6 Preprocessing of data

The learning speed, which means the convergence rate of parameters to the parameter set with the lowest cost-function, depends on the learning algorithm, as well as on the quality of the learning data. Of course the quality of the learning data depends on many factors such as:

- Noise on data sets.
- Errors in the data set.
- Offset and variance on data inputs.
- Correlation in data sets.

The two first factors, noise and errors, must be taken care of in an individual way for each data set. It could be that it is possible to remove some of the noise by filtering the data set by a high or low pass filter before using them in the neural network, but it depends on the problem and the noise. Errors in the data set are also necessary to handle individually depending on the data set. For some problems it is possible to remove the data sets which are wrong, but it requires that it is possible to locate them in the data set. For other problems it isn't possible to locate the wrong examples in the data set, and it would therefore be necessary to find network architectures and learning methods, which are robust for these examples. This could e.g. be different methods for rejecting the result by estimating a probability for a correct network result.

In the next sections some methods which preprocess the data set before using them in the neural networks algorithms are discussed. These methods are especially designed to take care of correlation in the inputs. In figure 2.5 the way the preprocessor works is sketched.

[3] MATLAB is a numerical tool which makes it possible to analyze matrices and plot results easily.

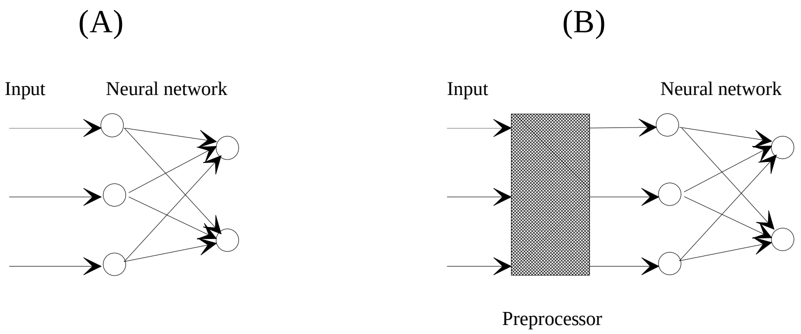


Figure 2.5: Preprocessing of data sets. In figure (A) the input is used directly in the neural network, while (B) shows the way a preprocessor is used on the data before they are used by the neural network.

In the next two subsections, some methods will be discussed which take care of the problem of learning from data sets with offset and variance on the data, and the problem with a lot of correlation in the data set.

2.6.1 Normalization input

The problem, when the data used to train the network isn't normalized (zero mean value and non-uniform variance), is that the convergence for the learning algorithm is slow. In the article [Le Cun et al., 1991] it has been shown that a mean value different from zero or a non-uniform variance in the inputs, influence the condition number of the Hessian matrix. Further it has been shown that the convergence rate depends on the condition number. The condition number κ for the Hessian matrix is defined as

$$\kappa = \sigma_{max}/\sigma_{min}, \tag{2.36}$$

where σ_{max} and σ_{min} are respectively the largest and smallest eigenvalue of the Hessian matrix.

For a simple first order algorithm (back-prop/gradient descent algorithm) it has been shown, e.g. [Le Cun et al., 1991] that the optimal learning parameter η depends on the eigenvalue σ in the Hessian, which corresponds to that weight. When only one learning parameter for all the weights is used, it is necessary to tune the learning parameter to the largest eigenvalue, which causes the convergence of weights with smaller eigenvalues to be slow. If the condition number is large the convergence rate for the entire model (all the weights) will therefore be slow.

In the second order Gauss-Newton algorithm proposed in the previous section, it is attempted to compensate for this phenomena by applying the second order term. This method uses the Hessian matrix to compensate for different update steps by using the inverse Hessian to calculate the size of the update steps for each parameter. A problem is that the calculation of the Hessian is not exact. Some approximations are used in the calculations, e.g. the Levenberg-Marquart approximation, which makes sure that the estimate of the Hessian is positive definite and the inverse Hessian exists; all the eigenvalues of the Hessian are positive. Therefore the second order algorithm, which is used to optimize the learning problem, can't take care of all the problems with slow convergence, which arise when the dispersion in the eigenvalues is large. It would for these reasons always be a good idea to normalize the inputs by removing the mean value in each input, and normalize to uniform variance.

In the article [Le Cun et al., 1991] it has been shown for a linear model with a quadratic cost-function, that a bias on the inputs results in a scaling of one of the eigenvalues with the bias term. A similar spread of the spectrum of the eigenvalues is present if the inputs have non-uniform variance, according to the article. These problems can of course be solved by subtracting the mean value and rescaling the inputs. For the quadratic optimization problem, it can also be solved by using a second order Newton method, while no approximations are needed; but as mentioned before some approximations are going into the second order methods if used for the non-linear neural network. Therefore it would be a better idea to normalize the input before using them.

Another conclusion which could be derived from this argumentation is why it is better to use a tanh activation function than a sigmoid activation function on the hidden units. For the tanh activation function it is possible to have a zero mean output value at the hidden units, while the sigmoid activation function always will have a mean value larger than zero. The output from the hidden units are going into optimization of the output weights as inputs to these weights. It is therefore important for this optimization to have zero mean value. However, there is no method that ensures zero mean output from the hidden units. Using the methods sketched in 2.4.3, where the problem is a quadratic optimization problem (quadratic cost-function and linear output units) it doesn't mean anything if the mean of the output from the hidden units is different from zero. The optimization of the output weights will for this case be exact.

2.6.2 Principal Components

Another method which can help in speeding up the learning process is the method of using principal components, e.g. [Conradsen, 1984b], to describe the learning set. The method uses a linear transformation, based on the eigenvalues and eigenvectors of the correlation matrix for the input patterns. The eigenvalues are sorted in descending order and a matrix consisting of the corresponding eigenvectors is formed. The matrix consisting of all the input patterns is then multiplied by the eigenvectors matrix, and the resulting matrix is then the transformed input matrix. The elements in these transformed inputs are then the principal components of each pattern. The part ρ of the total variance that can be explained by each principal component σ_k can be found as

$$\rho = \frac{\sigma_k}{\sigma_1 + \sigma_2 + \cdots + \sigma_m} \tag{2.37}$$

where m is the number of inputs (the number of eigenvalues/principal components).

While the eigenvalues are sorted in descending order it can from this equation be seen that the first principal components describe most of the variance of the input patterns.

If there are some very small eigenvalues the corresponding principal components describe almost no variance in the inputs. This means that if there is a large eigenvalue dispersion it is perhaps possible to describe the information in the data with fewer inputs, by using fewer principal components. Further, while the parameters corresponding to the small eigenvalues describe only little of the variance in the inputs, the parameters corresponding to these inputs don't mean anything for the performance of the neural network. If these parameters aren't trained to an optimal value it would mean little for the overall performance of the network. As described in the previous section about normalization of the inputs it would not be crucial for the learning algorithm, because most of the information in the inputs is described in the input patterns with the largest eigenvalues, which are learned fastest. It should though be emphasized that even though the small eigenvalues only account for a small part of the variance it can be the crucial part of the input signal, the part that should be used when the output should be learned. The preceding remarks are therefore likely to hold, but no more.

There are also some disadvantages in using the principal component preprocessing algorithm, some of which are

- It is impossible to remove "original" inputs by pruning the network, because these inputs are mixed up in the principal components, which are used as inputs to the neural network.
- It is difficult to interpret the structure of the resulting network.

These arguments are all going in the lost possibility of interpreting the resulting neural network. Especially the disadvantage that it isn't possible to remove any inputs from the network by pruning, thus eliminating the possibility of reducing the original input-space, must be emphasized.

2.6.3 Gram-Schmidt orthogonalization

Other preprocessing methods can be proposed. In [Orfanidis, 1988; Marshall et al., 1989; Orfanidis, 1990] the Gram-Schmidt preprocessing method which transforms the input patterns $\mathbf{x} = [x_0, x_1, \ldots, x_k]$ to the neural network to an orthogonal basis is described. The method uses some of the same principles as the principal component method described in the previous section, it calculates a new set of inputs $\mathbf{z} = [z_0, z_1, \ldots, z_k]$ with mutually uncorrelated components. Instead of the direct calculations done using principal components the coefficients of the preprocessor is not found explicitly, but is calculated in an iterative way by minimizing the correlation between the next input and the previous orthogonalized inputs, a way that look like gradient descent learning of neural network parameters. First an input x_0 is chosen and used directly as input z_0 to the neural network, as sketched in figure 2.6. Then the next input x_1 is used and the correlation between this input transformed to z_1 by the parameter b_{10} (as shown in the figure) is minimized by changing b_{10}. This can e.g. be done by a gradient-descent or Newton algorithm. Then the next coefficients b_{20} and b_{21} are calculated by minimizing the correlation between the third transformed input z_2 and the previously orthogonalized inputs z_0 and z_1. This method is used until all the inputs in $\mathbf{x}$ are transformed to a set of orthogonalized inputs in $\mathbf{z}$

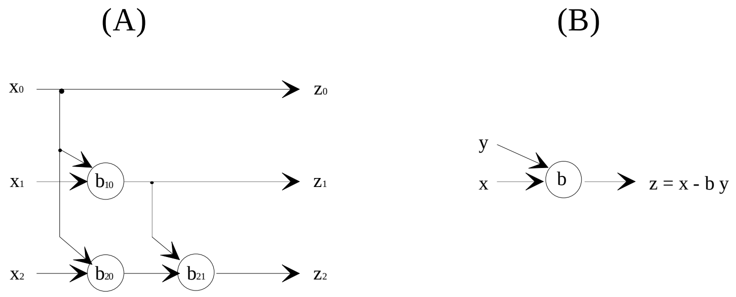


Figure 2.6: Gram-Schmidt orthogonalization of the inputs **x** with the coefficients b_{ij}, which results in a set of orthogonalized inputs **z** is shown in (A). In (B) a simple calculation element in the Gram-Schmidt algorithm is shown.

These orthogonalized inputs are then used as inputs to the network. The advantage of using this method is that the correlation between each input is minimized. All the inputs have new information which makes it possible to learn the parameters connecting this input with the neural network.

The disadvantages of the method are as mentioned for the principal component preprocessing that all the inputs are mixed up. This makes it impossible to eliminate some of the inputs if the connections to these inputs are removed. Another disadvantage of this method is that a new set of parameters has to be calculated by an iteratively algorithm, whenever new learning patterns are added.

Chapter 3

Neural Network Architecture Optimization

Optimizing a neural network to solve a problem involves in addition to optimization of the parameters, optimization of the network architecture. To obtain the optimal performance from a neural network, knowledge about the problem should be built into the network architecture before starting to learn the parameters. As an example, knowledge about receptive fields in the input space would be very helpful to build into the network, to limit the number of free parameters which should be optimized [Levin et al., 1990].

Unfortunately, it is often difficult to formulate knowledge or expectation about the network architecture. Discovery of the importance of network optimization has therefore caused big effort to be put into finding methods for designing the architecture of neural networks from data. It is believed that architecture optimization helps in

- improving the generalization performance of the network by reducing the number of examples needed to learn a problem,
- allowing faster adaption to changes in the environment,

and it is a fact that removing superfluous connections reduces the computational burden.

Different methods for finding the optimized network structures have been proposed. They can be divided into two main groups:

- Network pruning algorithms.
- Network growth algorithms.

It is believed that a combination of pruning and growth methods will be very helpful, but today no effective methods for combining them have been proposed. Of course the proposed methods for pruning and growth of the networks can be used iteratively, but many of especially the growth methods result in networks with strange architectures, which can be very difficult to prune. An example is the structure built by the cascade-correlation algorithm [Fahlman and Lebiere, 1990], which builds a network with many layers and connections between these layers. It will be difficult to use the pruning algorithms discussed in this chapter for pruning these irregular multi-layer network structures.

In this chapter the network pruning algorithms are described in the first section, while the next section gives an overview of some of the growth algorithms.

3.1 Pruning Algorithms

Pruning neural networks assumes that a neural network with superfluous resources (parameters) already has been trained. In this thesis only the straight feed-forward neural network type is used. Therefore pruning is the technique of removing connections between units in one layer and the next layer, which are superfluous for solving the problem. Whole units can also be removed; if all the connections (inputs or output) to/from a unit are removed, the unit itself is removed from the network.

Different methods have been proposed for removing weights from the network. Some of them will be discussed in the next subsections.

3.1.1 Weight-decay pruning

Weight-decay is a way of giving the parameters (weights) a tendency to decay to zero, unless they are reinforced by minimization of the rest of the cost-function. The cost-function with a weight-decay term can be written as

$$E = E_{\text{train}} + E_{\text{weight-decay}}, \tag{3.1}$$

where

E	-	Cost-function.
E_{train}	-	Cost-function, as defined in (2.5), (2.6) or other.
$E_{\text{weight-decay}}$	-	Weight-decay term of cost-function.

Different principles for defining the weight-decay term have been proposed. In the previous chapter the simplest weight-decay term is incorporated in the calculation of the learning algorithms for the weights in the hidden layer and the linear output layer. This kind of weight-decay is penalizing large parameters by a quadratic term. Weights are given a tendency to decay to zero. It can in the simplest form be written as

$$E_{\text{weight-decay}} = \frac{\alpha}{p} \sum_{l} u_l^2, \tag{3.2}$$

where

α - Parameter that adjust the level of weight-decay compared to the rest of the cost-function.
u_l - The l'th parameter in the network.
l - Parameter index.
p - Number of examples in learning set.

This kind of weight-decay can of course be more advanced by making it possible to adjust the weight-decay individually for each parameter, by adding an individual weight-decay parameter α_l for each parameter in the network. In this thesis the weight-decays described in equation (2.8) are used, with individual weight-decay for parameters in the non-linear input to hidden layer and for parameters in the linear hidden to output layer.

Other and more advanced weight-decay algorithms have been proposed [Hertz et al., 1991]. One such example is

$$E_{\text{weight-decay}} = \frac{\alpha}{p} \sum_i \frac{u_i^2}{u_0^2 + u_i^2}, \tag{3.3}$$

where u_0 is an additional parameter that can be used to tune the maximal size of the penalized weights. The α parameter is used in the same way as in equation (3.2). This way of removing parameters does not punish parameters which are "very large" compared to u_0. Therefore this kind of weight-decay will prefer a network with fewer large parameters, while the simple weight-decay given in equation (3.2) prefers network with more smaller weights. No method for choosing the parameter u_0 has been proposed. In [Weigend et al., 1990] it is mentioned that better results could be obtained on signal processing tasks e.g. prediction of time-series using the weight-decay term in (3.3). Experiments done in this Ph.D. project have shown that it is very difficult to adjust the extra parameter u_0 in the algorithm, to obtain any advantages from this type of weight-decay. The simple algorithm with as few parameters as possible has therefore been chosen throughout the experiments in this thesis.

These weight-decay techniques just force the parameters towards zero by adding a term to the cost-function that penalizes large weights. The weights are therefore not explicitly removed from the network architecture but just bounded. This means that the network contains less free parameters, but the parameters are still present in the network architecture. In most literature these techniques are therefore not considered to be pruning techniques. In this work the weight-decay technique is not used as pruning technique; it is used to help the optimization techniques by putting some bounds on the parameters, so it is possible to solve the optimization problem.

3.1.2 Magnitude based pruning

In this pruning scheme the weights are pruned by their magnitudes. This method uses the assumption that there is a correlation between the magnitudes of the weights and their influence on the cost-function, so removing small weights only increases the cost-function a little, while removing large weights increases the cost-function more. This assumption is only correct in some situations. There are other factors which influence the increase in the cost-function when removing weights. Some of them are:

- The amplitude of the inputs into a unit.
- For the parameters in a hidden layer, the parameters in the output layer will influence the change in the cost-function, when removing the parameters in the hidden layer.

Using magnitude based pruning these factors are not taken into consideration. This pruning method will therefore in many situations give unsatisfying results. In the next sections two methods which incorporate the other factors, which influence the increase in the cost-function when pruning weights, will be described.

3.1.3 Optimal Brain Damage method

The so-called **Optimal Brain Damage** method proposed in [Le Cun et al., 1990] was originally used to optimize network structures within the field of digit recognition. The network built and optimized originally had 10,000 weights, and incorporated 4 layers of neurons. It was trained on a database with 7,291 handwritten digits and 2,549 printed digits, taken from zip-codes. The test database consisted of 2,007 handwritten and 700 printed digits. By using the Optimal Brain Damage method, some architectural changes in the original network were indicated. These were made manually. The network then incorporated 2,600 weights, but still 4 layers of neurons. Using the Optimal Brain Damage on this network, it was possible to automatically remove further 60% of the weights (leaving only about 800 weights), without any significant increase in MSE (Mean Square Error) on the training or test set. The classification error for the 10-categories even decreased for the pruned network.

The Optimal Brain Damage method is based on an estimate of the increase in the cost-function, when removing weights from the network. This method compensates for the phenomena that was mentioned in the section with magnitude based pruning. It incorporates and compensates when:

- The input patterns are not scaled to the same variance.
- The weights in different layers influence each other.

Although the OBD algorithm takes care of scaling the inputs, when calculating the significance of different weights, it is still a good idea to scale the inputs to the network, so the variance of the inputs are uniform. This scaling will help the learning algorithm derived in section 2.2, because the condition number of the input correlation matrix will be smaller as explained in section 2.6.1.

The OBD method uses a second order approximation to the cost-function in each parameter. For a quadratic cost-function with a linear output unit this is an exact result for the output weights, and as described in the experiments in chapter 4 it is also a very good approximation for the input to hidden layer when the parameters are well optimized. The method calculates the so-called **saliency** for each parameter in the network. The parameters with the smallest saliency are then deleted and the network is retrained without these parameters.

The method uses the second order Taylor series in the parameters to approximate the cost-function. This can be written as

$$\begin{aligned} E &= E_0 + \left(\frac{\partial E}{\partial \mathbf{u}}\right)^T \delta\mathbf{u} + \frac{1}{2}\delta\mathbf{u}^T\mathbf{H}\delta\mathbf{u} + o(\|\delta\mathbf{u}\|^3) \\ &\approx E_0 + \left(\frac{\partial E}{\partial \mathbf{u}}\right)^T \delta\mathbf{u} + \frac{1}{2}\delta\mathbf{u}^T\mathbf{H}\delta\mathbf{u}, \end{aligned} \tag{3.4}$$

which is the same approximation for the cost-function as shown in equation (2.20). The different parameters in the equation are explained in section 2.4.2.

The goal is to find a set of parameters which can be deleted (pruned) without a large increase in the cost-function. Of course the cost-function can be calculated with one parameter deleted and the rest of the parameters untouched, but it will be very expensive computationally. Therefore the approximation sketched before is used. There is a problem in using this method; the full Hessian (all the h_{ij}) will be expensive to calculate. Instead the diagonal approximation to the Hessian matrix is used as described in section 2.4.2, where the pseudo Gauss-Newton method for learning the parameters was derived. This approximation assumes that the cost of deleting parameters in the network is additive.

The method used to estimate the change in the cost-function when removing a weight assumes that the network is well trained; close to a minimum in all weights. Then the gradient of the cost-function with respect to each parameter can be approximated by zero. The increase in the cost-function when changing a parameter u_i by δu_i can then be approximated by

$$\delta E_i = \frac{1}{2}h_{ii}(\delta u_i)^2 \tag{3.5}$$

where

$$h_{ii} = \frac{\partial^2 E}{\partial u_i^2} \tag{3.6}$$

If the parameter u_i is pruned, set to zero, the change is

$$\delta u_i = -u_i \tag{3.7}$$

The estimate of the change in the cost-function, when removing a weight from the network, can then be calculated as

$$S_{E_i} = \delta E_i = \frac{1}{2}h_{ii}u_i^2 \tag{3.8}$$

This expression is called the **saliency** S_{E_i} for each parameter.

The saliency is the change in total cost-function given in equation (3.1), but what is interesting is in most cases only, what the change in the training part (quadratic or entropic cost-function) of the cost-function is. The weight-decay part of the cost-function is only added for helping with numerical problems and with pruning by forcing the parameters in the direction of zero. It is therefore in most cases only interesting, what the increase in the training part of the cost-function is. This can easily be calculated by compensating for the weight-decay part. When calculating the saliency for the total cost-function it is assumed that the gradient is zero, because the network is well trained. This assumption does not hold when the saliency for the quadratic part should be calculated. Then it is necessary to compensate for a derivative different from zero. This can easily be done by differentiating the equation for the total cost-function. It can be written as

$$\begin{aligned} \frac{\partial E}{\partial u_i} &= \frac{\partial E_{\text{train}}}{\partial u_i} + \frac{2\alpha}{p} u_i = 0 \\ &\Updownarrow \\ & \frac{\partial E_{\text{train}}}{\partial u_i} = -\frac{2\alpha}{p} u_i \end{aligned} \tag{3.9}$$

Further, the Hessian matrix for the cost-function is changed by the weight-decay term. It can be calculated as

$$\begin{aligned} & \frac{\partial^2 E}{\partial u_i^2} = \frac{\partial^2 E_{\text{train}}}{\partial u_i^2} + \frac{2\alpha}{p} \\ \Updownarrow & \\ & h_{ii} = h_{\text{train},ii} + \frac{2\alpha}{p} \end{aligned} \tag{3.10}$$

The **saliency** for the parameters with respect to the training cost-function (called S_i) can therefore be calculated as

$$\begin{aligned} S_i = \delta E_{\text{train}_i} &= \frac{\partial E_{\text{train}}}{\partial u_i}\delta u_i + \frac{1}{2}\frac{\partial^2 E_{\text{train}}}{\partial u_i^2}(\delta u_i)^2 \\ &= \frac{2\alpha}{p}u_i^2 + \frac{1}{2}\frac{\partial^2 E_{\text{train}}}{\partial u_i^2}u_i^2 \\ &= \left(\frac{2\alpha}{p} + \frac{1}{2}h_{\text{train},ii}\right) u_i^2 \end{aligned} \tag{3.11}$$

This equation can be used to calculate the saliency for each parameter in the network. Equation (2.26) and (2.27) can be used to calculate the second derivative for the learning part of the cost-function.

There are as mentioned earlier also problems using this method. Some of them are

- The estimate is based on the diagonal approximation to the Hessian matrix.
- This method estimates the change in the cost-function before retraining[1] the network. It is the change in the cost-function after retraining the network which is really interesting.

It will be computationally expensive if only one weight is removed before retraining the network. Therefore it is advised in [Le Cun et al., 1990] that more than one parameter could be removed before retraining the network. In the article there is no recommendation of how many parameters could be removed, but as should be seen in section 5 the assumptions are all right if only a few parameters (compared to the total number of parameters in the network) are removed. The assumption is that the total change in the cost-function S_{total} when removing m parameters can be calculated as

$$S_{\text{total}} = \sum_{i=1}^{m} S_i \tag{3.12}$$

In this method for estimating the change in the cost-function only the diagonal terms of the Hessian matrix are used. This simplifies the calculations, but as mentioned before it is only possible to estimate what the change in the cost-function is before retraining and when only one parameter is removed. In the next section a method which can take care of these problems is described. The method uses the same approximation, the second order Taylor series for the cost-function, but instead of a diagonal approximation for the Hessian the full Hessian matrix is used.

3.1.4 Optimal Brain Surgeon method

The **Optimal Brain Surgeon** method by [Hassibi et al., 1993] extends the Optimal Brain Damage method. The difference is that the Optimal Brain Surgeon method uses the full Hessian matrix when calculating the change in the cost-function. Using this method the following advantages are obtained

- The method incorporates retraining in the estimate of the change in the cost-function.
- The method gives an estimate of the optimal parameters after removing several of them (it is not necessary to retrain the network if the second order approximation used is correct).
- The method gives an estimate of the change in cost-function, when removing more than one parameter.

Unfortunately there are some drawbacks when using the method, and especially the following should be mentioned

[1]When parameters are removed from the network, the network is not well trained in the parameter space any more. It is therefore necessary to train the parameters again (**retrain** the parameters) to get a lower training error.

- The method uses the full Hessian matrix, which holds $n(n+1)/2$ terms (n is the number of parameters in the model). This means that a lot of new parameters have to be estimated. This makes new demands to the training set, to have a sufficiently correct estimate of all the terms in the Hessian matrix.
- The estimated Hessian matrix has to be inverted. This is an $o(n^3)$ process.

It shall be mentioned that an iterative method where the inverse Hessian matrix is directly estimated is derived later in this section.

The Optimal Brain Surgeon method calculates the change in the error when changing the weights by using the second order Taylor series approximation as in the Optimal Brain Damage method described in equation (3.4). The change in the cost-function can therefore be approximated by

$$\delta E \approx \frac{1}{2}\delta\mathbf{u}^T\mathbf{H}\delta\mathbf{u}, \tag{3.13}$$

where $\delta\mathbf{u}$ is a vector of the changes in the parameters $\mathbf{u}$.

It is desired to prune a weight u_m to zero. This can be written as

$$\begin{aligned} & \delta u_m + u_m = 0 \\ or & \\ & \mathbf{e}_m^T\delta\mathbf{u} + u_m = 0, \end{aligned} \tag{3.14}$$

where $\mathbf{e}_m$ is a unit vector with same dimension as $\delta\mathbf{u}$ and a 1 corresponding to the weight u_m that should be pruned.

Under the constraints given by equation (3.14) the goal is to find the parameter u_m which gives the least increase in the cost-function E. Using equation (3.13), which gives the increase in cost-function when changing the parameter vector, equation (3.14), and a Lagrange multiplier λ the following Lagrangian equation can be set up

$$L_a = \frac{1}{2}\delta\mathbf{u}^T\mathbf{H}\delta\mathbf{u} + \lambda(\mathbf{e}_m^T\delta\mathbf{u} + u_m). \tag{3.15}$$

Finding the derivative of (3.15) with respect to $\delta\mathbf{u}$ and setting it to zero yields

$$\frac{\partial L_a}{\partial(\delta\mathbf{u})} = \delta\mathbf{u}^T\mathbf{H} + \lambda\mathbf{e}_m^T = 0. \tag{3.16}$$

If the Hessian is assumed to be positive definite so the Hessian Matrix $\mathbf{H}$ can be inverted, the update in the weight vector $\delta\mathbf{u}$ can be found as

$$\delta\mathbf{u} = -\lambda\mathbf{H}^{-1}\mathbf{e}_m, \tag{3.17}$$

By using equation (3.14) and (3.17) it is possible to find the Lagrangian multiplier as

$$\lambda = \frac{u_m}{\mathbf{e}_m^T \mathbf{H}^{-1} \mathbf{e}_m}. \tag{3.18}$$

Using this equation and equation (3.17) it is possible to estimate the update in the parameter vector as

$$\delta \mathbf{u} = -\frac{u_m}{\mathbf{e}_m^T \mathbf{H}^{-1} \mathbf{e}_m} \mathbf{H}^{-1} \mathbf{e}_m, \tag{3.19}$$

and the estimate of the change in the cost-function (called the Optimal Brain Surgeon **saliency** L_E [Hassibi et al., 1993]) can therefore be calculated as

$$\begin{aligned} L_E &= \frac{1}{2} \delta \mathbf{u}^T \mathbf{H} \delta \mathbf{u} \\ &= \frac{1}{2} \frac{u_m^2}{\mathbf{e}_m^T \mathbf{H}^{-1} \mathbf{e}_m}. \end{aligned} \tag{3.20}$$

As for the Optimal Brain Damage method this method has to be modified when weight-decay is used in the cost-function. It is for the "train" part of the cost-function the increase should be estimated and minimized when parameters are deleted. This could be estimated as shown in [Hansen, 1993a], where the assumption about the first derivative of the learning part of the cost-function is calculated from the weight-decay term, and the second derivative is partioned in two, one part which comes from the learning part and one from the weight-decay part of the cost-function. The increase in the "train" part of the cost-function L could then be written as

$$L = \delta E_{\text{train}} = \left(\frac{\partial E_{\text{train}}}{\partial \mathbf{u}} \right)^T \delta \mathbf{u} + \frac{1}{2} \delta \mathbf{u}^T \mathbf{H}_{\text{train}} \delta \mathbf{u}. \tag{3.21}$$

The calculation of the change in the parameter vector $\partial \mathbf{u}$ using a Lagrange multiplyer is the same as for the cost-function without weight-decay so it is as before given by equation (3.19).

The term $\partial E_{\text{train}} / \partial \mathbf{u}$ can be calculated from the assumption that $\partial E / \partial \mathbf{u}$ is zero as

$$\begin{aligned} \frac{\partial E}{\partial \mathbf{u}} &= \frac{\partial E_{\text{train}}}{\partial \mathbf{u}} + \frac{2\alpha}{p} \mathbf{u} = 0 \\ &\Updownarrow \\ & \frac{\partial E_{\text{train}}}{\partial \mathbf{u}} = -\frac{2\alpha}{p} \mathbf{u}, \end{aligned} \tag{3.22}$$

and the Hessian matrix for the cost-function with weight-decay $\mathbf{H}$ can be calculated as

$$\frac{\partial^2 E}{\partial \mathbf{u}^2} = \frac{\partial^2 E_{\mathrm{train}}}{\partial \mathbf{u}^2} + \frac{2\alpha}{p}\mathbf{I}$$
$$\Updownarrow$$
$$\mathbf{H} = \mathbf{H}_{\mathrm{train}} + \frac{2\alpha}{p}\mathbf{I} \qquad (3.23)$$

where $\mathbf{I}$ is the identity matrix with the same dimension as the Hessian matrix (the number of parameters).

The change in the "train" part of the cost-function (the "saliency" of the weights) can therefore using equation (3.21) and (3.22) as in [Hansen, 1993b] be calculated as

$$\begin{aligned} L &= \left(\frac{\partial E_{\mathrm{train}}}{\partial \mathbf{u}}\right)^T \delta\mathbf{u} + \frac{1}{2}\delta\mathbf{u}^T \mathbf{H}_{\mathrm{train}} \delta\mathbf{u} \\ &= \frac{2\alpha}{p} u_m \frac{\mathbf{u}^T \mathbf{H}^{-1} \mathbf{e}_m}{\mathbf{e}_m^T \mathbf{H}^{-1} \mathbf{e}_m} + \frac{1}{2} u_m^2 \frac{\mathbf{e}_m^T \mathbf{H}^{-1} \mathbf{H}_{\mathrm{train}} \mathbf{H}^{-1} \mathbf{e}_m}{(\mathbf{e}_m^T \mathbf{H}^{-1} \mathbf{e}_m)^2}, \end{aligned} \qquad (3.24)$$

where the Hessian for the full cost-function $\mathbf{H}$ can be calculated using equation (3.23).

It can be seen that this equation as expected vanish to equation (3.21) if the weight-decay parameter α is set to zero.

For this method there is a need to calculate the inverse of the full Hessian matrix for all the parameters, both in the hidden and output layers. This will be very costly, so instead a method is proposed where the inverse of the Hessian is directly estimated[2].

3.1.5 Iterative method for estimation of the inverse Hessian matrix.

The iterative method for estimation of the inverse Hessian matrix was described in [Hassibi et al., 1993], but is also used in the conventional recursive method for estimation of parameters in linear models [Ljung, 1987]. The full Hessian of the cost-function E can be written as shown in equation (2.23), where u_i is all the parameters in the network both in hidden layer w_{jk} and output layer W_{ij}. The diagonal elements in the Hessian matrix for the learning cost-function are described in equation (2.26) and (2.27), while the off-diagonal elements can be derived using equation (2.11) and (2.12). In the following equations the elements in the Hessian matrix for the total cost-function including the weight-decay term are described.

$$\begin{aligned} \frac{\partial^2 E}{\partial w_{jk}^2} &= \frac{\partial^2 E_{\mathrm{train}}}{\partial w_{jk}^2} + \frac{\partial^2 E_{\mathrm{weight-decay}}}{\partial w_{jk}^2} \\ &\approx \frac{2}{p}\sum_{i\mu}\left(W_{ij}\frac{\partial V_j^\mu}{\partial h_j^\mu}\xi_k^\mu\right)^2 + \frac{2\alpha_w}{p} \end{aligned} \qquad (3.25)$$

[2]Note, when the number of parameters in the network, N, is comparable to the number of training examples, p, direct inversion of equation (3.23) is computationally feasible.

$$\begin{aligned}\frac{\partial^2 E}{\partial W_{ij}^2} &= \frac{\partial^2 E_{\text{train}}}{\partial W_{ij}^2} + \frac{\partial^2 E_{\text{weight-decay}}}{\partial W_{ij}^2} \\ &\approx \frac{2}{p}\sum_{\mu}(V_j^{\mu})^2 + \frac{2\alpha_W}{p} \end{aligned} \tag{3.26}$$

$$\begin{aligned}\frac{\partial^2 E}{\partial w_{jk}\partial W_{ij}} &= \frac{\partial^2 E_{\text{train}}}{\partial w_{jk}\partial W_{ij}} + \frac{\partial^2 E_{\text{weight-decay}}}{\partial w_{jk}\partial W_{ij}} \\ &\approx -\frac{2}{p}\sum_{\mu} V_j^{\mu} W_{ij} \frac{\partial V_j^{\mu}}{\partial h_j^{\mu}} \xi_k^{\mu} \end{aligned} \tag{3.27}$$

$$\begin{aligned}\frac{\partial^2 E}{\partial w_{jk}\partial w_{j'k'}} &= \frac{\partial^2 E_{\text{train}}}{\partial w_{jk}\partial w_{j'k'}} + \frac{\partial^2 E_{\text{weight-decay}}}{\partial w_{jk}\partial w_{j'k'}} \\ &\approx \frac{2}{p}\sum_{i\mu}\left(W_{ij}\frac{\partial V_j^{\mu}}{\partial h_j^{\mu}}\xi_k^{\mu}\right)\left(W_{ij'}\frac{\partial V_{j'}^{\mu}}{\partial h_{j'}^{\mu}}\xi_{k'}^{\mu}\right) \end{aligned} \tag{3.28}$$

$$\begin{aligned}\frac{\partial^2 E}{\partial W_{ij}\partial W_{i'j'}} &= \frac{\partial^2 E_{\text{train}}}{\partial W_{ij}\partial W_{i'j'}} + \frac{\partial^2 E_{\text{weight-decay}}}{\partial W_{ij}\partial W_{i'j'}} \\ &\approx \frac{2}{p}\sum_{\mu} V_j^{\mu} V_{j'}^{\mu} \end{aligned} \tag{3.29}$$

Equation (3.25), (3.27) and (3.28) is simplified as equation (2.27) by using the Levenberg-Marquart approximation. This assumes that the error $(\zeta_i^{\mu} - O_i^{\mu})$ is uncorrelated with the input patterns ξ_k^{μ}.

The Hessian matrix can therefore in matrix notation be written as

$$\mathbf{H} = \mathbf{H}_p + \mathbf{H}_0, \tag{3.30}$$

where the two matrices are

$$\mathbf{H}_p = \frac{2}{p}\sum_{\mu=1}^{p} \mathbf{X}^{\mu}\mathbf{X}^{\mu T} \tag{3.31}$$

and

$$\mathbf{H}_0 = \mathbf{D}(\mathbf{\Lambda}). \tag{3.32}$$

The matrix $\mathbf{D}(\mathbf{\Lambda})$ is a diagonal matrix with the elements in the vector $\mathbf{\Lambda}$ placed as diagonal elements in the matrix. $\mathbf{X}$ and $\mathbf{\Lambda}$ are two vectors with dimension as the number of parameters. They are defined as

$$\mathbf{X}^{\mu} = [x^{\mu}_{h_1}, x^{\mu}_{h_2}, \ldots, x^{\mu}_{h_{N_h}}, x^{\mu}_{o_1}, x^{\mu}_{o_2}, \ldots, x^{\mu}_{o_{N_o}}]^T \tag{3.33}$$

and

$$\mathbf{\Lambda} = [\alpha_w, \alpha_w, \ldots, \alpha_w, \alpha_W, \alpha_W, \ldots, \alpha_W]^T, \tag{3.34}$$

where N_h is the number of parameters in input to hidden layer and N_o is the number of parameters in hidden to output layer. The number of α_w and α_W coefficients corresponds to the number of parameters in the two layers. The elements in the $\mathbf{X}$ vector can be found as

$$x^{\mu}_{h_p} = \sum_i \left(W_{ij} \frac{\partial V^{\mu}_j}{\partial h^{\mu}_j} \xi^{\mu}_k \right) \tag{3.35}$$

and

$$x^{\mu}_{o_p} = V^{\mu}_j, \tag{3.36}$$

where the parameter p is the index in respectively the hidden layer weights w_{jk} and the output layer weights W_{ij}.

The Hessian matrix can therefore be calculated iteratively as

$$\mathbf{H}_{m+1} = \mathbf{H}_m + \frac{2}{p}\mathbf{X}^{(m+1)}\mathbf{X}^{(m+1)T} \tag{3.37}$$

with $\mathbf{H}_0$ defined in equation (3.32) as the weight decay part of the Hessian matrix.

By using the standard Matrix inversion lemma [Ljung, 1987]

$$[\mathbf{A} + \mathbf{BCD}]^{-1} = \mathbf{A}^{-1} - \mathbf{A}^{-1}\mathbf{B}[\mathbf{DA}^{-1}\mathbf{B} + \mathbf{C}^{-1}]^{-1}\mathbf{DA}^{-1} \tag{3.38}$$

the inverse Hessian matrix can be estimated directly in a recursive way. This can be written as

$$\mathbf{H}^{-1}_{m+1} = \mathbf{H}^{-1}_m - \frac{\mathbf{H}^{-1}_m \mathbf{X}^{(m+1)}\mathbf{X}^{(m+1)T}\mathbf{H}^{-1}_m}{\mathbf{X}^{(m+1)T}\mathbf{H}^{-1}_m\mathbf{X}^{(m+1)} + \frac{p}{2}} \tag{3.39}$$

where $\mathbf{H}^{-1}_0$ is found by inverting all the elements in the diagonal. This means that it can be calculated as

$$\mathbf{H}^{-1}_0 = \mathbf{D}(\mathbf{\Lambda}^{-1}), \tag{3.40}$$

where $\mathbf{\Lambda}^{-1}$ is defined as

$$\mathbf{\Lambda}^{-1} = [1/\alpha_w, 1/\alpha_w, \ldots, 1/\alpha_w, 1/\alpha_W, 1/\alpha_W, \ldots, 1/\alpha_W]^T. \tag{3.41}$$

For $m + 1 = p$ the calculations of the inverse Hessian in this iteratively way gives that

$$\mathbf{H}^{-1} = \mathbf{H}_p^{-1} \tag{3.42}$$

which means that if the cost-function with weight-decay is used, the iterative scheme gives an exact calculation of the inverse Hessian. It is therefore possible to calculate the inverse Hessian that should be used in the equation (3.20). If the Optimal Brain Surgeon saliency should be calculated without the weight-decay term as in equation (3.24), this way of calculating the inverse Hessian has a problem, because it is necessary to initialize the inverse Hessian $\mathbf{H}_0^{-1}$, with something that is bigger than zero to have this method work.

The basis for this is that it is necessary to be sure to have non zero eigenvalues in the Hessian matrix before it is possible to calculate the inverse Hessian, and the weight-decay term is precisely helping in securing that no zero eigenvalues in the Hessian matrix exist. Therefore if this method should be used to calculate the inverse Hessian matrix, the best solution will probably be to use the weight-decay inverse Hessian $\mathbf{H}^{-1}$ as an estimate of the $\mathbf{H}_{\text{train}}^{-1}$ of the cost-function without weight-decay.

The requirements to the learning set when using this method are very crucial while all the terms in the Hessian matrix have to be estimated. In the examples shown in the next chapters and in the articles appended the more simple OBD method, where only the diagonal terms in the Hessian matrix have to be estimated, has therefore been used, while the number of examples in the learning set are very limited.

3.2 Growth Algorithms

Instead of starting with a large network with superfluous connections and units, methods have been proposed where the network is started up with only a few layers, units and connections. This idea is attractive because it often is easy to train the parameters in these small networks.

One of the first methods proposed was the **cascade-correlation** method by [Fahlman, 1989]. In this method the network initially has only one unit. When the parameters to this unit are well trained on the output signal they are fixed. Then a new hidden unit is placed in the network. This unit is trained to correlate to the error signal, which means the difference between the desired output signal and the output from the trained neural network. When this new unit is well-trained on the error signal, the weights connecting the unit are fixed. The output unit is then retrained with the output from the new hidden unit as an extra input. When the weights are well trained the performance of the network is tested. If the performance is satisfactory the algorithm is stopped, else wise a new hidden unit is added and this unit is trained in the same way as the first one. By adding new units in this way it is possible to obtain as good a performance as wanted on the training set.

The drawback of this method is that a network with a lot of layers is obtained. Each time a new unit is added an extra layer is added to the network structure. In some experiments done at the Electronics Institute [Hansen and Hansen, 1992] it has been shown that almost every time 50 units should be added to the network structure to solve the "two-spiral" problem (the same test problem as used in [Fahlman, 1989]). Further this network has a very complicated structure with many hidden layers. By just using an ordinary two layer feed-forward network the problem could be solved by a network with about 30 units! It seems as there is a problem in getting the optimal performance from each of the new units added. A lot of connections could therefore probably be removed, but the network with all the layers is not very well suited for a pruning method such as the Optimal Brain Damage method, due to the fact that this type of pruning method uses a second order approximation to the cost-function in each weight. With a network structure as the structure built by the cascade-correlation method with many layers of non-linear units, the second order approximation to the cost-function is probably not very good. This means that some "wrong" weights probably are removed by the method, weights that when removed change the cost-function more than weights still present in the network. This has not been tested though.

The network structure which is generated by the cascade-correlation method can be transformed to a structure with only one layer of hidden units. Experiments done during this Ph.D. study have shown, that if new units are added to already existing and well trained units in one layer, it is very difficult to obtain the optimal performance from these new units. The output from the new units seems to have a tendency to be correlated with the output from the already present and trained units in the network. A network with one hidden layer, where all the parameters are initialized with random values and then trained, is much easier to train to an optimal performance than a network with a couple of units which already are trained and then adding some new units with random weights and training all the weights. It is assumed that the number of units and weights in the resulting networks are the same. The explanation for this phenomenon must be that the optimization algorithm for the parameters in the small network must have found a local minima in the weights space, where it is difficult to escape from. When training the large network the training algorithm normally doesn't get stuck in these local minima.

In the paper [Fahlman, 1991] another variant of the cascade-correlation algorithm is proposed where recurrent connections in the network are introduced. It should give a more efficient network, where fewer connections are needed to solve a given problem. It will also give even more difficult problems though, when the performance of this type of network shall be analyzed, while no method has been found for analyzing the performance of networks with recurrent connections.

Other algorithms have been proposed for adding new units to an already existing and well trained network. One of them is the so-called **upstart algorithm** by [Frean, 1990], which also adds new units to a hidden layer to correct as many as possible of the errors made by the already existing network. This algorithm works only for classification problems, and the trick in this algorithm is that new units are only trained on examples which couldn't be learned by the original units. It has many similarities to the cascade-correlation algorithm. This method has not been tested during this Ph.D. study.

The above mentioned methods add new units with random weights and try to learn the error that the small networks can't do. In that way the total network (original and new weights) is optimized to minimize the error. The experiments have though shown that it is hard to learn the errors in this additive way, where new weights just are correlated to the error signal. Further it has to be analyzed if it is both the learning error and the generalization error that are going down by adding the new connections and minimizing the learning error.

Due to all the problems mentioned, when letting the networks grow (adding new units) and trying to achieve optimized performance from the new units, which means using their resources, not many experiments have been done with network growth algorithms. In the experimental part of this thesis no experiments with the growth algorithms will be mentioned.

3.3 Estimation of a stopping criterion for pruning algorithms

When using a pruning algorithm it is necessary to have a method to make out when the optimal number of parameters is reached. What does the optimal number of parameters in the model mean? There are two problems, too many parameters and too few parameters. If too few parameters are present in the model, it is not possible for the model to learn the rule of the learning set. When there are too many parameters in the model, the model fits the data points in the learning set too well. It learns, in addition to the rule of the data set, the noise in the learning set; examples of underfitting and overfitting problems were given in section 2.1. The problem of too many parameters in the model only arises for small learning sets, which means learning sets with too few examples or learning sets with a great deal of correlation between the different examples in the set. Therefore a method for estimating the optimal model order should both take the effective number of examples and the effective number of parameters into consideration. In the next sections a method for estimating the effective number of parameters in the model is described. In this method the correlation in the learning set, and therefore the effective number of examples, is taken into consideration, while the eigenvalues used are based on the set of training examples. A straight boundary for the number of parameters that could be incorporated in the model is determined by the fact that it is not possible to determine more parameters than the number of examples in the data set. If there are more parameters than the number of examples in the learning set, some of the parameters could be changed without any change in the performance of the model.

To be able to define a stopping criterion for the pruning algorithm, it is necessary to define what an **optimal model** is. It could be defined as the model with the lowest **generalization error**, which means the model with the lowest error on an independent data set, taken from the same distribution of data as the learning set. A method for finding the best model could therefore be based on an independent data set, which is used for testing the model. A disadvantage with this method is that it is necessary to reserve some of the data from the learning set for testing the model. It is often difficult or expensive to obtain new data. Therefore it is not very satisfactory to have to reserve some data sets just for testing. It would instead be very desirable if it was possible to find a method for estimating the optimal model order without any need for reserving some of the data for testing purposes. Instead all the data sets should be used for learning the parameters. In some of the experiments (chapter 5) done during this Ph.D project it has been shown that the method of using a test set for finding the optimal number of parameters in the model demands a "quite large" test set to get a confident estimate of the optimal number of parameters.

3.3.1 Stopping criterion defined by Akaike's Final Prediction-error Criterion

In this section a method for estimating the generalization error for problems with Gaussian noise on the learning set is defined. The methods are derived in appendix A. By using this estimate for the generalization error it is possible to define a stopping criterion for the pruning algorithm. In [Ljung, 1987] the method originally given by Akaike [Akaike, 1968; Akaike, 1969] for estimating the generalization error is derived. The basis for the derivation is the fact that the generalization error can be described by the second order Taylor series in the error function in the "correct" parameters as

$$E_{\text{generalization}} = E^* + \frac{1}{2}\delta\mathbf{u}^{*T}\mathbf{H}^*\delta\mathbf{u}^* \tag{3.43}$$

where $E_{\text{generalization}}$ is the generalization error and E^* and $\mathbf{H}^*$ are respectively the error function and the Hessian matrix in the "correct" parameters. $\delta\mathbf{u}^*$ is the variation of the parameters around the optimal parameter set $\mathbf{u}^*$ where the derivative of the cost-function is zero.

In appendix A Akaike's Final Prediction-Error estimate (FPE) of the generalization error $J(\mathcal{M})$ is derived. For the case without regularization in the cost-function the derivation is described in section A.1.1. It can be written as

$$J(\mathcal{M}) = K_{\text{FPE}}E_{\text{Learn}} = \frac{1 + N_{\mathcal{M}}/p}{1 - N_{\mathcal{M}}/p}E_{\text{Learn}}, \qquad N_{\mathcal{M}} < p, \tag{3.44}$$

where $N_{\mathcal{M}}$ is the model order (the number of free parameters (weights) in the model), p is the number of examples in the learning set, and E_{Learn} is the learning error.

It is known that the learning error approaches zero when the number of parameters in the model is going to ∞, while it can be seen that the FPE coefficient K_{FPE} is going from one to ∞, when the ratio $N_{\mathcal{M}}/N$ is changed from 0 to 1. In figure 3.1 this is sketched, together with the FPE estimate of the generalization error. From this figure it can be seen that a minimum of the generalization error must exist when the number of parameters is changed.

The figure shows that a minimum of the FPE exists in between (eventually in the bounds) zero parameters in the model and N parameters in the model, where N is the number of examples in the training set. When the pruning is started it is assumed that the model incorporates a number of parameters larger than the number which is given by

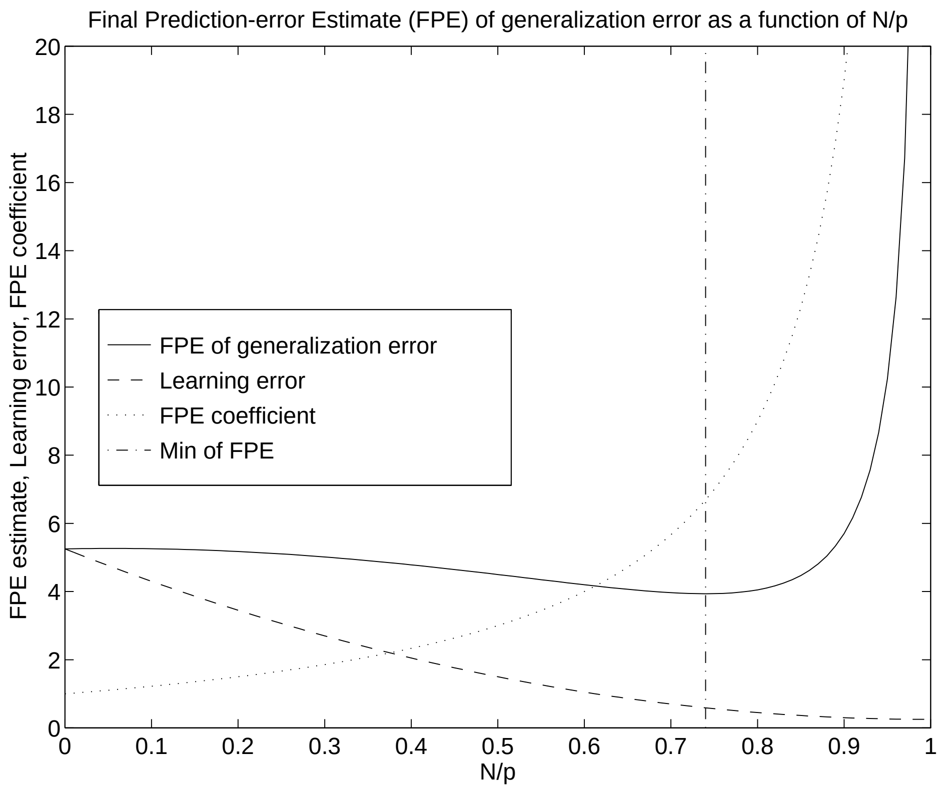


Figure 3.1: Final Prediction-error Estimate of the generalization error.

the minimum on the FPE curve. The pruning is then started and some parameters are removed from the model. Whenever the FPE has reached a minimum the pruning can be stopped and it is assumed that the optimal model is reached. In chapter 5 some examples of using this method for stopping the pruning of the neural network will be given. Further the estimate will be tested against a generalization error estimated by an independent test set.

In the article [Kashyap, 1980] the inconsistency of Akaike's information criterion (AIC) rule for model order estimation for autoregressive models is calculated. For Gaussian innovations (noise) in the model and if $N_{\mathcal{M}} << p$ can be assumed, which often is the case, the AIC and FPE calculations show the same results, so the conclusions for the inconsistency in the AIC rule can also be used for the FPE rule. In the article it is derived that a lower bound for the probability of an error decision for the model order using the AIC decision rule and thereby the FPE decision rule is 0.156. This means that the estimate given by the FPE rule for model order estimation is inconsistent for some cases. In some of the experiments done through this Ph.D. project it has indeed been seen that the FPE rule for some of the experiments give a wrong estimate of the model order. This is tested by having an independent set of examples for testing the FPE rule. To help with this problem, an ensemble of prunings starting from a fully connected network has been made for many of the examples. The "best" of the pruned network has then been chosen. How the best is chosen will be explained in the experiment sections.

There is a problem in the FPE estimate because the number of parameters used in equation (3.44) is assumed to be equal to the number of parameters (weights) in the network. The number of parameters in the equation is the number of free parameters in the model. In the minimization done when using the cost-function with weight-decay some bounds are put on the parameters, so the number of free parameters is less than the real number of parameters in the model. In [Moody, 1992; Larsen, 1993] a method for estimating the number of free parameters in the model is derived. The calculation of this is in the notation used throughout this thesis done in section A.1.1. The number of free parameters is called the effective number of parameters N_{eff} and can be calculated as

$$N_{\text{eff}} = \sum_{jk} \left(\frac{\sigma_{jk}}{\sigma_{jk} + 2\alpha_w/p} \right)^2 + \sum_{ij} \left(\frac{\sigma_{ij}}{\sigma_{ij} + 2\alpha_W/p} \right)^2 , \tag{3.45}$$

where σ_{jk} and σ_{ij} is the second derivative (eigenvalues) of the training part of the cost-function, calculated in equation (2.27) and (2.26). Further experiments have shown that using the second derivatives from the diagonal approximation to the Hessian matrix gives some problems, when they are used in this equation. As a symptom of the problems the estimated number of parameters in a model can be larger than the number of training examples, which can't be true. It is not possible for a learning set consisting of N examples to span a model with more than N parameters, and therefore the effective number of parameters in a model can't be larger than the number of examples in the training set. This problem doesn't appear when using the full Hessian matrix. The problem in using the full Hessian matrix is as mentioned before that it is very computationally expensive to estimate and calculate the eigenvalues.

As shown in appendix A the effective number of parameters N_{eff} can then be used in equation (3.44). The FPE estimate of the generalization error can then, when there is weight-decay at the cost-function, be calculated as

$$J(\mathcal{M}) = \frac{1 + N_{\text{eff}}/p}{1 - N_{\text{eff}}/p} E_{\text{Learn}}, \qquad N_{\text{eff}} < p. \tag{3.46}$$

This estimate of the generalization error is used throughout the examples shown in chapter 5.

3.3.2 Stopping criterion for classification problems

For classification problems the method sketched in the previous section can not be used, while noise at the training set can not be consider to be Gaussian. This assumption about the noise is used when the equation for the FPE estimate of the generalization error is derived. Therefore another method for choosing when to stop pruning of the network has to be used.

For problems where no noise is present in the training set, which means that all the examples in the learning set are 'correct', a way of defining a stopping criterion is to delete parameters in the network until the network can't do the learning set anymore. It is believed that the smallest network (the network with fewest parameters), is also the network that has the best generalization ability [Thodberg, 1991].

This way of choosing when to stop pruning can only be used when there is no noise at the learning set. When noise is present at the learning set the network chosen by this method will have too many parameters, because it can do both the correct examples and the examples which are wrong. A method for estimating the level of noise in the learning set then has to be found. No method for this is given; instead a test set has to be used. The pruning is then stopped when the best generalization ability (the lowest test error) is reached. Another way of choosing when to stop, where only one example from the learning set has to be spared for testing, is to use a leave one out cross validation method for estimating the test error. A network is trained for all learning set configurations where one example is spared for testing. Then the generalization error is found as the mean of all these test errors at one example. This method is very computationally expensive because so many different networks have to be trained and pruned. It is also a problem that almost every time a new network pruning is going on, a different combination of the weights is going into the pruned network. This means that it is difficult to choose the optimal network configuration using this method.

In chapter 4 some different examples of pruning of networks used for classification are shown. The contiguity problem is an example of a pruning session where no noise is present in the learning set, and the optimal network is therefore chosen by the first method described, where the smallest network that can do the task is used. In the problem where cell shapes are classified there is noise present in the training set, and a test set is used for choosing the optimal network configuration. The leave one out method for estimating the generalization error has been too computationally expensive to use.

3.4 Adaptive regularization

From the method for estimating the generalization error derived in appendix A an automatic method for tuning the weight-decay parameters can be derived as also shown in [Hansen and Rasmussen, 1994]. This method gives an "optimal" way of tuning the weight-decay parameter in the training algorithm, while it is based on an iterative minimization of Akaike's FPE estimate of the generalization error estimate with respect to the weight-decay parameters. The derived algorithm is iterative, while it hasn't been possible to find the exact solution. The following two equations describe the algorithm

$$\alpha_w(n+1) = \alpha_w(n) - \rho \frac{\partial E_{\text{generalization}}(\mathbf{u_0})}{\partial \alpha_w} \tag{3.47}$$

$$\frac{\partial E_{\text{generalization}}(\mathbf{u_0})}{\partial \alpha} = \frac{4}{p^2} \sum_{l=1}^{N_w} \frac{(\alpha_w u_l^2 - E_{\text{learn}}(\mathbf{u}^*))\sigma_l^2}{(\sigma_l + 2\alpha_w/p)^3} \tag{3.48}$$

where w is a set of weights which has the same weight-decay parameter α_w and σ_l is the eigenvalue for each of these weights. The weight-decay parameter can then be found by using these equations iteratively.

In some of the experiments shown in chapter 5 and some of the articles, this method for tuning the weight-decay parameters is demonstrated. The results are quite obvious; it is necessary to have a larger weight-decay in the network when many parameters are present in the network. When parameters are removed by pruning it is possible to let the weight-decay parameters decrease, which is also shown in some of the experiments.

Chapter 4

Examples: Classification systems

In this chapter different examples of using neural networks for solving classification tasks will be shown. It is especially shown how optimization of the neural network architectures by pruning helps in getting a better generalization ability from the classifier. It is in general believed that a good generalization ability of the model is associated with a minimal representation or minimization of the number of free parameters in the network model [Thodberg, 1991]. For the classification examples shown in this chapter a combination of the techniques is used, minimization of the number of free parameters (regularization by weight-decay methods) is used while superfluous parameters are removed from the networks architecture (the networks are pruned). In the method used to estimate the saliency of the parameters a second order approximation of the cost-function is used. If no regularization of the parameters is present the estimate of the Hessian matrix (full or diagonal approximation) can be singular, the regularization term in the cost-function also helps in solving this numerical problem, so the estimate of the Hessian matrix certainly is none singular and therefore can be used to estimate the saliencies for the network parameters.

Two different examples are given in this chapter: the first is a toy problem, the contiguity classification problem, while the other example is taken from classification of biological cells from their shape. In appendix D an article describing the contiguity problem and the Optimal Brain Damage method is appended, while in appendix G an article demonstrating the methods used for a cell classification problem is appended. In the next sections some further explanations for these tasks will be given.

4.1 Classification in the contiguity problem

The contiguity problem is in general used as a benchmark problem for testing the performance of neural networks in classification tasks. The contiguity problem is discussed in e.g. [Denker et al., 1987]. In the article appended in appendix D the performance of the neural network methods used throughout this thesis is discussed.

The problem is also called the *two or three clumps* problem. It has a boolean input field consisting of ±1's. The number of $+1$'s is counted and the patterns with two and three clumps of $+1$'s are selected. The boolean output is set to -1 for the inputs with two groups of $+1$'s and to $+1$ for the inputs with three groups of $+1$'s. In the examples shown in this section an input consisting of 10 boolean inputs is used. For this input size there exist 792 input patterns of which 432 have three clumps and 360 have two clumps. A full description of the problem is appended in the article in appendix D. It can be shown that a simple sparse symmetric network structure as shown at figure 3 in the article can solve the problem. The hidden units can be understood as a kind of edge detectors. If the network structure is initialized to this simple structure, with receptive fields of 2 or 3 in the input space for the hidden units and some restrictions on the values that the weights can take, it can be shown that the number of examples needed to learn the problem is reduced to 30-40 examples [Denker et al., 1987; Schwartz et al., 1990] which is very few examples compared to the number of examples that should be used when a fully connected network should learn the problem (see experiments in next section).

In the next subsections some curves showing the learning curves for the problem and some of the simple network structures are presented. The example is also used to test the approximations done in the Optimal Brain Damage method for pruning the network.

4.1.1 Learning curves

The learning curves quantify how many examples should be used to learn a given problem. To analyze how the network architecture optimization methods help in giving a learning transition for fewer parameters in the network, the learning curve for the fully connected network is first pictured[1]. In figure 4.1 the learning curve is shown for fully connected networks. The generalization ability is calculated at the part of the data set which hasn't been used for training purposes. At each training set size ten networks has been trained with the optimization methods described in chapter 2.2 with constant weight-decays $\alpha_w = 0.03$ and $\alpha_W = 0.03$.

From the figure it can be seen that the ensemble performance using the majority vote gives an error that is in the lower part of the errors that each network gives individually. This shows that it is possible to obtain a better performance of the neural network method by using an ensemble of network, trained on the same training set, but initialized by different randomly chosen weights. This has also been shown in [Hansen and Salamon, 1990]. From the graph it can further be seen that the learning transition for the individual fully connected networks is located around 200 examples, while the ensemble method with 10 networks used moves the learning transition to approximately 160 examples.

It is desired to test the performance of the pruning method. This is done by pruning and retraining the fully connected networks shown in figure 4.1. The optimal network configuration for each network pruning session is then chosen as the smallest network that still can do the learning set. In figure 4.2 the generalization ability for these networks is shown. A solid line is drawn through the smallest networks that still can do the learning task for each training set size. It can be seen that the network chosen by this heuristic

[1]The learning curve shows the network performance (the generalization ability or test error) as a function of the number of examples in the training set.

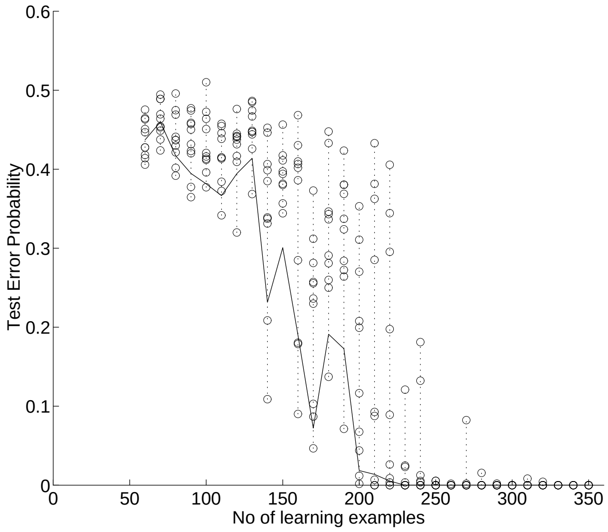


Figure 4.1: Learning curve for the fully connected network architectures (10-12-1). At each training set size ten networks have been trained on the same randomly chosen training set. The majority vote of the ensemble is shown by the solid line.

method is also a network with a quite optimal performance; for most training set sizes the smallest network is also the network with the lowest generalization error. By using this method for choosing the optimal network it can be seen that the learning transition is located around 90 examples. This shows that the pruning method helps in getting a better generalization ability of the networks for smaller training set size.

4.1.2 Choose most pruned network

This strategy of choosing the most pruned network that still can do the training set from an ensemble of networks trained on the same training set can also be illustrated by training a bunch of networks at the same training set size, and then looking at their generalization ability. In figure 4.3 50 networks are trained at the same training set consisting of 140 examples chosen randomly in between the 792 data sets, all nets trained at the same training set. The rest of the data set is used for testing the generalization performance. It can clearly be seen that the networks which have the best performance are the smallest networks, with approximately 40 weights present in their architecture.

If the training set size is large enough, the network structure generated by the pruning algorithm is quite close to the network structure shown at figure 3 in the article, appendix D. As described earlier a quite obvious architecture which can solve the problem is the symmetric network architecture, where each of the hidden units acts as a kind of edge detector. In figure 4.4 an optimal network structure which has been found by using the pruning algorithm is shown. The network is trained at a training set size of $p = 160$. The network structure has manually been unfolded with respect to receptive fields of the

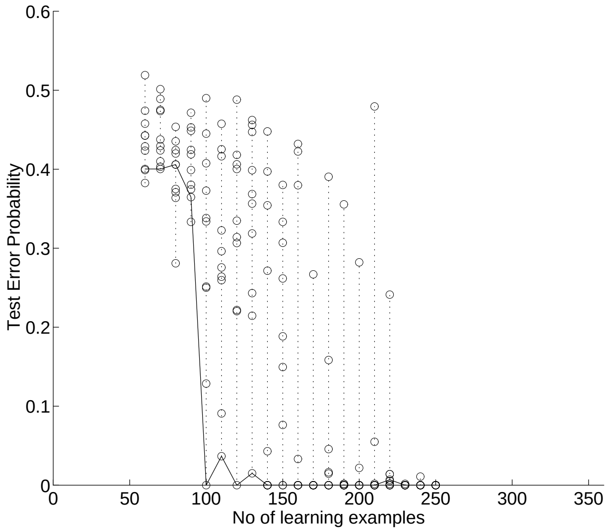


Figure 4.2: Learning curve for the pruned network architectures. Each pruning session is started with one of the fully connected networks which has been shown in figure 4.1. The solid line shows the performance of the smallest pruned network which still can do the training set.

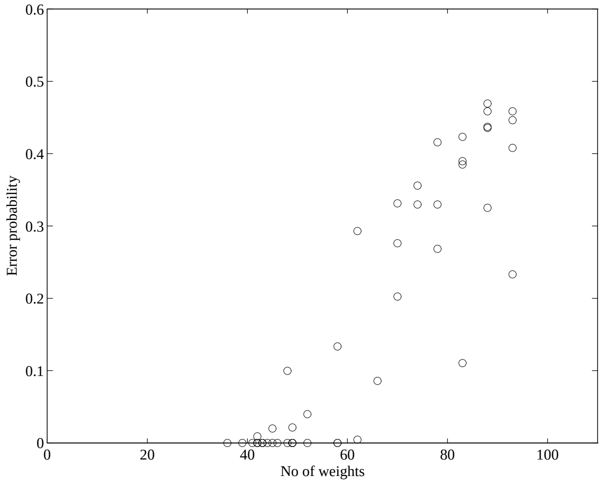


Figure 4.3: Test error probability as a function of weights in the smallest network that still can do the training set. Fifty networks are trained at the same training set consisting of $p = 140$ examples.

hidden units. It can also be seen that only 8 hidden units are used, while the "human" solution uses 9 hidden units.

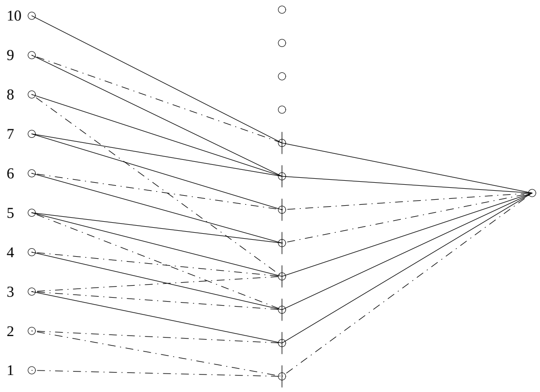


Figure 4.4: The figure shows one of the optimal network structures found by training the network at a randomly chosen training set consisting of p = 160 examples. The network has manually been unfolded by permutation of the hidden units with respect to the receptive fields of the hidden units. Solid line indicates positive weights, while dot-dashed lines indicate negative weights. A vertical bar through a unit indicates that a threshold value is present for this unit.

4.1.3 Approximations in OBD and learning

The optimal brain damage method uses the assumption that the weights in the network do the same task even though superfluous weights are removed from the network structure. This means that the fluctuations in the values of the weights still present in the network during a pruning session (removing superfluous weights and retraining the rest) are expected to be quite slow. In figure 4.5 the evolution of the weights during a pruning session is illustrated. It is seen as expected that the weights are only adjusted a little after a pruning; the assumption that the weights still are doing the same task after the pruning has taken place can be said to be fulfilled.

Another assumption made in the pruning scheme is that the cost-function can be approximated by a second order approximation in each of the weights. In figure 4.6 it can be seen that this approximation holds quite well for both the fully connected and pruned networks. For two of the weights with largest error in the estimate of the saliency, figure 4.7 shows the estimate of the saliency and the calculated error when changing the weight values. It can be seen that the "correct" error function can not be approximated well by the second order approximation for these weights. There are however many problems in using higher order approximations when estimating the saliency, while even more parameters have to be calculated which means that problems with singularity of the estimated parameters will be even bigger. It can also from figure 4.6 be seen that the approximation in general is very good.

The reason why weight-decay at the cost-function helps in getting a better estimate of the

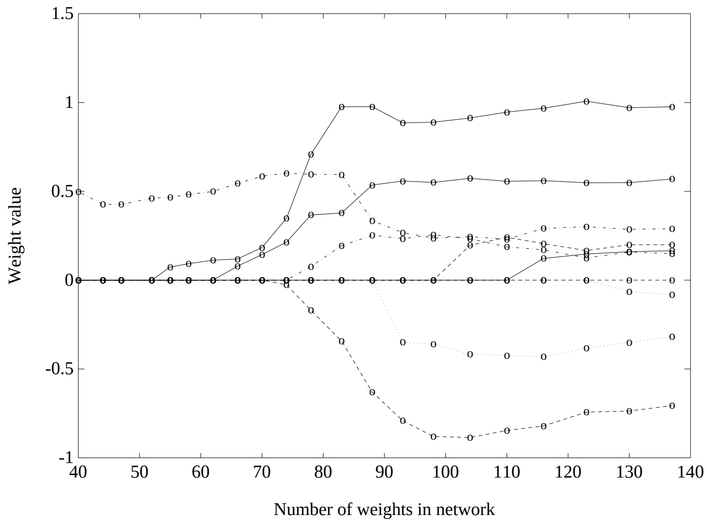


Figure 4.5: Evolution of the weight during a pruning session. Network pruning is started with a fully connected network. Only a small fraction of the weights are shown at the figure, they are chosen randomly. The training set consist of $p = 200$ randomly chosen examples.

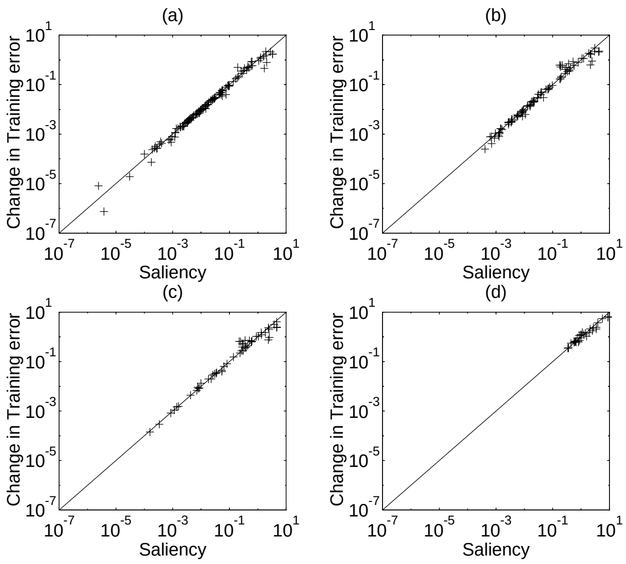


Figure 4.6: Experimental test of the OBD approximation on the contiguity problem at four different stages during a pruning session: (a) fully connected (145 weights), (b) 104 weights, (c) 66 weights, and (d) 42 weights. The training set consists of $p = 200$ patterns. Only input-hidden weights are shown.

saliencies for the weights using a second order approximation can also be seen in figure 4.7. It is seen that if the minima of the cost-function for the weights is "near" zero the

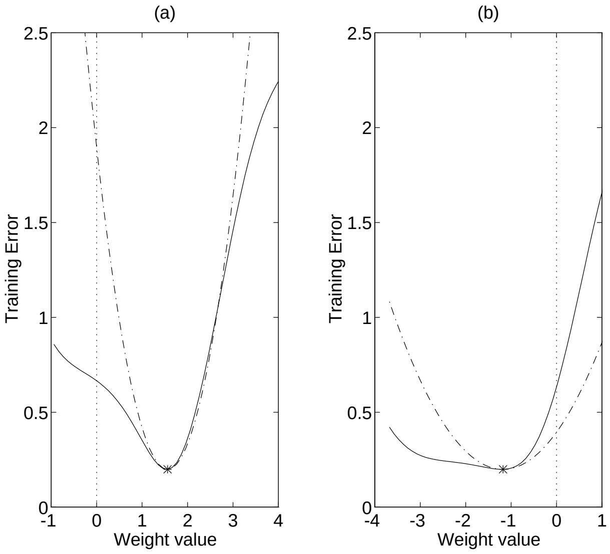


Figure 4.7: Test of second order approximation for two weights with high saliency and a significant error in the saliency estimate. The second order approximation of the cost-function is shown as a dot-dashed line, while the solid line shows the actual error as a function of the the weight. In a) the error is overestimated by a factor 2.7 while b) underestimates the saliency by a factor 0.6. For both these weights the cost-function is poorly approximated by the second order approximation.

approximation is very good. The approximated curve and the "correct" error curve follow each other very well near the minima. Therefore a weight-decay which ensures that the minima is not too far from zero (the weights are pulled in the direction of zero) helps in having the OBD method to give a quite "correct" estimate of the actual cost-function change when removing weights from the network (the weight is set to zero). If the distance between the weight value at the minima of the cost-function and zero is larger it is also seen at the curves that more sophisticated functions (higher order approximations to the cost-function in the weights) have to be used, to get "correct" estimates of the change in the cost-function when setting weights to zero.

4.2 Classification of cell shapes from images

Within the areas of medicine and biology there is a need for fast and flexible methods for identification and classification of cells in microscope imagery. The ccd camera/microscopes have made it possible to easily digitalize samples of cells. In this section examples of classification of images of cells will be shown. The neural network technique has successfully been used to classify cells from the contour of the cells. In the Ph.D. study cells from different areas have been worked at, from classification of algae from the sea to classification of human sperm cells. During the project it has been found that it is very difficult to get enough images of sufficiently high quality to have a data set that could be used for neural network training and evaluation.

In section 4.2.1 the preprocessing techniques used to normalize the cell shape images are described. The examples shown for illustrating the methods used for preprocessing are from classification of some cells in two classes; one class of circular cells and one class of cells with pseudopodias. The data has been produced by [Wellhouse and Salamon, 1992]. The database is artificial cell shapes sketched by an operator after having inspected cell imagery.

In appendix G an article describing experiments done in classification of human sperm cells is appended. Some additional experiments done on the data are described in subsection 4.2.2.

In images as well as for other classification problems, it is essential that items which should be classified as belonging to the same class also look like each other in the data description. Therefore it is necessary to find methods which normalize the data description as much as possible, so a relative invariant set of descriptors for similar cells has similar input descriptors. For recognition of items in images different descriptions of the items can be chosen. If it is the shape of the items which is essential for the recognition, a very powerful method is to use Fourier descriptors. A method for normalizing the Fourier descriptors for cell shapes will be described in the next subsection.

4.2.1 Fourier preprocessing technique

The contour of a shape e.g. a cell, a particle in the air, an airplane or a ship is sampled in the x-y plane. Many parameters play a role for sampling of a shape

- The number of sampling points around the shape.
- Variations in the starting point of the sampling sequence.
- Differences in size of the shapes, varying with e.g. the distance to the object.
- The rotation of the shape.
- Origo of the x-y coordinate system (translations in the plane).

Often it is desirable to make the description invariant under changes in all these parameters. In the following, transformations which make the data description of the contour of the shapes invariant, will be given. Of course some of the parameters could be essential for classification problems, e.g. the size of the cell, but then these transformations could be left out. The example used to illustrate the techniques is taken from [Wellhouse and Salamon, 1992; Wellhouse, 1993], where the problem is to classify two different types of cells (pseudopodia cells and circular cells), as it will be illustrated by the figures in the next sections. In the articles [Granlund, 1972; Kuhl and Giardina, 1982; Kerley and Knisley, 1993; Wu and Schowengerdt, 1991; Diaz et al., 1989] the techniques are also used in a more or less systematic way.

The approach is as mentioned based on a sample of the shape in the x-y plane. The shape is then represented by M samples of complex numbers z_k as

$$z_k = x_k + i \cdot y_k, \qquad k = 0, 1, \ldots, M-1, \tag{4.1}$$

where x_k, y_k are the samples in the x-y plane, and i is the complex operator.

In figure 4.8 the shape of 4 different cells from the two different classes (pseudopodia cells, and circular cells) are shown. In the figure it is seen that if just the x, y coordinates are used as a description of the shape, there are a lot of differences from one cell belonging to a class to the next, both in the starting point (shown with a '+' at the figure) and the number of samples around the shape. For the four cells the number of samples are 89, 130, 59 and 61. If all the samples should be used as inputs to a classification system (a neural network), it will give a lot of problems both due to varying number of inputs but also due to the correlation in all these inputs

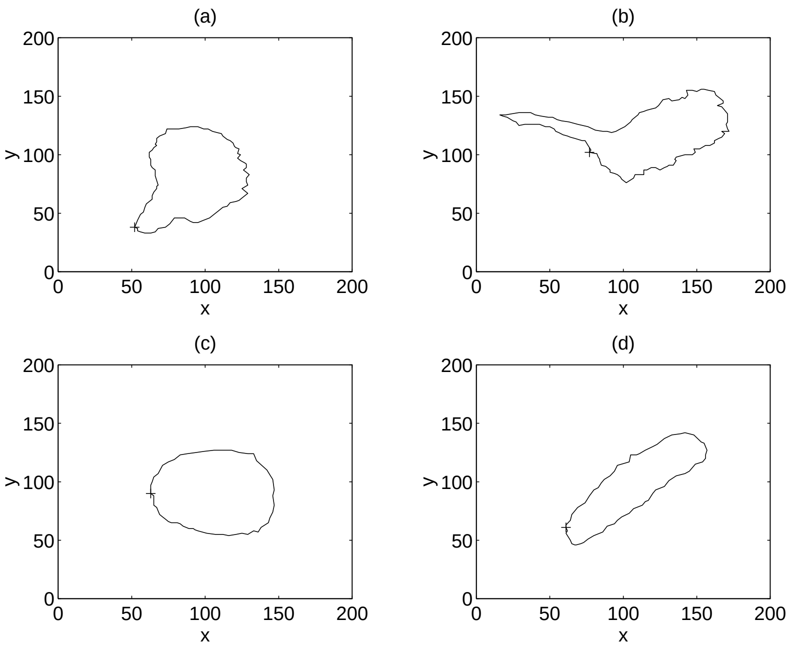


Figure 4.8: Plot of the contour of 4 different cells, 2 pseudopodia cells (a) and (b), and 2 circular cells (c) and (d). They are varying in the number of samples around the contour, with the starting point of the coordinate system, with the size, with the orientation, and with the starting point for the sampling. The (+) is marking the starting point for the sampling sequence.

Instead a Fourier description of the shape is used. It has the feature that the sample of the shape is easily converted to a description which is invariant to the transformations described above. In the Fourier technique the shape coordinates are transformed by the **Fourier transformation**. The Fourier coefficients F_n are calculated as

$$F_n = \sum_{k=0}^{M-1} z_k e^{-i2\pi kn/M}, \qquad n = 0, 1, \ldots, M-1, \tag{4.2}$$

where z_k is the complex coordinates of the contour of the shape as described by equation (4.1).

The cell shape is then described by a set of Fourier components as

$$\mathbf{F} = [F_0, F_1, F_2, \ldots, F_{M-2}, F_{M-1}] \tag{4.3}$$

The Fourier components describe the contour of shape as well as the coordinates of the shape, but as should be seen in some of the following sections they are easier to simplify and normalize. The Fourier transformation is a linear transformation and it is possible to transform the Fourier description of the shape back to the x-y coordinates for the shape. This is done by the **inverse Fourier transformation**.

$$z_k = \frac{1}{M} \sum_{n=0}^{M-1} F_n e^{-i2\pi nk/M}, \qquad k = 0, 1, \ldots, M-1 \tag{4.4}$$

In the following subsections the advantages of using the Fourier components as descriptors for a contour of a shape will be described. It is assumed that the surface of a shape is sampled with sufficiently high sampling rate to be represented well by the samples, so the sampling theorem is fulfilled.

Translation invariance

Translation invariance is very easy to obtain within the Fourier description. The zero'th component

$$F_0 = \frac{1}{M} \sum_{k=0}^{M-1} z_k, \tag{4.5}$$

is the center of mass for the samples of the x-y coordinates. Therefore setting the zero'th component to zero and using the inverse Fourier transformation (4.4), the shape is transformed to a standard reference system. The Fourier descriptors in the translation invariant form are

$$\mathbf{F}_{\text{trans.inv.}} = [0, F_1, F_2, \ldots, F_{M-2}, F_{M-1}] \tag{4.6}$$

where the rest of the Fourier components ($k \neq 0$) are unchanged from equation (4.3).

In figure 4.9 the translation invariant description of the shapes are shown, together with the original shapes.

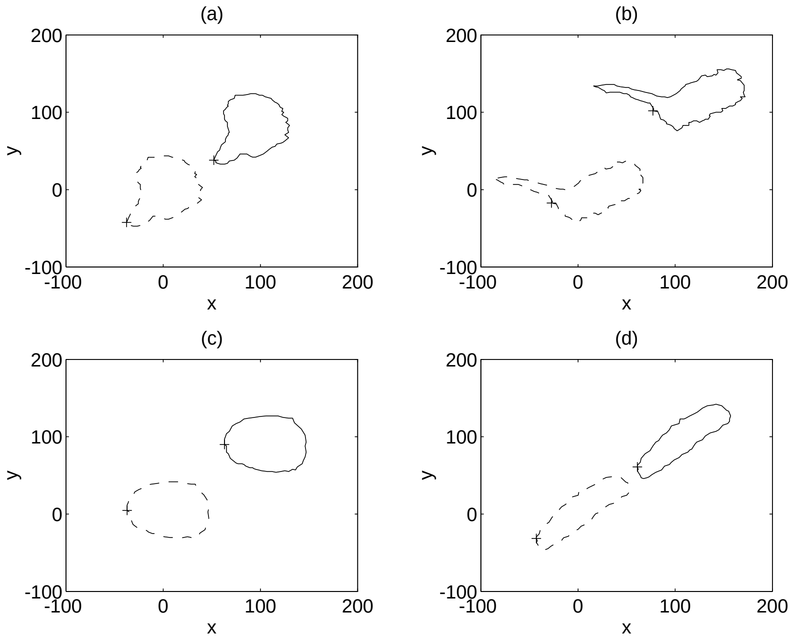


Figure 4.9: Plot of the contour of 4 different cells, 2 pseudopodia cells (a) and (b), and 2 circular cells (c) and (d) (same as shown in figure 4.8). The shapes are transformed to be translation invariant by setting the zero'th Fourier component to zero. The signature is: solid line - original shape, dashed line - translation invariant shape (the zero'th component is set to zero).

Invariance in the number of descriptors for a shape

As mentioned before the number of samples around the contour of a shape is often varying. It is therefore very difficult to use the x-y coordinates as standardized inputs to a neural network. The number of inputs has to be constant for all the patterns that should be classified. The Fourier representation makes it possible to standardize this by using the most significant components[2] as descriptors for the shape. This will be illustrated by the shapes shown in figure 4.9. These shapes are first Fourier transformed, and made translational invariant by setting the zero'th component to zero. They are then described by a set of Fourier components as shown in equation (4.6).

The most significant components for the shape contour are the components F_1 and F_{M-1}. When only these two components are used (the rest are set to zero) it can be seen that the inverse Fourier transformation is described by

$$
\begin{aligned}
z_k &= \left(F_1 e^{-i2\pi k/M} + F_{M-1} e^{-i2\pi(M-1)k/M}\right) \\
&= \left(F_1 e^{-i2\pi k/M} + F_{M-1} e^{i2\pi k/M}\right), \qquad k = 0, 1, \ldots, M-1, \qquad (4.7)
\end{aligned}
$$

[2]The most significant components in the Fourier description of a shape are meant to be the components which give most information about the contour of the shape.

corresponding to samples of an ellipse. This is in the following called the major ellipse for the shape.

The components F_2 and F_{M-2} are also giving an ellipse, which is rotating two times, when k is varying from 0 to $M-1$. Further the calculation of z_k is a sum of the different contributions, so this ellipse can be regarded as an extra ellipse, which is rotating at the periphery of the major ellipse. The next components are then describing an ellipse, which is turning three times when k varies. This goes on for all the Fourier components [Kuhl and Giardina, 1982]. The most significant Fourier components for the shape are therefore the components giving the major ellipse, the ellipse rotating at the periphery of this, and so on.

A method for reducing the number of descriptors for the shape and making it invariant is therefore to set some of the "lower" Fourier components to zero. This corresponds to low pass filtering the x, y coordinates, or removing noise from the data. This is described in equation (4.8), where only the k most significant components are used.

$$\mathbf{F}_{\text{no.inv.}} = [0, \overbrace{F_1, \overbrace{F_2, \ldots, \underbrace{F_k, 0, \ldots, 0, F_{M-k}}_{k'th\ components}, \ldots, F_{M-2}}^{2'nd\ components}, F_{M-1}}^{1'st\ components}] \tag{4.8}$$

The zero'th component is also set to zero, while it already is seen that it only makes the shape translation invariant.

In figure 4.10 the shapes are shown for a varying number of Fourier components. From this figure it can be seen that using the first components (the major ellipse) gives a fine description for simple shapes. Using a couple of components more gives also for more "advanced" shapes a fine description as can be seen for the pseudopodia cells in the figure using five components. Further it can be concluded from the figure that a way of standardizing the number of descriptors for the shape is found by using the Fourier components as descriptors for the shape. Using the most significant Fourier components for the shapes, an invariant number of descriptors for the different shapes can be found, although the contour is sampled by a different number of x-y coordinates.

Scale invariance

Often the form of the shape and not the size should enter the classification of the shape, therefore it might be necessary to normalize the image after the size. For this purpose the radius of the first harmonic circle [Kuhl and Giardina, 1982] can be used. The radius of the first harmonic circle (based on the major ellipse) is found as

$$K_{\text{scale}} = \sqrt{\|F_1\|^2 + \|F_{M-1}\|^2}, \tag{4.9}$$

where F_1 and F_{M-1} are the ordinary Fourier coefficients.

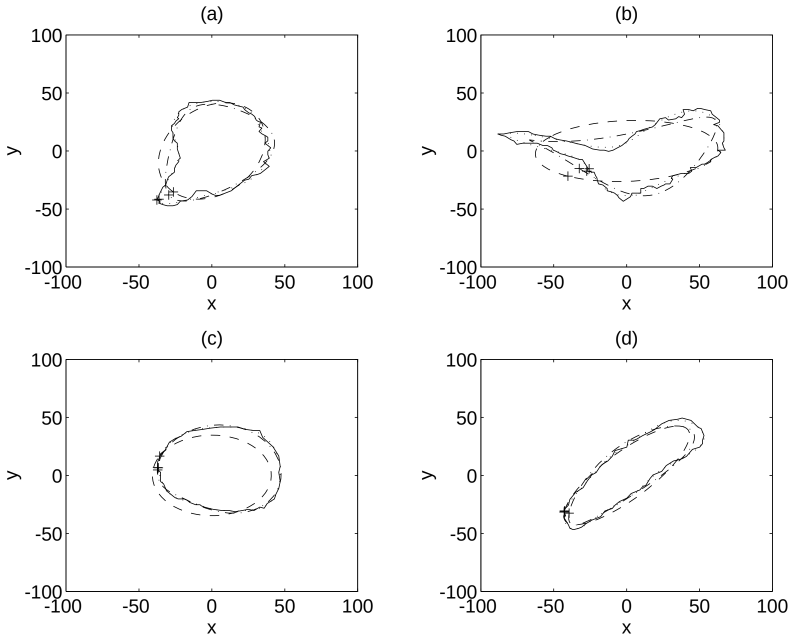


Figure 4.10: Plot of the contour of 4 different cells, 2 pseudopodia cells (a) and (b), and 2 circular cells (c) and (d) (same as shown in figure 4.8). The signature is: solid line - original shape, dashed line - shape using only 1 component (k=1) (major ellipse), dashed-dotted line - shape using 2 components (k=2), dotted line - shape using 5 components (k=5).

Normalization of the Fourier coefficients is then done by dividing all the used coefficients by this normalization coefficient as

$$F_{n,\text{scale.inv.}} = F_n / K_{\text{scale}}, \qquad n = 0, 1, \ldots, M-1. \tag{4.10}$$

In figure 4.11 the 4 cells are scale normalized using this method.

Rotation invariant

The shapes can still be rotated as can be seen in figure 4.11, where the two contours of the circular cells obviously are rotated if compared with each other, especially this could be seen if the major ellipses are compared. The phase of the major ellipse is therefore used to normalize the rotation of the shapes. This is done by a factor calculated as

$$K_{\text{rotate}} = e^{-i((\text{phase}(F_1)+\text{phase}(F_{M-1}))/2)} \tag{4.11}$$

where phase is the complex phase of the Fourier coefficient.

Normalization is done by multiplying all the Fourier coefficients by this coefficient as

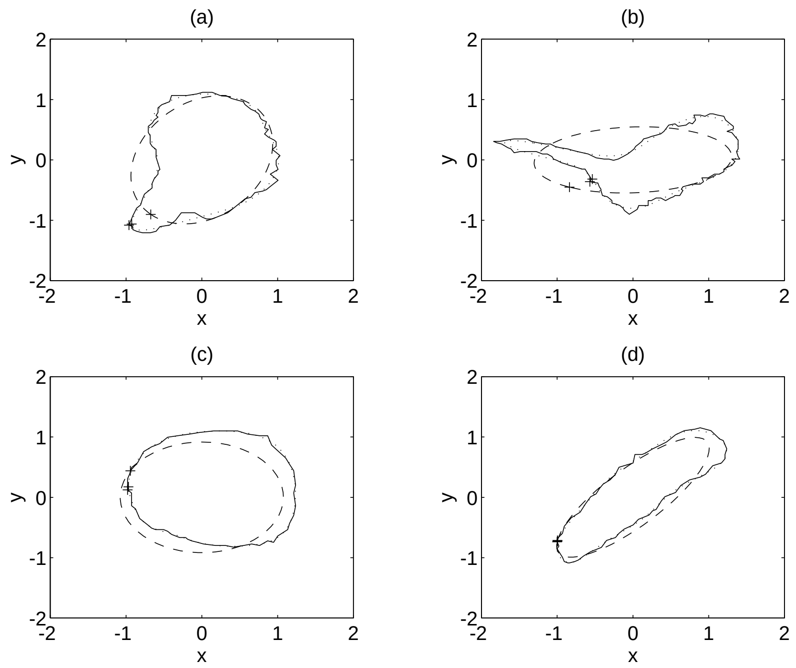


Figure 4.11: Plot of the contour of 4 different cells, translation and scale normalized, 2 pseudopodia cells (a) and (b), and 2 circular cells (c) and (d) (same as shown in figure 4.8). The signature is: solid line - original shape, dashed line - shape using only 1 component (k=1) (major ellipse), dotted line - shape using 5 components (k=5).

$$F_{n,\text{rotate.inv.}} = F_n \cdot K_{\text{rotate}}, \qquad n = 0, 1, \ldots, M-1. \tag{4.12}$$

In figure 4.12 the rotation invariant cells are sketched, the major axis in the major ellipse is the x-axis in the plot. It is seen that there still are some problems with the pseudopodia cells ((c) and (d)), where the pseudopodias are pointing in two different directions. In a later subsection a method that helps on this problem will be proposed; no fully satisfactory solution has been found for the problem.

Invariance for the starting point in the sampling sequence

As can be seen in figure 4.12 there is also a need for making the description invariant to the choice of starting point for the sampling sequence, This can be done by e.g. placing the first sampling point on the positive x-axis. This can be done by using a transformation of the Fourier coefficients as

$$F_{n,\text{inv}} = \begin{cases} F_n e^{i v_{\text{inv}} n} & n <= (M-1)/2 \\ F_n e^{i v_{\text{inv}} (n-M)} & n > (M-1)/2 \end{cases} \tag{4.13}$$

It is still the major ellipse which is used when normalizing the Fourier coefficients.

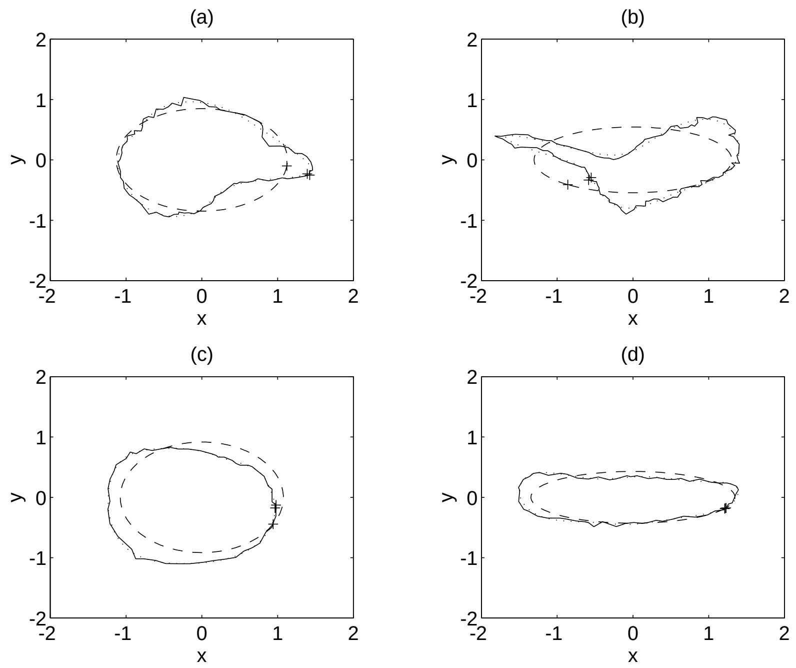


Figure 4.12: Plot of the contour of 4 different cells, translation, scale and rotation normalized, 2 pseudopodia cells (a) and (b), and 2 circular cells (c) and (d) (same as shown in figure 4.8). The signature is: solid line - original shape, dashed line - shape using only 1 component (k=1) (major ellipse), dotted line - shape using 5 components (k=5).

The angle v_{inv} which is used, when normalizing the Fourier coefficients, can be calculated by using the equation for the coordinates of the major ellipse. It is given as

$$z_{k,\text{inv}} = \frac{1}{M}\left(F_1 e^{iv_{\text{inv}}} e^{-i2\pi k/M} + F_{M-1} e^{-iv_{\text{inv}}} e^{i2\pi k/M}\right) \tag{4.14}$$

The purpose of the transformation was that the first point should be placed on the x-axis. This means that the imaginary part of the first coordinate ($k = 0$) should be zero. It is therefore possible to calculate v_{inv} using the following equation

$$\begin{aligned}
\text{imag}(z_{0,\text{inv}}) &= \frac{1}{M}\text{imag}\left(F_1 e^{iv_{\text{inv}}} + F_{M-1} e^{-iv_{\text{inv}}}\right) = 0 \\
&\Updownarrow \\
&\quad \text{real}(F_1)\sin(v_{\text{inv}}) + \text{imag}(F_1)\cos(v_{\text{inv}}) + \\
&\quad \text{real}(F_{M-1})\sin(-v_{\text{inv}}) + \text{imag}(F_{M-1})\cos(-v_{\text{inv}}) = 0 \\
&\Updownarrow \\
&\quad (\text{real}(F_1) - \text{real}(F_{M-1}))\sin(v_{\text{inv}}) = \\
&\quad -(\text{imag}(F_1) + \text{imag}(F_{M-1}))\cos(v_{\text{inv}}) \\
&\Updownarrow
\end{aligned}$$

$$\frac{\sin(v_{\text{inv}})}{\cos(v_{\text{inv}})} = -\frac{(\text{imag}(F_1) + \text{imag}(F_{M-1}))}{(\text{real}(F_1) - \text{real}(F_{M-1}))}$$

$$\Updownarrow$$

$$v_{\text{inv}} = -\arctan\left(\frac{\text{imag}(F_1) + \text{imag}(F_{M-1})}{\text{real}(F_1) - \text{real}(F_{M-1})}\right) \tag{4.15}$$

Using this angle v_{inv} in equation 4.13 makes it possible to calculate a new set of Fourier coefficients describing the ellipse with the starting point $z_{0,\text{inv}}$ placed on the x-axis.

In figure 4.13 the transformed cell shapes are shown. It can be seen that it is only the starting point for the sampling sequence that is changed compared to figure 4.12. This was also the purpose for the transformation.

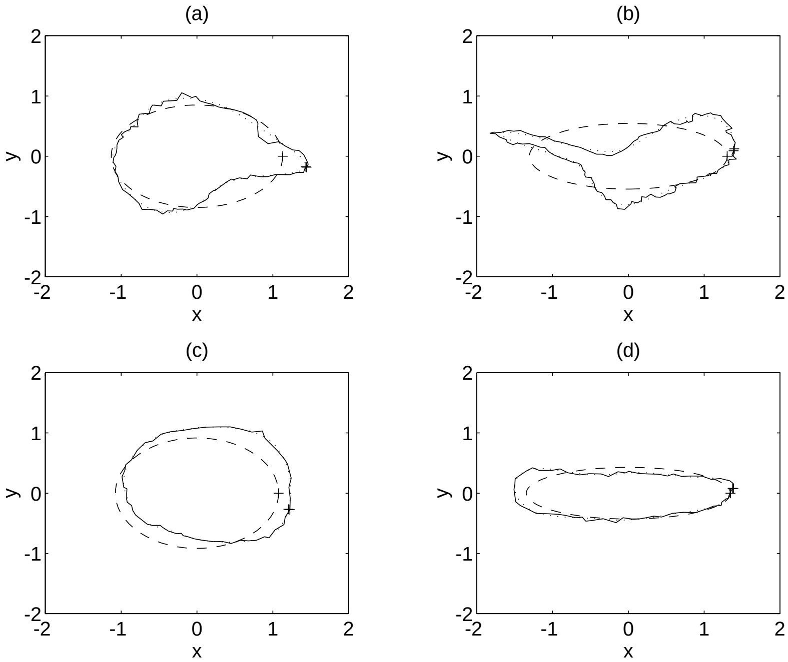


Figure 4.13: Plot of the contour of 4 different cells, translation, scale, rotation and starting point normalized, 2 pseudopodia cells (a) and (b), and 2 circular cells (c) and (d) (same as shown in figure 4.8). The signature is: solid line - original shape, dashed line - shape using only 1 component (k=1) (major ellipse), dotted line - shape using 5 components (k=5).

From the figures it can be seen that there still are some problems with the invariant description of the shapes. The pseudopodia cells (a) and (b) are normalized with its translation, size, rotation of the major ellipse, and starting point, but still it can be seen that the pseudopodia for the two cells go in two different directions. It is due to the problem that the major ellipse is normalized to have its major axis at the x-axis, but the information about the pseudopodias is coded in the higher order Fourier components, which has not been used during normalization. No method for using these components during normalizing has been found, but some of the problems seem to be solved by the method mentioned in the following section, where the Fourier components are changed to a magnitude-phase description.

Real/imaginary part description versus magnitude/phase description of the Fourier components

In the previous section it has been mentioned that though all these normalizations have taken place, it can be seen from the images that there still are problems e.g. with the placement of the pseudopodias. One possibility for solving this in a classification system is to let the classifier take care of the problem.

Instead, a description where the Fourier components are translated as shown in the following equation can be used.

$$F_n = F_{n,x} + iF_{n,y} = r_n e^{i\phi_n} \tag{4.16}$$

where r_n is called the magnitude of the Fourier coefficient and ϕ_n is the phase of the coefficient. $F_{n,x}$ and $F_{n,y}$ is real/imaginary part description of the Fourier coefficients.

The magnitudes of the Fourier coefficients are invariant for the placement of e.g. the pseudopodias and the transformation done has therefore been able to give an invariant set of descriptors. It is in the phase part of the coefficients that the information of e.g. the placement of pseudopodias is coded, and it is this part of the coefficients which still isn't invariant. This problem can't be solved without finding a method for using the rest of the Fourier coefficients for normalizing the placement of the pseudopodias (rotating the major axis in the major ellipse).

Especially for this problem, where the cells have to be classified for the size of the pseudopodias and not the placements, this doesn't seem to be a problem, because it is in the invariant set of magnitudes parts of the Fourier components that the information about the size of the pseudopodias is coded. For other cell classification problems this could be a more serious problem e.g. a problem as classifying dividing cells from non-dividing cells. Some examples of this are shown in figure 4.14, where it is the placement of thin part of the cells which gives the information about the division of the cell. Fortunately this information is not coded in the absolute value of the phases for the Fourier components, but in the differences between phases of different components. This information about the difference in the phases has not been spoiled by the missing invariance in the phases.

If the real/imaginary part description is used instead it would have been more difficult to claim that these transformations have made an invariant description of the cells, because the invariant information from the phases of the pseudopodias has been coded in the real/imaginary parts of the Fourier components through a non-linear cosine respectively sine function.

4.2.2 Human sperm cells

Another cell classification task where the methods have been tested is classification of human sperm cells. This work has been carried out in a collaboration with E. Bostofte, Hvidovre Hospital and the fertility clinic at the Clinic of the medical doctors in Copenhagen, A. Giwercman and N. Jørgensen from the department of growth and reproduction at the University Hospital in Copenhagen, and C. Linneberg and J. Meyrowitsch from

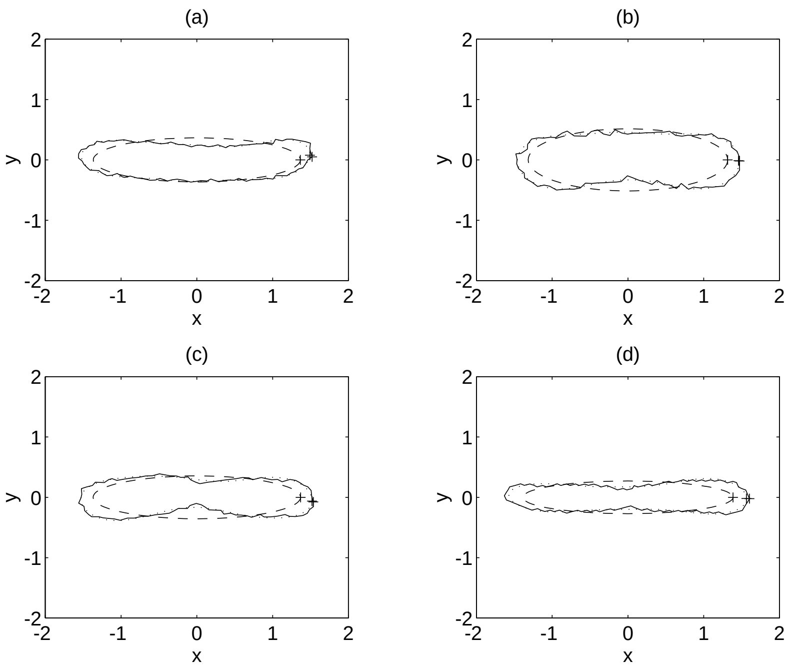


Figure 4.14: Plot of the contour of 4 different cells, translation, scale, rotation and starting point normalized, 2 non-dividing cells (a) and (b), and 2 dividing cells (c) and (d). The signature is: solid line - original shape, dashed line - shape using only 1 component (k=1) (major ellipse), dotted line - shape using 5 components (k=5).

DynaVision. The two medical clinics have delivered the samples and some laboratory technicians from the clinics have made the classification of the samples which have been used to train the networks. At DynaVision the images have been digitalized and the sperm cells have been sampled. In appendix G an article presented at the *1994 IEEE Workshop on Neural Networks for Signal Processing* is appended. The experiments in the article are done by C. Linneberg.

The purpose for classification of the sperm cells is that recent research has shown that the semen quality has been halved during the past 50 years. To find the factors that influence the semen quality, it is necessary to have reliable methods for classification of the sperm cells. Today the classification is done manually by laboratory technicians. This gives a significant inter-laboratory and inter-personal variation. It has been attempted to have the same samples classified into two classes (normal and non-normal cells) by technicians from different laboratories and by different technicians from the same laboratory. Further it has been attempted to have the same samples classified by the same technician 4 times. In the following tables the results are shown.

It can from these tables be seen that there is a lot of variation between technicians from different laboratories and between technicians from the same laboratory. Even for the same technician when the classification is done more than once the variation is significant, only 88% agreement with the consensus is obtained.

To minimize this large inter-laboratory and inter-personal variation it is very interesting

Laboratory	Percentage normal cells (60 cells)
Fertility Clinic	0.60 +/- 0.028
Growth and Rep. lab	0.46

Table 4.1: Inter-laboratory variation. Result of having the same 60 samples classified by 5 technicians at the Fertility clinic and 1 technician at the Growth and Rep. laboratory. It can clearly be seen that the technicians at the two laboratories don't use the same level for classification into normal/unnormal (the technician from Growth and Rep. lab is clearly out of the range 0.60 +/- 0.028 from the other laboratory).

Laboratory	Percentage versus consensus (60 cells)
Fertility Clinic	0.85 +/- 0.079

Table 4.2: Inter-technician variation. Results of having the same 60 samples classified by 5 technicians from the same laboratory. The results from each technician are compared with the consensus of the the 5 technicians. It is seen that the variation in the classification between the different technicians is quite large.

to study automatic methods for classification of the sperm cells. Further the time spent by the technicians when they classify the samples is rather large and could presumedly be reduced.

From the images of the semen samples the periphery of the sperm cells is sampled. Then the methods described in the previous sections for making the Fourier descriptors invariant are used for preprocessing the data. The five most significant Fourier components are used as input to the neural network classifier. The Fourier descriptors are described by their magnitudes and their phases. Further a descriptor for the percentage of the cell, which is the acrosome, is used as input to the neural network classifier. In figure 4.15 an image of a semen sample is shown. At the image the periphery and the line that divides the cells into acrosome and nucleus are shown. In figure 4.16 the same samples are shown together with the classification of each sperm cell. It can from these images be seen that it is very difficult to find the difference between the two classes by the human eye.

The neural network used to classify the samples is an ordinary feed-forward neural network

Laboratory	Percentage versus consensus (185 cells)
Fertility Clinic	0.88 +/- 0.15

Table 4.3: Variation for same technician. Results of having the same technician classify 160 samples 4 times. The results are compared with the consensus of the classification. The variation is quite large. The data set used is not the same as used when generating the results for table 4.1 and 4.2.

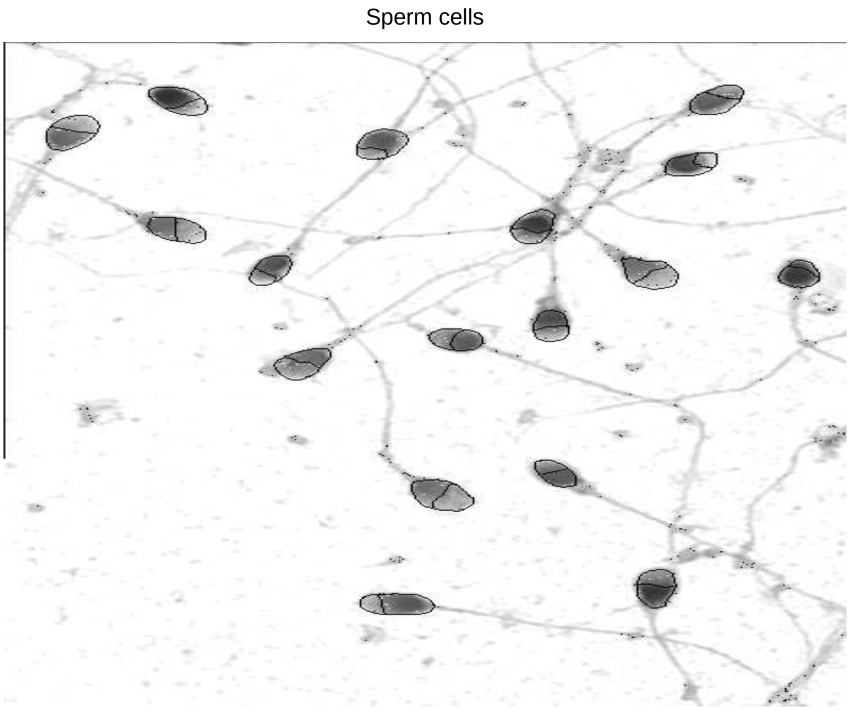


Figure 4.15: Image showing sperm cell sample. The contour of the cell and the line dividing the cell into the acrosome and nucleus part are drawn.

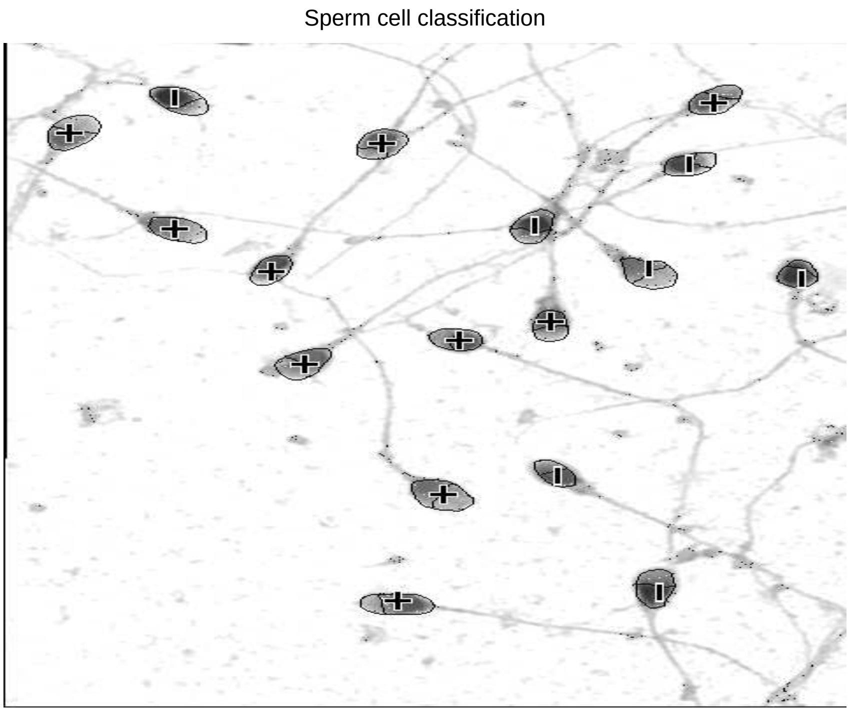


Figure 4.16: Image showing sperm cells, the classification is shown with +/− for normal/non-normal cells.

as described in section 2.1.3 and it is trained using the algorithms described in section 2.2 and 2.4. Further it is pruned using the methods described in section 3.1.3. There are 327 examples in the data set and 218 of these are randomly chosen for training the neural network while 109 examples are spared for testing. The initial neural network configuration is a fully connected neural network with 19 inputs (10 Fourier-component magnitudes, 8 Fourier-component phases and the placement of the line dividing the cell in nucleus and acrosome) 8 hidden units and 1 output unit. In figure 4.17 the normalized quadratic error is shown for a typical case during pruning. It can be seen that while parameters are removed from the neural network the learning error is increasing, but as should be expected the test error (generalization error) is decreasing. Further figure 4.18 shows that the classification error at the learning set is increasing while the test error is decreasing while pruning. The minimum of the classification error at the test set is 22%, where the classification error by the same technician has been appr. 12% (table 4.3) and between technicians appr. 15% (table 4.2).

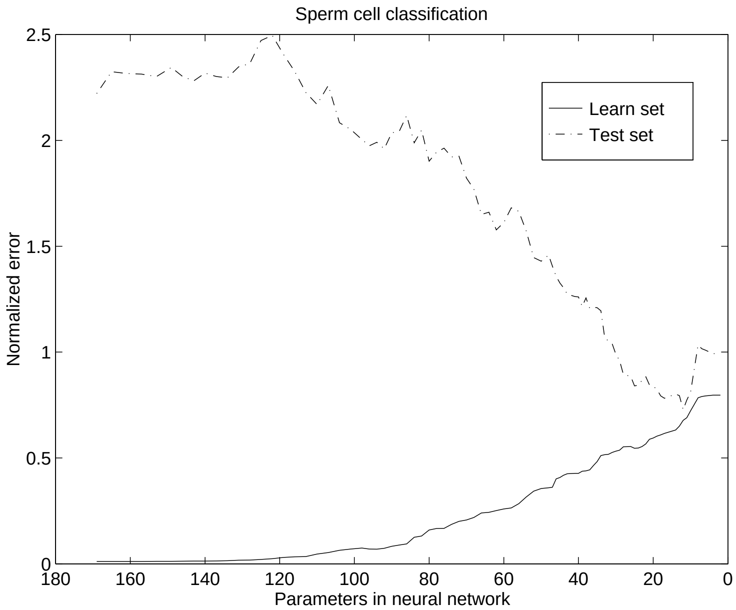


Figure 4.17: The normalized error (sum of the squared error of the learning and test set normalized by the variance of the learning set) for the sperm cell classification problem is shown during a pruning session. 218 examples are used for learning and 109 for testing.

No method but cross-validation has yet been found for estimating a stop criterion for the pruning algorithm in this case with classification of data sets with noise. The problem in using cross-validation is that it computationally is very expensive, for every combination of training sets (with a few examples drawn from the training set for validation) a new network has to be trained, and the validation error has to be calculated at the spared data set. The test error is then estimated as a mean of the error at the data sets that have not been used when training the network. Instead it is chosen to spare a larger part of the data set for validation and for choosing the optimal network configuration.

It can be seen from the curves that the data set used throughout the experiments is in

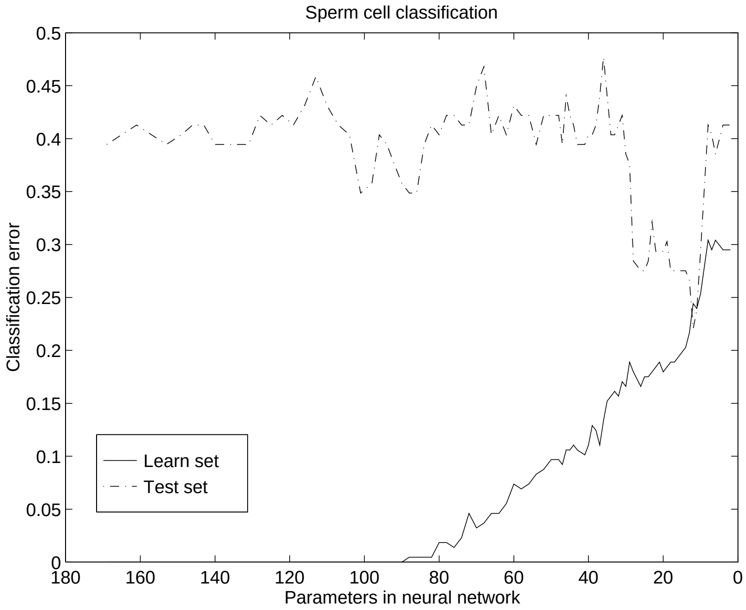


Figure 4.18: The classification error for the sperm cell classification problem is shown during a pruning session. The same pruning session as shown in figure 4.17.

the lower limit of what is needed. The training error is increasing very much before the minima of the test error is found. If a larger training set is used it should be expected that the difference between the test error and the learning error for a fully connected network is a lot smaller and the learning error doesn't increase so much before the test error starts to decrease.

The optimal network configuration which is found using the test set as validation set (shown in figure 4.17) is shown in figure 4.19. It can be seen that some of the most significant Fourier-components magnitude are used (input x(1), x(6) and x(9)) and only one phase is used (input x(15)). Further it can be seen that the dividing line between acrosome and nucleus is used (input x(19)). This is not very surprising since all the technicians have said that this is a significant descriptor for the cells. It shall here be mentioned that to test if this network is the optimal in general it is necessary to have another data set, which neither has been used for training or validation (selection of the optimal network structure).

From this experiment it can be seen that the pruning method is improving the neural network's generalization ability for this classification task. It can help the technicians in screening samples which easily can be classified so the technicians only have to manually classify difficult and extreme samples. The experiments also show that more refined preprocessing techniques have to be used. It has been tried to show the technicians the reconstructed sperm cells where only the shape and the placement of the dividing line between acrosome and nucleus has been drawn. It has been almost impossible for the technicians to classify the cells. This indicates that they use other features than these when they do the manual classification. Methods with information about the interior of the cell and about the tails will be helpful for the classification task. The method of using

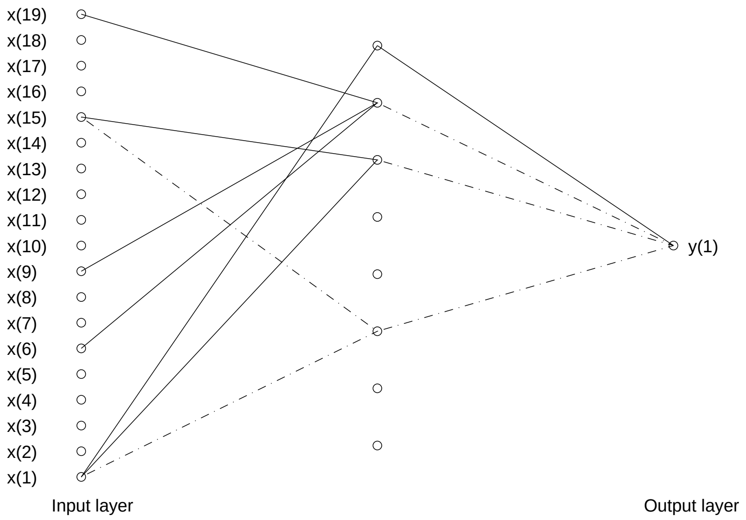


Figure 4.19: The optimal net for the sperm cell classification problem is shown. It can be seen that only a subset of the input space is used to solve the problem. Input 1-10 is the magnitude of the Fourier components, while 11-18 is the phases. Input 19 is the placement of the dividing line between the acrosome and the rest of the cell. At the figure it can be seen that almost none of the phases are used to solve the classification problem.

neural network techniques is though very promising for the classification of this kind of samples.

Chapter 5

Examples: Continuous output systems

Another important application area where neural networks can be used is within analysis of non-linear time-series. In the book [Weigend and Gershenfeld, 1994] many different application areas for neural networks, from forecasting the sunspot activity to analyzing the stock exchange market, are discussed. The ultimate tool for analyzing time-series must be fully recurrent neural network structures, while large time constants often is present in time-series. This is also known from traditional signal analysis, where in Finite Impulse Response (FIR) models there are only forward connections, while in Infinite Impulse Response (IIR) models there are recurrent connections. The IIR models can often, with fewer parameters, build a satisfactory model for time-series with long impulse responses. The problem is that no theory has been given for analyzing the performance, especially the generalization ability for recurrent neural networks. Therefore the most used neural network structure for signal processing tasks is still the feed-forward network type. The inputs to the neural network models are prior samples from the time-series and eventually other signals which are believed to influence the time-series.

In this chapter examples where neural networks have been used for short time prediction of chaotic time-series will be shown. Off course long time forecasts can be ruled out if the systems are chaotic. One problem that recently has been established as a benchmark problem for short time estimation of chaotic time-series is the sunspot problem [Weigend et al., 1990]. Another problem which has been used for testing the short time prediction performance of non-linear models is the Mackey-Glass chaotic time-series. In this chapter examples of using pruned two layer feed-forward network for prediction of the time-series will be shown. Further an example used in [Day et al., 1992] of estimation of the inverse transfer function of a transmission channel will be introduced.

5.1 Sunspot time-series

Prediction of the sunspot series has been used as a benchmark problem for time-series prediction algorithms by many different people as discussed in [Priestly, 1988] and [Weigend et al., 1990]. In the article [Svarer et al., 1993a] appended in appendix C, this problem has been used to show the performance of the pruning and training algorithms used throughout this Ph.D study. In the article [Weigend et al., 1990] the sunspot series has been divided into a training set (1700-1920) and two test sets (1921-1955) and (1956-1979). The reason why two test sets have been defined is that the variance of the observations of the sunspot series seems to change dramatically around 1955. This must be due to an actual change in the activity while the registration method has been checked carefully.

The sunspot series is a record of the yearly average sunspot activity (an activity related to the magnetic fields at the surface of the sun) for the period 1700-1979. It is believed that the activity has some influence on the weather on the earth due to results reported by some meteorologists at DMI (The danish institute of meteorology). The results have been shown in a news program on danish television. The sunspot activity is believed to be generated by a noisy, chaotic, dynamic system. Before analyzing the time-series the measured activity level has been rescaled to a maximum value of 1. An analysis of the frequence spectrum of the series shows that it is dominated by a frequency corresponding to a 12 year period. At figure 5.5 the time-series is shown. The 12 year period of the time-series is clearly seen at the figure.

In this section the major results reported in the article will be emphasized, together with some of the figures.

5.1.1 Training of neural network

The neural network is trained using the cost-function with a weight-decay term. As weight-decay parameters are used $\alpha_w = 0.02$ (input-hidden) and $\alpha_W = 0.01$ (hidden-output), with *tanh* activation functions at the hidden layer units and a linear output unit. The training is started with a fully connected neural network with 8 hidden units and a lag space of 12 values as input. By a lag space of 12 is meant the sunspot activity the last 12 years, which means that the input vector can be written as $\mathbf{x}(k) = [x(k-1), x(k-2), \ldots, x(k-12)]^T$ and the signal that should be predicted by the model is then the sunspot activity this year $x(k)$. In figure 5.1 the training of the fully connected network is shown. It can at this figure be seen that overfitting of the network to the learning set is present. The learning error is decreasing throughout the training session, while both test errors start to increase after 100 training epochs. This means that the weight-decay scheme with the parameter values used is not enough to overcome the overfitting problem. It would have been possible to increase the weight-decay parameters, which has also been tried, but when increasing the weight-decay parameters the quadratic error term in the cost-function increases. This is an undesirable effect, as it is this part of the cost-function that is especially interesting and should be minimized as much as possible. Therefore a pruning scheme is used to remove superfluous parameters from the network architecture.

A network pruning session is therefore carried out where 2% of the remaining weights are removed per retraining round. In figure 5.2 the result of the pruning session is shown. The FPE estimate of the test error (see appendix A) is used to point out which network structure is the optimal. It can be seen that the network structure pointed out by the method also is the network structure with the lowest test errors, both at test set (1921-

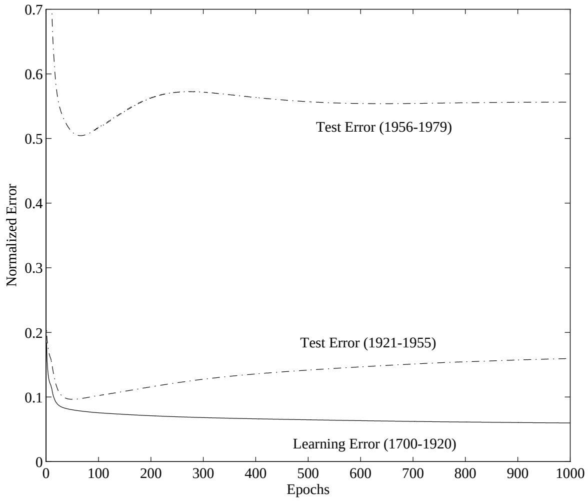


Figure 5.1: Sunspot time-series. Evolution of the quadratic training and test error functions, when training the fully connected network with a cost-function with weight-decay ($\alpha_w = 0.02$, input-hidden and $\alpha_W = 0.01$, hidden-output). An 'Epoch' is a full sweep through the training set.

1955) and at the test set (1956-1979). In figure 5.3 this optimal network structure is shown. It can be seen that only 3 hidden units are used and only a part of the input lag space is used (5 inputs).

It has been tried to train an ensemble of 11 networks to test how well the algorithms perform. From the original 11 networks, 2 of the network pruning sessions stop with a network with over 25 weights as the optimal network structure selected on the FPE criteria. The structure of these 2 networks are very different from the other 9 networks. In general it is believed that the smallest networks have the best generalization ability [Thodberg, 1991], therefore these 2 networks have been sorted out. The 9 other networks end up with a network structure with 12-16 weights, in the table 5.1 their performance is listed.

5.1.2 Retraining of optimal network

It is then tried to retrain the optimal network structure to see if it is possible to overcome the problem of overtraining when the optimal network structure already has been found. In figure 5.4 the retraining of the pruned network structure is shown. It can there clearly be seen that the problem with overfitting the network parameters to the learning set is overcome, by only having the network parameters present in the network structure that is needed to find a good solution for the problem. Both the learning and the two test errors are decreasing throughout the learning session.

In figure 5.5 the optimal retrained neural network is used for prediction of the sunspot series. It can can be seen that the neural network predicts the sunspot series very well

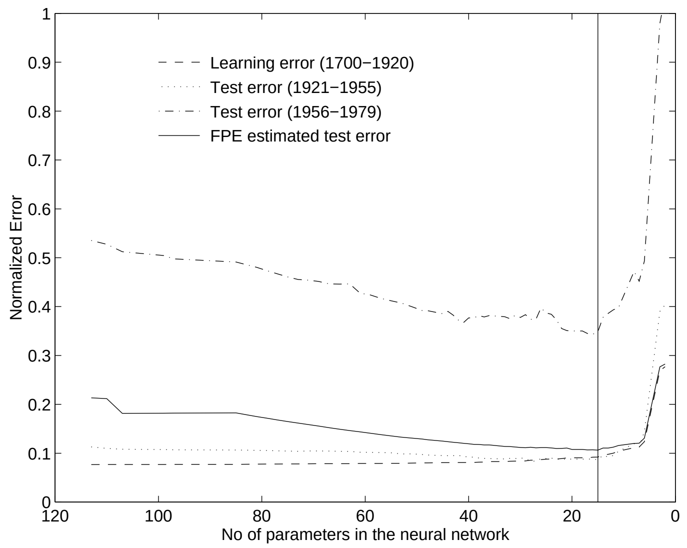


Figure 5.2: The evolution of training and test errors during pruning (2% of the remaining weights are removed in every pruning iteration). The FPE estimate of the test error is shown at the figure, this estimate is used to find out when to stop pruning. The vertical line shows the optimal network structure pointed out by the FPE estimate of the test error. It can be seen that this network structure for both test sets also is the network with the lowest test error.

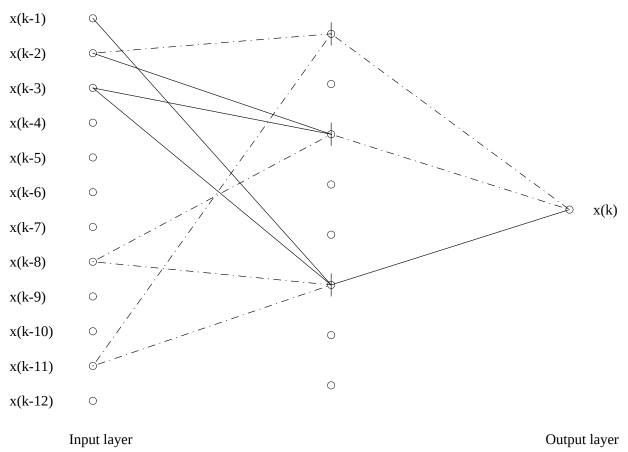


Figure 5.3: Optimal neural network for prediction of the sunspot series. The network has only 15 weights and 3 hidden units. It uses 5 of the original 12 inputs to the network. Solid lines show positive weights, while dot-dashed line show negative weights. A vertical bar through a unit indicates that a threshold is present for that unit.

until 1955, where the variance of the time-series seems to change.

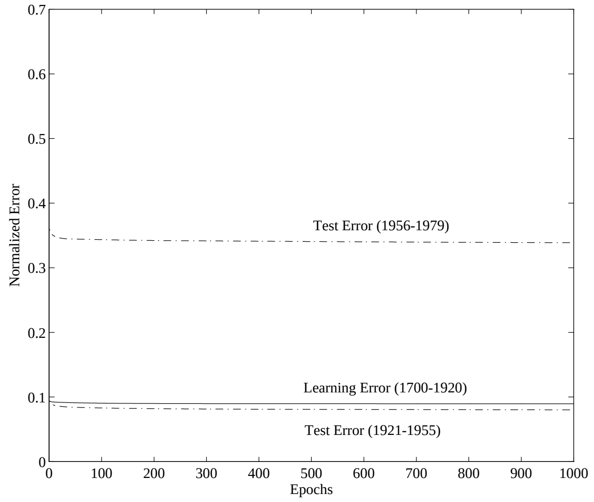


Figure 5.4: Training and test error evolution, when retraining the pruned network shown in figure 5.3. The cost-function used has no weight-decay ($\alpha_w = 0$, input-hidden and $\alpha_W = 0$, hidden-output). An 'Epoch' is a full sweep through the training set.

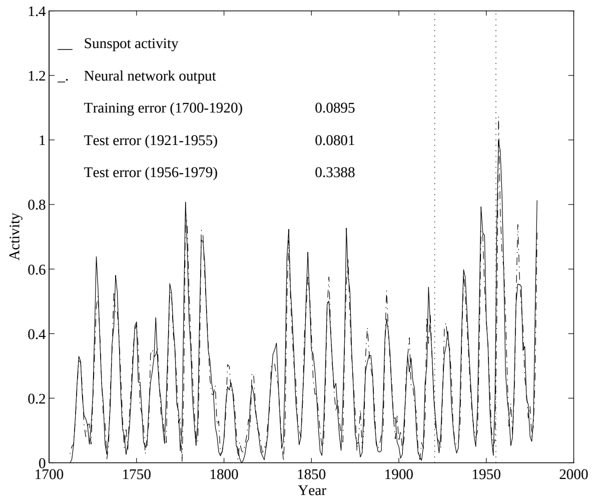


Figure 5.5: Sunspot activity shown as the solid line. The dot-dashed line shows the sunspot activity predicted by the pruned neural network shown in figure 5.3.

In table 5.1 the results obtained by using the neural network architecture optimization algorithms are compared to the results described in the literature and to the performance of a linear model. It can from this be seen that the pruning of the feed-forward neural networks is doing very well compared to the performance of especially the linear model, but also the other network structures proposed for solving this time-series estimation problem.

NORMALIZED ERROR

Model	Train (1700-1920)	Test (1921-55)	Test (1956-79)	Number of parameters
Tong and Lim [Tong and Lim, 1980]	0.097	0.097	0.28	16
Weigend *et al.* [Weigend et al., 1990]	0.082	0.086	0.35	43
Linear model[1]	0.132	0.130	0.37	13
Fully connected network[2]	0.078 ± 0.002	0.104 ± 0.005	0.46 ± 0.07	113
Pruned network[3]	0.090 ± 0.001	0.082 ± 0.007	0.35 ± 0.05	$12 - 16$

Table 5.1: 1) The linear model is a single linear unit. 2) The initial pre-pruned networks, trained with the same weight-decay terms as used during pruning. 3) Pruned networks retrained without weight-decay. The mean and standard deviation are based on the networks selected for retraining (9 of the initial set of networks 11). The normalized error is the sum of squared errors normalized by the variance of the total data set (learning and test sets).

5.2 Mackey-Glass time-series

Another example which has been extensively used as a bench-mark test for the performance of neural networks within time-series prediction, is the Mackey-Glass time-series. This time-series is a nonlinear chaotic time-series given by the following differential equation

$$\frac{dz(t)}{dt} = -bz(t) + a\frac{z(t-\tau)}{1+z(t-\tau)^{10}} \tag{5.1}$$

where the constants are chosen to have the values $a = 0.2$, $b = 0.1$ and $\tau = 17$, and a sampling period of 1. It is standard to use these values, when the time-series is used as a benchmark for testing the prediction capability for neural networks. As input to the predictor the 12 previous samples with time step 6 is used. The predictor is then used as a 6 step ahead predictor. In the articles [Svarer et al., 1993b; Hansen et al., 1994], which are appended in appendix C and E, this example is extensively used as a benchmark test for the network performance. In the next subsections some new experiments will be shown. These results show more clearly than the results from the articles, what can be obtained by using the pruning methods, the stop criterion and the adaptive method for tuning the weight-decay parameters, which has been proposed in the previous chapters.

In figure 5.6 a three dimensional plot of the Mackey-Glass attractor is shown. This should be used for comparison of the results obtained by using the neural network as a predictor for the time-series.

In the following examples the input to the artificial neural network models and the other models, which are used for comparison of the performance, consist of the vector $\mathbf{x}(k) = [z(k-6), z(k-12), \ldots, z(k-96)]$ and the signal that the models should predict is $y(k) = z(k)$.

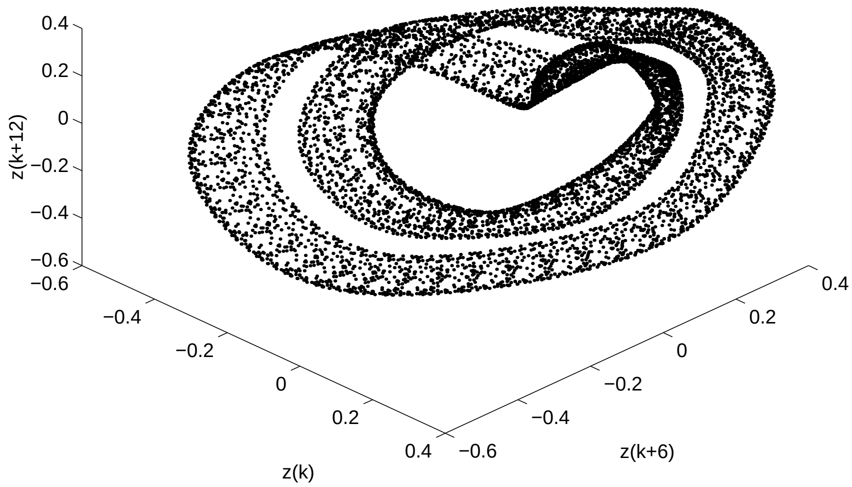


Figure 5.6: Three dimensional plot of the Mackey-Glass time-series, with $(z(k), z(k+6), z(k+12))$ at the three axis. 8400 points are shown at the figure.

5.2.1 Article IEEE-SP93

In the article in appendix C it is shown how pruning of the network helps in getting better generalization ability for the network. The weight-decay parameters were kept constant during the training and pruning session, with values $\alpha_w = 0.001$ and $\alpha_W = 0.001$. In figure 1 from the article it can be seen that during pruning the performance of the neural network is not changed very much. For a training set consisting of 1000 examples the test error is only decreasing very little during the pruning session, while for a training set size of 500 examples it can be seen that the the test error decreases a little more. This is also expected while a larger training set span the input space better and therefore will give a better generalization ability.

In table 1 in the article the performance of the networks is compared to a 5-nearest-neighbors model and a linear model. It can be seen that there is no significant difference between the performance of the fully connected network and the 5-nearest-neighbors-models. Compared to the linear model the test error is though a lot better. Using the FPE criterion to select the optimal network structure, shown by the vertical line in figure 1 from the article it can be seen that the performance compared to the fully connected network almost does not change. It is first when retraining the network without weight-decay that the performance of the neural network model significantly changes. Using a training set size of 1000 examples, table 1 shows that the performance at the training set has been improved by approximately 52% compared to the pruned network trained with weight-decay and approximately 50% at the test set. This shows that it is essential for the network performance to remove superfluous parameters from the network architecture, while it is first when the optimized network architecture is obtained that the network parameters can be retrained without weight-decay, with the result that both learning and generalization error decreases.

5.2.2 Retraining after pruning

When the optimal network structure has been found using a cost-function with weight-decay and the OBD technique, it has been shown in the article that it is possible to retrain the network without weight-decay. Table 1 in the article shows that both the training error and the test error are decreased by 50 percent. This means that if an appropriate network structure for the problem is found, the risk of overfitting the data to the training set is minimized.

It has been tried to use the same technique on a training set consisting of only 500 examples, but using retraining of the network without weight-decay has shown that the learning error as should be expected decreases but the test error increases. This means that the network architecture identified by the method for a training set size consisting of only 500 examples is not the optimal one. Therefore the network will adapt to noise in the training set and the performance on the test set deteriorates.

Another method for comparing the different network structures is to use correlation coefficients between the true attractor and the attractor generated by the network. The square holding the attractor is sampled with a grid of 20×20 subsquares. Using the network in a fully recurrent way to estimate point from the model of the attractor is then correlated to the points at the true attractor. Taking two samples of 8850 points gives a correlation coefficient of 0.991. For the fully connected network trained at 1000 examples the correlation coefficient is 0.922, while for the pruned network trained without weight-decay it is 0.948. This shows also a significantly better performance by the pruned network structure; it is however relatively difficult to see this in the plots of figure 2 in the article.

5.2.3 Adaptive weight-decay

Some new experiments have been setup on the Mackey-Glass problem to test the performance of the method for tuning the weight-decay parameters adaptively. This method is introduced in the article in appendix E and derived in appendix A. A training set consisting of only 500 examples is used for training. This training set is shown in figure 5.7. Comparing the training set to figure 5.6, which shows the Mackey-Glass attractor with 8400 points, it can be seen that only a small part of the attractor spanned in the test set is represented in the training set.

The network is then trained using the methods for adaptive tuning of the weight-decay parameters described in appendix A. Two weight-decay parameters are used throughout the experiments as also described when using constant weight-decays, one for the input-hidden layer weights and one for hidden-output layer weights. For the fully connected network the methods lead to a set of weight-decay parameters with the value of 1.4e-5 for the input-hidden layer parameter and for the hidden-output layer parameter a value of 4.0e-5. When using the fully connected network to estimate the Mackey-Glass attractor this leads to the points shown in figure 5.8. It can here be seen that the learned attractor

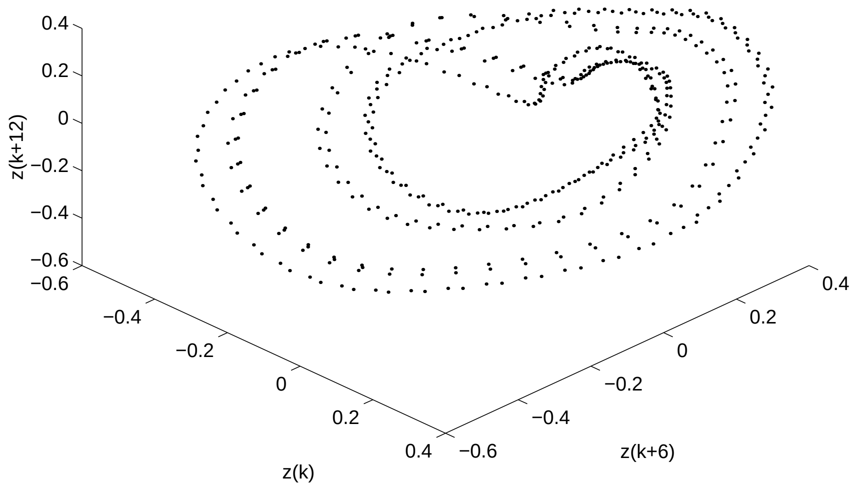


Figure 5.7: Three dimensional plot of the Mackey-Glass time-series, with $(z(k), z(k+6), z(k+12))$ at the three axis. The 500 points in the learning set are shown.

has the same structure as seen at the correct attractor shown in figure 5.6, but some parts of the attractor are not very well described. The characteristic of the learning set at figure 5.7 is also seen in the data from the learned attractor, where only a part of the attractor spanned by the test set is represented in the learned attractor.

The pruning session is then started and 5 percent of the remaining weights is removed from the network for each iteration. In figure 5.9 the learning, measured and estimated test errors are shown during a pruning session. It is seen that both the measured and estimated test error decreases while pruning the network. Further it is seen that the optimal network structure pointed out by the FPE method for estimating the generalization ability of the network, is the same network as that pointed out by having a test set. In figure 5.10 this network structure is shown. It can be seen that only 5 of the original 10 hidden neurons are used. This means that many degrees of freedom have been removed from the neural network model.

In figure 5.11 the weight-decay parameters are shown during the pruning session. It can be seen that the weight-decay parameters are decreased during pruning, until the optimal network is reached. This is also expected as for large network structures with many superfluous weights large values of the weight-decay parameters should be used to remove these extra degrees of freedom from the network. While it for the optimal network structure isn't necessary to have any weight-decay, as shown in the experiments from the article in appendix C.

The attractor found by using the optimal network structure shown in figure 5.10 to predict the Mackey-Glass time-series recursively is drawn in figure 5.12. By comparing this figure to the original attractor curve shown in figure 5.6 and the attractor generated by the

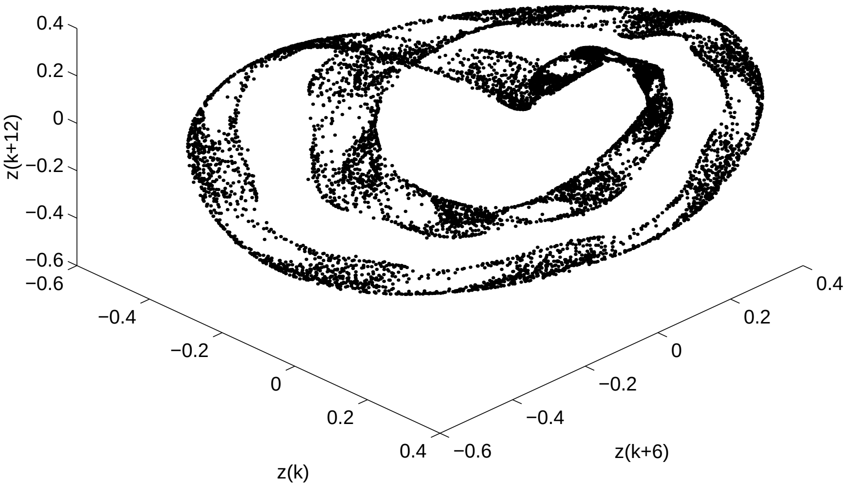


Figure 5.8: Three dimensional plot of the Mackey-Glass time-series, with $(z(k), z(k+6), z(k+12))$ at the three axis. A fully connected network is used to generate the 8400 points recursively (the recursion is started at the attractor). The results can be compared with figure 5.6 where the correct attractor curve is shown.

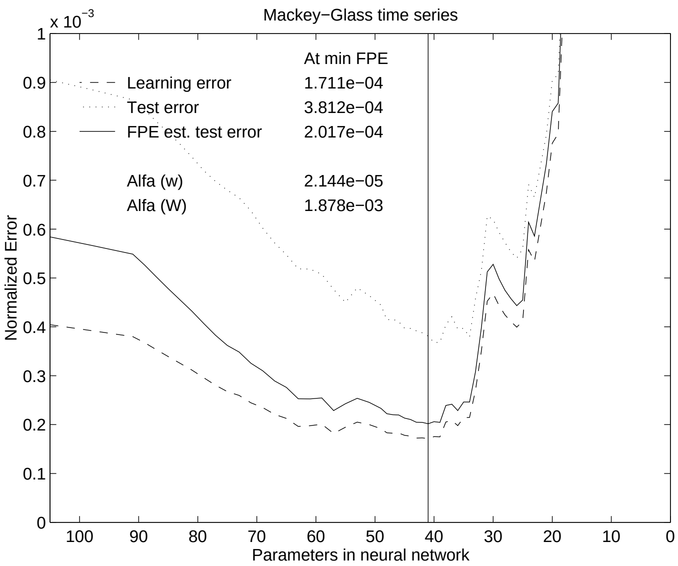


Figure 5.9: Pruning of the neural network for the Mackey-Glass series. The network is trained at the 500 examples shown in figure 5.7. The optimal network is chosen based at the FPE criterion. This network is shown at figure 5.10.

fully connected network shown in figure 5.8 it can be seen that the attractor generated by the pruned network is spanning the three-dimensional room better than the attractor

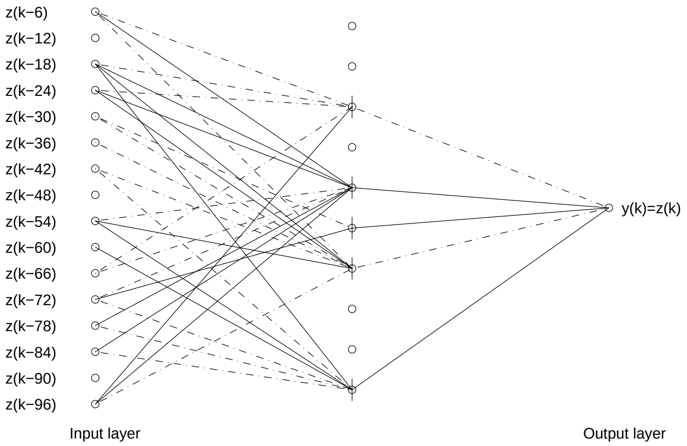


Figure 5.10: The optimal pruned network for the Mackey-Glass time-series. The network is trained and pruned at the 500 points shown in figure 5.7. Solid line - positive weights, Dot-dashed line - negative weights.

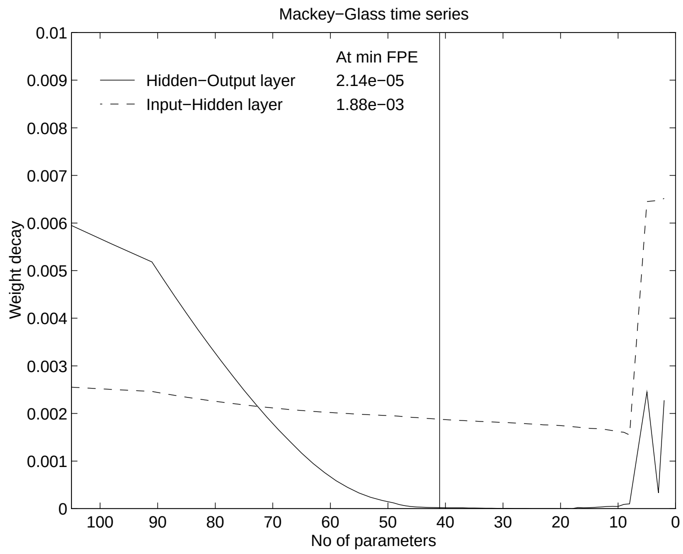


Figure 5.11: Pruning of the neural network for the Mackey-Glass series. While the network is pruned the method for estimating the optimal weight-decay parameter is used to estimate the optimal value of the two weight-decay parameters.

generated by using the fully connected network. It can though be difficult to see which of the figures that does the Mackey-Glass attractor best. To analyze this the correlation coefficients between the true attractor (a test set) and the output set generated using the fully connected and the pruned network models recursively is calculated. This is done by using the 8400 points shown in figure 5.6, 5.8 and 5.12 and sampling them in a two-dimensional grid of 20×20 points, with $z(k)$ and $z(k+6)$ at the axes (the two dimensional axes is used to be able to compare the results with them from the article in appendix C). Using two independent sets of 8400 points of the true attractor the correlations coefficient is $R = 0.991$. In figure 5.13 the correlation coefficient calculation for the fully connected network is illustrated, with the number of samples in each grid point for the true attractor at the x-axis, and at the y-axis for the fully connected network. The correlation coefficient is $R = 0.822$. For the pruned network this is illustrated in figure 5.14, and the correlation coefficient is $R = 0.919$. From this it can be seen that the attractor spanned using the pruned network to recursively estimate the points is really giving better results than using the fully connected network.

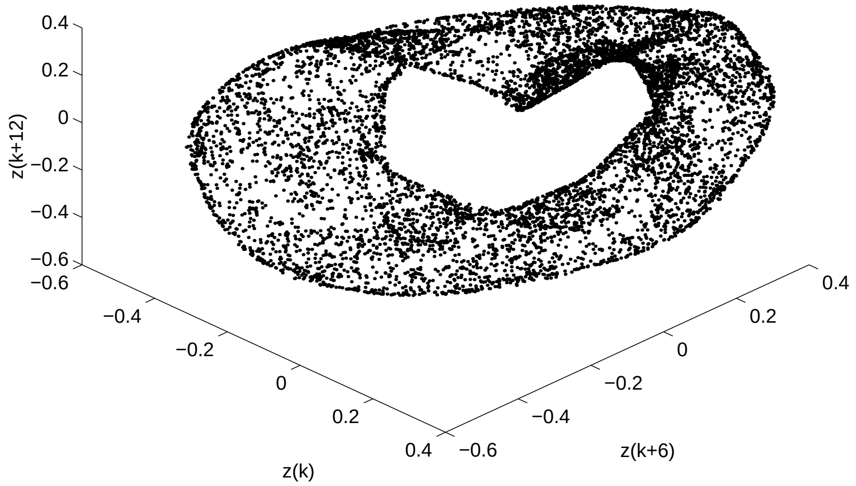


Figure 5.12: Three dimensional plot of the Mackey-Glass time-series, with $(z(k), z(k+6), z(k+12))$ at the three axis. The pruned network shown in figure 5.10 is used to generate the 8400 points recursively (the recursion is started at the attractor). The results can be compared with figure 5.6 and 5.8.

In table 5.2, the results are compared to what can be reached by a linear model and a k-nearest-neighbors model (k=5). The table shows that the performance of the pruned neural network is approximately 60% better than the performance of the fully connected network. Comparing the performance of the neural network to a 5-nearest-neighbors model it can be seen that the test error is approximately 50% better. Further it is seen that the linear model both at the training and test set has serious problems learning the task.

The experiments on using this adaptive weight-decay tuning method have shown that once in a while it is giving some problems, the weight-decay parameter seems to be unstable. Current work therefore concerns stabilizing the weight-decay adaption routine.

5.2.4 Learning of the attractor

Often one of the purposes of training the network is to use it for short-time predictions of time-series, which means that the network shall be used to predict what the next couple

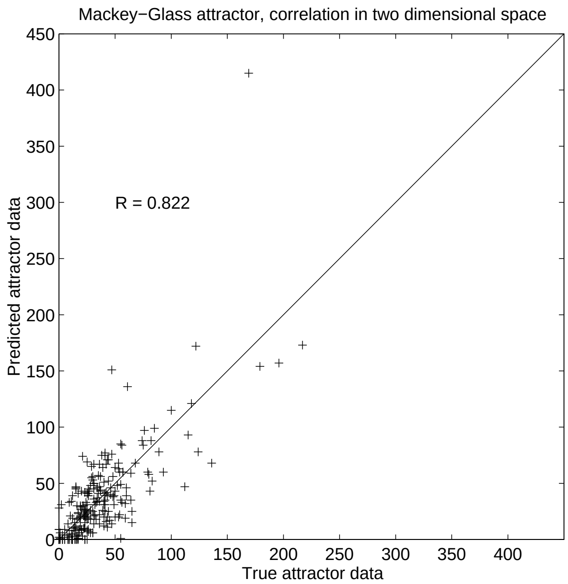


Figure 5.13: The figure shows the sampling of the points in a 20×20 grid, where the x-axis is from the true attractor and the y-axis is from the recursive estimate of points using a fully connected network. R is the correlation coefficient. The attractor is shown in figure 5.8.

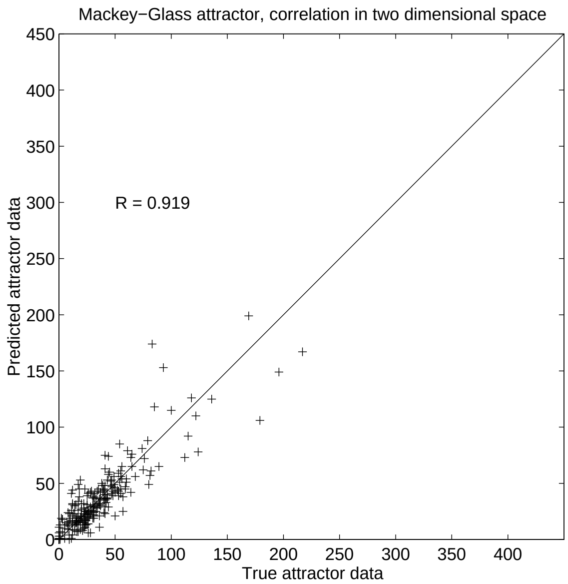


Figure 5.14: The figure shows the sampling of the points in a 20×20 grid, where the x-axis is from the true attractor and the y-axis is from the recursive estimate of points using the pruned network shown in figure 5.10. R is the correlation coefficient. The attractor is shown in figure 5.12.

of samples of the time-series shall be. This can be used for many different purposes, from

Normalized mean square error

	Train	Test	Reduc.	No. of par.	Reduc.
5-nearest-neighbors	4.27×10^{-8}	7.07×10^{-4}	–	–	–
Linear model	8.93×10^{-2}	9.77×10^{-2}	–	17	–
Fully connected network[a]	4.00×10^{-4}	9.00×10^{-4}		181	
Pruned network[b]	1.71×10^{-4}	3.81×10^{-4}	57.7%	41	77.3%

Table 5.2: Result on the Mackey-Glass time-series using the adaptive pruning scheme (11 networks are trained to give the statistics). All models are trained on 500 examples and tested on 8400 examples, Neural network models are a) Initial fully connected network. b) Pruned network.

weather forecasts to prediction of stock exchanges. The ability to predict can be tested with the Mackey-Glass series by using the neural network to estimate the next couple of samples from a test set. In figure 5.15 this is drawn. The estimation is started at the attractor by using the last samples as inputs and it is then run in a recursive way where the estimates are used as new inputs to the neural network. From the figure it can be seen that the estimates diverge within the attractor. By this is meant that the points are still on the attractor (as the previous shown curves also indicate) but the distance between the estimated point and the "correct" point is growing. This is also expected since the Mackey-Glass differential equation is a chaotic time-series, which can't be modeled fully by the neural network equations. Therefore it must diverge after a couple of samples as always will be the result in chaotic time-series [Parker and Chua, 1989]. It is though seen that for 10 to 15 iterations the neural network method seems to give quite a nice estimate of the expected value. It will therefore be possible to use the neural network for short-time prediction of the time-series.

5.3 Channel Equalizer

Experiments with the methods for automatic tuning of the weight-decay parameter have also been done at the inverse modeling problem described in [Day et al., 1992], where an inverse model for a non-linear transmission channel is estimated. In the article appended in appendix C the experiments made without automatic tuning of the weight-decay parameter are illustrated. The transmission channel can be described by the non-linear equation

$$v = 5u|u|(1-u^2) \Big/ (1+u^2) \,, \tag{5.2}$$

sandwiched between two low-pass filters with identical transfer functions

$$H_1(z) = 0.2602 - 0.9298z^{-1} + 0.2602z^{-2}. \tag{5.3}$$

The driving signal is a Gaussian i.i.d. sequence (unit variance, zero mean) filtered through

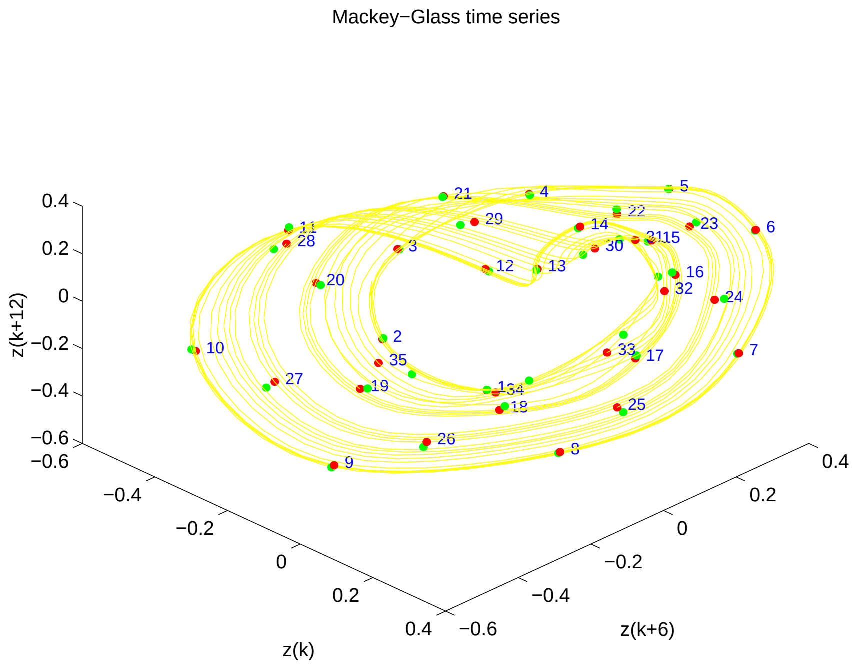


Figure 5.15: Testing of the possibility to do short-time forecasting with the pruned network show in figure 5.10. The network is used in a recursive way where the last output is used as input to the network when doing the next forecast. It is seen that the network is doing short-time forecasting quite well, but after 15 to 20 recursions the forecast starts to diverge. The numbers in the figure tells the iteration number after prediction has been started.

a third-order *Butterworth* low-pass filter with the transfer function

$$H_2(z) = \frac{0.727 + 2.18z^{-1} + 2.18z^{-2} + 0.727z^{-3}}{1.000 - 2.327z^{-1} + 2.072z^{-2} - 0.686z^{-3}} \cdot 10^{-3} \tag{5.4}$$

The network is then used to invert the transmission channel and estimate the input signal to the channel. This is illustrated by the block diagram shown in figure 5.16.

The training set consists of 2000 samples from input and output signals to the channel. As input to the neural network, a tapped delay line consisting of 25 taps of the noisy output from the channel is used. The signal that the neural model should predict is a 15 time steps delayed sampling of the input signal to the channel. The signal has to be delayed in order to ensure causal filtering.

In this example it is tried to compare the results using the automatic method for tuning the weight-decay to the results that could be reached by just tuning the weight-decay parameters manually. In figure 5.17 the learning error, the test error and the estimated test error are drawn, when the weight-decay parameters are tuned manually, by looking at the test error. This means that it is necessary to spare a part of the data set for evaluation, so not all the the data can be used for training. In figure 5.18 the curves when using the automatic scheme for tuning the weight-decay parameters is shown. It can from these

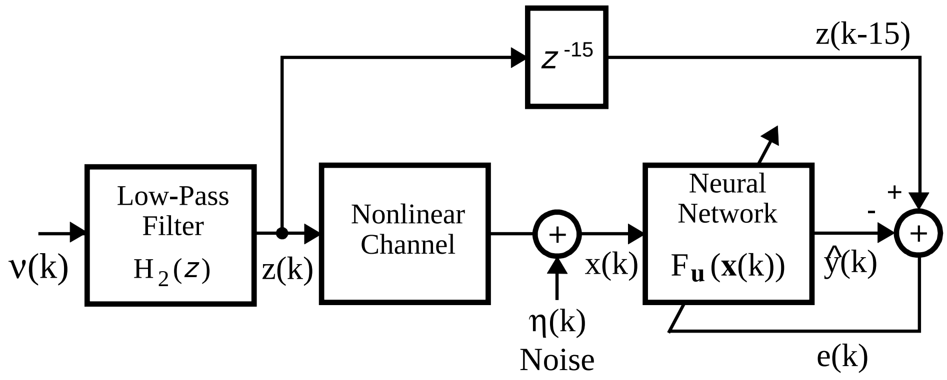


Figure 5.16: Block diagram showing the structure of the transmission channel. In the block *Nonlinear Channel* the equations (5.2) and (5.3) are used.

two curves be seen, that the results when using the automatic scheme is comparable to the results that can be obtained by manually tuning the parameters. This means that the method for this example gives a way of tuning the weight-decay parameters, without sparing any of the data sets for testing.

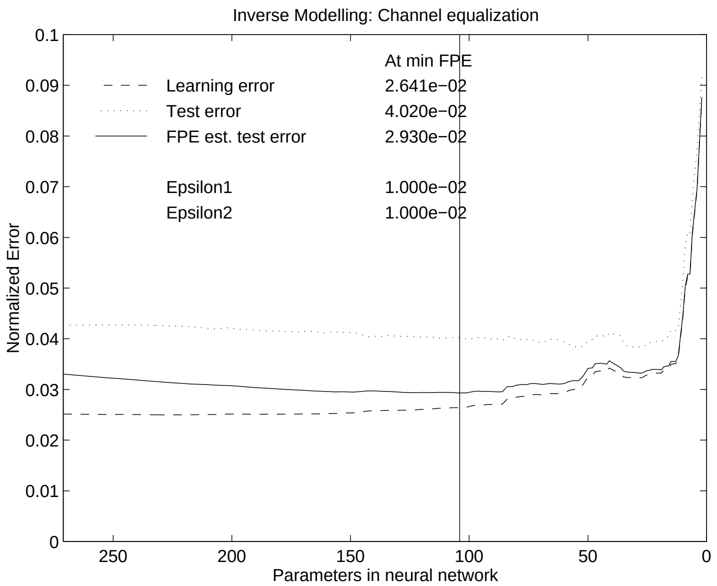


Figure 5.17: Plot of the learning, test and estimated test error for the channel equalizer during a pruning session. The weight-decay parameters are kept constant at the values $\alpha_w = 0.01$ (input-hidden) and $\alpha_W = 0.01$ (hidden-output).

In figure 5.19 it can be seen how the weight-decay parameters are changing during pruning

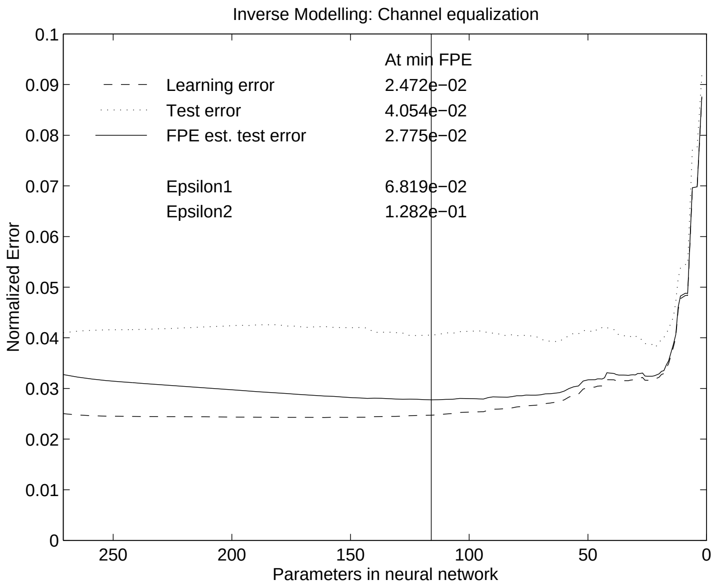


Figure 5.18: Plot of the learning, test and estimated test error for the channel equalizer during a pruning session. The weight-decay parameters are tuned adaptively during the pruning session. For the network structure with lowest estimated test error the weight-decay values are $\alpha_w = 0.068$ (input-hidden) and $\alpha_W = 0.128$ (hidden-output).

of the weights. The parameters still have a quite large value when the FPE estimate of the generalization error has a minima at figure 5.18. In this figure it can though be seen that the test error is still decreasing when the pruning continues until at network size consisting of only 22 weights. At this network size the weight-decay parameters are also decreased compared to the net chosen by the FPE estimate. For the network with the lowest test error the weight-decay parameters have a size of ($\alpha_w = 3.4e - 3$) and ($\alpha_W = 1.48e - 3$). From this it can be concluded that the scheme for tuning the weight-decay parameters seems to work well, giving as a general result that when the network size is reduced, the weight-decay parameters needed to prevent overfitting of the learning set are a lot smaller than for an overparameterized model.

The optimal network pointed out by Akaike's FPE criterion is shown in figure 5.20. It can be seen that almost all the the inputs are used although it should be clear (due to the constraint that the model must be causal) that it is only possible for the model to get new information from the inputs ($x(k-10)$ - $x(k-25)$). From this it can therefore be concluded that the pruning session and the FPE criterion have not found the optimal network configuration but have pointed out a network with too many weights. This can also be seen at figure 5.18 where the test error decreases to a network size of only 22 parameters. It is known from the literature [Ljung, 1987] that the FPE criterion has a tendency to overparameterize the system which also seems to be the case here. As mentioned in a previous chapter it is shown in [Kashyap, 1980] that there is not less than a 15% chance that a wrong model is chosen by using the FPE criteria. One of the reasons for that is that it is the test error that is estimated by the FPE criteria and not the actual "correct" model order. The estimate of the test error for the different pruned networks is

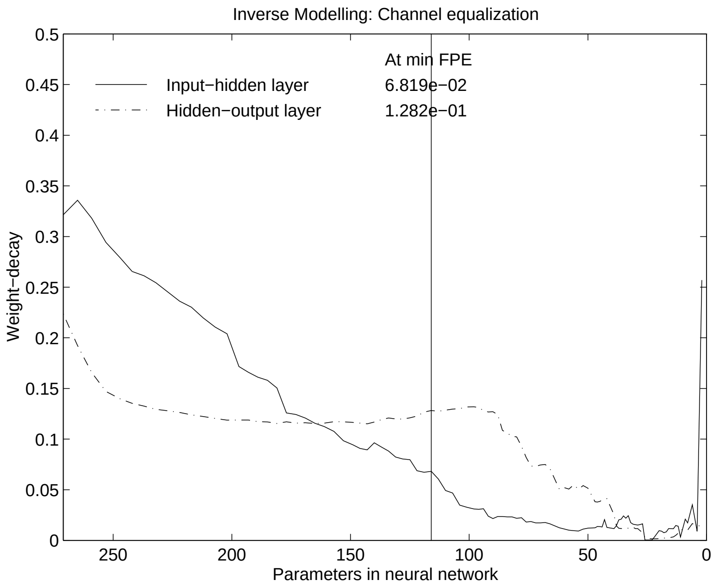


Figure 5.19: Plot of the evaluation of the weight-decay parameters during a pruning session. The vertical bar is drawn for the optimal network configuration estimated by Akaike's FPE. This is shown in figure 5.18.

then used to find the optimal architecture. In figure 5.21 the optimal network chosen by the test error is drawn and it can here be seen that almost all the non causal connections in the model have been removed. There is only one connection to $x(k-1)$ and a connection to $x(k-7)$. The test set is used for selecting this network structure and it is therefore necessary to have another data set for testing the performance of the selected model.

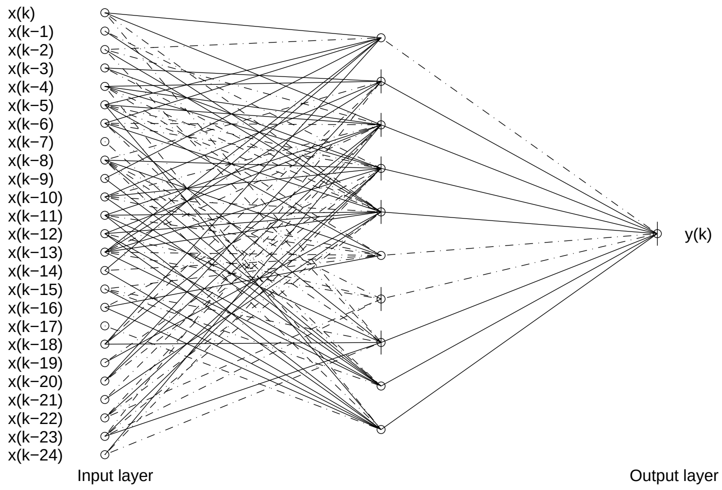


Figure 5.20: Optimal net configuration chosen by Akaike's FPE criterion. The model is still overparameterized while connections to the input values ($x(k)$ - $x(k-9)$) are giving a non causal model. The neural network models has 122 weights.

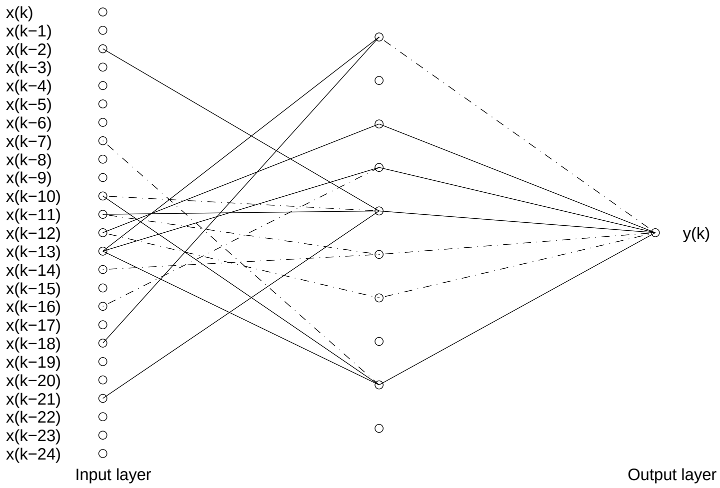


Figure 5.21: Optimal net configuration chosen by the minimum of the test error. There are only 22 weights present in this model structure.

The performance of the neural network technique has also been compared to the performance of the k-nearest-neighbors model (with $k = 5$) and a linear model. In table 5.3 these results are listed. Comparing the 5-nearest-neighbors model and the pruned neural network using adaptive weight-decay the performance at the test set is approximately 56% better, while it for the linear model is approximately 60% better. For the neural network model it can be seen that the pruning scheme, if using the same weight-decay values during the pruning, is giving a performance improvement of approx. 6%, while the number of parameters is reduced by 61%. For the pruning session where weight-decay tuning is used there is almost no reduction of the error of the test set. This must be due to the large weight-decay values used for the fully connected network, which prevent overfitting to the learning set. The reduction of the number of parameters is 57%.

Normalized mean square error

	Train	**Test**	**Reduc.**	**No. of par.**	**Reduc.**
5-nearest-neighbors[1]	2.74×10^{-14}	9.36×10^{-2}	–	–	–
Linear model[2]	1.05×10^{-1}	1.30×10^{-1}	–	26	–
Fully connected network[3a]	2.51×10^{-2}	4.27×10^{-2}		271	
Pruned network[3b]	2.64×10^{-2}	4.02×10^{-2}	5.85%	104	61.3%
Fully connected network[4a]	2.50×10^{-2}	4.10×10^{-2}		271	
Pruned network[4b]	2.47×10^{-2}	4.05×10^{-2}	1.22%	116	57.2%

Table 5.3: 1) Trained on 2000 examples. 2) Linear model trained on 2000 examples (a single linear unit). 3) Neural Networks trained on 2000 examples using constant weight-decay parameters (w=0.01, W=0.01), a) Initial fully connected network. b) Pruned network. 4) Neural Networks trained on 2000 examples using the weight-decay optimization technique, a) Initial fully connected network. b) Pruned network. All models are tested on 2000 examples.

Chapter 6

Conclusion

In this thesis and the articles appended I have focused on what can be achieved in generalization ability of neural networks through optimization of the network architectures. The networks analyzed are of the feed-forward type with one hidden layer of neurons. For the hidden units tanh activation functions are used while the output units have linear activation functions. The learning algorithm used to optimize the network parameters is based on the Gauss-Newton method.

Network training is based on overparameterized fully connected networks which are then pruned to a compact size by elimination of superfluous resources (weights). Pruning is done using the Optimal Brain Damage (OBD) method for estimation of the saliencies of the network weights. A main objective has been the development of a stopping criterion which can tell when the optimal network configuration is reached. For this purpose Akaike's Final Prediction Error (FPE) estimate of the generalization error is used for problems with continuous output, while a more heuristic method has been proposed to stop pruning in networks for classification tasks.

The network optimization algorithm depends on a number of parameters. In the ordinary back-propagation algorithm a gradient-descent parameter has to be chosen. This parameter is found using learning algorithms of the Gauss-Newton type. Furthermore a regularization (weight-decay) parameter is present in the cost-function. A method for estimating this parameter for problems with continuous output is found, based on a minimization of Akaike's FPE generalization error estimate with respect to the weight-decay parameter. Earlier experiments have shown that this parameter is crucial for the network's generalization ability.

Experiments on the "contiguity" classification problem addressed the question of the necessary number of examples for learning of a given problem. When pruning the networks the result shows that the number of examples needed for learning the problem is reduced by a factor 2 from about 200, without pruning, to 100 examples with pruning. In this classification problem the smallest network with no training errors was chosen. The method can of course only be used for problems without noise; else wise a noise level has to be estimated and the network pruning should instead be stopped when this error level is reached. The experiments done on the "contiguity" problem have also shown that when an ensemble of networks is trained and pruned on the same data set, the most pruned network shall be chosen as the optimal one, implying that "simple" is good.

The quality of the second order diagonal approximation used in the optimization algorithm and in the OBD method for estimating the weight saliencies has been tested using data from this problem. It has been shown that the approximation for most weights is very good and can be used without any risk of introducing large errors. The cell classification task has shown that the methods also can be used with success on classification problems where noise is present in the data. Very compact networks with better generalization ability than the fully connected network have been designed.

For the time-series problems, involving prediction and model building, it has also been shown that the pruning method gives superior results in generalization compared to the results obtained using fully connected networks. Especially for the sunspot time-series the obtained results are convincing. For the Mackey-Glass chaotic time-series the experiments using adaptive weight-decay parameters show that it is possible to tune these parameters automatically and obtain results that are comparable to what can be obtained by a method where the parameters are tuned by hand using a validation data set.

Even though good results have been obtained, using the diagonal approximation to the second order terms in the optimization methods (Gauss-Newton method), the method for estimating the parameter saliencies (the OBD method) and the method for estimating the generalization error (Akaike's FPE method), better results could perhaps be obtained using the full Hessian matrix for the cost-function. This has to be examined by new experiments; but as mentioned this will also make new demands to the quality of the data set that is used when learning the problem.

To summarize, the major results that have been obtained are:

- The FPE criteria can be used as a stopping criterion for network pruning in problems with continuous output.
- By using the pruning technique it is possible to get better generalization ability by the networks or it is possible to use fewer examples in the training set to learn the problem.
- The most pruned network that can do the learning task shall be chosen as the optimal one (the network that has the best generalization ability) if an ensemble of networks is pruned on the same data set.
- Pruning of the network prevents overfitting to noise in the learning set.
- It is possible to use adaptive weight-decay algorithms to tune the regularization parameters in the network optimization algorithms and obtain results that are comparable to what can be reached when tuning the parameters by hand using a validation set.

Appendix A

FPE estimate of generalization error

A.1 Introduction

In this appendix methods for choosing a suitable model structure are described. The methods are based on the theory described in [Akaike, 1968] and in [Ljung, 1987; Hansen and Rasmussen, 1994; Larsen, 1993]. Using a second order approximation to the cost-function an estimate for the generalization error is developed. The method is first derived for the case where no regularization is used in the cost-function, afterwards for the case where a regularization by weight-decay term is included in the cost-function.

Further, a method for adaptive choosing of the correct value of the regularization parameters (weight-decay parameters) for neural networks is derived. This method is based on the results derived when calculating the FPE estimate of the cost-function with regularization. In the article appended in appendix E some experiments using the adaptive regularization method on tasks within digital signal processing are described.

A.1.1 Akaike's Final Prediction Error estimate.

The derivation of the Final Prediction Error estimate is based on the methods described in [Ljung, 1987; Hansen and Rasmussen, 1994; Larsen, 1993]. The notation is changed to follow the notation used throughout this thesis. The method is based on a second order approximation to the cost-function. This is sketched in figure A.1.

As shown in the figure, it is assumed that both the learning error function (the cost-function minimized) and the generalization error function can be approximated by a second order approximation. The error functions can be written as

$$\mathbf{E}_{\text{learn}} = E_0 + \left(\frac{\partial E}{\partial \mathbf{u}}\right)^T \delta\mathbf{u} + \frac{1}{2}\delta\mathbf{u}^T \mathbf{H} \delta\mathbf{u} + o(\|\delta\mathbf{u}\|^3), \tag{A.1}$$

and

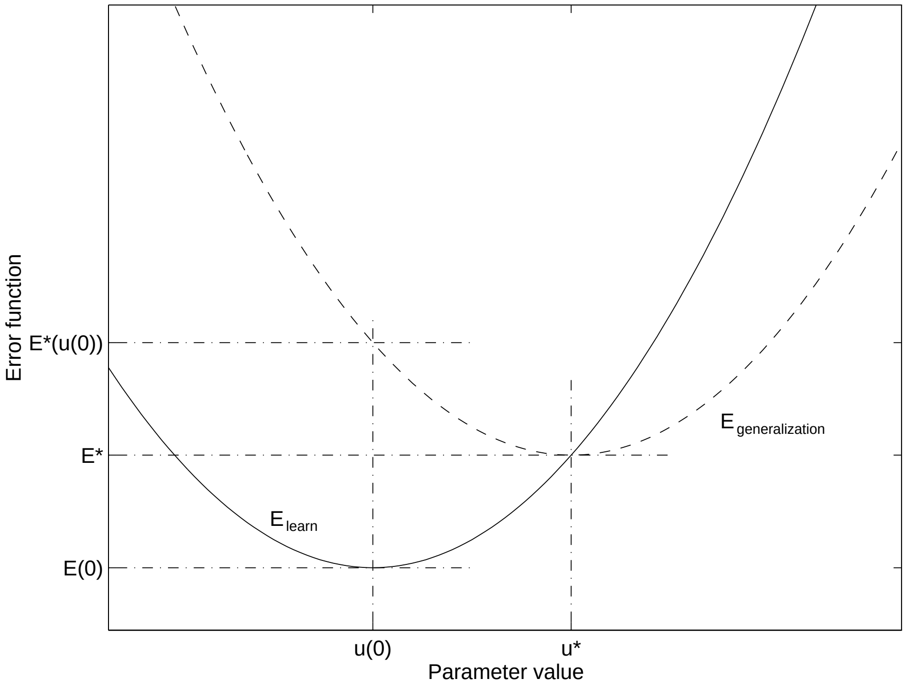


Figure A.1: Second order approximation to the learning and the generalization error functions.

$$\mathbf{E}_{\text{generalization}} = E^* + \left(\frac{\partial E^*}{\partial \mathbf{u}}\right)^T \delta\mathbf{u}^* + \frac{1}{2}\delta\mathbf{u}^{*T}\mathbf{H}^*\delta\mathbf{u}^* + o(\|\delta\mathbf{u}^*\|^3), \tag{A.2}$$

where $\mathbf{E}_{\text{learn}}$ is the learning error and $\mathbf{E}_{\text{generalization}}$ is the generalization error. Only up to the the second order terms are fully written. $\delta\mathbf{u}$ is a small change of the parameter (weight) vector around the minimum for the learning error function, while $\delta\mathbf{u}^*$ is a small change of the parameter vector around the minimum of the generalization error function. $\frac{\partial E}{\partial \mathbf{u}}$ and $\frac{\partial E^*}{\partial \mathbf{u}}$ are the first order derivatives of the two error functions. $\mathbf{H}$ and $\mathbf{H}^*$ are the Hessian matrices (second order derivatives in the parameters) of the two error functions.

In the notation used for the learning and generalization error estimate, it is the mean of the errors for all combinations of learning sets of a given size p, which is meant by $\widehat{E}_{\text{learn}}$ and $\widehat{E}_{\text{generalization}}$. This can also be written as

$$\widehat{E}_{\text{learn}} = \langle E_{\text{learn}} \rangle_p \tag{A.3}$$

$$\widehat{E}_{\text{generalization}} = \langle E_{\text{generalization}} \rangle_p \tag{A.4}$$

where the notation $\langle \ldots \rangle_p$ describes that it is the mean of errors calculated on all learning sets of size p. The derivatives in the minimum of the two error functions, for the parameter vector u_0 and u^* respectively, can be set to zero, so the estimates E_{learn} and $E_{\text{generalization}}$ can be written as

$$E_{\text{learn}} = E_0 + \frac{1}{2}\delta\mathbf{u}^T\mathbf{H}\delta\mathbf{u} \tag{A.5}$$

$$E_{\text{generalization}} = E^* + \frac{1}{2}\delta\mathbf{u}^{*T}\mathbf{H}^*\delta\mathbf{u}^* \tag{A.6}$$

In figure A.1 it can be seen that the test error (the generalization error in the parameter vector $\mathbf{u_0}$) can be calculated using the generalization error approximation in equation (A.6). Using this equation it is therefore possible to calculate the test error for a given parameter vector $\mathbf{u}$. The problem is then that E^*, $\mathbf{u}^*$ and $\mathbf{H}^*$ are unknown. It is in the following assumed that the network is well trained, so the distance between $\mathbf{u_0}$ and $\mathbf{u}^*$ is small. It can then be assumed that the second order derivative of the generalization error function can be approximated by the second order derivative of the learning error function. This means that equation (A.6) can be written as

$$E_{\text{generalization}} \approx E^* + \frac{1}{2}\delta\mathbf{u}^{*T}\mathbf{H}\delta\mathbf{u}^* \tag{A.7}$$

In the following some further calculations will be done at the term with the Hessian matrix. For the learning error functions the second order term can be written as

$$\begin{aligned} E_{\text{second\_order\_term}} &= \frac{1}{2}\delta\mathbf{u}^{*T}\mathbf{H}\delta\mathbf{u}^* \\ &= \frac{1}{2}(\mathbf{u}-\mathbf{u_0})^T\mathbf{H}(\mathbf{u}-\mathbf{u_0}) \\ &= \frac{1}{2}tr\left(\mathbf{H}(\mathbf{u}-\mathbf{u_0})(\mathbf{u}-\mathbf{u_0})^T\right) \end{aligned} \tag{A.8}$$

where tr is the trace of the matrix within the parenthesis.

This equation can be used in equation (A.5) and (A.7) to calculate the second order term. Therefore these two equations can be reformulated as

$$E_{\text{learn}}(\mathbf{u}) = E_0 + \frac{1}{2}tr\left(\mathbf{H}(\mathbf{u}-\mathbf{u_0})(\mathbf{u}-\mathbf{u_0})^T\right) \tag{A.9}$$

$$E_{\text{generalization}}(\mathbf{u}) \approx E^* + \frac{1}{2}tr\left(\mathbf{H}(\mathbf{u}-\mathbf{u}^*)(\mathbf{u}-\mathbf{u}^*)^T\right) \tag{A.10}$$

In these two equations there are some unknown terms which will be calculated in the following sections for the two different cases, without and with regularization at the learning cost-function. The goal for these calculations is to calculate an estimate of the generalization error in the parameter vector u_0 that minimizes the learning error. By combining the two equations (A.9) and (A.10) the unknown parameters E^* and u^* could be removed from the equations. In equation (A.9) $\mathbf{u}$ is set to $\mathbf{u}^*$. The learning error in the learned parameter vector $\mathbf{u_0}$ could then be expressed as a function of the learning error in the 'correct' parameter vector $\mathbf{u}^*$. This could be written as

$$E_{\text{learn}}(\mathbf{u}^*) \approx E_{\text{learn}}(\mathbf{u_0}) + \tfrac{1}{2} tr \left(\mathbf{H} \, (\mathbf{u}^* - \mathbf{u_0}) \, (\mathbf{u}^* - \mathbf{u_0})^T \right)$$

$$\Updownarrow$$

$$E_{\text{learn}}(\mathbf{u_0}) \approx E_{\text{learn}}(\mathbf{u}^*) - \tfrac{1}{2} tr \left(\mathbf{H} \, (\mathbf{u}^* - \mathbf{u_0}) \, (\mathbf{u}^* - \mathbf{u_0})^T \right) \tag{A.11}$$

Using (A.10) it is possible to find an expression for the generalization error in the parameter vector $\mathbf{u_0}$. This could be written as

$$E_{\text{generalization}}(\mathbf{u_0}) \approx E_{\text{learn}}(\mathbf{u}^*) + \frac{1}{2} tr \left(\mathbf{H} \, (\mathbf{u_0} - \mathbf{u}^*) \, (\mathbf{u_0} - \mathbf{u}^*)^T \right) \tag{A.12}$$

In these equations (A.11) and (A.12) the term $(\mathbf{u_0} - \mathbf{u}^*) \, (\mathbf{u_0} - \mathbf{u}^*)^T$ is unknown and should be calculated. In the next sections the calculations will be done for the two cases without and with regularization (weight-decay) included in the optimized cost-function.

As discussed before all these calculation are done at the assumption that the error functions have been calculated at the mean of all combinations of training sets of size p. A more correct notation for the last two equations would therefore be

$$\begin{aligned} \widehat{E}_{\text{learn}}(\mathbf{u_0}) &= \left\langle E_{\text{learn}}(\mathbf{u_0}) \right\rangle_p \\ &\approx \left\langle E_{\text{learn}}(\mathbf{u}^*) \right\rangle_p - \frac{1}{2} \left\langle tr \left(\mathbf{H} \, (\mathbf{u}^* - \mathbf{u_0}) \, (\mathbf{u}^* - \mathbf{u_0})^T \right) \right\rangle_p \end{aligned} \tag{A.13}$$

$$\begin{aligned} \widehat{E}_{\text{generalization}}(\mathbf{u_0}) &= \left\langle E_{\text{generalization}}(\mathbf{u_0}) \right\rangle_p \\ &\approx \left\langle E_{\text{learn}}(\mathbf{u}^*) \right\rangle_p + \frac{1}{2} \left\langle tr \left(\mathbf{H} \, (\mathbf{u_0} - \mathbf{u}^*) \, (\mathbf{u_0} - \mathbf{u}^*)^T \right) \right\rangle_p \end{aligned} \tag{A.14}$$

Cost-function without regularization

A cost-function without regularization is in the neural network terms a cost-function without weight-decay. When using the quadratic cost-functions and assuming that there is only one output from the neural network this can be written as

$$\begin{aligned} E_p(\mathbf{u}) &= E_{\text{quadratic}}(\mathbf{u}) \\ &= \frac{1}{p} \sum_{\mu=1}^{p} (e^\mu)^2 \\ &= \frac{1}{p} \sum_{\mu=1}^{p} (\zeta^\mu - O^\mu)^2 \end{aligned} \tag{A.15}$$

By using a second order Taylor development of this around the 'correct' parameter set $\mathbf{u}^*$ this can be written as

$$E_p(\mathbf{u}) \approx E_p(\mathbf{u}^*) + \left(\frac{\partial E_p(\mathbf{u}^*)}{\partial \mathbf{u}^*}\right)^T (\mathbf{u}^* - \mathbf{u}) + \frac{1}{2}(\mathbf{u}^* - \mathbf{u})^T \mathbf{H_p} (\mathbf{u}^* - \mathbf{u}) \tag{A.16}$$

where $\mathbf{H_p}$ is the Hessian matrix of the cost-function in the parameter vector $\mathbf{u}^*$.

By differentiating this equation it can be found that

$$\frac{\partial E_p(\mathbf{u})}{\partial \mathbf{u}} \approx -\frac{\partial E_p(\mathbf{u}^*)}{\partial \mathbf{u}^*} - \mathbf{H_p}(\mathbf{u}^* - \mathbf{u}) \tag{A.17}$$

For $u = u_0$ the derivative is zero. Therefore the difference between the 'correct' parameter vector and the parameter vector estimated based on the a training set of p can be calculated as

$$(\mathbf{u}^* - \mathbf{u_0}) \approx -(\mathbf{H_p})^{-1} \frac{\partial E_p(\mathbf{u}^*)}{\partial \mathbf{u}^*} \tag{A.18}$$

Using equation (A.15) the derivative and second order derivative of the cost-function can be calculated as

$$\frac{\partial E_p(\mathbf{u})}{\partial \mathbf{u}} = -\frac{2}{p}\sum_{\mu=1}^{p}\left(\frac{\partial O^\mu(\mathbf{u})}{\partial \mathbf{u}}\right)(e^\mu) \tag{A.19}$$

$$\begin{aligned}\frac{\partial^2 E_p(\mathbf{u})}{\partial \mathbf{u}^2} = \mathbf{H_p} &= \frac{2}{p}\sum_{\mu=1}^{p}\left(\frac{\partial O^\mu(\mathbf{u})}{\partial \mathbf{u}}\right)\left(\frac{\partial O^\mu(\mathbf{u})}{\partial \mathbf{u}}\right)^T - \left(\frac{\partial^2 O^\mu(\mathbf{u})}{\partial \mathbf{u}^2}\right)(e^\mu) \\ &\approx \frac{2}{p}\sum_{\mu=1}^{p}\left(\frac{\partial O^\mu(\mathbf{u})}{\partial \mathbf{u}}\right)\left(\frac{\partial O^\mu(\mathbf{u})}{\partial \mathbf{u}}\right)^T \end{aligned} \tag{A.20}$$

The approximation is the Levenberg-Marquart approximation, it holds for large training sets ($p \approx \infty$). Then the error e^μ, can be approximated by a white noise sequence independent of the derivative of the output O^μ. Therefore this part of the expression could be set to zero. Using this together with equation (A.18) the difference between the 'correct' parameter vector and the parameter vector estimated based on a training set of p examples can be written as

$$(\mathbf{u}^* - \mathbf{u_0}) \approx (\mathbf{H_p})^{-1}\frac{2}{p}\sum_{\mu=1}^{p}\left(\frac{\partial O^\mu(\mathbf{u})}{\partial \mathbf{u}}\right)(e^\mu) \tag{A.21}$$

The term that is unknown in equation (A.14) is $< (\mathbf{u}^* - \mathbf{u_0})(\mathbf{u}^* - \mathbf{u_0})^T >_p$. It is not possible to calculate that while only one training set of size p is applicable. Instead the best estimate would be the error calculated based on one training set of size p. The estimator for that can therefore, based on equation (A.21) be calculated as

$$
\begin{aligned}
&\left\langle (\mathbf{u}^* - \mathbf{u_0})(\mathbf{u}^* - \mathbf{u_0})^T \right\rangle_p \\
&= \left\langle (\mathbf{H_p})^{-1} \frac{2}{p} \sum_{\mu=1}^{p} \left(\frac{\partial O^\mu(\mathbf{u})}{\partial \mathbf{u}} \right) (e^\mu) \frac{2}{p} \sum_{\mu'=1}^{p} (e^{\mu'})^T \left(\frac{\partial O^{\mu'}(\mathbf{u})}{\partial \mathbf{u}} \right)^T (\mathbf{H_p})^{-1} \right\rangle_p \\
&= \frac{4}{p^2} (\mathbf{H_p})^{-1} \left\langle \sum_{\mu=1}^{p} \left(\left(\frac{\partial O^\mu(\mathbf{u})}{\partial \mathbf{u}} \right) (e^\mu)(e^\mu)^T \left(\frac{\partial O^\mu(\mathbf{u})}{\partial \mathbf{u}} \right)^T \right) \right\rangle_p (\mathbf{H_p})^{-1} \\
&= \frac{4}{p^2} (\mathbf{H_p})^{-1} E^* \left\langle \sum_{\mu=1}^{p} \left(\left(\frac{\partial O^\mu(\mathbf{u})}{\partial \mathbf{u}} \right) \left(\frac{\partial O^\mu(\mathbf{u})}{\partial \mathbf{u}} \right)^T \right) \right\rangle_p (\mathbf{H_p})^{-1} && \text{(A.22)} \\
&= \frac{2}{p} (\mathbf{H_p})^{-1} E^* \mathbf{H_p} (\mathbf{H_p})^{-1} && \text{(A.23)} \\
&\approx \frac{2}{p} (\mathbf{H_p})^{-1} E^* && \text{(A.24)}
\end{aligned}
$$

where it in (A.22) is assumed that the noise e^μ is white. In (A.23) the expression from equation (A.20) is inserted.

Using the result from equation (A.24) and equation (A.13) and (A.14), where $\mathbf{H}$ is approximated by the best estimator $\mathbf{H_p}$ when only p examples is applicable, the best estimator for the learning and generalization error can be found as

$$
\begin{aligned}
\widehat{E}_{\text{learn}}(\mathbf{u_0}) &\approx E_{\text{learn}}(\mathbf{u}^*) - \frac{1}{2} tr \left(\mathbf{H_p} (\mathbf{u}^* - \mathbf{u_0})(\mathbf{u}^* - \mathbf{u_0})^T \right) \\
&= E_{\text{learn}}(\mathbf{u}^*) - \frac{1}{2} tr \left(\mathbf{H_p} \frac{2}{p} \mathbf{H_p}^{-1} E^* \right) \\
&= E_{\text{learn}}(\mathbf{u}^*) \left(1 - \frac{1}{p} tr \left(\mathbf{H_p} \mathbf{H_p}^{-1} \right) \right) \\
&= E_{\text{learn}}(\mathbf{u}^*) \left(1 - \frac{N_{\mathcal{M}}}{p} \right) && \text{(A.25)}
\end{aligned}
$$

$$
\begin{aligned}
\widehat{E}_{\text{generalization}}(\mathbf{u_0}) &\approx E_{\text{learn}}(\mathbf{u}^*) + \frac{1}{2} tr \left(\mathbf{H_p} (\mathbf{u_0} - \mathbf{u}^*)(\mathbf{u_0} - \mathbf{u}^*)^T \right) \\
&= E_{\text{learn}}(\mathbf{u}^*) + \frac{1}{2} tr \left(\mathbf{H_p} \frac{2}{p} \mathbf{H_p}^{-1} E^* \right) \\
&= E_{\text{learn}}(\mathbf{u}^*) \left(1 + \frac{N_{\mathcal{M}}}{p} \right) && \text{(A.26)}
\end{aligned}
$$

where $N_{\mathcal{M}}$ is the order (dimension) of the parameter vector.

Combining these two equations it is possible to remove the unknown noise variance E^* from the equations and calculate the generalization error for the estimated parameter vector $\mathbf{u_0}$ as

$$\widehat{E}_{\text{generalization}}(\mathbf{u_0}) \approx \frac{(1+N_{\mathcal{M}}/p)}{(1-N_{\mathcal{M}}/p)}\widehat{E}_{\text{learn}}(\mathbf{u_0}) \tag{A.27}$$

This expression is called Akaike's Final Prediction Error estimate of the generalization error [Akaike, 1968] and it makes it possible to estimate the generalization error from a given network configuration, whenever the number of parameters in the network and the learning error of a given learning set with p examples are known.

The generalization error can be used to compare the performance of different network configurations, when they are trained at the same training set.

This formula can only be used when no regularization is included in the minimized cost-function. In the next section the calculations when a simple weight-decay term is included in the cost-function are done.

Cost-function with regularization

With regularization a term is added to the cost-function in equation (A.15). This term regularizes the parameters in the neural network model. The most used regularization method appends a simple weight-decay term to the the cost-function. The parameters (weights) are punished by a quadratic term in the parameter values. This means that the parameters are regularized in the direction of zero. When using the quadratic cost-functions, the simple weight-decay term and assuming that there is only one output from the neural network, the minimized cost-function can be written as

$$\begin{aligned} F_p(\mathbf{u}) &= E_{\text{quadratic}}(\mathbf{u}) + E_{\text{regularization}}(\mathbf{u}) \\ &= \frac{1}{p}\sum_{\mu=1}^{p}(e^{\mu})^2 + \frac{\alpha}{p}\sum_{l=1}^{N_{\mathcal{M}}} u_l^2 \\ &= \frac{1}{p}\sum_{\mu=1}^{p}(\zeta^{\mu} - O^{\mu})^2 + \frac{\alpha}{p}\sum_{l=1}^{N_{\mathcal{M}}} u_l^2 \end{aligned} \tag{A.28}$$

where F_p is equivalent to E_p for the cost-function without weight-decay. u_l is the values of the weights in network and $N_{\mathcal{M}}$ is the number of weights.

By using a second order Taylor approximation developed around the 'correct' parameter vector this can be written as

$$F_p(\mathbf{u}) \approx F_p(\mathbf{u}^*) + \left(\frac{\partial F_p(\mathbf{u}^*)}{\partial \mathbf{u}^*}\right)^T (\mathbf{u}^* - \mathbf{u}) + \frac{1}{2}(\mathbf{u}^* - \mathbf{u})^T \mathbf{G_p} (\mathbf{u}^* - \mathbf{u}) \tag{A.29}$$

where $\mathbf{G_p}$ is the Hessian matrix of the cost-function F_p in the parameter vector $\mathbf{u_0}$ estimated based on a learning set of p examples.

By differentiating this we find that

$$\frac{\partial F_p(\mathbf{u})}{\partial \mathbf{u}} \approx -\frac{\partial F_p(\mathbf{u}^*)}{\partial \mathbf{u}^*} - \mathbf{G_p}\left(\mathbf{u}^* - \mathbf{u}\right) \tag{A.30}$$

When $\mathbf{u} = \mathbf{u}_0$ the derivative is zero. Therefore the difference between the 'correct' parameter vector and the parameter vector estimated based on the a training set of p examples can be calculated as

$$\left(\mathbf{u}^* - \mathbf{u_0}\right) \approx -(\mathbf{G_p})^{-1}\frac{\partial F_p(\mathbf{u}^*)}{\partial \mathbf{u}^*} \tag{A.31}$$

Using equation (A.28) the derivative and second order derivative of the cost-function can be calculated as

$$\frac{\partial F_p(\mathbf{u})}{\partial \mathbf{u}} = -\frac{2}{p}\sum_{\mu=1}^{p}\left(\frac{\partial O^\mu(\mathbf{u})}{\partial \mathbf{u}}\right)(e^\mu) + \frac{2\alpha}{p}\sum_{l=1}^{N_{\mathcal{M}}} u_l \tag{A.32}$$

$$\begin{aligned}
\frac{\partial^2 F_p(\mathbf{u})}{\partial \mathbf{u}^2} &= \mathbf{G_p} = \frac{2}{p}\sum_{\mu=1}^{p}\left(\frac{\partial O^\mu(\mathbf{u})}{\partial \mathbf{u}}\right)\left(\frac{\partial O^\mu(\mathbf{u})}{\partial \mathbf{u}}\right)^T - \left(\frac{\partial^2 O^\mu(\mathbf{u})}{\partial \mathbf{u}^2}\right)(e^\mu) + \frac{2\alpha}{p}\mathbf{I} \\
&\approx \frac{2}{p}\sum_{\mu=1}^{p}\left(\frac{\partial O^\mu(\mathbf{u})}{\partial \mathbf{u}}\right)\left(\frac{\partial O^\mu(\mathbf{u})}{\partial \mathbf{u}}\right)^T + \frac{2\alpha}{p}\mathbf{I} \\
&\approx \mathbf{H_p} + \frac{2\alpha}{p}\mathbf{I}
\end{aligned} \tag{A.33}$$

where $\mathbf{I}$ is an identity matrix with the dimension of the parameter vector and $\mathbf{H_p}$ is the same Hessian matrix as calculated in the situation without weight-decay equation (A.20). The approximation used is again the Levenberg-Marquart approximation, which holds for large training sets where ($p \approx \infty$). The error e^μ can then be approximated by a white noise sequence independent of the output O^μ from the network.

Using these two equations together with (A.31), $(\mathbf{u}^* - \mathbf{u_0})$ can be calculated as

$$\left(\mathbf{u}^* - \mathbf{u_0}\right) \approx \left(\mathbf{H_p} + \frac{2\alpha}{p}\mathbf{I}\right)\left(\frac{2}{p}\sum_{\mu=1}^{p}\left(\frac{\partial O^\mu(\mathbf{u})}{\partial \mathbf{u}}\right)(e^\mu) + \frac{2\alpha}{p}\sum_{l=1}^{N_{\mathcal{M}}} u_l\right) \tag{A.34}$$

As for the case with no regularization the term that is unknown in equation (A.14) is $< \left(\mathbf{u}^* - \mathbf{u_0}\right)\left(\mathbf{u}^* - \mathbf{u_0}\right)^T >_p$. This can using equation (A.34) be calculated as

$$
\begin{aligned}
&\left\langle (\mathbf{u}^* - \mathbf{u})(\mathbf{u}^* - \mathbf{u})^T \right\rangle_p \\
&= \left\langle \left((\mathbf{H_p} + \frac{2\alpha}{p}\mathbf{I})^{-1} \left(\frac{2}{p} \sum_{\mu=1}^{p} \left(\frac{\partial O^\mu(\mathbf{u})}{\partial \mathbf{u}} \right) (e^\mu) + \frac{2\alpha}{p} \sum_{l=1}^{N_{\mathcal{M}}} u_l \right) \right) \right. \\
&\quad \left. \left(\left(\frac{2}{p} \sum_{\mu'=1}^{p} (e^{\mu'})^T \left(\frac{\partial O^{\mu'}(\mathbf{u})}{\partial \mathbf{u}} \right)^T + \frac{2\alpha}{p} \sum_{l=1}^{N_{\mathcal{M}}} u_l \right) (\mathbf{H_p} + \frac{2\alpha}{p}\mathbf{I})^{-1} \right) \right\rangle_p \\
&= \frac{4}{p^2}(\mathbf{H_p} + \frac{2\alpha}{p}\mathbf{I})^{-1} \left\langle \sum_{\mu=1}^{p} \left(\left(\frac{\partial O^\mu(\mathbf{u})}{\partial \mathbf{u}} \right) (e^\mu)(e^\mu)^T \left(\frac{\partial O^\mu(\mathbf{u})}{\partial \mathbf{u}} \right)^T \right) \right\rangle_p (\mathbf{H_p} + \frac{2\alpha}{p}\mathbf{I})^{-1} + \\
&\quad (\mathbf{H_p} + \frac{2\alpha}{p}\mathbf{I})^{-1} \left(\frac{2\alpha}{p} \right)^2 \left\langle \sum_{l=1}^{N_{\mathcal{M}}} u_l \sum_{l=1}^{N_{\mathcal{M}}} u_l \right\rangle_p (\mathbf{H_p} + \frac{2\alpha}{p}\mathbf{I})^{-1} \qquad \text{(A.35)} \\
&= \frac{4}{p^2}(\mathbf{H_p} + \frac{2\alpha}{p}\mathbf{I})^{-1} E^* \left\langle \sum_{\mu=1}^{p} \left(\left(\frac{\partial O^\mu(\mathbf{u})}{\partial \mathbf{u}} \right) \left(\frac{\partial O^\mu(\mathbf{u})}{\partial \mathbf{u}} \right)^T \right) \right\rangle_p (\mathbf{H_p} + \frac{2\alpha}{p}\mathbf{I})^{-1} + \\
&\quad \left(\frac{2\alpha}{p} \right)^2 (\mathbf{H_p} + \frac{2\alpha}{p}\mathbf{I})^{-1} \left\langle \sum_{l=1}^{N_{\mathcal{M}}} u_l \sum_{l=1}^{N_{\mathcal{M}}} u_l \right\rangle_p (\mathbf{H_p} + \frac{2\alpha}{p}\mathbf{I})^{-1} \qquad \text{(A.36)} \\
&\approx \frac{2}{p}(\mathbf{H_p} + \frac{2\alpha}{p}\mathbf{I})^{-1} E^* \mathbf{H_p} (\mathbf{H_p} + \frac{2\alpha}{p}\mathbf{I})^{-1} + \\
&\quad \left(\frac{2\alpha}{p} \right)^2 (\mathbf{H_p} + \frac{2\alpha}{p}\mathbf{I})^{-1} \left\langle \sum_{l=1}^{N_{\mathcal{M}}} u_l \sum_{l=1}^{N_{\mathcal{M}}} u_l \right\rangle_p (\mathbf{H_p} + \frac{2\alpha}{p}\mathbf{I})^{-1} \qquad \text{(A.37)}
\end{aligned}
$$

where it in (A.36) is assumed that the noise e^μ is white. In (A.37) the expression from equation (A.33) for $\mathbf{H_p}$ is filled in.

Using this result the learning and generalization error can be found using equation (A.13) and (A.14), where the best estimator that can be found for $\mathbf{H}$ when only one learning set consisting of p examples is applicable, is $\mathbf{H_p}$. Using this it can be found that

$$
\begin{aligned}
&E_{\text{learn}}(\mathbf{u_0}) \\
&\approx E_{\text{learn}}(\mathbf{u}^*) - \frac{1}{2} tr \left(\mathbf{H_p} (\mathbf{u}^* - \mathbf{u_0})(\mathbf{u}^* - \mathbf{u_0})^T \right) \\
&= E_{\text{learn}}(\mathbf{u}^*) - \frac{1}{2} tr \left(\mathbf{H_p} \frac{2}{p} (\mathbf{H_p} + \frac{2\alpha}{p}\mathbf{I})^{-1} E^* \mathbf{H_p} (\mathbf{H_p} + \frac{2\alpha}{p}\mathbf{I})^{-1} \right) - \\
&\quad \frac{1}{2} tr \left(\left(\frac{2\alpha}{p} \right)^2 \mathbf{H_p} (\mathbf{H_p} + \frac{2\alpha}{p}\mathbf{I})^{-1} \sum_{l=1}^{N_{\mathcal{M}}} u_l \sum_{l=1}^{N_{\mathcal{M}}} u_l (\mathbf{H_p} + \frac{2\alpha}{p}\mathbf{I})^{-1} \right) \\
&= E_{\text{learn}}(\mathbf{u}^*) \left(1 - \frac{1}{p} tr \left(\mathbf{H_p} (\mathbf{H_p} + \frac{2\alpha}{p}\mathbf{I})^{-1} \mathbf{H_p} (\mathbf{H_p} + \frac{2\alpha}{p}\mathbf{I})^{-1} \right) \right) - \\
&\quad \left(\frac{2}{p} \right)^2 \frac{1}{2} tr \left(\alpha^2 \mathbf{H_p} (\mathbf{H_p} + \frac{2\alpha}{p}\mathbf{I})^{-1} \sum_{l=1}^{N_{\mathcal{M}}} u_l \sum_{l=1}^{N_{\mathcal{M}}} u_l (\mathbf{H_p} + \frac{2\alpha}{p}\mathbf{I})^{-1} \right)
\end{aligned}
$$

$$= \quad E_{\text{learn}}(\mathbf{u}^*)\left(1 - \frac{N_{\text{eff}}}{p}\right) - \frac{2}{p^2}\sum_{l=1}^{N_{\mathcal{M}}}\sigma_l\frac{\alpha^2 u_l^2}{(\sigma_l + 2\alpha/p)^2} \tag{A.38}$$

$$\begin{aligned} E_{\text{generalization}}(\mathbf{u_0}) &\approx E_{\text{learn}}(\mathbf{u}^*) + \frac{1}{2}tr\left(\mathbf{H_p}\,(\mathbf{u_0} - \mathbf{u}^*)\,(\mathbf{u_0} - \mathbf{u}^*)^T\right) \\ &= E_{\text{learn}}(\mathbf{u}^*)\left(1 + \frac{N_{\text{eff}}}{p}\right) + \frac{2}{p^2}\sum_{l=1}^{N_{\mathcal{M}}}\sigma_l\frac{\alpha^2 u_l^2}{(\sigma_l + 2\alpha/p)^2} \end{aligned} \tag{A.39}$$

where the parameter N_{eff} can be calculated as

$$\begin{aligned} N_{\text{eff}} &= tr\left(\mathbf{H_p}(\mathbf{H_p} + \frac{2\alpha}{p}\mathbf{I})^{-1}\mathbf{H_p}(\mathbf{H_p} + \frac{2\alpha}{p}\mathbf{I})^{-1}\right) \\ &= \sum_{l}^{N_{\mathcal{M}}}\left(\frac{\sigma_l}{\sigma_l + 2\alpha/p}\right)^2 \end{aligned} \tag{A.40}$$

where $N_{\mathcal{M}}$ is the dimension of the parameter vector and σ_l is the l'th eigenvalue in the Hessian matrix $\mathbf{H_p}$.

In equation (A.38) and (A.39) there is a second order term in the regularization parameter α. For small values of this regularization parameter these equations can be approximated with the first order approximation in the regularization parameter, so they are reduced to

$$\widehat{E}_{\text{learn}}(\mathbf{u_0}) \approx E_{\text{learn}}(\mathbf{u}^*)\left(1 - \frac{N_{\text{eff}}}{p}\right) \tag{A.41}$$

$$\widehat{E}_{\text{generalization}}(\mathbf{u_0}) \approx E_{\text{learn}}(\mathbf{u}^*)\left(1 + \frac{N_{\text{eff}}}{p}\right) \tag{A.42}$$

This result shows to be the same result as calculated for the case without weight-decay. The number of parameters $N_{\mathcal{M}}$ is replaced by N_{eff} in equation (A.40). From this equation it can be seen that for α equal to zero N_{eff} is exactly $N_{\mathcal{M}}$ the number of parameters in the network. N_{eff} is called the **effective number of parameters** in the network.

Using the results from equation (A.41) and (A.42) the generalization ability can be calculated as

$$\widehat{E}_{\text{generalization}}(\mathbf{u_0}) \approx \frac{(1 + N_{\text{eff}}/p)}{(1 - N_{\text{eff}}/p)}\widehat{E}_{\text{learn}}(\mathbf{u_0}) \tag{A.43}$$

This equation can be used for calculation of the regularization ability of the neural network for different sizes of the network, where the quadratic cost-function with weight-decay is used as minimization criteria when estimating the parameters.

All the calculations done with weight-decay are done with only one weight-decay parameter common for all the parameters. If more than one weight-decay parameter is used (the limit is individual weight-decay parameters for all the weights) most of the calculations are not changed. It is only the way the effective number of parameters in the network is calculated in equation (A.40) that has to be changed. It can instead be written as

$$N_{\text{eff}} = \sum_{l}^{N_1} \left(\frac{\sigma_l}{\sigma_l + 2\alpha_1/p} \right)^2 + \ldots + \sum_{l}^{N_k} \left(\frac{\sigma_l}{\sigma_l + 2\alpha_k/p} \right)^2 \tag{A.44}$$

where k is the number of different groups of weights, N_1 and N_k is the number of weights in each group, and α_1 and α_k is the weight-decay parameter for each group.

A.1.2 Adaptive regularization by tuning weight-decay parameters

Methods for tuning the weight-decay parameter introduced in the previous section can be found using the equation for the generalization error (A.39). The method is a gradient descent algorithm using the derivative of the generalization error with respect to the weight-decay parameter α to optimize the weight-decay parameter. Therefore the full-expression in equation (A.39) has to be used, when differentiating the expression for the generalization error. When differentiating the equation it can be found that

$$\begin{aligned}
&\frac{\partial E_{\text{generalization}}(\mathbf{u_0})}{\partial \alpha} \\
&= \frac{\partial \left(E_{\text{learn}}(\mathbf{u}^*) \left(1 + \frac{N_{\text{eff}}}{p}\right)\right)}{\partial \alpha} + \frac{\partial \left(\frac{2}{p^2} \sum_{l=1}^{N_{\mathcal{M}}} \sigma_l \left(\frac{\alpha u_l}{\sigma_l + 2\alpha/p} \right)^2 \right)}{\partial \alpha} \\
&= \frac{-4E_{\text{learn}}(\mathbf{u}^*)}{p^2} \sum_{l=1}^{N_{\mathcal{M}}} \frac{\sigma_l^2}{(\sigma_l + 2\alpha/p)^3} + \\
&\quad \frac{4}{p^2} \sum_{l=1}^{N_{\mathcal{M}}} \left(\frac{\alpha \sigma_l u_l^2}{(\sigma_l + 2\alpha/p)^2} - \frac{2\alpha^2}{p} \frac{\sigma_l u_l^2}{(\sigma_l + 2\alpha/p)^3} \right) \\
&= -\frac{4E_{\text{learn}}(\mathbf{u}^*)}{p^2} \sum_{l=1}^{N_{\mathcal{M}}} \frac{\sigma_l^2}{(\sigma_l + 2\alpha/p)^3} + \\
&\quad \frac{4}{p^2} \sum_{l=1}^{N_{\mathcal{M}}} \frac{\alpha \sigma_l u_l^2 (\sigma_l + 2\alpha/p) - 2\alpha^2/p \sigma_l u_l^2}{(\sigma_l + 2\alpha/p)^3} \\
&= -\frac{4E_{\text{learn}}(\mathbf{u}^*)}{p^2} \sum_{l=1}^{N_{\mathcal{M}}} \frac{\sigma_l^2}{(\sigma_l + 2\alpha/p)^3} + \frac{4}{p^2} \sum_{l=1}^{N_{\mathcal{M}}} \frac{\alpha (\sigma_l u_l)^2}{(\sigma_l + 2\alpha/p)^3} \\
&= \frac{4}{p^2} \sum_{l=1}^{N_{\mathcal{M}}} \frac{(\alpha u_l^2 - E_{\text{learn}}(\mathbf{u}^*)) \sigma_l^2}{(\sigma_l + 2\alpha/p)^3}
\end{aligned} \tag{A.45}$$

In this equation the only unknown parameter is $E_{\text{learn}}(\mathbf{u}^*)$. This can be found using equation (A.38). Then by using the simple gradient descent algorithm in the parameter α

it is possible to minimize the generalization error in this parameter. The update scheme for the weight-decay parameter can then be written as

$$\alpha(n+1) = \alpha(n) - \rho \frac{\partial E_{\text{generalization}}(\mathbf{u_0})}{\partial \alpha}, \qquad (given \quad \alpha = \alpha(n)) \tag{A.46}$$

Equation (A.45) can easily be changed to incorporate individual weight-decay on each parameter or group of parameters, while none of the equations assumes that it is the same weight-decay that is used at all the parameters in the network. This situation is often the case where individual weight-decay is used on the input layer weights and the output layer weights in a 2 layer neural network. The equation for the derivative of the generalization error, in the situation where the same weight-decay α_w is used on the parameters in a given group of parameters (size of group N_w), can then be written as

$$\frac{\partial E_{\text{generalization}}(\mathbf{u_0})}{\partial \alpha_w} = \frac{4}{p^2} \sum_{l=1}^{N_w} \frac{(\alpha_w u_l^2 - E_{\text{learn}}(\mathbf{u}^*))\sigma_l^2}{(\sigma_l + 2\alpha_w/p)^3} \tag{A.47}$$

and equation (A.46) is then used to update the weight-decays for all the groups of parameters in the network individually.

Appendix B

ICNN-93 article: On Design and Evaluation of Tapped-Delay Neural Network Architectures.

Reference for the article: [Svarer et al., 1993a]

On Design and Evaluation of Tapped-Delay Neural Network Architectures

Claus Svarer, Lars Kai Hansen, and Jan Larsen
CONNECT, Electronics Institute, B349
Technical University of Denmark,
DK-2800 Lyngby, Denmark
emails: claus, lars, jan@eiffel.ei.dth.dk

Abstract— **We address pruning and evaluation of Tapped-Delay Neural Networks for the sunspot benchmark series. It is shown that the generalization ability of the networks can be improved by pruning using the Optimal Brain Damage method of Le Cun, Denker and Solla. A stop criterion for the pruning algorithm is formulated using a modified version of Akaike's Final Prediction Error estimate. With the proposed stop criterion the pruning scheme is shown to produce succesful architectures with a high yield.**

I. Introduction

Needless to say, processing of time series is an important application area for neural networks, and the quest for application-specific architectures penetrates current network research. While the ultimate tool may be fully recurrent architectures, many problems arise during adaptation of these. Even worse, the generalization properties of recurrent networks are not well understood, hence, model optimization is difficult. However, the conventional Tapped-Delay Neural Net (TDNN) [11] may be analysed using statistical methods and the results of such analysis can be applied for model optimization. Here we demonstrate the power of this strategy within time series prediction. We aim at designing compact TDNN's using the so-called *Optimal Brain Damage* (OBD) method of Le Cun *et al.* [5]. The benefits from compact architectures are three-fold: They generalize better, they carry less computational burden, and they are faster to adapt if the environment changes. Further we show that the generalization ability of the network may be estimated, without extensive cross-validation, using a modification of Akaike's *Final Prediction Error* (FPE) estimate [1].

II. Time Series Prediction

The possibility of predicting the future fascinates. The techniques invoked through history cover oracles, crystal balls, feed forward neural networks and many more. While we can rule out long time predictions for chaotic systems; short time predictions may still be viable. Recent work by Priestly [9], and Weigend *et al.* [11] have established the *sunspot series* as a benchmark for time series prediction algorithms. The series is a scaled record of the yearly average sunspot activity for the period 1700-1979. The sunspot series is believed to be generated by a noisy, chaotic, dynamical system. The spectrum is dominated by a frequency corresponding to a 12 year period. Weigend *et al.* applied a weight decay scheme to design feed-forward networks that generalize better than conventional algorithms from the training set to an independent test set. In this work the OBD *pruning* method is shown to produce very compact networks for this problem, having only around 15 free parameters. The networks that we obtain use around one third of the parameters of the network published by Weigend *et al.* while having comparable performance.

We start the pruning procedure from the same initial network configuration as in [11]. The network is a *tapped delay line* architecture with 12 input units, 8 hidden sigmoid units and a single linear output unit, see Fig. 1. The initial network is fully connected between layers and implements a non-linear mapping from lag space $\mathbf{z}(k) = [x(k-1), ..., x(k-L)]$, $L = 12$, to the real axis:

$$\widehat{x}(k) = F_{\mathbf{u}}(\mathbf{z}(k)) \qquad \widehat{x} \in \mathcal{R}, \tag{1}$$

where $\mathbf{u}$=(w, W) is the N-dimensional weight vector and $\widehat{x}(k)$ is the prediction of $x(k)$.

The non-linear mapping can be written as:

$$F_{\mathbf{u}}(\mathbf{z}(k)) = \sum_{j=1}^{n_H} W_j \tanh\left(\sum_{i=1}^{L} w_{ij}x(k-i) + w_{i0}\right) + W_0, \tag{2}$$

where n_H is the number of hidden units. W_j are the hidden-to-output weights while w_{ij} connect the input and hidden units.

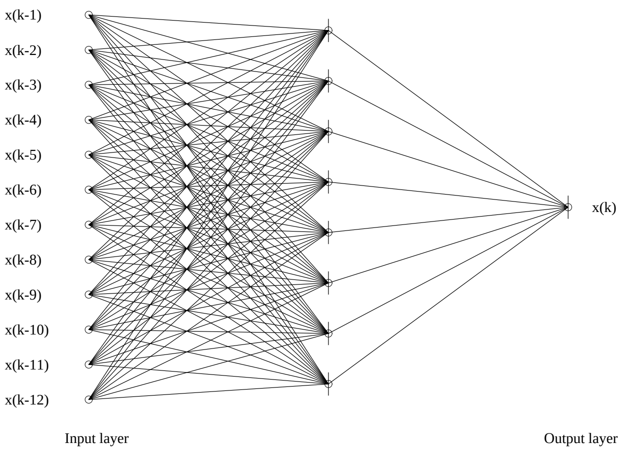


Fig. 1 Fully connected network used as a starting point for the pruning procedure. A vertical bar through a unit indicates an active threshold.

III. Training

The objective of the training procedure is single-step prediction. Hence, the network weights, $\mathbf{u}$, are trained to recognize the short time structure of the chaotic time series. We use the sum of squared errors to measure the prediction ability of the current network:

$$E_{train} = \frac{1}{p} \sum_{k=1}^{p} \left[x(k) - F_{\mathbf{u}}(\mathbf{z}(k))\right]^2 , \tag{3}$$

where p is the number of training examples.

A state of the art simulator has been developed based on *batch mode*, second order local optimization. The second order scheme is implemented as a direct matrix-inversion identification of the hidden-to-output weights [2], while a pseudo Gauss-Newton method is used for identification of input-to-hidden weights, see e.g. [3].

To ensure numerical stability and for assisting the pruning procedure we augment the cost-function with a weight decay term. The cost-function can then be written as:

$$E = E_{train} + \frac{\alpha_w}{p} \sum_{ij}^{N_w} w_{ij}^2 + \frac{\alpha_W}{p} \sum_{j}^{N_W} W_j^2, \tag{4}$$

where N_w, N_W are the numbers of weights and thresholds in hidden and output units, respectively.

The second order pseudo Gauss-Newton method used for identification of input-to-hidden weights can be written as:

$$\Delta w_{ij} = -\eta \left(\frac{\partial E_{train}}{\partial w_{ij}} + \frac{2\alpha_w}{p} w_{ij} \right) \Big/ \left(\frac{\partial^2 E_{train}}{\partial w_{ij}^2} + \frac{2\alpha_w}{p} \right) \tag{5}$$

where the parameter η is used to secure that all the weight updates lead to a decrease in the cost-function. η is initialized to 1 before each step, and iteratively dimished by powers of two, until the step leads to a decrease in the cost-function. As in [5] we approximate the second derivative by the positive semi-definite expression:

$$\frac{\partial^2 E_{train}}{\partial w_{ij}^2} \approx \frac{2}{p} \sum_{k=1}^{p} \left(\frac{\partial F_{\mathbf{u}}(\mathbf{z}(k))}{\partial w_{ij}} \right)^2 . \tag{6}$$

IV. Pruning by Optimal Brain Damage

The OBD scheme proposed by Le Cun *et al.* [5] was succesfully applied to reduce large networks for recognition of handwritten digits [6]. The basic idea is to estimate the effect on the *training error* when deleting weights. The estimate is formulated in terms of weight saliencies s_l:

$$\delta E_{train} = \sum_{l \in D} s_l \equiv \sum_{l \in D} \left(\frac{2\alpha}{p} + \frac{1}{2} \frac{\partial^2 E_{train}}{\partial u_l^2} \right) u_l^2, \tag{7}$$

where u_l is a component of $\mathbf{u}$. The saliency definition used here takes into account that the weight decay terms force the weights to depart from the minimum of the training set error. The sum runs over the set of deleted weights D.

The following assumptions enter the derivation of OBD:

- The terms of third and higher orders in the deleted weights can be neglected.
- The off-diagonal terms in the Hessian, $\frac{\partial^2 E_{train}}{\partial u_l \partial u_{l'}}$, can be neglected.

Computationally the second order (diagonal) terms are reused from the training scheme (5), in particular we refrain from working on the full Hessian, which would scale poorly for large networks.

The recipe allows for *ranking* the weights according to saliency. The question of how many weights it may be possible to delete was not answered in [5]. To evaluate a network, hence, formulate a pruning *stop criterion*, we note that there are three objectives of pruning:

- Improve the generalization performance by limiting the network resources.
- Reduce the computational burden of prediction.
- Allow for fast on-line adaptation.

In this presentation we emphasize the first of these objectives. However, since the generalization error by definition, involves test on an independent data set, we cannot directly use the error on the training set, as estimated by OBD, to formulate a stop criterion. We may indeed accept an increased error on the training set if better generalization is obtained. Also, among networks with the same estimated test error we still prefer the minimal, because it has a lower computational burden, and typically needs less training examples for retraining if the environment changes. The latter is very important for on-line adaptation. If data are abundant we can formulate a stop criterion based on a validation set (an independent subset of the training set). This approach was criticized by Weigend *et al.*: The training data set is scarce for the sunspot series – and indeed for many other applications. We support their conclusion by the observation that even for the (90%)/(10%) splitting of the training set (in training and validation sets, respectively), as used by [11], the estimated validation error is an extremely noisy quantity.

In the usual case of limited data sets we follow the standard approach within *system identification* [7] and estimate the generalization error of the pruned networks using statistical arguments. In particular, we apply Akaike's FPE estimate [1, 7] of the test error in terms of the training error. In its standard form it reads:

$$\widehat{E}_{test} = \frac{p+N}{p-N} E_{train}, \tag{8}$$

where p is the number of training samples, and N is the number of parameters in the model. The left hand side of (8) is the average generalization error, averaged over all possible training sets of size p. The estimate is based on linearization of the networks as regards the fluctuations in the weights resulting from different training sets. The relation express the fact that the training error is a biased estimate of the noise level because each parameter during training has "absorbed" some of the noise in the training samples.

Since we have regularized the training procedure by weight-decay terms α_w, α_W, hence, suppressed the ability of the (otherwise) ill-determined parameters to model noise, we need to modify the classical FPE estimate by replacing the total number of parameters with the *effective* number of parameters see e.g. [4, 8]:

$$\widehat{E}_{test} = \frac{p+N_{eff}}{p-N_{eff}} E_{train}, \tag{9}$$

$$N_{eff} = \sum_{ij}^{N_w} \left(\frac{\lambda_{ij}}{\lambda_{ij} + 2\alpha_w/p} \right)^2 + \sum_{j}^{N_W} \left(\frac{\Lambda_j}{\Lambda_j + 2\alpha_W/p} \right)^2 . \tag{10}$$

Where the λ's are the second derivatives already computed in (7), $\lambda_{ij} \equiv \partial^2 E_{train} / \partial w_{ij}^2$, $\Lambda_j \equiv \partial^2 E_{train} / \partial W_j^2$.

In brief, the following assumptions enter the derivation of (8-10):

- Independence of input and error on output.
- Sufficient capacity, i.e., the network must be able to implement the rule.
- Many examples pr. weight: $N/p \rightarrow 0$.
- The off-diagonal elements of the second derivative matrix can be neglected.

With the above tool we can obtain a generalization error estimate for each pruned network. By selecting the network with the lowest estimated generalization error we have developed the stop criterion sought.

V. Experiments

Following Weigend *et al.* [11], the sunspot data are partioned into a training set (1700-1920) and a test set (1921-1979), and further we compute the test error for two separate sets, namely the periods 1921-1955 and 1956-1979. The sunspot series is rather non-stationary and the latter period is atypical for the series as a whole. The normalized errors on the training set and on the two test sets are calculated as:

$$E_{set} = \frac{1}{\sigma_{total}^2 \cdot p_{set}} \sum_{k=1}^{p_{set}} \left[x(k) - F_{\mathbf{u}}(\mathbf{z}(k)) \right]^2 , \tag{11}$$

where p_{set} is the number of examples in the data set in question. The squared errors of each data set are normalized as in [11] by the variance of the total data set σ_{total}^2.

An ensemble of 11 networks were trained and pruned. The weight decay parameters were set as: $\alpha_w = 0.02$ and $\alpha_W = 0.01$.

Fig. 2 shows the normalized training error and the two test set errors during training of the fully connected network. Note that the training set error decreases monotonously, while the error on the test sets start out decreasing, but after some training increase again. This is a generic *over-training scenario* in which the network overfits the training set.

In order to prevent the network from overfitting we limit its resources by pruning. We use OBD, rank the remaining weights according to saliency, and delete a number of these determined by $|D| = \lceil 0.02 \cdot N_{remaining} \rceil$, in a simple iterative procedure. The evolution of training and test errors during pruning are recorded for a specific network in Fig. 3. Further this figure shows the estimated test error as given by (9). The FPE estimate of the test error lies

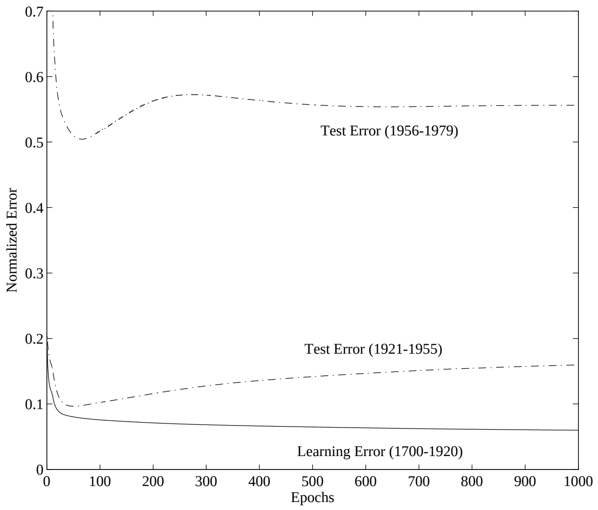


Fig. 2 Training and test error when training the fully connected network. An 'Epoch' is a full sweep through the training set.

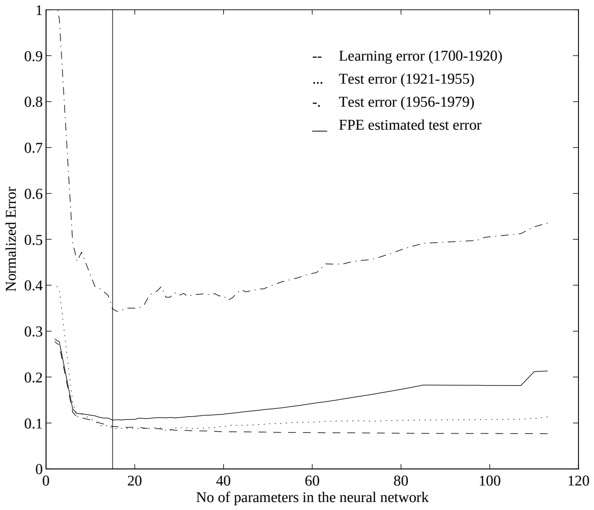


Fig. 3 The evolution of training and test errors during pruning. The FPE estimate of the test error is based on equation (9). The vertical line indicates the network for which the *estimated* test error is minimal.

between the two test sets. Most importantly, the estimate reproduce the common trend of the test sets, sharing a generalization error minimum just below 20 parameters, indicating that the statistical approach is viable. In the particular run, the stop criterion selects a network having 12 weights and 3 thresholds as indicated by the vertical line in Fig. 3.

Among the eleven networks pruned, the procedure selected 9 nets with a number of parameters in the range $12-16$, and two nets with more than 25 parameters. The 9 small architectures were considered succesfully pruned, and used as an ensemble for computing significance levels for the procedure. In order to fine tune the nine small networks, they were retrained without weight decay, resulting in the normalized errors on the three data sets: 0.090 ± 0.001 (1700-1920), 0.082 ± 0.007 (1921-55), and 0.35 ± 0.05 (1956-79). Four nets had 15 parameters, another four nets had 16 parameters, while one net had 12 parameters. The latter appears to be overpruned and carried higher errors than the rest. In Table I we compare our findings with other reported results for the sunspot series.

We illustrate the properties of one of the compact networks (presented in Fig. 4 and in Table II), in two ways: First we show, in Fig. 5, the retraining history of this network after pruning. As expected we see no overtraining. Both the training error and the two test errors are monotonously decreasing.

It is interesting to note that the network does not use

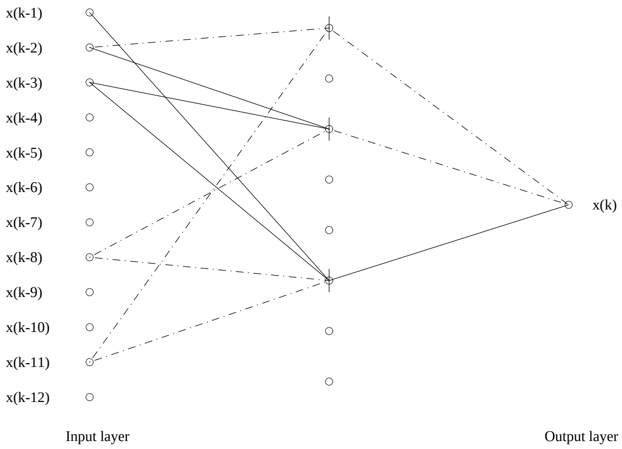


Fig. 4 Pruned network with 12 weights and 3 thresholds. Note that the network only uses a subset of the Tapped-Delay line. Dash-dotted lines indicate negative weights, and solid lines positive weights. The network parameters are given in Table II.

the full set of inputs. We interpret this to be a result of the finite training set. Within the noise level of the sunspot time series it is harmful to use more than a carefully selected subset of the available lag space. Among the succesfully pruned networks there is some consensus regarding which inputs to use. They all use the two most recent inputs, and one or more inputs among the "oldest" part of lag space.

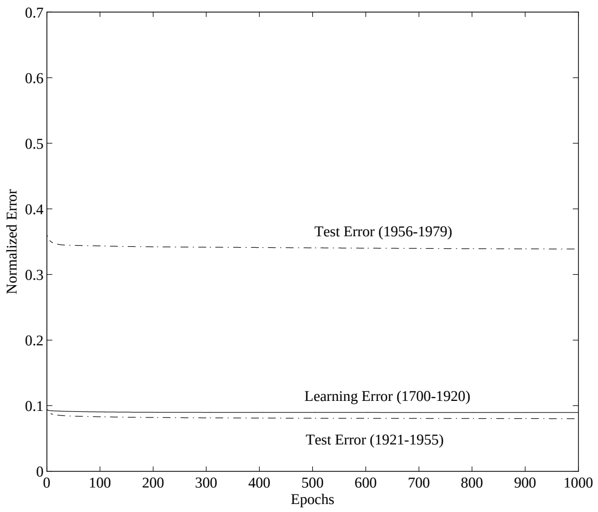


Fig. 5 Training and test error when re-training the pruned network without weight decay.

Secondly, in Fig. 6, the predicted sunspot activity using the pruned and fine tuned network is shown. We note that the variance of the sunspot activity is increased significantly in the period after about 1960.

In Fig. 7 we show the effective number of parameters as computed from (10). Judged from this figure the weight decays are important in the initial pruning phase where they limit the number of parameters to about 85. With higher weight decays we were unable to train the networks to error levels like those reported. On the other hand, with smaller weight decays the second order optimization scheme is plagued by numerical problems also leading to higher errors and a lower yield of useful architectures.

VI. Conclusion

We have discussed pruning and evaluation of Tapped-Delay Neural Networks. We have shown that the generalization ability on the sunspot data can be improved by pruning using the Optimal Brain Damage method. In particular, we have identified a set of compact networks with three hidden units employing around 15 weights and thresholds. These networks generalize well compared to

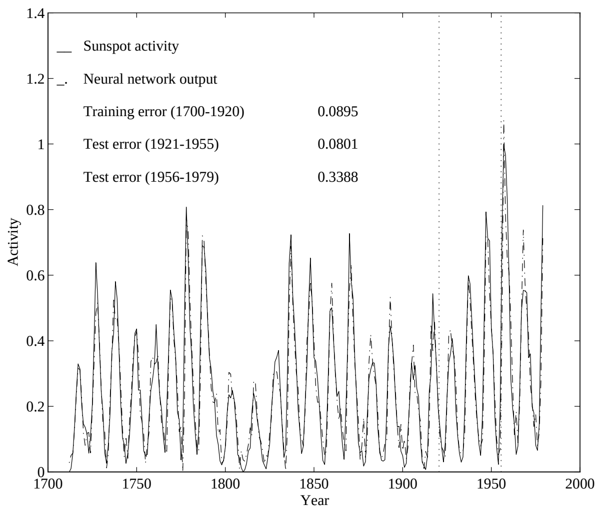


Fig. 6 Predicted sunspot activity using the pruned feed-forward network shown in Fig. 4.

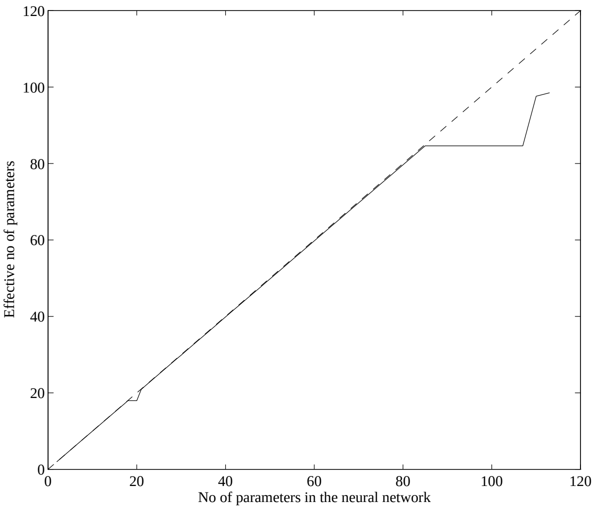


Fig. 7 The effective number of parameters in the neural network during the pruning session.

TABLE I
NORMALIZED ERROR

Model	Train (1700-1920)	Test (1921-55)	Test (1956-79)	Number of parameters
Tong and Lim [10]	0.097	0.097	0.28	16
Weigend *et al.* [11]	0.082	0.086	0.35	43
Linear model[1]	0.132	0.130	0.37	13
Fully connected network[2]	0.078 ± 0.002	0.104 ± 0.005	0.46 ± 0.07	113
Pruned network[3]	0.090 ± 0.001	0.082 ± 0.007	0.35 ± 0.05	$12 - 16$

1) Linear model is a single linear unit. 2) The initial pre-pruned networks, trained with the same weight decay terms as used during pruning. 3) Pruned networks retrained without weight decay. The mean and standard deviation are based on the networks selected for retraining (90 % of the initial set of networks).

TABLE II
PRUNED NETWORK WEIGHTS

Input	Hidden unit 1	Hidden unit 3	Hidden unit 6
Lag 1	0	0	1.399
Lag 2	-0.562	0.944	0
Lag 3	0	1.035	1.068
Lag 8	0	-0.435	-0.408
Lag 11	-0.279	0	-0.259
Threshold	0.192	0.236	0.411

Hidden	Output unit
Unit 1	-1.1544
Unit 3	-1.5537
Unit 6	1.5636
Threshold	0

previous studies. Further we have shown that the network performance may be evaluated using statistical methods and that the trend (in error versus number of parameters) of the estimate is in good agreement with those of the test sets. We have shown how the estimated generalization error may be used for selection of the optimal network architecture during a pruning session. The yield of the procedure was 90%: Out of eleven networks the procedure found nine useful architectures.

For the sunspot series we note that non-stationarity is a problem insofar that the normalized test error for the period 1956-1979 is four times higher than the test error on the more representative test set comprising the period 1921-1955. This means, that it is important to corroborate our results on other problems and time series, such work is in progress.

Acknowledgments

This research is supported by the Danish Natural Science and Technical Research Councils through the Computational Neural Network Center (CONNECT).

References

[1] H. Akaike: "Fitting Autoregressive Models for Prediction". *Ann. Inst. Stat. Mat.* , vol. 21, 243-247, (1969).

[2] S.A. Barton: "A Matrix Method for Optimization a Neural Network". *Neural Computation*, vol. 3, 450-459 (1990).

[3] J. Hertz, A. Krogh and R.G. Palmer: *Introduction to the Theory of Neural Computation*, Addison Wesley, New York (1991).

[4] J. Larsen: *Design of Neural Network Filters.* Ph. D. Thesis, Electronics Institute, Technical University of Denmark. In preparation, (1993).

[5] Y. Le Cun, J.S. Denker, and S.A. Solla: "Optimal Brain Damage". In *Advances in Neural Information Processing Systems 2*, 598-605, Morgan Kaufman. (1990).

[6] Y. Le Cun, B. Boser, J.S. Denker, D. Henderson, R.E. Howard, W. Hubbard, and L.D. Jakel: "Handwritten Digit Recognition with a Back-Propagation Network", In *Advances in Neural Information Processing Systems 2*, 396-404. Morgan Kaufman. (1990)

[7] L. Ljung: *System Identification: Theory for the user*, Prentice-Hall, Information and System Sciences series (1987).

[8] J.E. Moody: "Note on Generalization, Regularization and Architecture Selection in Nonlinear Systems". In *Neural Networks For Signal Processing*; Proceedings of the 1991 IEEE-SP Workshop, (Eds. S.Y. Kung, B.H. Juang, and C. Kamm), IEEE Service Center, 1-10, (1991).

[9] M.B. Priestly: *Non-linear and Non-stationary Times Series Analysis*, Academic Press (1988).

[10] H. Tong and K. S. Lim: "Threshold autoregression, limit cycles and cyclical data". *Journ. Roy. Stat. Soc. B*, vol. 42, 245 (1980).

[11] A.S. Weigend, B.A. Huberman, and D.E. Rumelhart: "Prediction the future: A Connectionist Approach", *Int. J. of Neural Systems*, vol. 3, 193-209 (1990).

Appendix C

IEEE-SP93 article: Designer Networks for Time Series Processing

Reference for the article: [Svarer et al., 1993b]

DESIGNER NETWORKS FOR TIME SERIES PROCESSING

C. Svarer, L. K. Hansen, J. Larsen and C. E. Rasmussen
CONNECT, Electronics Institute B349
Technical University of Denmark,
DK-2800 Lyngby, Denmark
Tel: +45 4593 4207 / Fax: +45 4288 0117
emails: claus,lars,jan,ed@eiffel.ei.dth.dk

INTRODUCTION

The conventional Tapped-Delay Neural Net [14] may be analyzed using statistical methods and the results of such analysis can be applied to model optimization. In this presentation we review and extend recent efforts to demonstrate the power of this strategy within time series processing. We aim at designing compact networks using the so-called *Optimal Brain Damage* (OBD) method of Le Cun *et al.* [7]. The benefits from compact architectures are three-fold: Their generalization ability is at least comparable – mostly better, they carry less computational burden, and they are faster to adapt if the environment changes. Further, we show that the generalization error of the network may be estimated, without extensive cross-validation, using a modification of Akaike's *Final Prediction Error* (FPE) estimate [1]. The minimal FPE constitutes a useful stopping criterion for pruning [12].

TIME SERIES PROCESSING

Time series processing is an important application area for neural networks. While long time forecasts can be ruled out for chaotic systems, short time predictions may still be viable. Recent work by Priestly [11], and Weigend *et al.* [14] have established the *sunspot series* as a benchmark for time series prediction algorithms. Recently we showed how pruning by OBD can produce very compact networks for this problem [12]. We obtained networks using around one third of the parameters of the network published by Weigend *et al.* while having comparable performance. In this paper we corroborate our results on the sunspot series by application to two new fields, identification of chaotic systems (Mackey-Glass) and inverse modeling of transmission channels (channel equalization).

The basic network is characterized by a *tapped delay line* architecture with L input units, n_H hidden sigmoid units and a single linear output unit. The initial network is fully connected between layers and implements a non-linear mapping from lag space $\mathbf{x}(k) = [x(k), ..., x(k-L+1)]$, ($L$ is the length of the tapped delay line), to the real axis:

$$\widehat{y}(k) = F_{\mathbf{u}}(\mathbf{x}(k)) \qquad \widehat{y} \in \mathcal{R}, \tag{1}$$

where $\mathbf{u} = [\mathbf{w}, \mathbf{W}]$ is the N-dimensional weight vector and $\widehat{y}(k)$ is the prediction of the target signal $y(k)$. The non-linear mapping can be written as:

$$F_{\mathbf{u}}(\mathbf{x}(k)) = \sum_{j=1}^{n_H} W_j \tanh\left(\sum_{i=0}^{L-1} w_{ij} x(k-i) + w_{i0}\right) + W_0, \tag{2}$$

where n_H is the number of hidden units. W_j are the hidden-to-output weights while w_{ij} connect the input and hidden units.

TRAINING

The objective of the training procedure is model identification. Hence, the network weights, $\mathbf{u}$, are trained to recognize the short time structure of the time series. We use the sum of squared errors to measure the performance of the current network:

$$E_{\text{train}} = \frac{1}{p}\sum_{k=1}^{p}\left[y(k) - F_{\mathbf{u}}(\mathbf{x}(k))\right]^2, \tag{3}$$

where p is the number of training examples.

A simulator has been developed based on *batch mode*, second order local optimization. A direct matrix-inversion method is used for identification of the hidden-to-output weights [2], while a pseudo Gauss-Newton method (diagonal approximation) is used for identification of input-to-hidden weights, see e.g. [5]. To ensure numerical stability and for assisting the pruning procedure we augment the cost-function with a weight decay term. The cost-function can then be written as:

$$E = E_{\text{train}} + \frac{\alpha_w}{p}\sum_{ij}^{N_w} w_{ij}^2 + \frac{\alpha_W}{p}\sum_{j}^{N_W} W_j^2, \tag{4}$$

where N_w, N_W are the numbers of weights and thresholds in hidden and output units, respectively. Further, α_w, α_W is the weight decay parameters of the hidden and output layers, respectively.

The second order pseudo Gauss-Newton method used for identification of input-to-hidden weights can be written as:

$$\Delta w_{ij} = -\eta\left(\frac{\partial E_{\text{train}}}{\partial w_{ij}} + \frac{2\alpha_w}{p} w_{ij}\right) \Bigg/ \left(\frac{\partial^2 E_{\text{train}}}{\partial w_{ij}^2} + \frac{2\alpha_w}{p}\right) \tag{5}$$

where the gain parameter η is used to secure that all weight updates lead to a decrease in the cost-function. Before each step η is initialized to 1 and iteratively diminished by powers of two until the step leads to a decrease in the cost-function. As e.g. in [7] we approximate the second derivative by the positive semi-definite expression:

$$\frac{\partial^2 E_{\text{train}}}{\partial w_{ij}^2} \approx \frac{2}{p} \sum_{k=1}^{p} \left(\frac{\partial F_{\mathbf{u}}(\mathbf{x}(k))}{\partial w_{ij}} \right)^2 . \tag{6}$$

PRUNING BY OPTIMAL BRAIN DAMAGE

The OBD method proposed by Le Cun *et al.* [7] was successfully applied to reduce large networks for recognition of handwritten digits [8]. The basic idea is to estimate the increase in the *training error* when deleting weights. The estimate is formulated in terms of weight *saliencies* s_l:

$$\delta E_{\text{train}} = \sum_{l \in D} s_l \equiv \sum_{l \in D} \left(\frac{2\alpha}{p} + \frac{1}{2} \frac{\partial^2 E_{\text{train}}}{\partial u_l^2} \right) u_l^2 , \tag{7}$$

where u_l is a component of $\mathbf{u}$ and the sum runs over the set of deleted weights D. The saliency definition used here takes into account that the weight decay terms force the weights to depart from the minimum of the training set error.

The major assumptions entering the derivation of OBD are: 1) Terms of third and higher orders in the deleted weights can be neglected. 2) The off-diagonal terms in the Hessian, $\partial^2 E_{\text{train}} / \partial u_l \partial u_{l'}$, can be neglected. Computationally, the second order (diagonal) terms are reused from the training scheme eq. (6). We refrain from operations involving the full Hessian, which scales poorly for large networks. The recipe allows for *ranking* the weights according to saliency. The question of how many weights it may be possible to delete was answered in [12]. We applied Akaike's FPE estimate [1, 9] of the test error in terms of the training error. In its standard form it reads:

$$\widehat{E}_{\text{test}} = \frac{p+N}{p-N} E_{\text{train}} , \tag{8}$$

where p is the number of training samples, and N is the number of parameters in the model. The left hand side of eq. (8) is the average generalization error, averaged over all possible training sets of size p. The estimate is based on linearization of the networks as regards the fluctuations in the weights resulting from different training sets. For a discussion of the approximations entering this estimate see [6]. The relation expresses the fact that the training and test errors are biased estimates of the noise level because each parameter during training has "absorbed" noise from the training samples.

Since we have regularized the training procedure by weight-decay terms α_w, α_W, hence, suppressed the ability of the (otherwise) ill-determined parameters to model noise, we need to modify the standard FPE estimate by replacing the total number of parameters with the *effective* number of parameters see e.g. [6, 10, 12]:

$$\widehat{E}_{\text{test}} = \frac{p + N_{\text{eff}}}{p - N_{\text{eff}}} E_{\text{train}}, \tag{9}$$

$$N_{\text{eff}} = \sum_{ij}^{N_w} \left(\frac{\lambda_{ij}}{\lambda_{ij} + 2\alpha_w/p} \right)^2 + \sum_{j}^{N_W} \left(\frac{\Lambda_j}{\Lambda_j + 2\alpha_W/p} \right)^2 . \tag{10}$$

where the λ's are the second derivatives already computed in eq. (6), $\lambda_{ij} \equiv \partial^2 E_{\text{train}} / \partial w_{ij}^2$, $\Lambda_j \equiv \partial^2 E_{\text{train}} / \partial W_j^2$.

With the above tool we can obtain a generalization error estimate for each pruned network. By selecting the network with the lowest estimated generalization error we have developed the stop criterion sought.

EXPERIMENTS

In [12] we tested the system on the classical sunspot series. In this paper the methods are tested on two other typical signal processing problems. The first is a standard problem of nonlinear dynamics viz. the Mackey-Glass chaotic time series, and the second concerns channel equalization. For both systems errors are computed as

$$E_{\text{set}} = \frac{1}{\sigma^2_{\text{total}} \cdot p_{\text{set}}} \sum_{k=1}^{p_{\text{set}}} \left[y(k) - F_{\mathbf{u}}(\mathbf{x}(k)) \right]^2 , \tag{11}$$

where p_{set} is the number of examples in the data (train or test) set in question, and σ^2_{total} is the total variance of $y(k)$ on the training and test set.

Mackey-Glass Chaotic Dynamics

The Mackey-Glass attractor is a non-linear chaotic system described by the following equation:

$$\frac{dz(t)}{dt} = -bz(t) + a\frac{z(t-\tau)}{1 + z(t-\tau)^{10}} \tag{12}$$

where the constants are $a = 0.2$, $b = 0.1$ and $\tau = 17$. The series is resampled with sampling period 1 according to standard practice.

We aim at identifying the underlying dynamic model, from this chaotic time series. The network configuration is $L = 16$, $n_H = 10$ and we train to implement a six step ahead prediction. That is, $\mathbf{x}(k) = [z(k-6), z(k-12), \cdots, z(k-6L)]$ and $y(k) = z(k)$.

In Fig. 1 the normalized training and test errors cf. eq. (11), and the FPE error are sketched for a training set size of 500 and 1000 examples, the test set comprises 8500 examples. Weight decays were set to $\alpha_w = 0.001$, $\alpha_W = 0.001$. With higher weight decays we were unable to train the networks to error levels like those reported. On the other hand, with smaller weight decays the second order optimization scheme is plagued by numerical problems, also leading to higher errors and a lower yield of useful architectures. It is seen that the stop criterion is able to select the optimal network for both training set sizes. As expected, a small training set can only "justify" a network with few parameters.

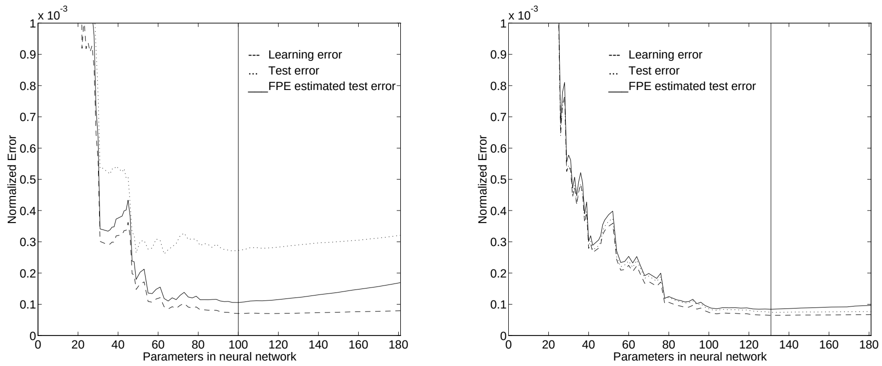


Figure 1: The evolution of training and test errors during pruning for the Mackey-Glass time series (left) training set size 500, (right) training set size 1000. The FPE estimate of the test error is based on eq. (9). The vertical line indicates the network for which the *estimated* test error is minimal. The figures show that a small training set will favor a simple model with few parameters.

In Fig. 2 we show the attractors for the original Mackey-Glass series (a), and three network architectures whose outputs are fed back as new inputs, thus performing iterative predictions. Panel (b) displays the pruned network. In (c) the attractor for the fully connected network trained on 1000 examples is shown, and finally the attractor of the pruned network trained on 1000 examples is shown in (d).

In Table 1 we compare the performances of pruned networks with those of fully connected nets, a linear model, and with a K-nearest-neighbor linear model [4]. It is interesting to note that the performance of the networks is similar to the nearest neighbor estimate. The latter involves finding the neighbors among the 1000 training examples and computing a regularized linear estimate of the output. The model has been optimized by leave-one-out cross-validation and the computational overhead (time and storage) is significant for this model. Also it is noteworthy that the pruned network, retrained without weight decay, is clearly superior. We have found that weight decay greatly assist pruning and optimization for the networks with superfluous resources; however, for the optimized pruned architectures we remove the weight decay and retrain to fine tune the performance [12].

In each case we compute the correlation with the true Mackey-Glass at-

Normalized error (mean square) on Mackey-Glass time series

	Train	**Test**	**No. of par.**
5-nearest-neighbors	4.80×10^{-7}	8.37×10^{-5}	–
Linear model[1]	9.70×10^{-2}	9.58×10^{-2}	17
Fully connected network[2]	6.30×10^{-5}	7.67×10^{-5}	181
Pruned network[3]	6.45×10^{-5}	7.45×10^{-5}	131
Pruned network[4]	3.12×10^{-5}	3.68×10^{-5}	131

Table 1: 1) Linear model is a single linear unit. 2) The initial pre-pruned networks, trained with the same weight decay terms as used during pruning. 3) Pruned network. 4) Pruned networks retrained *without* weight decay.

tractor, as described in [15]. The attractor is sampled on a grid of 20×20 points for the correlation estimate. The correlation coefficients for two independent samples of 8850 points of the true attractor is 0.991. For the fully connected network trained on 1000 examples it is 0.922, for the pruned network trained on 1000 examples 0.948. In order to compare with [15] we also completed a pruning session based on on 500 examples, and for the pruned network we found a correlation coefficient of 0.934 (with weight decay), while the coefficient for a feed-forward network with second order couplings were found to be 0.917 in [15]. For networks trained on 500 examples only, it turns out that, retraining without weight decay deteriorates the performance, indicating that pruning has not identified the optimal architecture.

We conclude that the Designer Networks more accurately identify the underlying dynamics, than straight fully connected networks.

Inverse Modeling: Channel Equalization

We have also implemented the toy-model studied by Day *et al.* [3], involving a non-linear channel sandwiched between low-pass filters with identical transfer functions $H_1(z) = 0.2602 - 0.9298z^{-1} + 0.2602z^{-2}$ and the non-linearity of the channel is given by the simple map $v = 5u|u|(1-u^2)\big/(1+u^2)$. The driving signal is a Gaussian i.i.d. sequence (unit variance, zero mean) filtered through a third-order *Butterworth* low-pass filter with the transfer function

$$H_2(z) = \frac{0.727 + 2.18z^{-1} + 2.18z^{-2} + 0.727z^{-3}}{1.000 - 2.327z^{-1} + 2.072z^{-2} - 0.686z^{-3}} \cdot 10^{-3}, \tag{13}$$

The network is trained, using weight decays $\alpha_w = 0.01$, $\alpha_W = 0.01$, to invert the channel and reconstruct the input as illustrated in Fig. 3. As input to the *network* we use a tapped delay line with 25 taps (noisy outputs from the the channel). The signal to be predicted is delayed 15 samples in order to ensure causal filtering. The training set comprises 2000 examples, which is a minute fraction of the training set used in [3]. In the left panel of Fig. 4 the

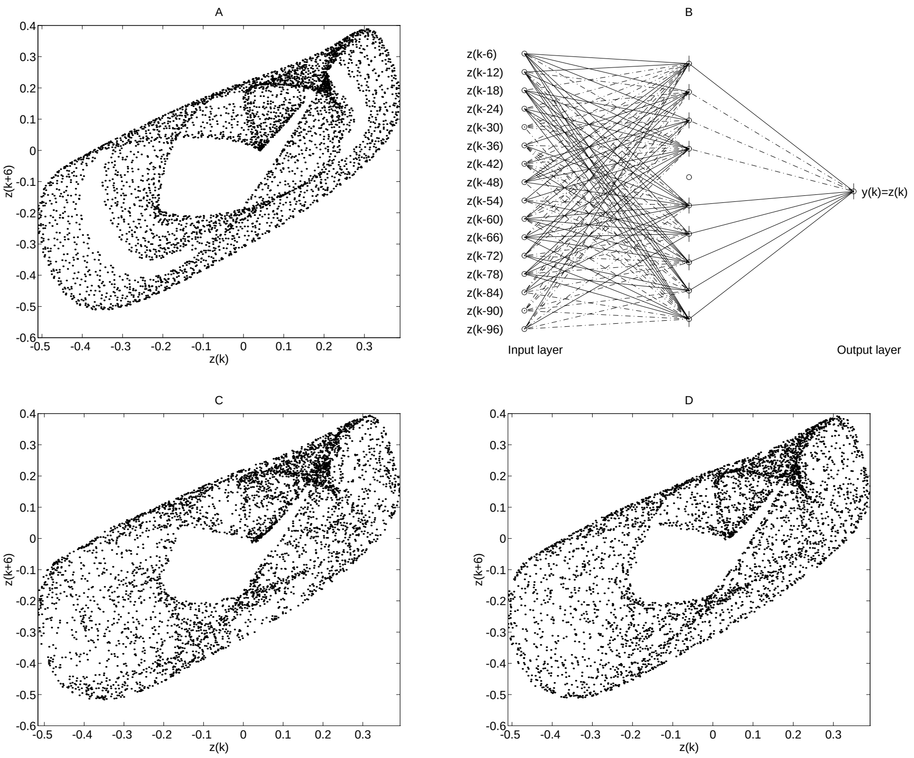


Figure 2: Phase-state plot ($z(k)$ versus $z(k+6)$) of a) The Mackey-Glass attractor. The correlation coefficient for two independent samples of 8850 points is 0.991. b) The pruned network obtained with a training set of 1000 examples. c) The estimated attractor for fully connected network trained on 1000 examples, having 181 weights and thresholds and a correlation coefficient of 0.922. d) The estimated attractor for the pruned network trained on 1000 examples (retrained without weight decay) having 131 weights and a correlation coefficient of 0.956.

evolution of the training and test errors during a pruning session are shown, and further the FPE estimate of the test error. We note that the minima of the estimated test error and the experimental test error (2000 samples) are compatible, hence confirming that the statistical approach is viable. In the right panel of Fig. 4 we show the pruned network obtained. As expected we see that the network make use of the inputs which in lag space are closest (in time) to the output to be estimated.

We present our test errors relative to the performance of a linear model. With a linear model with input as the networks obtains a normalized mean-square error defined in eq. (11) of 0.130. For the network the test error is reduced to 0.039. This is a reduction of the test error with a factor 0.3. In

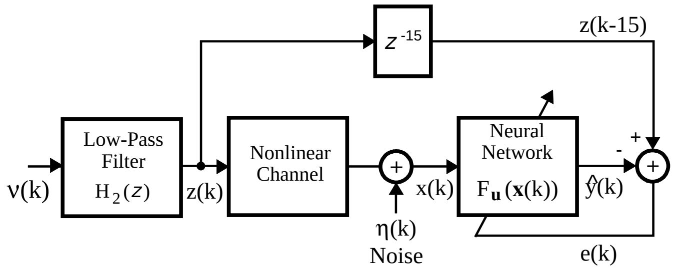


Figure 3: Channel equalization. The non-linear channel is driven by a low-pass filtered white noise signal $z(k)$. Output from the channel is denoted $x(k)$. The neural network is trained to reproduce the channel input $z(k)$ based on a sample of the noisy output $x(k)$. The variance of the channel Gaussian i.i.d. noise $\eta(k)$ is 30dB below the variance of the channel output.

[3] the reduction was from 0.0625 to 0.0256 for the back-prop net[1], which amounts to a reduction by a factor of 0.41.

CONCLUSION

We have corroborated our earlier results for the sunspot series [12] by new results for chaotic time series prediction and channel equalization. We suggested to optimize network architectures using the Optimal Brain Damage pruning scheme combined with our new statistical stopping criterion.

For the Mackey-Glass time series the pruned networks used almost all delayed input signals, while the networks for the channel equalizer mainly used the delayed inputs which are closest in time to the output signal. This shows that the pruning scheme effectively captures the relevant connections and removes unimportant hidden units.

We succeeded in designing compact neural networks which generalized better than the unpruned networks. A significant gain in generalization abil-

[1] The errors in [3] are defined as the normalized root-mean-square error. Furthermore, it should be noted that the error for the linear network on our data is significantly higher than the error of [3]. This is not due to the different training set sizes used, since the linear model of [3] converged very quickly. Rather, the non-linearity in our data set is more noticeable. It may therefore be more difficult to invert the channel with the present data set.

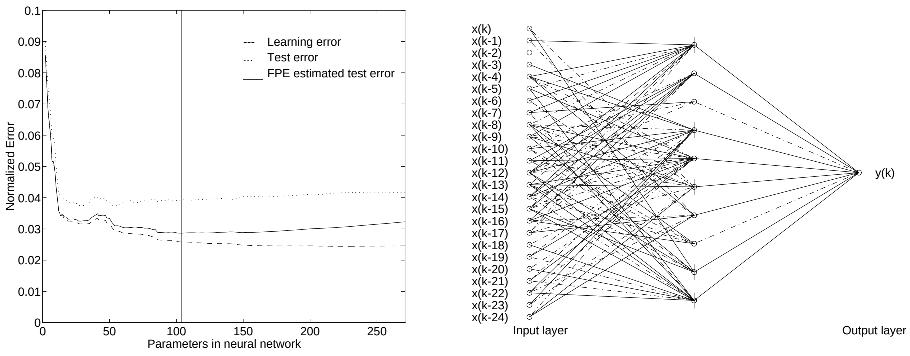


Figure 4: The left panel shows the evolution of training and test errors during pruning for channel equalization. The FPE estimate of the test error is based on eq. (9). The vertical line indicates the network for which the *estimated* test error is minimal. In the right panel we show the resulting, pruned, network. A vertical bar through a unit indicates an active threshold, full lines indicate positive connections, dash-dotted lines negative connections.

ity is achieved by retraining the pruned network without weight decay. However, note that weight decay is essential for the success of the pruning. Hence, we recommend pruning by Optimal Brain Damage supplemented by our newly developed stopping criterion as a tool for adapting "Designer Networks" for time series processing.

ACKNOWLEDGMENTS

This research was supported by the Danish Natural Science and Technical Research Councils through the Computational Neural Network Center (CONNECT).

REFERENCES

[1] H. Akaike: "Fitting Autoregressive Models for Prediction". Ann. Inst. Stat. Mat. vol. 21, 243-247, 1969.

[2] S.A. Barton: "A Matrix Method for Optimization a Neural Network". Neural Computation, vol. 3, 450-459, 1990.

[3] A.P. Day, M.R. Davenport, and D.S. Camporese: "Dispersive Networks for Nonlinear Adaptive Filters". In Neural Networks For Signal Processing II; Proceedings of the 1992 IEEE-SP Workshop, (Eds. S.Y. Kung, F. Fallside, J.Aa. Sørensen, and C.A. Kamm), IEEE Service Center, 464-473, 1992.

[4] J.D. Farmer, and J.J. Sidorowich: "Exploiting Chaos to Predict the Future and Reduce Noise", Technical Report LA-UR-88, Los Alamos National Laboratory, 1988.

[5] J. Hertz, A. Krogh and R.G. Palmer: Introduction to the Theory of Neural Computation, Addison Wesley, New York, 1991.

[6] J. Larsen: Design of Neural Network Filters. Ph. D. Thesis, Electronics Institute, Technical University of Denmark, March 1993.

[7] Y. Le Cun, J.S. Denker, and S.A. Solla: "Optimal Brain Damage". In Advances in Neural Information Processing Systems 2, 598-605, Morgan Kaufman, 1990.

[8] Y. Le Cun, B. Boser, J.S. Denker, D. Henderson, R.E. Howard, W. Hubbard, and L.D. Jakel: "Handwritten Digit Recognition with a Back-Propagation Network". In Advances in Neural Information Processing Systems 2, 396-404. Morgan Kaufman, 1990.

[9] L. Ljung: System Identification: Theory for the user, Prentice-Hall, Information and System Sciences series, 1987.

[10] J.E. Moody: "Note on Generalization, Regularization and Architecture Selection in Nonlinear Systems". In Neural Networks For Signal Processing; Proceedings of the 1991 IEEE-SP Workshop, (Eds. B.H. Juang, S.Y. Kung, and C. Kamm), IEEE Service Center, 1-10, 1991.

[11] M.B. Priestly: Non-linear and Non-stationary Times Series Analysis, Academic Press, 1988.

[12] C. Svarer, L.K. Hansen, and J. Larsen: "On Design and Evaluation of Tapped Delay Line Networks", In Proceedings of the 1993 IEEE International Conference on Neural Networks, San Francisco, vol. 1, 46-51, 1993.

[13] H. Tong and K. S. Lim: "Threshold autoregression, limit cycles and cyclical data". Journ. Roy. Stat. Soc. B, vol. 42, 245, 1980.

[14] A.S. Weigend, B.A. Huberman, and D.E. Rumelhart: "Prediction the future: A Connectionist Approach", Int. J. of Neural Systems, vol. 1, 193-209, 1990.

[15] N.H. Wulff and J.A. Hertz: "Prediction with Recurrent Networks". In Neural Networks For Signal Processing II; Proceedings of the 1992 IEEE-SP Workshop, (Eds. S.Y. Kung, F. Fallside, J.Aa. Sørensen, and C.A. Kamm), IEEE Service Center, 464-473, 1992.

Appendix D

IJNN article: A Quantitative Study of Pruning by Optimal Brain Damage

Reference for the article: [Gorodkin et al., 1993]. The layout of this article is in the journal changed a little while it has been manually typeset in two columns format in the journal.

A QUANTITATIVE STUDY OF PRUNING BY OPTIMAL BRAIN DAMAGE

J. Gorodkin[a)], L.K. Hansen[b)], A. Krogh[b)], C. Svarer[b)], and O. Winther[a)]

[a)]CONNECT, Niels Bohr Institute
Blegdamsvej 17, DK-2100 København Ø, Denmark

[b)]CONNECT, Electronics Institute
Technical University of Denmark, B. 349, DK-2800 Lyngby, Denmark

SUPERrevised, June 1993

Abstract

The Optimal Brain Damage (OBD) scheme of Le Cun, Denker, and Solla, for pruning of feed-forward networks, has been implemented and applied to the contiguity classification problem. It is shown that OBD improves the learning curve (the test error as function of the number of examples). By inspecting the architectures obtained through pruning it is found that the networks with less parameters have the smallest test error, in agreement with "Ockhams Razor". Based on this, we propose a heuristic which selects the smallest successful architecture among a group of pruned networks and we show that it leads to very efficient optimization of the architecture. The validity of the approximations involved in OBD are discussed and it is found and they are surprisingly accurate for the problem studied.

1 Introduction

Much attention is currently directed towards the problem of identifying optimal *application-specific* neural network architectures. It is generally believed that good generalization is associated with minimal representation and it may indeed be proven within many models that the generalization (or test) error is decreased by removing superfluous resources.

Pruning schemes have been successfully applied to layered neural networks [11, 12, 16, 17, 7]. Apart from the identification of more compact representations, significant improvements in performance have been obtained. The *Optimal Brain Damage* (OBD) scheme of Le Cun, Denker and Solla [12], stands out for its success in reducing the complexity of a well-trained network for identification of hand-written ZIP-codes [11, 12]. The method is based on the computation of the *saliency* of a weight, which is an estimate of the increase in training error induced by deleting the given weight.

Recently Hassibi and Stork proposed a generalization of OBD, the Optimal Brain Surgeon (OBS) [7, 8]. OBS was first found to perform well on a set of Boolean problems [7], and it has later been applied to the NETtalk problem. While OBD is based on the diagonal approximation of the second derivative matrix (the *Hessian*), OBS uses the full Hessian, which makes it possible to estimate the combined effect of pruning and retraining — assuming the second order approximation is valid, retraining is not necessary in OBS. Because of the approximations entering OBD, it is recommended to use it iteratively: alternating pruning of a small set of weights followed by retraining [11]. When applied to larger problems like NETtalk, Hassibi *et al.* adapted the OBS prescription to also include retraining, and reduced the complexity of the full Hessian by a block-diagonal approximation [8].

In this study we provide some quantitative results on the performance of OBD for Boolean classification problems. A separate presentation addresses regression problems[15]. In the original studies [11, 12] OBD was used to prune an architecture already optimized by hand (the zip-code reading network). One of the main motivations for this work is to see if OBD can prune a large redundant and unstructured network to a near-optimal one with only a fraction of the original number of weights.

For the *contiguity* classification problem we show numerically that OBD indeed enhances the learning performance, and in particular that the learning curve (*i.e.*, the generalization ability as a function of the size of the training set) is improved by OBD. The level of generalization is higher and the *learning transition*, a “knee” in the learning curve which identifies the minimum number of training examples needed for good generalization [13], occur earlier for the networks pruned by OBD. A fully connected network can thus be used if sufficient training examples are available, but there is a range of training set sizes for which OBD is crucial for successful learning. The scaling of this improvement with problem size is an open problem, *i.e.*, whether learnability in the sense of Valiant [19] is affected by OBD.

In this work we show that OBD may be further improved by a strategy in which a group of networks are pruned and the smallest final network picked, leading to a dramatic improvement in generalization ability. The number of training examples needed to perfectly learn our particular test problem consistently drops to around 100, whereas at least 200 examples are needed to obtain a fully connected network that generalizes well. Even a majority vote among 10 fully connected independently trained networks has a learning transition around 150 examples. This shows that OBD is capable of finding a near-optimal network architecture in a moderate number of trails. When the smallest of 10 networks is chosen the learning transition becomes very sharp: with 90 examples the generalization error is around 40%, and above 100 examples it is very close to zero.

In order to check the validity of the approximations used in OBD, we compare the estimated saliencies with the observed ones, finding surprising agreement. This is in line with the results of Le Cun *et al.* on recognition of handwritten digits [12].

The paper is organized as follows: in the next section OBD is derived and some of it's inherent difficulties are discussed. Our specific two-layer feed forward network is introduced in section three, and the saliency estimators are derived and discussed. Section four contains the results of our numerical investigations of OBD. The validity of the various approximations entering the saliency estimation is discussed, and a result for the "monks1 problem" is presented.

2 Pruning by Optimal Brain Damage

Present generalization theories [1, 5, 9, 14] estimate the test error from the training error and model capacity. The basic idea is that the superfluous parameters allow for too many generalizations from the training set, hence the "correct" generalization is picked with a small probability. The design rule derived from this insight would be:

> Minimize the capacity of the network under the constraint that it is able to implement the training set.

This strategy is often referred to as *Ockhams Razor*, see *e.g.* [16], since the above principle is a special form of a general guideline proposed by the medieval philosopher. A difficulty appears when applying the razor to problems with noise or inconsistencies in the training set, because a certain level of the training error is unavoidable, and in fact zero training error would lead to poor generalization ability. See [15] for a discussion of this problem in the context of non-linear regression. Throughout this presentation we restrict ourselves to the case of a noise free teacher that provides only correct examples.

Assuming that the network capacity is an increasing function of the number of parameters [3], Ockhams Razor may be implemented by *brute force* pruning as proposed by Thodberg [16, 17]. Alternatively, following Le Cun *et al.* [12], we may attempt to estimate the saliency and use this measure for ranking the parameters as candidates for pruning. For completeness we rederive and discuss the OBD method.

2.1 Weight saliency

Optimal Brain Damage is based on a second order approximation to the increase in training error resulting from elimination of a weight. If the weights w_i are perturbed by δw_i the change in the training error E_T is, to second order,

$$\delta E_T \simeq \sum_{i=1}^{N} \frac{\partial E_T}{\partial w_i}\delta w_i + \frac{1}{2}\sum_{i=1}^{N} \frac{\partial^2 E_T}{\partial w_i^2}\delta w_i^2 + \frac{1}{2}\sum_{j=1}^{N}\sum_{i=1, i\neq j}^{N} \frac{\partial^2 E_T}{\partial w_i \partial w_j}\delta w_i \delta w_j, \tag{1}$$

where N is the number of weights in the network. The first-order term is zero if the training has reached a minimum of E_T. If deletion of only one weight is considered ($\delta w_i = 0$ for all i except $\delta w_j = -w_j$), the off-diagonal part of the second order term is also zero, and all is left is

$$\delta E_T \sim s_j \equiv \frac{1}{2}\frac{\partial^2 E_T}{\partial w_j^2} w_j^2, \tag{2}$$

which is called the *saliency*. Figure 1 illustrates Equation (2). In the original work of Le Cun *et al.* the saliency estimator was used for deleting sets of weights $\{w_k | k \in D\}$, and the cumulative increase was approximated by

$$\delta E_T \sim \sum_{j \in D} s_j, \tag{3}$$

with the additional assumption that off-diagonal terms in the second derivative matrix can be neglected.

If the error function is additive, *i.e.*, $E_T = \frac{1}{p}\sum_{\alpha=1}^{p} E^\alpha$, where E^α is the error on example α, the diagonal elements of the second derivative matrix are given by:

$$\frac{\partial^2 E_T}{\partial w_j^2} = \frac{1}{p}\sum_{\alpha=1}^{p}\left[\frac{\partial^2 E^\alpha}{\partial (V^\alpha)^2}\left(\frac{\partial F_w(x^\alpha)}{\partial w_j}\right)^2 + \frac{\partial E^\alpha}{\partial V^\alpha}\frac{\partial^2 F_w(x^\alpha)}{\partial w_j^2}\right] \tag{4}$$

where F_w is the function implemented by the network, and $V^\alpha = F_w(x^\alpha)$ is the output corresponding to input x^α. The parameter p denotes the number of examples in the training set. For the most commonly used cost functions the second term can be neglected since $\frac{\partial E}{\partial V}$ is proportional to the error on example α (see below).

For the sum of squared errors $E_T = \frac{1}{2p}\sum_\alpha (y^\alpha - F_w(x^\alpha))^2$, Equation (4) becomes

$$\frac{\partial^2 E_T}{\partial w_j^2} = \frac{1}{p}\sum_{\alpha=1}^{p}\left[\left(\frac{\partial F_w(x^\alpha)}{\partial w_j}\right)^2 + (y^\alpha - F_w(x^\alpha))\left(-\frac{\partial^2 F_w(x^\alpha)}{\partial w_j^2}\right)\right] \tag{5}$$

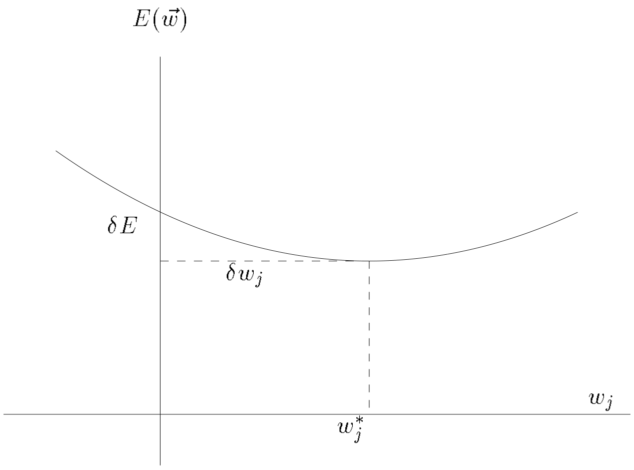


Figure 1: After training the network is assumed to be in a training error minimum (or close to it) with $w_j = w_j^*$ as illustrated. This weight is removed by setting $w_j = 0$. The cost is δE, which is in turn estimated by the saliency.

where the training set consists of the p pairs of input and target (x^α, y^α). The Levenberg-Marquardt approximation is invoked by Le Cun *et al.* [12], to drop the second term. If the network has learned to reproduce the examples exactly, the errors: $e^\alpha = y^\alpha - F_w(x^\alpha)$ would be zero. In general, one would expect that a well-trained network should have "random" errors, *i.e.*, the e^α would have zero mean. The correlation between errors and the second derivative of the network-function is thus expected to vanish for large training sets. An additional motivation for neglecting the second term of Equation (5) is *computational*: the remaining first term of the second derivative matrix is non-negative, and contains quantities already computed during training. The final estimate of the training error increase when deleting weight j is

$$s_j = w_j^2 \frac{1}{2p} \sum_\alpha \frac{\partial^2 E^\alpha}{\partial (V^\alpha)^2} \left(\frac{\partial F_w(x^\alpha)}{\partial w_j} \right)^2 . \tag{6}$$

If E_T is the sum of squared errors, $\frac{\partial^2 E}{\partial V^2} = 1$.

2.2 Inherent problems in OBD

A network pruning method based on the saliencies rests on two crude assumptions. First, the training error is assumed to be well approximated by a second order expansion around its minimum point, even far from it. It is not difficult to think of situations where this is not the case. Assume, for instance, that a standard sigmoid like tanh is used as activation function for a hidden unit which saturates for most inputs. The error surface will be very flat, and the second order expansion will be a poor approximation. A small weight decay regularizer can be used to avoid saturation of the sigmoids; we have found that such a technique greatly improves the performance of OBD.

The second crude assumption is that a large increase in error *before* retraining ($\delta E_{before}(w_j)$) when removing weight w_j also means a large error increase *after* retraining ($\delta E_{after}(w_j)$). Again one can easily think of cases where the ranking of weights according to δE_{before} would differ from the ranking based on δE_{after}. Ideally one would like to use the latter, and that is essentially what the methods of Thodberg [16] and OBS [8] aim at. It is worth noting that since the error always decreases during retraining, δE_{before} is an *upper bound* for δE_{after}, *i.e.*,

$$\delta E_{after}(w_j) \leq \delta E_{before}(w_j). \tag{7}$$

Therefore using OBD might be a good trade-off when considering the computational cost of the two other algorithms mentioned above.

3 OBD for a two-layer network

To learn Boolean classifications of M-dimensional binary inputs, we have chosen a family of functions that map the M-dimensional hypercube into the reals:

$$F_w(x^\alpha) = \sum_{j=1}^{n_H} W_j \tanh\left(\sum_{k=0}^{n_I} w_{jk} x_k^\alpha\right) + W_0, \tag{8}$$

n_H, n_I are the numbers of hidden and input units respectively, (here $n_I = M$). The thresholds are implemented as weights connecting to clamped units, $x_0 = 1$, in the preceding layer. The binary classification is given by the sign of the output. This is a standard two-layer network with one linear output unit, where w_{jk} denotes weights from input to hidden units and W_j the weights from hidden to output. The fully connected architecture is shown in Figure 2.

For training error we use the usual sum of squares:

$$E_T = \frac{1}{2p} \sum_{\alpha=1}^{p} \left(y^\alpha - F_w(x^\alpha)\right)^2. \tag{9}$$

One could argue that a linear output unit is not the obvious choice for a Boolean classification problem. The main virtue of (8) is that it allows for efficient optimization, as shown below.

Saliencies of two different functional forms are obtained, corresponding to the two layers of weights. The saliency of an output weight is:

$$s_j = W_j^2 \frac{1}{2p} \sum_{\alpha=1}^{p} \tanh^2\left(\sum_{k=0}^{n_I} w_{jk} x_k^\alpha\right) \tag{10}$$

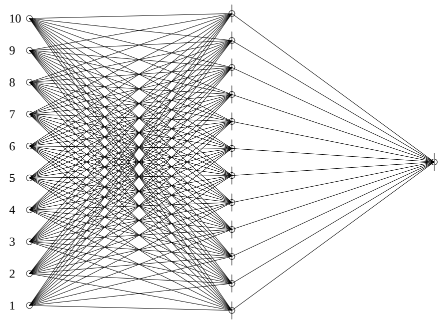


Figure 2: Fully connected architecture (10-12-1), an active threshold is indicated by a vertical line through the unit.

The saliency of an input-to-hidden weight is found to be:

$$s_{jk} = W_j^2 w_{jk}^2 \frac{1}{2p} \sum_{\alpha=1}^{p} \left[1 - \tanh^2 \left(\sum_{k'=0}^{n_I} w_{jk'} x_{k'}^{\alpha} \right) \right]^2 , \tag{11}$$

since $(x_k^\alpha)^2 = 1$ for Boolean inputs. Note that the training set output (*i.e.*, the specific Boolean function) only enters the saliencies implicitly through the weight distribution of the trained network.

Magnitude based pruning, in which the weights are ranked solely according to magnitude, has been criticized in [12] and [7]. Such criticism can be supported by analysis of equations (11) and (10). In the OBD context, magnitude based pruning is equivalent to the assumption that the second derivative is a constant for all weights. For the hidden to output weights, the second derivative is determined by the activation level of the hidden unit, which need not be the same for all hidden units. The second derivative with respect to an input-to-hidden weight includes the corresponding output weight, which need not be the same for all hidden units either.

Note that the accumulated saliency of a deleted set of weights is an estimate of the increase in the *training error*. If we adopt the standard procedure and let the network classification be determined by the *sign* of the output, some problems might tolerate significant increases in squared error, without making training set classification errors. This observation suggests a *stop criterion*: in the "noise-free" case, where all examples are correct, we prune the networks until classification errors appear on the training set.

4 Experiments

4.1 The contiguity problem

The contiguity (or "two or three clumps") problem was discussed in Denker *et al.* [5], and used by Thodberg for testing a pruning algorithm [16]. The Boolean input field (± 1) is interpreted as a one dimensional "image", and connected clumps of $+1's$ are counted. Two classes of patterns are considered: those with two and three clumps. The Boolean function is defined by assigning the output -1 to the two-clump inputs and $+1$ to the three-clump inputs. In particular we study the $n_I = 10$ case which has a total of 792 legal input patterns of which 432 have three clumps and 360 have two clumps. It is known that this problem can not be solved by a simple perceptron, *i.e.*, it is not linearly separable and hidden units are needed. A simple symmetric solution is illustrated in Figure 3; only a few (shared) parameters are necessary. In this network the hidden units work as *edge-detectors*. It was shown that only 20 to 30 examples are needed to obtain good generalization if this architecture is chosen [13].

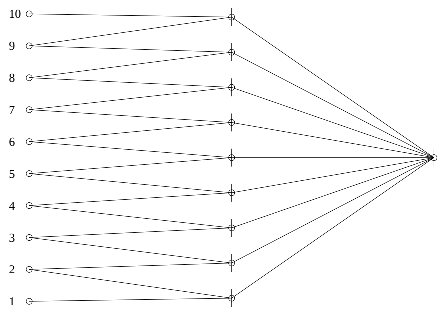


Figure 3: A sparse symmetric architecture for the contiguity problem

4.2 The simulator

The learning algorithm is based on mixed back-propagation and second order batch mode optimization. Since we have chosen a linear output unit, the cost function is quadratic in the hidden-to-output weights, allowing for direct optimization of these weights by matrix inversion [2]. The weights to the hidden units are initially adapted by batch mode gradient descent, and then refined by the second order *pseudo-Newton Rule* [9]:

$$\Delta w_{ij}(t) = -\eta \frac{\partial E}{\partial w_{ij}} \Bigg/ \frac{\partial^2 E}{\partial {w_{ij}}^2} \, . \tag{12}$$

The parameter η is used to secure that all weight updates in the "second order phase" lead to a decrease in the cost function. The learning rate η is initialized to 1 before each step, and iteratively diminished by powers of two until the step leads to a decrease in the cost function. We proceed by iteration, alternating updates between the two layers of weights.

To ensure numerical stability and to assist in the pruning procedure we add to the cost function a small *weight decay* term [9]:

$$E = E_T + E_W = \frac{1}{2p}\sum_{\alpha=1}^{p}\left(y^\alpha - F_w(x^\alpha)\right)^2 + \frac{\epsilon}{2p}\left(\sum_{j=1}^{n_H}\sum_{k=0}^{n_I} w_{jk}^2 + \sum_{j=0}^{n_H} W_j^2\right), \qquad (13)$$

with $\epsilon = 0.005$. Calculating the derivatives, the update rule for the weights in the first layer of the network is given by:

$$\Delta w_{jk}(t) = -\eta\left(\frac{\partial E_T}{\partial w_{jk}} + \frac{\epsilon}{p} w_{jk}\right) \Big/ \left(\frac{\partial^2 E_T}{\partial {w_{jk}}^2} + \frac{\epsilon}{p}\right). \qquad (14)$$

The matrix inversion scheme is also regularized by the weight decay term.

In each iteration of the pruning session, learning proceeds until no further decrease of the cost function can be obtained. The saliencies are then estimated and used to rank the weights. In our present version of OBD we proceed as in [12], and speed up the pruning procedure by removing several weights in each iteration (5% of the remaining weights). After pruning, the network is retrained using the mixed second order scheme described above. The procedure is carried out iteratively and stops when the network cannot be retrained to zero classification errors on the training set. This stop criterion is meaningful as long as the training set contains no false examples.

4.3 Learning curves

Our main results are shown in figures 4-5: learning curves for a fully connected network and networks trained and pruned using OBD. For each training set size, ten experiments were performed with a randomly chosen (but fixed) training set and random initial configurations of the weights. No strong dependency was observed on the particular choice of training set.

These learning curves comply with statistical theories on generalization: an S-shaped learning curve with a learning transition characteristic of the given problem and the given architecture. The location of this transition is a useful measure of the suitability of an architecture or an architecture optimization algorithm, to a specific task. Figure 4 shows the learning curve for the fully connected network. The learning transition is located around 200 examples: note that even with more than 200 examples some networks perform almost as bad as random guessing. The characteristics of the learning curve do not depend on the particular training sets chosen; the fluctuation observed is due to the success or failure of the learning algorithm (keep in mind that each network is error free on the given training set). The scatter in the performance of the solutions could be stabilized by a majority vote of the *ensemble* [6], as indicated by the line in the figure.

Figure 5 shows the learning curve for the pruned networks, where the learning transition is now located at only 150 examples. The fluctuations are significant, so the transition is not a sharp one. Picking the smallest network among the solutions obtained results in the line shown in the figure. Although based on simple heuristics, the strategy is remarkably stable and it produces a very sharp learning transition just below 100 examples. Figure 6 illustrates the correlation between network size and test error for a set of 50 networks trained on 140 examples, and it provides strong evidence in favor of Ockhams Razor, as found by Thodberg [16]. We conclude that OBD can be improved by a ensemble search combined with Ockhams Razor.

An example of the optimized architectures is shown in Figure 7. This particular architecture was trained and pruned on 160 examples. It makes no errors on the legal inputs. It is interesting to compare this network with the symmetric one Figure 3. The solution obtained through pruning uses only eight hidden units and 36 connections (including thresholds), while the symmetric network has nine hidden units and 37 connections. While the symmetric network is easy to interpret, it is not at all obvious how the network of Figure 7 works. Some of the hidden units are apparently "edge-detectors", while others are more complex (triple input) feature detectors, but these features are still local.

4.4 Test of the saliency estimator

A basic condition for the use of OBD is that the saliencies (10)-(11) give reliable estimates of "true" saliencies, *i.e.*, the increase in cost function when deleting a weight. In Figure 8 we test the validity of the two approximations entering the estimate of the individual saliencies. The experimental (true) saliency is plotted versus the estimated saliency at four different stages of the pruning process for the input-to-hidden weights. The correlation is quite remarkable for the network shown, and the result is reproducible when using the pseudo-Newton optimization. Since the uncertainty of the estimates is much smaller than the range of saliencies, it never happens that weights with "large" saliencies are deleted prematurely.

We note that the uncertainty is largest for the very small and the very large saliencies. For the small weights the imprecision is due to numerical problems, while for the large saliencies it is due to the non-linearity of the network. The largest error of the saliency estimate for large saliencies is an overestimate by a factor of 2.7. In Figure 9 the true cost function and the estimated cost function are shown for two different weights with large saliencies and significant errors. The second order approximation used by OBD (and by OBS) does not give an accurate description of the cost function. Note that these weights are exceptions to the general behav-

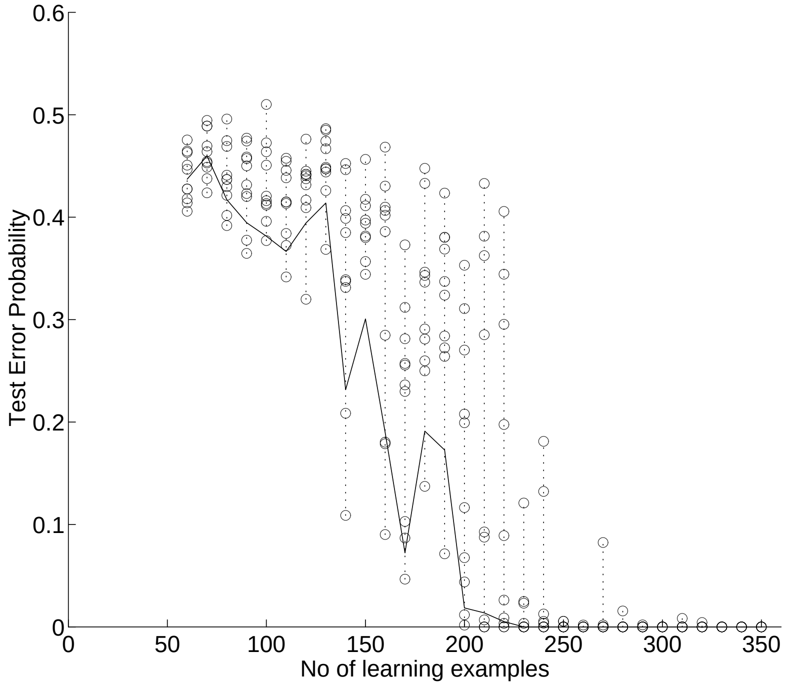


Figure 4: Learning curve of a fully connected architecture (10-12-1). At each training set size (vertical dotted lines) ten networks have been trained on the same random training set. Their test errors are indicated by circles. The *majority* vote of the ensemble is shown by the line.

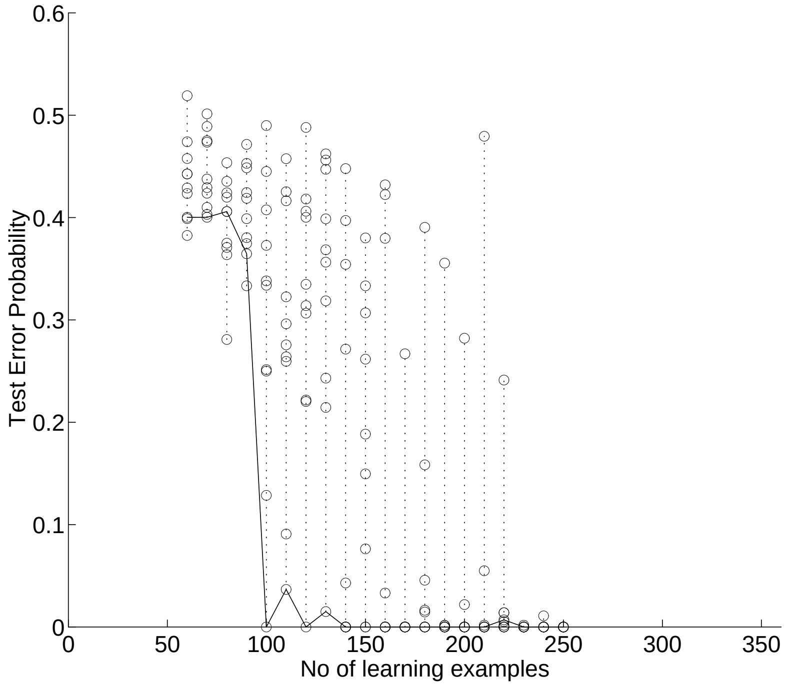


Figure 5: Learning curve for networks trained and pruned *on the run* by Optimal Brain Damage. At each training set size (vertical dotted lines) ten networks have been trained as indicated by circles. Picking the smallest network among the ten leads to test errors as shown by the line.

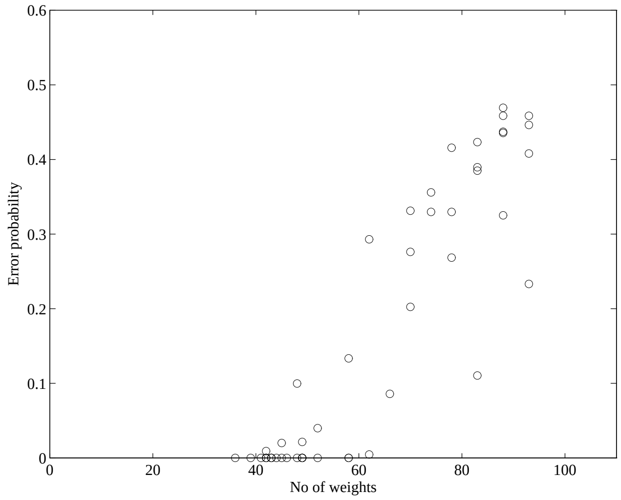


Figure 6: Probability of test error versus number of weights in a pruned network. Fifty networks were trained and pruned on a training set of size $p = 140$.

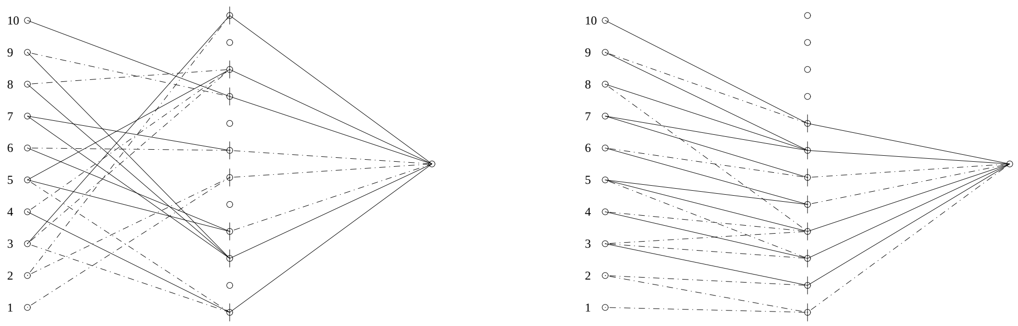


Figure 7: Optimized architecture trained on 160 randomly chosen examples. The network as it dropped out of the simulator is shown to the left. The network to the right has been unfolded with respect to receptive fields by a permutation of the hidden units. Solid lines indicate positive weights, dash-dotted lines indicate negative weights. A vertical line through a unit indicates an active threshold

ior. For the majority of weights the estimate is accurate enough to ensure correct ranking.

The experiment has been repeated with a straight back-prop optimizer. In this case the *small* saliencies are somewhat less accurately estimated, presumably because the gradient is not exactly zero at the "local minimum". We conclude, for the present network, that the effect of single weight pruning is well estimated by the saliency. Note that the off-diagonal terms do not enter the saliency for single weight deletion,

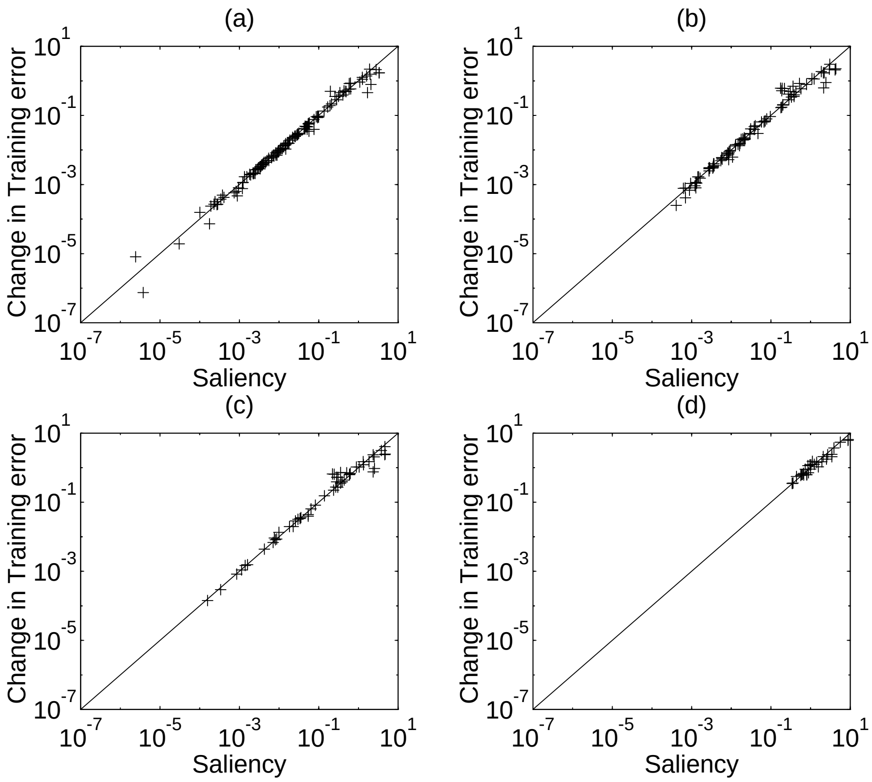


Figure 8: Experimental test of OBD approximations on the contiguity problem at four different stages in the pruning procedure: a) fully connected (145 weights), b) 104 weights, c) 66 weights, and d) 42 weights. The training set consists of 200 patterns. Only input-to-hidden weights are shown.

hence Figure 8 is testing the second order approximation to the cost function in the given minimum and the success of the optimizer in locating it.

It is surprising to see that most of the weights of this network, have saliencies that can be computed within the second order approximation. Inspecting equations (10)-(11) we note that the saliencies could be small due to the saturated activities (the hidden units saturate if the input-to-hidden weights are of large magnitude), so a small saliency would not always mean that the weight is unimportant. For large weights (and no weight decay) the error surface can be like a "bath tub" with steep walls and low curvature at the bottom. This kind of surface is poorly described by a quadratic approximation. The weight decay regularizer seems to prevent the system from entering the flat and non-quadratic bottom of the "bath tub" leading to good agreement between the measured and estimated saliencies in most cases.

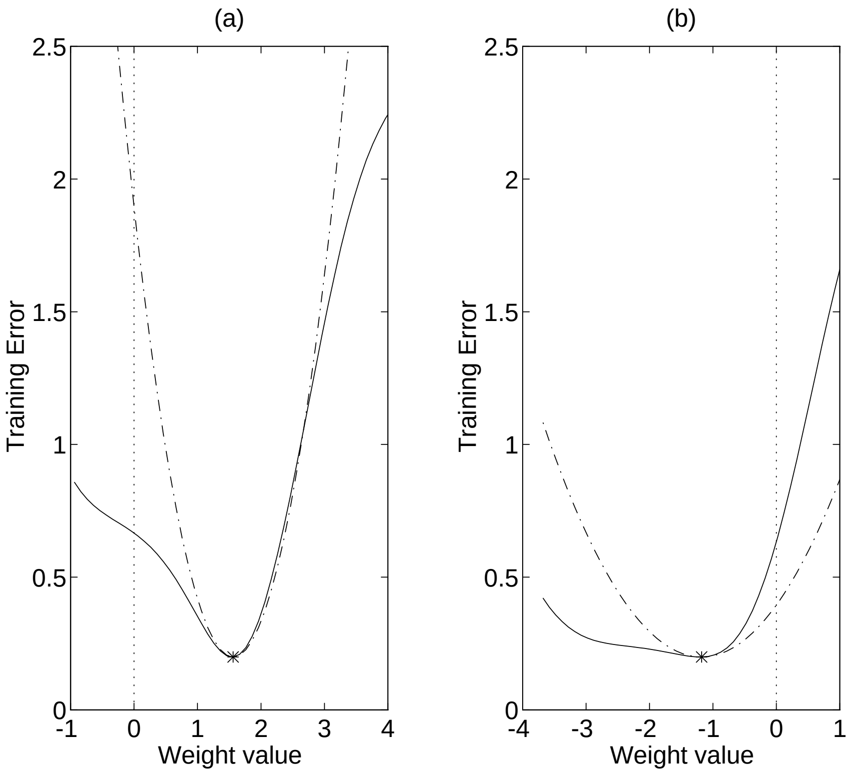


Figure 9: Test of the second order approximation for two weights with high saliency and a significant error of the saliency estimate. The second order approximation of the cost function is shown as a dash-dotted line, while the solid line represents the actual error as function of the given weight. In a) the estimate is overestimated by a factor of 2.7 and in b) the saliency is underestimated by factor of 0.6. We note that the cost function is poorly approximated by the second order approximation for both of these weights.

4.5 Experiments with the "monks problem"

The method has been tested on a few other Boolean problems. In particular the "first monk problem" [18], also used in the OBS experiments [7].

The first monk problem (*monk1*) [18] was formulated within an artificial robot domain in which robots are described by six different attributes. Back-propagation with weight decay showed the best performance among the considered classifiers [18]. The *monk1* problem can be represented by a net with 17 inputs and one output, and is defined by a total number of 432 legal examples. In [18] the training set size was 124 examples and the number of hidden neurons was 3. With this training set size Hassibi and Stork [7] report a net with 14 connections, while Lautrup [4] found a net with 14 connections using back-prop with weight decay and magnitude based pruning. For the same specifications, using back-prop, weight decay and a non-linear output we have been able to find a net with only 11 connections.[1] The network is shown in Figure 10 below; note that all thresholds are pruned away. Solutions with only two hidden units but more connections (down to 14) have been found as well.

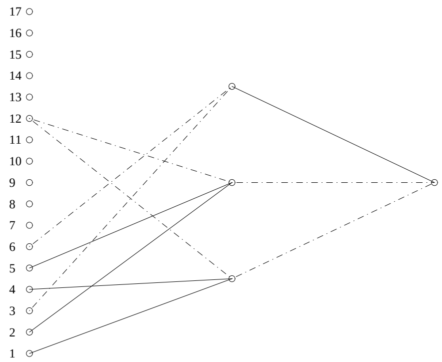


Figure 10: The network for *monk1* with 11 connections. The input-to-hidden weights have magnitudes about 0.8. The hidden-to-output weights have magnitudes about 1.6. Solid lines indicate positive weights, dash-dotted lines indicate negative weights.

5 Concluding remarks

We have investigated the Optimal Brain Damage scheme for pruning of feed forward networks. In particular we have studied the problem of learning a Boolean function from examples. Since our test problem is known to be solvable by a sparsely connected network, a fully connected network is highly redundant and a pruning procedure should be able to improve generalization. Our numerical experiments confirm this convincingly by showing that the learning curve of the iteratively trained and pruned network is significantly improved relative to the learning curves of fully connected networks. Furthermore we have shown that it is possible to improve the learning curve even more by chosing the smallest network among a small ensemble of pruned solutions. This ensemble search heuristic provides a learning curve with a remarkably sharp learning transition, much sharper than that of the fully connected architecture.

A qualitative understanding of the success of OBD follows from the observation that some of the underlying approximations are well satisfied. We believe that our careful optimization scheme and the use of weight decay play important roles in achieving these results.

[1] In response to our results, David Stork has informed us that *monk1* networks with 11 connections have been found with OBS in a new series of experiments.

Although OBS [7] uses a more sophisticated technique for ranking of weights, both OBD and OBS rely on the quadratic approximation. Future studies will have to determine whether the computational overhead is used better on OBS, on ensemble search with more OBD networks, or on strategies that go beyond the second order approximation.

It should also be mentioned that pruning is not necessarily the best way to optimize the network architecture. Much work has been devoted to incremental algorithms that build a network by gradually adding units. We believe that both constructive and destructive methods are important, and should eventually be combined in order to identify models with good generalization at a minimal computational cost.

Acknowledgements

We thank Jan Larsen, Benny Lautrup, Sara Solla, and David Stork for valuable discussions. We thank David Stork for providing us with preprints of the work on the Optimal Brain Surgeon method. Sara Solla is gratefully acknowledged for many comments on the manuscript. This research is supported by the Danish Research Councils for the Natural and Technical Sciences through the Danish Computational Neural Network Center (CONNECT).

References

[1] H. Akaike, *Fitting Autoregressive Models for Prediction.* Annals of The Institute of Statistical Mathematics **21**, 243-347 (1969).

[2] S. A. Barton, *A Matrix Method for Optimization a Neural Network* Neural Computation **3**, 450-459 (1990).

[3] E. B. Baum and D. Haussler, *What Size Net Gives Valid Generalization*, Neural Computation **1**, 151-160 (1989).

[4] B. Lautrup, *private communication.*

[5] J. Denker, D. Schwartz, B. Wittner, S.A. Solla, R. Howard, L. Jackel, J. Hopfield, *Large Automatic Learning, Rule Extraction, and Generalization*, Complex Systems **1**, 877-922 (1987).

[6] L.K Hansen and P. Salamon, *Neural Network Ensembles*, IEEE Transactions on Pattern Analysis and Machine Intelligence **12**, 993-1001 (1990).

[7] B. Hassibi and D. G. Stork: *Second Order Derivatives for Network Pruning: Optimal Brain Surgeon.* Preprint, NIPS 1992, in press (1992).

[8] B. Hassibi, D. G. Stork, and G. J. Wolff, *Optimal Brain Surgeon and General Network Pruning*, in Proceedings of the 1993 IEEE International Conference on Neural Networks, San Francisco (Eds. E.H. Ruspini *et al.*) pp. 293-299 (1993).

[9] J. Hertz, A. Krogh and R.G. Palmer, *Introduction to the Theory of Neural Computation*, Addison Wesley, New York (1991).

[10] E. Levin, N. Tishby and S.A. Solla, *A Statistical Approach to Learning and Generalization in Layered Neural Networks*, in Proc. 2nd Ann. Workshop on Computational Learning Theory (COLT'89), eds. R. Rivest, D. Haussler, and M.K. Warmuth. San Mateo, CA: Morgan Kaufmann, pp. 280-295 (1989), and Proceedings of the IEEE **78** 1574 (1990).

[11] Y. Le Cun, B. Boser, J.S. Denker, D. Henderson, R.E. Howard, W. Hubbard, and L.D. Jakel: *Handwritten Digit Recognition with a Back-Propagation Network*, in Advances in Neural Information Processing Systems II (Denver 1989) ed. D.S.Touretzsky. San Mateo: Morgan Kaufman, pp. 396-404 (1990)

[12] Y. Le Cun, J.S. Denker, S.A. Solla: *Optimal Brain Damage*, in Advances in Neural Information Processing Systems II (Denver 1989) ed. D.S.Touretzsky. San Mateo: Morgan Kaufman, pp. 598-605 (1990)

[13] D.B. Schwartz, V.K. Samalam, S.A. Solla, and J.S. Denker, *Exhaustive Learning*, Neural Computation **2**, 371-382 (1990).

[14] S.A. Solla: *Capacity Control in Classifiers for Pattern Recognizers*, in 'Neural Networks For Signal Processing'; Proceedings of the 1992 IEEE-SP Workshop, (Eds. S.Y. Kung, F. Fallside, J. Aa. Sørensen, and C.A. Kamm), IEEE Service Center, Piscataway NJ, pp. 255-266 (1992).

[15] C. Svarer, L.K. Hansen, and J. Larsen, *On Design and Evaluation of Tapped-Delay Neural Network Architectures*, in Proceedings of the 1993 IEEE International Conference on Neural Networks, San Francisco (Eds. E.H. Ruspini *et al.*), pp. 46-51 (1993).

[16] H.H. Thodberg, *Improving Generalization of Neural Networks Through Pruning.* Int. Journal of Neural Systems **1**, 317-326 (1991).

[17] H.H. Thodberg, *The Neural Information Processing System used for Pig Carcasses Grading in Danish Slaughterhouses*, preprint 1989, The Danish Meat Research Institute, Denmark.

[18] S. B. Thrun *et al.*, *The MONK's Problems - A performance comparison of different learning algorithms.* Technical Report, CMU-CS-91-197 Carnegie-Mellon U. Department of Computer Science, 1991.

[19] L. Valiant, *A theory of the learnable*, Communication of the ACM **27**, 1134-1142 (1984).

Appendix E

IEEE-SP94 article: Adaptive Regularization

Reference for the article: [Hansen et al., 1994]

ADAPTIVE REGULARIZATION

L. K. Hansen, C. E. Rasmussen,* C. Svarer, and J. Larsen
CONNECT, Electronics Institute B349
Technical University of Denmark,
DK-2800 Lyngby, Denmark
email: lkhansen,ed,csvarer,jlarsen@ei.dtu.dk

Abstract. Regularization, e.g., in the form of weight decay, is important for training and optimization of neural network architectures. In this work we provide a tool based on asymptotic sampling theory, for iterative estimation of weight decay parameters. The basic idea is to do a gradient descent in the estimated generalization error with respect to the regularization parameters. The scheme is implemented in our *Designer Net* framework for network training and pruning, i.e., is based on the diagonal Hessian approximation. The scheme does not require essential computational overhead in addition to what is needed for training and pruning. The viability of the approach is demonstrated in an experiment concerning prediction of the chaotic Mackey-Glass series. We find that the optimized weight decays are relatively large for densely connected networks in the initial pruning phase, while they decrease as pruning proceeds.

INTRODUCTION

Learning based on the conventional feed-forward net may be analyzed with statistical methods and the result of such analysis can be applied to model optimization [5, 6, 10, 11, 12]. We have shown how pruning and regularization can be combined to design compact networks for time series prediction [11, 12]. Our "Designer Net" framework is based on the *Optimal Brain Damage* (OBD) method of Le Cun *et al.* [7] and we use simple weight decay for regularization. The benefits from compact architectures are three-fold: Their generalization ability is better, they carry less computational burden, and they are faster to adapt if the environment changes. Further, we have shown how the generalization error of the network may be estimated – without extensive cross-validation – using a modification of Akaike's *Final Prediction Error* (FPE) estimate [1]. The minimal FPE constitutes a useful stopping

*Present address: Dept. of Computer Science, University of Toronto, Canada.

criterion for pruning. However, our previous work has been conditioned on the correct setting of several parameters, most prominently the weight decay parameters. *In this contribution we provide the possibility of adapting regularization parameters within the Designer Net framework.*

The results obtained can be viewed as a *sampling theory* alternative to the Bayesian or *Evidence* based techniques for adaptive regularization developed by MacKay [8, 9]. An analytical comparison of these two techniques has recently been given in [2].

LEARNING

The use of *system identification* tools for neural net learning has been pioneered by Moody (see e.g., [10]) who derived estimators for the average generalization error. The main source of uncertainty in the learning process is the shortage of training data. Other important contributions to uncertainty are: Lack of fit, noise in the training process, and non-stationarity of the data-generating environment. Lack of fit[1] was discussed in, e.g., [5], while noise in the training process has been discussed in [3]. In this presentation we will neglect these three effects. Lack of fit can be minimized by starting the pruning process from large enough networks, while noise in the training process can be relieved by careful search in weight space. Non-stationarity is a hard problem that will be pursued in future work, here we will assume stationarity.

NETWORK ARCHITECTURE AND TRAINING

The basic network is a *tapped delay line architecture* with L input units, n_H hidden sigmoid units and a single linear output unit. The initial network is fully connected between layers and implements a non-linear mapping from lag space $\mathbf{x}(k) = [x(k), ..., x(k-L+1)]$, ($L$ is the length of the tapped delay line), to the real axis:

$$\widehat{y}(k) = F_{\mathbf{u}}(\mathbf{x}(k)) \qquad \widehat{y} \in \mathcal{R}, \tag{1}$$

where $\mathbf{u} = [\mathbf{w}, \mathbf{W}]$ is the N-dimensional weight vector and $\widehat{y}(k)$ is the prediction of the target signal $y(k)$. The particular family of non-linear mappings considered here can be written as:

$$F_{\mathbf{u}}(\mathbf{x}(k)) = \sum_{j=1}^{n_H} W_j \tanh\left(\sum_{i=1}^{L} w_{ij} x(k-i-1) + w_{i0}\right) + W_0, \tag{2}$$

where n_H is the number of hidden units, W_j are the hidden-to-output weights, while w_{ij} connect the input and hidden units.

[1]Lack of fit is also sometimes described as "the teacher does not belong to student space" or "incomplete modeling".

A simulator based on *batch mode*, second order local optimization has been developed, as described in [11, 12]. The scheme is based on the *diagonal approximation* of the cost-function Hessian (the second derivative matrix). We use the sum of squared errors to measure the performance of the current network:

$$E_{\mathrm{train}} = \frac{1}{p} \sum_{k=1}^{p} \left[y(k) - F_{\mathbf{u}}(\mathbf{x}(k)) \right]^2 , \tag{3}$$

where p is the number of training examples. To ensure numerical stability and for assisting the pruning procedure we augment the cost-function with a weight decay term:

$$E = E_{\mathrm{train}} + \frac{\alpha_w}{p} \sum_{ij}^{N_w} w_{ij}^2 + \frac{\alpha_W}{p} \sum_{j}^{N_W} W_j^2 , \tag{4}$$

where N_w, N_W are the numbers of weights and thresholds in hidden and output units, respectively. Further, α_w, α_W are the weight decay parameters of the hidden and output layers, respectively. The objective of the training procedure is to optimize the networks ability to predict near future values of a given time series. Hence, the network weights, $\mathbf{u}$, are trained to recognize the short time structure of the training set time series.

PRUNING

The OBD method proposed by Le Cun *et al.* [7] was successfully applied to reduce large networks for recognition of handwritten digits. The basic idea is to estimate the increase in the *training error* when deleting weights. The estimate is formulated in terms of weight *saliencies* s_l:

$$\delta E_{\mathrm{train}} = \sum_{l \in D} s_l \equiv \sum_{l \in D} \left(\frac{2\alpha}{p} + \frac{1}{2} \frac{\partial^2 E_{\mathrm{train}}}{\partial u_l^2} \right) u_l^2 , \tag{5}$$

where u_l is a component of $\mathbf{u}$ and the sum runs over the set D of weights to be deleted. The saliency definition used here takes into account that the weight decay terms force the weights to depart from the minimum of the training set error. As in [7] we approximate the second derivative by the positive semi-definite expression:

$$\frac{\partial^2 E_{\mathrm{train}}}{\partial u_j^2} \approx \frac{2}{p} \sum_{k=1}^{p} \left(\frac{\partial F_{\mathbf{u}}(\mathbf{x}(k))}{\partial u_j} \right)^2 . \tag{6}$$

The major assumptions entering the derivation of OBD are: 1) Terms of third and higher orders in the deleted weights can be neglected. 2) The off-diagonal terms in the Hessian, $\partial^2 E_{\mathrm{train}} / \partial u_l \partial u_{l'}$, can be neglected. Computationally, the second order (diagonal) terms, eq. (6), are reused from the training scheme. We refrain from operations involving the full Hessian, which scales poorly for large networks. The recipe allows for *ranking* the weights according to saliency.

GENERALIZATION

The generalization error is defined as the average squared error on an example from the example distribution function $P(\mathbf{x}, y)$. The examples are assumed to be generated by a *teacher* function of the same form as the model and with a set of *unknown* weights $\mathbf{u}^*$ and degraded by additive noise:

$$y(k) = F_{\mathbf{u}^*}(\mathbf{x}(k)) + \nu(k) \tag{7}$$

where the noise samples $\nu(k)$ are independent identically distributed variables of unknown variance σ^2. Further, we assume that the noise terms are independent of the corresponding inputs. The generalization error of a given network is by definition the average error on a random example. A more interesting quantity is the training set average of the generalization error, viz., the average over an ensemble of networks in which each network is provided with its individual training set. Using the diagonal approximation for the Hessian this error (also referred to as the test error) can be estimated as [6]:

$$\begin{aligned} \widehat{E}_{\text{test}} &= \left(1 + \frac{N_{\text{eff}}}{p}\right)\sigma^2 \\ &\quad + 2\sum_{ij}^{N_w} \lambda_{ij} \left(\frac{\alpha_w}{p} \cdot \frac{w_{ij}^*}{\lambda_{ij} + 2\alpha_w/p}\right)^2 \\ &\quad + 2\sum_{j}^{N_W} \Lambda_j \left(\frac{\alpha_W}{p} \cdot \frac{W_j^*}{\Lambda_j + 2\alpha_W/p}\right)^2 + R, \end{aligned} \tag{8}$$

with

$$N_{\text{eff}} = \sum_{ij}^{N_w} \left(\frac{\lambda_{ij}}{\lambda_{ij} + 2\alpha_w/p}\right)^2 + \sum_{j}^{N_W} \left(\frac{\Lambda_j}{\Lambda_j + 2\alpha_W/p}\right)^2, \tag{9}$$

where the λ's are the second derivatives already computed in eq. (6): $\lambda_{ij} \equiv \partial^2 E_{\text{train}}/\partial w_{ij}^2$, $\Lambda_j \equiv \partial^2 E_{\text{train}}/\partial W_j^2$. The rest term R contains higher order quantities and terms that do not affect the estimate of the regularization parameters, see [5, 6] for further discussion. The estimate is based on linearization of the networks as regards the fluctuations in the weights resulting from different training sets.

The generalization error estimates were also used for answering the question of how many weights it may be possible to delete in a pruning session in [11, 12]. We applied Akaike's FPE estimate [1] of the test error in terms of the training error which reads:

$$\widehat{E}_{\text{test}} = \frac{p+N}{p-N} E_{\text{train}}, \tag{10}$$

where p is the number of training samples, and N is the number of parameters in the model. The left hand side of eq. (10) is the average generalization error, averaged over all possible training sets of size p.

The relation expresses the fact that the training and test errors are biased estimates of the noise level because each parameter during training has "absorbed" noise from the training samples.

Since we have regularized the training procedure by weight-decay terms α_w, α_W, hence, suppressed the ability of the (otherwise) ill-determined parameters to model noise, we need to modify the standard FPE estimate by replacing the total number of parameters with the *effective* number of parameters, see [10, 11, 12][2]:

$$\widehat{E}_{\text{test}} = \frac{p + N_{\text{eff}}}{p - N_{\text{eff}}} E_{\text{train}}, \tag{11}$$

With the above tool we can obtain a generalization error estimate for each pruned network. By selecting the network with the lowest estimated generalization error we obtain a stopping criterion for pruning.

Note that the estimated average generalization eq. (9) error is a function of the regularization parameters, hence, it is possible to vary these and search for minimal test error. In MacKay's Evidence framework a similar strategy was adopted, however, with the purpose of maximizing the so-called Evidence. We find it more natural to optimize the quantity that is our basic objective, namely the test error. It is at present not clear what the relation between the Evidence and the generalization error is. Empirically, they have been found to be related [2, 8].

We use a simple gradient descent procedure for minimization of the generalization error:

$$\alpha(n+1) = \alpha(n) - \rho \left. \frac{\partial E_{\text{test}}}{\partial \alpha} \right|_{\alpha=\alpha(n)}, \tag{12}$$

where ρ is a gradient descent parameter, and n is the iteration index (one epoch). The Designer Net approach is based on the diagonal approximation to the Hessian. In terms of the diagonal elements the recursion above reads,

$$\alpha_w(n+1) = \alpha_w(n) + \frac{4\rho}{p^2} \sum_{ij}^{N_w} \frac{(\sigma^2 - \alpha_w(n)(w_{ij}^*)^2)\lambda_{ij}^2}{(\lambda_{ij} + 2\alpha_w(n)/p)^3}. \tag{13}$$

A similar expression applies for the hidden-to-output weight decay parameter α_W, in fact an arbitrary set of weight decay parameters can be defined and estimated using this recipe[3]. Expression (13) contains two unknown quantities: the teacher weights w_{ij}^* and the noise variance σ^2. The teacher weights are replaced by the *current estimated weights* of the network (see [2] for a

[2] In fact the notion of an effective number of parameters is quite delicate see [6].

[3] In the derivative of the test error we have kept the dependence $\lambda^2/(\lambda + 2\alpha/p)^3$ (rather than $1/\lambda$) providing a stabilizing effect similar to the Moore-Penrose pseudo inverse discussed in [6].

discussion), while the noise variance is estimated from the training error in the same approximation as in eq. (11):

$$\widehat{\sigma}^2 = \left(1 - \frac{N_{\text{eff}}}{p}\right)^{-1} E_{\text{train}}. \tag{14}$$

EXPERIMENTS

We illustrate the virtues of the adaptive regularization scheme on two time series forecasting problems. The first experiment explores the functional dependence of the derivative of the estimated test error cf. equation (13). The forecasting problem is the sunspot benchmark involving estimation of the yearly sunspot activity from the past twelve years activity (see, e.g., [11] for a detailed description of the benchmark). To simplify we consider a linear model for which the parameters are uniquely determined when using the least squares cost function. The sunspot benchmark involves three data sets: A training set and two test sets. In figure 1 we show the weight decay dependence of the two test errors and of the derivative of the *estimated* test error. Note that both test sets have shallow minima at values just below $\alpha = 0.1$, and that the derivative of the estimated test error passes through zero at a compatible value. Also note that the particular functional form of the derivative implies that the iterative scheme will converge to the zero point of the gradient, hence, provide near-optimal regularization with improved generalization errors. To further illustrate the role of adaptive regularization in the Designer Net framework we present tentative results on a standard problem of nonlinear dynamics, viz. the Mackey-Glass chaotic time series. This forecasting problem was previously studied in [12]. The Mackey-Glass attractor is a non-linear chaotic system described by the following equation:

$$\frac{dz(t)}{dt} = -bz(t) + a\frac{z(t-\tau)}{1 + z(t-\tau)^{10}} \tag{15}$$

where the constants are $a = 0.2$, $b = 0.1$ and $\tau = 17$. The series is resampled with sampling period 1 according to standard practice. We aim at identifying the underlying dynamic model, from this chaotic time series. The network configuration is $L = 16$, $n_H = 10$, with a total of 181 parameters, and we train to implement a six step ahead prediction. That is, $\mathbf{x}(k) = [z(k-6), z(k-12), \cdots, z(k-6L)]$ and $y(k) = z(k)$.

The errors are computed as:

$$E_{\text{set}} = \frac{1}{\sigma^2_{\text{total}} \cdot p_{\text{set}}} \sum_{k=1}^{p_{\text{set}}} \left[y(k) - F_{\mathbf{u}}(\mathbf{x}(k))\right]^2, \tag{16}$$

where p_{set} is the number of examples in the data (train or test) set in question, and σ^2_{total} is the total variance of $y(k)$ on the training and test set.

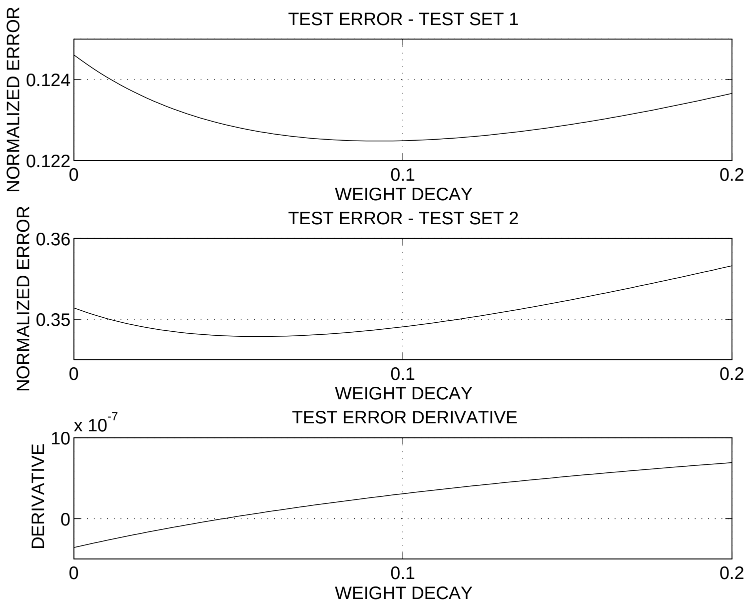


Figure 1: The role of weight decay regularization for a linear model on the sunspot benchmark series. The two upper figures show the errors (see [11] for a definition) on test sets both having shallow minima just below a weight decay of 0.1, while the bottom figure shows the derivative of the estimated test error as function of weight decay. Note that a gradient descent procedure will converge to the zero point.

There are several ways of implementing the adaptation scheme; here we initially set the weight decays to fixed values $\alpha_w = \alpha_W = 0.005$ for 100 epochs, then the network is trained with simultaneous adaptation of weight decay for 8000 epochs using eq. (13) with $\rho = 0.1$. After the initial training phase, further pruning and adaptation took place with pruning of 2% of the remaining weights per retraining round (400 epochs). In line with [12] it is seen that the stop criterion is able to select the optimal network. In figure 2 the normalized training errors, test errors cf. eq. (16), and the corresponding FPE error (after the initial training phase) are sketched for a training set size of 500 examples and the test set comprises 8500 examples. In [12] we

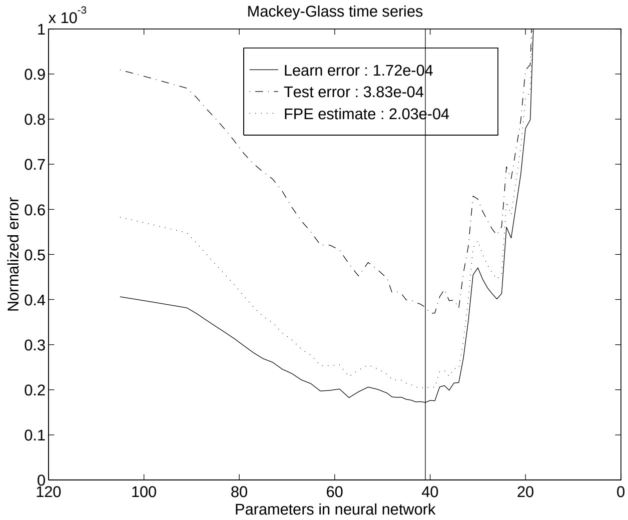


Figure 2: The evolution of training and test errors during pruning for the Mackey-Glass time series for a training set of size 500. The FPE estimate of the test error is based on eq. (11). The vertical line indicates the network for which the *estimated* test error is minimal.

compared the performances of pruned networks with those of fully connected nets, a linear model, and with a K-nearest-neighbor linear model. It was noted that the performance of the networks is similar to the nearest neighbor estimate. While the two weight decays previously were set manually we here adapt them according to equation (13). In figure 3 the development of the two weight decays is depicted as pruning proceeds. Note that the adaptive regularization scheme “chooses” relatively high regularization for the large network as should be expected. These networks have superfluous resources that could potentially harm generalization through overfitting. Eventually, at the end of the pruning session the test error estimates are rather biased (the network is underfitting) and the adaptive scheme does not provide reliable estimates. We have observed that the scheme in it present form has some

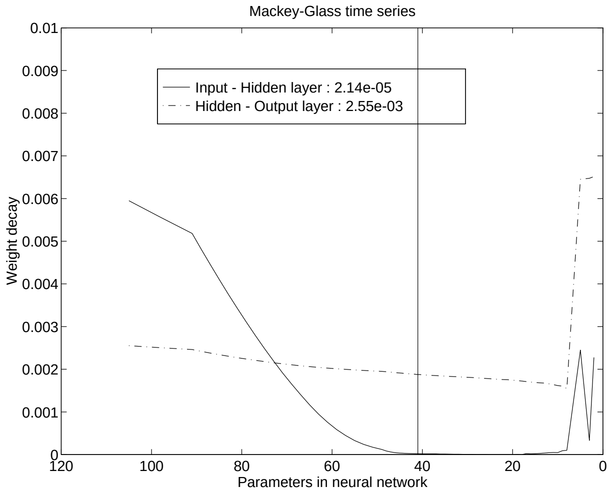


Figure 3: The evolution of weight decays during pruning for the Mackey-Glass time series. The vertical line indicates the network for which the *estimated* test error is minimal.

dependence on initialization of weights and weight decays; this is a topic for current research. The very low value of the input-to-hidden weight decay for the small networks is also in line with our earlier observations, namely that one can retrain the optimal architecture without weight decay and get slightly improved generalization [11, 12].

CONCLUSION

A scheme has been derived for adaptation of weight decay parameters. The scheme is based on asymptotic sampling theory. Two examples were given to illustrate the virtues of such adaptation. First, we showed that the functional form of the derivative of the estimated test error will provide convergence to near-optimal values for a linear model on the sunspot benchmark. Secondly, it was shown how the Designer Net framework can be applied with adaptive regularization, hence, relieving, manual tuning of these important parameters.

ACKNOWLEDGMENTS

This research was supported by the Danish Natural Science and Technical Research Councils through the Computational Neural Network Center (CONNECT).

REFERENCES

[1] H. Akaike, "Fitting Autoregressive Models for Prediction," Ann. Inst. Stat. Mat., **21**, 243-247, (1969).

[2] L.K. Hansen and C.E. Rasmussen, "Pruning from Adaptive Regularization," Accepted for Neural Computation, CONNECT Electronics Institute, Technical University of Denmark, preprint (1993).

[3] L.K. Hansen, "Stochastic Linear Learning: Exact Test and Training Error Averages," Neural Networks **6**, 393-396, (1993).

[4] J. Hertz, A. Krogh and R.G. Palmer, Introduction to the Theory of Neural Computation, Addison Wesley, New York, (1991).

[5] J. Larsen, Design of Neural Network Filters, Ph. D. Thesis, Electronics Institute, Technical University of Denmark, March (1993).

[6] J. Larsen and L.K. Hansen, "Generalization Performance of Regularized Neural Network Models," In proceedings of the IEEE Workshop on Neural Networks for Signal Processing NNSP'94.

[7] Y. Le Cun, J.S. Denker, and S.A. Solla, "Optimal Brain Damage," In Advances in Neural Information Processing Systems 2, 598-605, Morgan Kaufman, (1990).

[8] D. MacKay, "Bayesian interpolation". Neural Computation **4** 448-472, (1992).

[9] D. MacKay, "A practical framework for backpropagation networks". Neural Computation **4** 415-447 (1992).

[10] J.E. Moody, "Note on Generalization, Regularization and Architecture Selection in Nonlinear Systems," In Neural Networks For Signal Processing Proceedings of the 1991 IEEE-SP Workshop (Eds. B.H. Juang, S.Y. Kung, and C. Kamm), IEEE Service Center, 1-10, (1991).

[11] C. Svarer, L.K. Hansen, and J. Larsen, "On Design and Evaluation of Tapped Delay Line Networks," In Proceedings of the 1993 IEEE International Conference on Neural Networks San Francisco, 46-51, (1993).

[12] C. Svarer, L.K. Hansen, and J. Larsen, and C.E. Rasmussen, "Designer Networks for Time Series Processing," Proceedings of the 1993 IEEE Workshop on Neural Networks for Signal Processing (NNSP'93) Baltimore (Eds. C.A. Kamm et al.), 78-87, (1993).

Appendix F

IEEE-SP94 article: Neural Estimation of Kinetic Rate Constants from Dynamic PET-Scans

Reference for the article: [Fog et al., 1994]

NEURAL ESTIMATION OF KINETIC RATE CONSTANTS FROM DYNAMIC PET-SCANS

Torben Fog, Lars Hupfeldt Nielsen and Lars Kai Hansen
CONNECT, Electronics Institute B349
Technical University of Denmark,
DK-2800 Lyngby, Denmark
email: lkhansen@eileen.ei.dth.dk

Søren Holm, Ian Law, Claus Svarer, and Olaf Paulson
Dept. of Neurology,
The University Hospital of Copenhagen
DK-2100 Copenhagen Ø, Denmark

Abstract. A feed forward neural net is trained to invert a simple three compartment model describing the tracer kinetics involved in the metabolism of [^{18}F]flourodeoxyglucose in the human brain. The network can estimate rate constants from Positron Emission Tomography sequences and is about 50 times faster than direct fitting of rate constants using the parametrized transients of the compartment model.

INTRODUCTION

Positron Emission Tomography (PET) is an important tool for mapping of brain metabolism and functionality [2]. The primary target of PET is reconstruction of concentrations of certain radioactive tracers. The usefull tracers emit positrons that are locally annihilated to produce two 511 keV gamma rays propagating in opposite directions. The 3D distribution of the tracer can be reconstructed from the geometric constraints of coincident counts, using standard techniques (filtered backprojection). An important class of tracers are chemically equivalent to compounds that enter the basic brain metabolism. By reconstructing such tracer distributions important aspects of brain metabolism have been revealed. Furthermore, by investigating the *transient response* to tracer injection, it is possible to identify fundamental kinetic rate constants. In this study we investigate the latter approach. The basic kinetic model was proposed by Sokoloff et al. [1]; in subsequent studies the model was used to estimate rate constant in lumped regions. In the work of Kanno et al. [3] pixel by pixel estimation of the rate-constants was introduced. This scheme has, however, not found wide spread use due to the complexity of the task of fitting the kinetic model transient to the large number of individual pixel transients. *In this work we show how a neural network may substitute for such tedious parameter fitting procedures. The neural network system is trained to produce a smooth map relating a given transient with its most likely rate constants. This will provide a much faster estimation time for the individual pixel rate constants*

SOKOLOFF's KINETIC MODEL

We consider here the kinetics of the compound [^{18}F]flourodeoxyglucose (FDG). The kinetics of this tracer is similar to glucose in the initial metabolic steps. It passes through the blood-brain barrier (BBB), and is phosphorylized in a process past the BBB analogous to glucose. Then it ceases to react further and is effectively trapped. The kinetics can be modelled by a compartemental model involving one compartment representing the tracer density in the arterial blood outside the BBB, C^*_{P}; one compartment representing the socalled *precursor pool*, C^*_{E}; and finally a compartment representing the phosphorylized fraction behind the BBB, C^*_{M}; see figure 1. In current experiments the arterial concentrations are measured continuously along with the scan, hence the concentration C^*_{P} can be considered a control parameter for the compartment model.

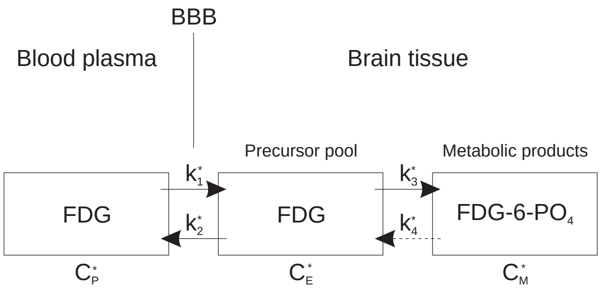


Figure 1: Sokoloff's three compartment model applied to phosphorylization of [^{18}F]flourodeoxyglucose (FDG). The star on the concentrations signifies that we consider tracer amounts and constants

Following the injection of the tracer, hence, the rise of the arterial blood concentration C^*_{P}, the flow through the BBB starts. The measured PET tracer activity is the sum of the activities of the two compartments to the right of the BBB c.f. figure 1,

$$C_i^* = C_E^* + C_M^* \tag{1}$$

The dynamics of the three compartment model is given by:

$$\frac{dC_i^*}{dt} = \frac{dC_E^*}{dt} + \frac{dC_M^*}{dt} \tag{2}$$

with

$$\frac{dC_M^*}{dt} = k_3^* C_E^*, \tag{3}$$

and

$$\frac{dC_E^*}{dt} = k_1^* C_P^* - k_2^* C_E^* - k_3^* C_E^*. \tag{4}$$

The reverse reaction rate constant (k_4^*) corresponding to k_3^* is neglected. These equations are straightforward to integrate providing the two time dependent concentrations,

$$C_E^*(t) = k_1^* e^{-(k_2^*+k_3^*)t} \int_0^t e^{(k_2^*+k_3^*)t'} C_P^*(t')\ dt' \tag{5}$$

$$C_M^*(t) = k_1^* k_3^* \int_0^t \left[e^{-(k_2^*+k_3^*)t'} \int_0^t e^{(k_2^*+k_3^*)t''} C_P^*(t'')\ dt'' \right] dt' \tag{6}$$

Following injection these solutions describe the transient activity in terms of the measured $C_{\mathrm{P}}^*(t)$ and the three rate constants. Conversely, for a given transient $C_{\mathrm{P}}^*(t)$ and for given measured sum of concentrations $C_{\mathrm{i}}^*(t)$ we may fit the three rate constants. An example is shown in figure 2. We use a simple least squares cost function for the fit, hence implicitly assuming Gaussian residuals. Optimization over the three parameters was carried out using a second order Newton scheme[1]

There are two different approaches from here. Up til now most studies assume that the rate constant are homogeneous in regions, see e.g. [1, 2], and the rate constants are fitted from the regional average activity transient. Alternatively we can fit individual rate constants for each pixel in the reconstructed volume [3], and *analyse* for homogeneity. However, since it is quite tedious to fit the kinetic model, the latter approach has not found widespread use. In the upper panel of figure 3 we show the result of such a pixel by pixel fit. The artifacts outside the elliptic area of the brain are due to the reconstruction scheme used (Filtered backprojection).

The database used for these experiments are PET data collected at the PET center at Rigshospitalet, Copenhagen. The subject described in this paper is a 43 year old woman with multiple sclerosis. Data are aquired on a

[1] Based on the solution to the kinetic model it is straightforward to compute the second derivatives.

GE4096 plus (General Electric Medical Systems), sampling 15 slices simultaneously. The dynamic scans after injection of 200 MBq F-18 labelled FDG are performed over 60 minutes, yelding 34 contiguous time frames of increasing duration in order to provide a reasonable sampling of the C_i^* curve: (10@6 sec; 3@20 sec; 8@60 sec; 5@120 sec; 8@300 sec). A single such curve is shown in figure 2. Images were reconstructed in $128x128$ matrices ($2mm^2$ pixels) by standard Filtered backprojection (Ramp filter with Hann window). Correction for attenuation is based on a separate transmission scan with a rotating Germanium pin source. For further introduction to PET scan techniques see e.g., [2]

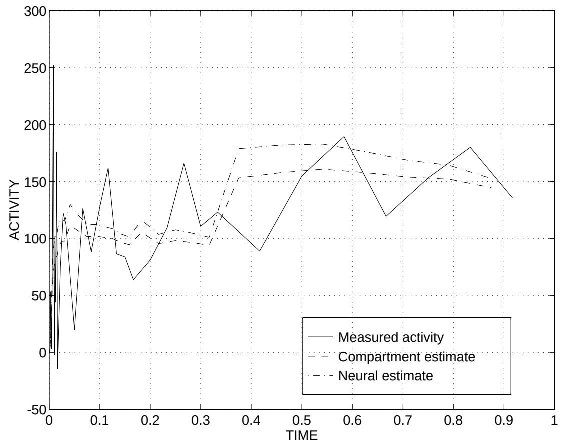
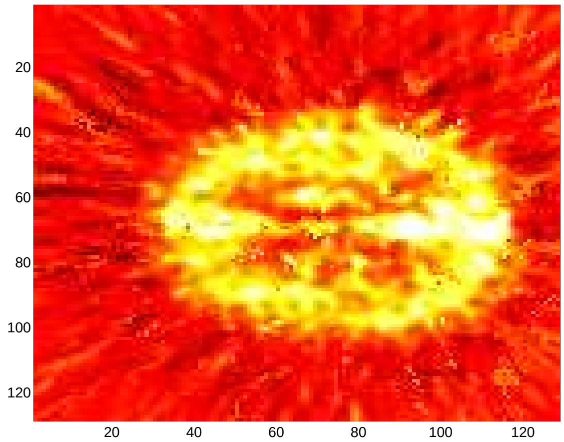


Figure 2: C_i^*(t) as measured by PET for a single pixel, as produced by the kinetic compartment model with the fitted rate constants, and as estimated by the neural network.

To avoid the tedious fitting procedure we here investigate the possibility of identifying the *inverse model* of the kinetics: we search for a *map* that provides an estimate of the three rate constants for a given observed transient. Our basic vehicle will be a simple feed-forward network.

What should be used for inputs?. The PET transients depend, c.f. equations (5-6), on the rate constants and on the time dependent arterial concentration (C_{P}^*). If we want to generalize from one set of pictures to another set (possibly another subject) we would need to provide both the observed PET transient and $C_{\mathrm{P}}^*(t)$. This will be pursued in future studies. In this work we tentatively train the network to predict the rate constants for pictures of a *single sequence* of PET images, hence, C_{P}^* is the same for all pixels and we need not provide it as input. In particular we train the network on transients from a small subsample of one slice of the PET volume scan Subsequently we apply the trained network to get the rate constants for the (large) remaining set of pixels.



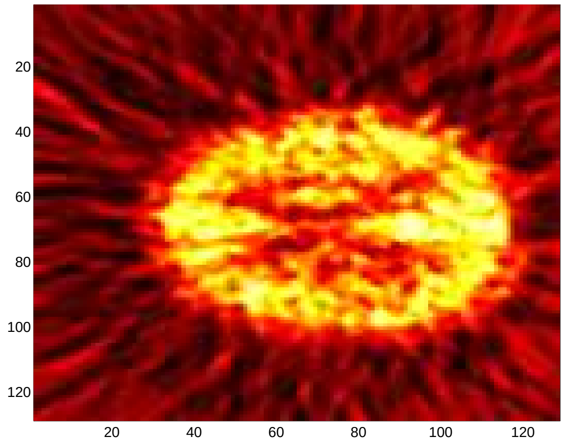


Figure 3: Image (slices) showing the k_1^* rate constant as determined by fitting the kinetic model (pixel by pixel) using a second Newton scheme (upper panel) and as determined by the neural net operating as inverse model for the kinetics (lower panel). The artifacts outside the elliptic area of the brain are due to the "Filtered backprojection" algorithm used for reconstruction.

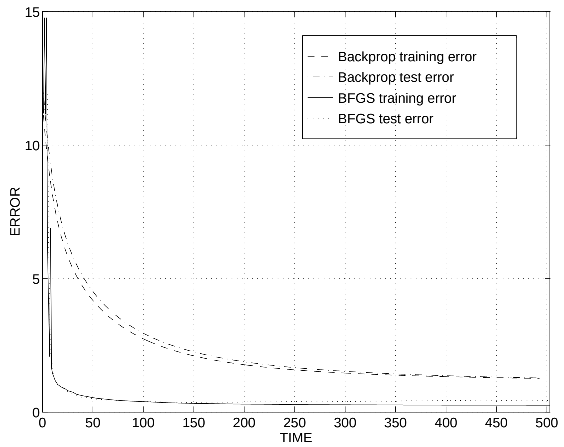


Figure 4: Comparison of time developments for the training process of standard Backpropagation and the BFGS scheme.

NETWORK DESIGN

The network considered is a standard feed forward net with a single layer of hidden units. The activation function of the hidden units are hyperbolic tangents, while the output units are linearly activated. The particular network for this study had 34 inputs corresponding to the activity transient of a given pixel. The net had ten hidden units and three output units. We trained the network by a pseudo second order scheme, the Broyden-Fletcher-Goldfarb-Shanno (BFGS) algorithm see e.g. [4]. This algorithm is a variable metric method that constructs a sequence of matrices approximating the inverse Hessian. Using BFGS instead of e.g. standard Backpropagation [5] provides a significant speed-up. This is quantified in figure 4, showing the time development of the training process with backprop and with the BFGS method respectively. Note the scale is in arbitrary *CPU time units* not iterations, since a backprop iteration is faster than a BFGS iteration.

For most adaptive systems the objective is not to learn the training set, but rather to perform well on a much larger set of conceivable inputs, i.e., generalization. The generalization ability depends on architecture and on size of the training set. Hence, for a given architecture, it is important to estimate the necessary number of training cases (pixels) to obtain good generalization. This relation is quantified by the so-called *learning curve* of the architecture as shown in figure 5. We note that a mere 4000 pixels are needed to obtain the asymptotic level of the test error.

Finally we apply the trained network to produce a full estimate of a "slice" of the PET scan as presented in the lower panel of figure 3. It is quite remarkable that the image quality of the rate constants reconstructed by the

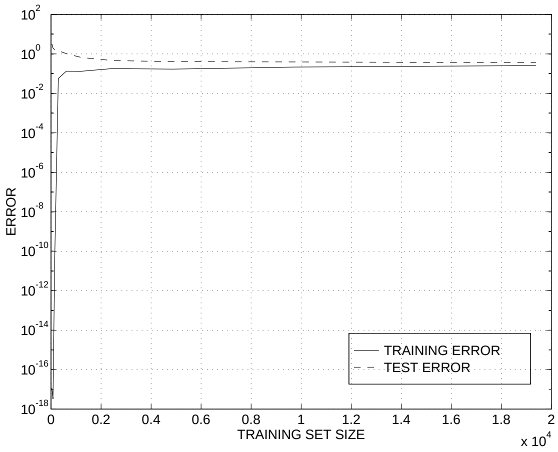


Figure 5: Learning curve for the feed forward net trained by BFGS, approximately 5000 pixels are needed to reach the asymptotic generalization level (about a quarter of the pixels in an image (slice)).

networks inverse model is less noisy than quality obtained from the fitting procedure. The reason is that the network capacity is limited by the inherent constraints of the network architecture, while the Newton fit can lead to arbitrarily (wrong) rate constants for a given pixel if the transient is very noisy. We also note that the execution time for the feed forward network is about two percent of the average time needed to obtain the kinetic constants by fitting the transients with the Newton method.

CONCLUSION

We have shown how a feed forward net may be used for identification of the inverse model for three compartment PET tracer kinetics. Not only is the use of the feed forward net significantly faster than fitting the kinetic model, but our tentative results seem to indicate that the rate constant distribution is regularized by the constraints imposed by the network architecture.

ACKNOWLEDGMENTS

This research was supported by the Danish Natural Science and Technical Research Councils through the Computational Neural Network Center (CONNECT).

REFERENCES

[1] L. Sokoloff, M. Reivich, C. Kennedy, M.H. Des Rosiers, C.S. Patlak, K.D. Pettigrew, O. Sakurada, and M. Shinohara: "The C-14-Deoxyglucose Method for the Measurement of local Cerebral Glucose Utilization: Theory, Procedures and Normal Values in the Conscious and Anesthetized Albino Rat" Journal of Neurochemistry **28** 897-916, (1977).

[2] M.E. Phelps: "Positron Emission Tomography (PET)". In: J.C. Maziotta and S. Gilman (Eds.): *Clinical Brain Imaging, Principles and Applications*, F.A. Davis Company, Philadelphia (1992).

[3] H. Sasaki, I. Kanno, M. Murakami, F. Shishido, and K. Uemuda: "Tomographic Mapping of KInetic Rate Constants in the FDG model Using Dynamic PET". Journal of Cerebral Blood Flow **6** 447-454 (1986).

[4] W. Press et al.: *Numerical Recipes in 'C'.* Cambridge University Press.

[5] D. E. Rumelhart and J. L. McClelland: "Back-propagation of errors" in *Parallel Distributed Processing. Explorations in the Microstructure of Cognition*, Vols. 1–2, MIT Press, Cambridge, Massachusetts (1986).

Appendix G

IEEE-SP94 article: Towards Semen Quality Assessment using Neural Networks

Reference for the article: [Linneberg et al., 1994]

TOWARDS SEMEN QUALITY ASSESSMENT USING NEURAL NETWORKS

Chr. Linneberg, P. Salamon, C. Svarer, and L.K. Hansen
CONNECT, Electronics Institute, build. 349
Technical University of Denmark,
DK-2800 Lyngby, Denmark
email: linne,salamon,csvarer,lkhansen@eileen.ei.dth.dk

and

J. Meyrowitsch
DynaVision, Copenhagen Science Park, SYMBION
DK-2100 Copenhagen Ø, Denmark

Abstract. The paper presents the methodology and results from a neural net based classification of human sperm head morphology. The methodology uses a preprocessing scheme in which invariant Fourier descriptors are lumped into "energy" bands. The resulting networks are pruned using *Optimal Brain Damage*. **Performance comparable to the error rate for human technicians is obtained.**

1 INTRODUCTION

Semen quality assessment is important to fertility studies and standards have been introduced by the World Health Organization[1]. Recent research has demonstrated that semen quality has decreased by 50% during the past 50 years in the western world, possibly as result of the increasing exposure to pollutants and changes in diet [4, 5]. The Atlas of Sperm Morphology[2] defines 19 classes of sperm cells based on morphology. The characteristic abnormalities involve shape modifications, including multiple heads and tails. Furthermore, sperm cell motility is also important for definition of abnormality. Since sperm cells show great variety, the morphological classification problem represents a complex task. Furthermore, the appearence of the specimen depends on several factors that are only partially controllable, such as reduced imaging quality, caused by co-fixation of precipitates and presence of non-sperm cells. Hence flexible and robust classification tools are necessary.

In this communication we analyse a neural net approach to automatic cell shape classification. Neural net learning makes it possible to compensate for the problems of specific imaging devices, and the user is not forced to produce an algorithmic description of the discriminant. Rather the same general software can adapt to a given setting based on examples produced by a skilled technician. Robustness is achieved by the networks' ability to generalize. The prime objective in machine learning is the ability to discriminate appropriately for *test* cases, i.e., examples that are different from the examples used for training of the neural system. Additional robustness can be obtained by careful *preprocessing* by which we mean model-based information processing, such as extraction of known salient features.

As a first step towards automatic quality assessment we have developed preprocessing algorithms for locating sperm cells and for extraction of salient features of the "head" and the "tail". In this presentation we will discuss the neural classifier for head shape classification. A convenient preprocessing strategy for shape discrimination has been proposed by [18, 11] and is based on Fourier analysis. A polygon in the image plane is isomorphic to a periodic sequence in the complex plane. For reviews and further analysis see also [11, 19, 16, 17, 14, 8]. Fourier descriptors are robust to the number of points used in sampling the shape and readily provide features which are invariant to changes in position, orientation, and starting point [18, 11].

Neural networks in conjunction with Fourier descriptors have been applied for shape discrimination previously in [13] and more recently in [8]. The objective of [13] was to recognize tools from a mechanical toolbox. The Fourier coefficients were crudely preprocessed by keeping only a few manually selected amplitudes. The results were promising; good performance was obtained with very few examples using a fully connected feed forward neural net trained by standard Backpropagation. However for sperm cell classification we have found that the inherent variability forces us to invoke optimized classifiers and more sophisticated preprocessing schemes. Our neural classifier approach was first described in [6], and in [8] it was tested on two sets of artificial cell shape data. Here we apply the system to real world data. In the context of cell discrimination the approach involves a new feature: the neural networks used have *adaptive* architectures. In particular, the networks are *pruned* to obtain the optimal connectivity. The advantage of the pruned networks is that they perform better on test data, ie. they do not simply memorize the training data but are able to *generalize* better than fully connected architectures. They are also able to select out the relevant inputs and thereby can mitigate the results of using too many input descriptors. More details on the simulator for the design of application specific architectures can be found in [7, 6]. Based on statistical theories of generalization, the best generalization is expected from the least complex networks. In our simulator we search for the best network with a pruning scheme based on the Optimal Brain Damage technique of [10]. We first train a large network that can easily implement the training set. Subsequently we compute the *saliency* of the weights of the network and delete a fraction of the weights with the lowest saliency. The network is re-trained and the procedure repeated as long as the pruned network is able to implement the training set. For noisy problems, of course, one would tolerate a certain amount of error on the training set in order not to *overfit*[1]. Our results indicate that pruning improves performance but, however, should be terminated well before reaching the smallest network that is able to do the training set.

2 SHAPE PREPROCESSING

As stated in the introduction, we used Fourier *shape* descriptors, i.e. the inputs to our neural networks were computed from the Fourier coefficients of the sequence of (x, y) coordinate pairs viewed as complex numbers: $z = x+iy$. The complex representation of a sequence of N points becomes periodic with the definition: $z_{N+k} = z_k$. The Fourier representation has the advantage that the description is rather robust to sampling the actual cell shape with different points and is rather insensitive to the number of points in the sample [3]. It also allows for finding descriptors which are invariant to translation, rotation, and the choice of a starting point in our description of the curve [18, 11]. The invariances are important because they effectively increase the size of the training set. Using invariant inputs is roughly equivalent to enlarging the training set by adding rotated, renumbered etc. versions of each training sample.

The invariances are achieved by transforming the shape to "standard form". For example, the centroid of the shape can be moved to the origin by setting the zeroth Fourier coefficients equal to zero. The standard form inherent in the popular "elliptic Fourier descriptors" [11] is based on the elliptic approximation for the shape which is obtained by truncating the Fourier series to terms with frequency one. This so-called *primary ellipse* then serves through its major axis to provide a natural starting point and a natural coordinate system which takes care of rotation and scale. The resulting transformed Fourier coefficients are invariants which provide a complete description of the shape and are called the elliptic Fourier coefficients.

To further reduce the complexity of the input representation, hence the network, we lumped the component by summing the energy in certain frequency bands.

3 NEURAL CLASSIFIER

Our simulation tool for design of adaptive neural architectures was described in [7, 6]. The initial architecture is an ordinary feed-forward network with one hidden layer. The standard output coding scheme for multi-class problems is used, hence, each class is represented by a specific output neuron. The

[1]For an implementation of this within time series prediction see [7].

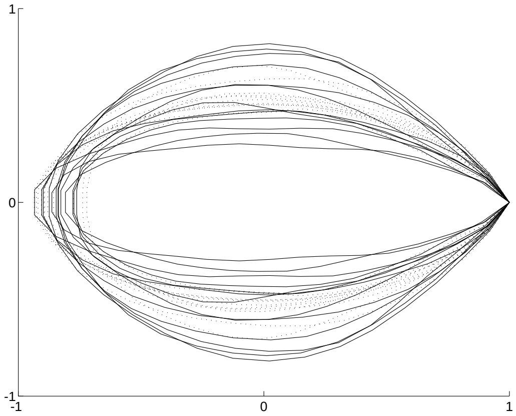


Figure 1: Reconstruction of cells from reduced sets of Fourier coefficient. (Solid: normal cell shapes; dotted: amorphous cell shapes)

activation functions of the hidden neurons are hyperbolic tangents, while the activation functions of the output neurons are linear. The training error is the usual sum of squared errors. The learning algorithm adjusts the weights of the network in order to minimize the error function. For the hidden layer, a second order modification of the back-propagation algorithm with batch learning is used, and, for the output neuron, the matrix inversion algorithm, described in [9] is used.

The algorithm used to adapt the neural network architecture is in brief:

1. Train the fully connected neural network using the learning algorithm.
2. Use the Optimal Brain Damage method to calculate the saliencies of all the weights in the network.
3. Find the weight in the network with the lowest saliency and remove this weight by setting it (permanently) to zero. If a hidden neuron is disconnected from the output neuron the hidden neuron is removed.
4. Retrain the network using the learning algorithm.

4 THE IMAGE DATABASE

The image database used in this work consisted of digitized images of fixated and Papanicolau stained semen specimens collected at the Department of Growth and Reproduction, Rigshospitalet Copenhagen. Each image obtained contains a variable number of manually classified cells. The classifications were far from certain and show that the problem includes significant overlap between the categories. Six independent laboratory technicians classified the images and on the average the technicians differed from the consensus classification on more than 15% of the samples. To simplify matters, we have restricted our attention to binary discrimination using two categories: "normal" and "abnormal". The set consisted of 50 cells; 25 from each category. The set was divided into a training set of 13 cells from each category, and a test set of the remaining 24 cells.

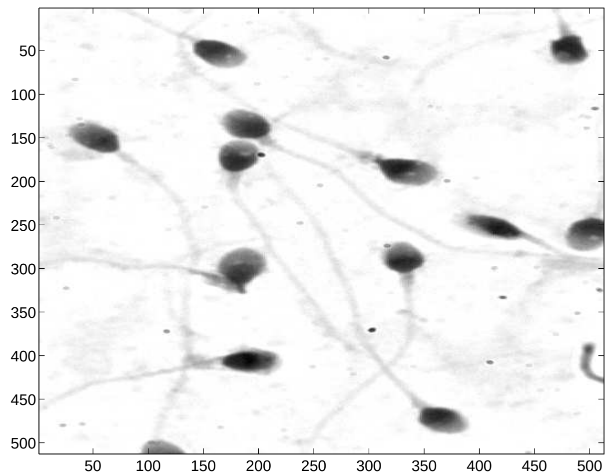


Figure 2: An original grey scale image used for manual labeling of the sperm cells

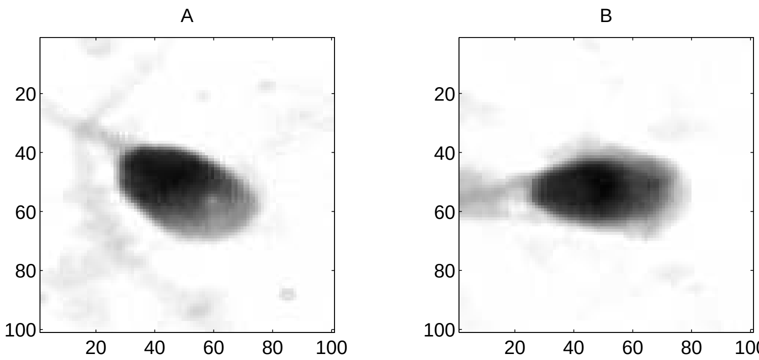


Figure 3: Examples of the two kinds of sperm cells. A) Normal cell shape B) Amorphous shape

5 EXPERIMENTAL RESULTS

The feed forward net was configured with 20 input units. The first 16 input units contained normalized Fourier amplitudes corresponding to 8 low-frequency pairs. The remaining four inputs represented the accumulated energies of 2 intermediate-frequency bands. The phases of all the Fourier components were discarded along with the amplitudes for the high-frequency components . The network has been initialized with random weights and a simple feed-forward architecture with 6 hidden neurons. It was found that the nets could be pruned significantly and that such pruning improved generalization. The training and test errors during a pruning run are depicted in figure 5. Contrary to previous experience using the pruning strategy [7, 6], the network with minimum test error was **not** the network with the smallest architecture which correctly implements the training set. Furthermore, we suspect that this is generally to be expected for problems with small training sets and significant overlap between categories. We are presently working on a crossvalidation methodology for choosing the best architecture along a pruning sequence.

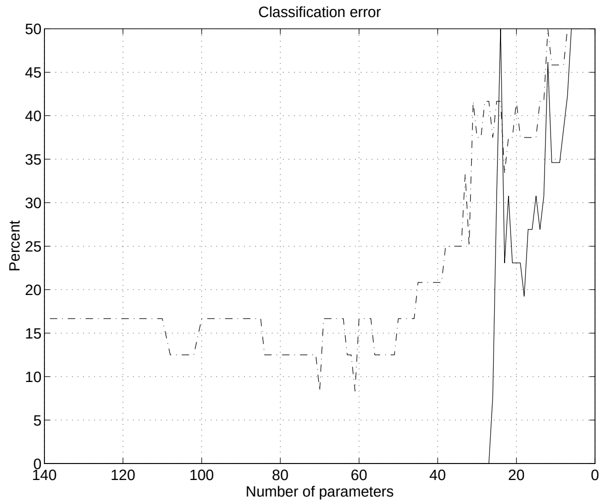


Figure 4: Training (solid line) and test errors (dot-dashed) as pruning progresses. For reference, the test error level of a technician is about 15%.

Inspecting the test errors it is found that pruning can decrease the error by about 25%.

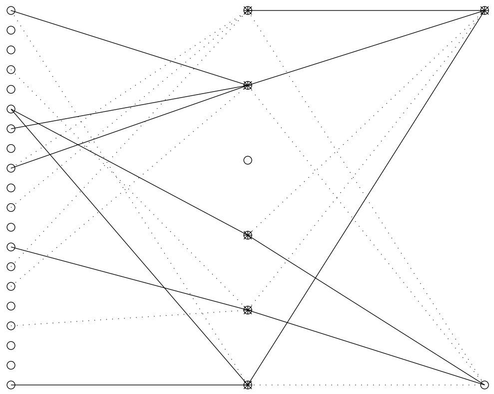


Figure 5: Typical pruned network that can learn the training set (Solid line - Positive weight, Dotted line - Negative weight).

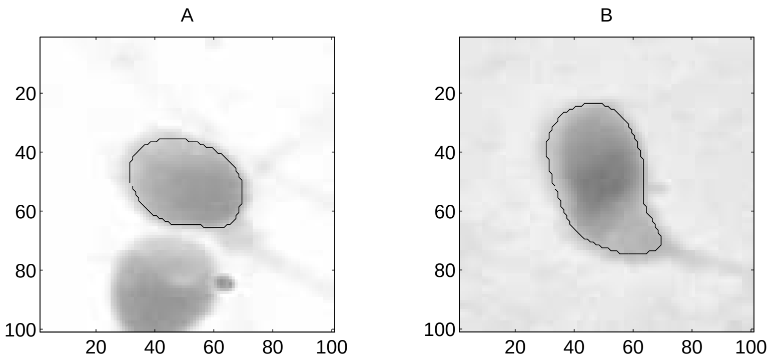


Figure 6: Two cell shapes from the test set that were not learned by the above network. A) normal cell B) amorphous cell.

6 CONCLUSION

It has been shown that neural networks are useful for morphological classification of cell shapes. Application specific architectures can be designed automatically which generalize well to an independent test set. The optimal architectures showed that only a few neurons and connections are necessary. Also it has been shown that preprocessing the cell shape coordinates using the complex Fourier transformation provides a well-suited and compact representation of the data. The optimal networks use less input information than the human eye apparently found necessary. Current work concerns larger databases, multi-class classification, and crossvalidation methodologies for choosing the best network among a pruning sequence.

ACKNOWLEDGMENTS

This research is supported by the Danish Research Councils for the Natural and Technical Sciences through the Danish Computational Neural Network Center. We thank A. Gewercman and N. Jørgensen (Dept. of Growth and Reproduction, Rigshospitalet, Copenhagen) and Erik Bostofte (Fertilitetsklinikken, Hvidovre Hospital and The Clinic of the medical doctors of Copenhagen) for useful discussions and for providing data for this project.

REFERENCES

[1] "WHO Laboratory Manual for the Examination of Human Semen and Semen-Cervical Mucus Interactions" Cambridge University Press, Cambridge 1992.

[2] M.M. Adelman and E.M. Cahill "Atlas of Sperm Morphology" ASCP Press, Chicago (1989).

[3] G.W. Strang "Introduction to Applied Mathematics" Cambridge University Press, Cambridge 1985.

[4] E. Carlsen, A. Gewercman, and N.E. Skakkebæk: "Evidence for decreasing quality of semen during past 50 years" British Medical Journal **305**, 609-613 (1992).

[5] R.M. Sharpe and N.E. Skakkebæk: "Are oestrogens involved in falling sperm counts and disorders of the male reproductive tract?" The Lancet **341**, 1392-1395 (1993).

[6] J. Gorodkin, L.K. Hansen, A. Krogh, C. Svarer, and O. Winter: "A Quantitative Study of Pruning by Optimal Brain Damage" Int. Journ. Neural Systems (1993).

[7] C. Svarer, L.K. Hansen, and J. Larsen: "On Design and Evaluation of Tapped-Delay Neural Network Architectures" The 1993 IEEE Int. Conference on Neural Networks San Francisco. Eds. H.R. Berenji et al., 45-51, (1993)

[8] C. Svarer, L.K. Hansen B. Wellhouse and P. Salamon: "Classification of Cell Shapes using Designer Networks". In preparation (1994).

[9] Simon A. Barton: "A Matrix Method for Optimizing a Neural Network", Neural Computation **3**, 450-459, (1991)

[10] Yann Le Cun, John S. Denker and Sara A. Solla: "Optimal Brain Damage", In Advances in Neural Information Processing Systems II (Denver 1989), ed. D.S. Touretzky, 396-404. San Mateo: Morgan Kaufmann, (1989)

[11] F.P. Kuhl and C.R. Giardina: "Elliptic Fourier Features of a Closed Contour" Computer graphics and image processing **18**, 226-258 (1982)

[12] Bill Wellhouse, Department of Mathematical Science, San Diego State University: "Description of cell shape data", Private communication, (1992)

[13] H-H. Wu and R.A. Schowengerdt: "Shape Discrimination Using inveriant Fourier Representation and a neural network classifier" SPIE **1569** Stochastic and Neural Methods in Signal Processing, Image Processing and, Computer Vision 147-154 (1991).

[14] L.M. Kerley and J.R. Knisley: "Complex Vectors and Image Identification" The College Mathematics Journal **24**, 166-174 (1993).

[15] J.C. Schon, J.T. Torre-Bueno, and G.B. Stefano: "Microscopic computer-assisted analysis of conformational state: reference to neuroimmunology" Advances in Neuroimmonology **1**, 252-259 (1991).

[16] G. Diaz, A. Zuccarelli, I. Pelligra, and A. Giani: "Elliptic Fourier Analysis of Cell and Nuclear Shapes" Computers and Biomedical Research **22**, 405-414 (1989).

[17] G. Diaz, D. Quacci, and C. Dell'Orbo: "Recognition of cell surface modulation by elliptic Fourier analysis" Computer Methods and Programs in Biomedicine **31**, 57-62 (1990).

[18] G.H. Granlund: "Fourier Preprocessing for Hand Print Character Recognition" IEEE Transactions on Computers **C-21**, 195-201 (1972).

[19] A.W. Partin, J.S. Schoeniger, J.L. Mohler, and D.S. Coffey: "Fourier analysis of cell motility: Correlation of motility with metastatic potential" Proc.Natl.Acad.Sci **86**, 1254-1258, (1989).

Bibliography

Akaike, H. (1968). On the use of a linear model for the identification of feedback systems. *aism*, 20:425–439.

Akaike, H. (1969). Fitting autoregressive models for prediction. *Ann. Inst. Stat. Mat.*, 21:243–247.

Anderson, J.A. and Rosenfeld, E. , editors (1988). *Neurocomputing: Foundations of Research.* MIT Press, Cambridge.

Barton, S.A. (1991). A matrix method for optimizing a neural network. *Neural Computation*, 3:450–459.

Buntime, W.L. and Weigend, A.S. (1993). Computing second derivatives in feed-forward networks: a review. *To appear in: IEEE Transactions on Neural Networks.*

Conradsen, K. (1984a). *En introduktion til statistik, bind 2A (in danish).* IMSOR, Lyngby, Denmark.

Conradsen, K. (1984b). *En introduktion til statistik, bind 2B (in danish).* IMSOR, Lyngby, Denmark.

Cover, T.M. (1965). Geometrical and statistical properties of systems of linear inequalities with applications in pattern recognition. *IEEE Transactions on Electronic Computers*, 14:326–334.

Cybenko, G. (1989). Approximation by superpositions of a sigmoidal function. *Mathematics of Control, Signals, and Systems*, 2:303–314.

Day, A.P. , Davenport, M.R. , and Camporese, D.S. (1992). Dispersive networks for nonlinear adaptive filters. In Kung, S.Y. et al., editor, *Neural Networks for Signal Processing II*, pages 464–473, Piscataway, NJ. (Helsingør 1992), IEEE.

Denker, J. , Schwartz, D. , Wittner, B. , Solla, S. , Howard, R. , Jackel, L. , and Hopfield, J. (1987). Large automatic learning, rule extraction, and generalization. *Complex Systems*, 1:877–922.

Diaz, G. , Zuccarelli, A. , Pelligra, I. , and Ghiani, A. (1989). Elliptic fourier analysis of cell and nuclear shapes. *Computers and Biomedical Research*, 22:405–414.

Fahlman, S.E. (1989). Fast-learning variations on back-propagation: An empirical study. In Touretzky, D. , Hinton, G. , and Sejnowski, T. , editors, *Proceedings of the 1988 Connectionist Models Summer School*, pages 38–51, San Mateo. (Pittsburg 1988), Morgan Kaufmann.

Fahlman, S.E. (1991). The recurrent cascade-correlation architecture. *CMU-CS-91-100, Intern report: Computer Science, Carnegie Mellon University*, pages 1–8.

Fahlman, S.E. and Lebiere, C. (1990). The cascade-correlation learning architecture. In Touretzky, D.S. , editor, *Advances in Neural Information Processing Systems*, volume 2, pages 524–532, San Mateo. (Denver 1989), Morgan Kaufmann.

Fog, T. , Nielsen, L.H. , Hansen, L.K. , Holm, S. , Law, I. , Svarer, C. , and Paulson, O. (1994). Neural estimation of kinetic rate constants from dynamic pet-scans. In Vlontzos, J. et al., editor, *Neural Networks for Signal Processing IV*, pages 616–623, Piscataway, NJ. (Ermioni 1994), IEEE.

Frean, M (1990). The upstart algorithm: A method for constructing and training feedforward neural networks. *Neural Computation*, 2:198–209.

Gorodkin, J. , Hansen, L.K. , Krogh, A. , Svarer, C. , and Winther, O. (1993). A quantitative study of pruning by optimal brain damage. *International Journal of Neural Systems*, 4:159–169.

Granlund, G.H. (1972). Fourier preprocessing for hand print character recognition. *IEEE Transactions on Computers*, C-21:195–201.

Hansen, L.K. (1993a). Controlled growth of cascade correlation nets. *Unpublished notes, Electronics Institute, The Tecnical University of Denmark.*

Hansen, L.K. (1993b). Notes on generalization. *Unpublished handwritten notes.*

Hansen, L.K. and Hansen, A. (1992). *Private Communication, Experiments at The Two-spirale Problem using Fahlmanns Cascade-correlation Algorithm.* Electronics Institute, DTH, Lyngby.

Hansen, L.K. and Rasmussen, C.E. (1994). Pruning from adaptive regularization. *Neural Computation*, 6(6):1223–1232.

Hansen, L.K. , Rasmussen, C.E. , Svarer, C. , and Larsen, J. (1994). Adaptive regularization. In Vlontzos, J. et al., editor, *Neural Networks for Signal Processing IV*, pages 78–87, Piscataway, NJ. (Ermioni 1994), IEEE.

Hansen, L.K. and Salamon, P. (1990). Neural network ensembles. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 12:993–1001.

Hassibi, B , Stork, D.G. , and Wolff, G.J. (1993). Optimal brain surgeon and general network pruning. *Submitted for publication*, pages 1–8.

Hertz, J , Krogh, A. , and Palmer, R.G. (1991). *Introduction to the Theory of Neural Computation.* Addison-Wesley, New York.

Hornik, K. (1991). Approximation capabilities of multilayer feedforward networks. *Neural Networks*, 4:251–257.

Hornik, K. , Stinchcombe, M. , and White, H. (1989). Multilayer feedforward networks are universal approximators. *Neural Networks*, 2:359–366.

Kashyap, R.L. (1980). Inconsistency of the AIC rule for estimating the order of autoregressive models. *IEEE Transactions on Automatic Control*, AC-25:996–998.

Kerley, L.M. and Knisley, J.R. (1993). Complex vectors and image identification. *The College Mathematics Journal*, 24-2:166–174.

Kuhl, F.P. and Giardina, C.R. (1982). Elliptic fourier features of a closed contour. *Computer Graphics and Image Processing*, 18:236–258.

Larsen, J. (1993). *Design of Neural Networks*. Ph.D Thesis, Electronics Institute, DtH, Lyngby.

Le Cun, Y. , Boser, B. , Denker, J.S. , Henderson, D. , Howard, R.E. , Hubbard, W. , and Jackel, L.D. (1989). Backpropagation applied to handwritten zip code recognition. *Neural Computation*, 1:541–551.

Le Cun, Y. , Denker, J.S. , and Solla, S.A. (1990). Optimal brain damage. In Touretzky, D.S. , editor, *Advances in Neural Information Processing Systems*, volume 2, pages 598–605, San Mateo. (Denver 1989), Morgan Kaufmann.

Le Cun, Y. , Kanter, I. , and Solla, S.A. (1991). Eigenvalues of covariance matrices: Application to neural-network learning. *Physical Review Letters*, 66(18):2396–2399.

Levin, E. , Denker, J.S. , and Solla, S.A. (1990). A statistical approach to generalization in layered neural networks. *Proceedings of the IEEE*, 78(10):1568–1574.

Linneberg, C. , Salamon, P. , Svarer, C. , and Hansen, L.K. (1994). Towards semen quality assessment using neural networks. In Vlontzos, J. et al., editor, *Neural Networks for Signal Processing IV*, pages 509–517, Piscataway, NJ. (Ermioni 1994), IEEE.

Ljung, L. (1987). *System Identification: Theory for the User*. Prentice-Hall, Inc., New Jersey.

MacKay, D.J.C. (1992a). Bayesian interpolation. *Neural Computation*, 4:415–447.

MacKay, D.J.C. (1992b). A practical framework for backpropagation networks. *Neural Computation*, 4:448–472.

Marshall, D.F. , Jenkins, W.K. , and Murphy, J.J. (1989). The use of orthogonal transforms for improving performance of adaptive filters. *IEEE Transactions on Circuits and Systems*, 36(4):474–483.

McCulloch, W.S. and Pitts, W. (1943). A logical calculus of ideas immanent in nervous activity. *Bulletin of Mathematical Biophysics*, 5. Reprinted in [Anderson and Rosenfeld, 1988].

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

Moody, J.E. (1992). The effective number of parameters: An analysis of generalization and regularization in nonlinear learning systems. In Moody, J.E.. et al.., editor, *Advances in Neural Information Processing Systems 4, Proceedings of the 1991 Conference*, volume 4, pages 847–854, San Mateo, CA. (Denver 1992), Morgan Kaufmann Publishers.

Moody, J. and Darken, C. (1988). Learning with localized receptive fields. In Touretzky, D. , Hinton, G. , and Sejnowski, T. , editors, *Proceedings of the 1988 Connectionist Models Summer School*, pages 133–143, San Mateo. (Pittsburg 1988), Morgan Kaufmann.

Moody, J. and Darken, C. (1989). Fast learning in networks of locally-tuned processing units. *Neural Computation*, 1:281–294.

Oppenheim, A.V. and Schafer, R.W. (1975). *Digital Signal Processing.* Prentice-Hall, Inc., New Jersey.

Orfanidis, S.J. (1988). *Optimum Signal Processing.* 2nd ed., McGraw-Hill, New York.

Orfanidis, S.J. (1990). Gram-Schmidt neural nets. *Neural Computation*, 2:116–126.

Parker, T.S. and Chua, L.O. (1989). *Practical Numerical Algorithms for Chaotic Systems.* Springer-Verlag, New York.

Priestly, M.B. (1988). *Non-linear and Non-stationary Times Series Analysis.* Academic Press, New York.

Rosenblatt, F. (1958). The perceptron: a probabilistic model for information storage and organization in the brain. *Psychological Review*, 65:386–408.

Rosenblatt, F. (1962). *Principles of Neurodynamics.* Spartan, New York.

Rumelhart, D.E. , Hinton, G.E. , and Williams, R.J. (1986a). Learning representations by back-propagating errors. *Nature*, 323:533–536. Reprinted in [Anderson and Rosenfeld, 1988].

Rumelhart, D.E. , McClelland, J.L. , and the PDP Research Group (1986b). *Parallel Distributed Processing: Explorations in the Microstructure of Cognition*, volume 1. MIT Press, Cambridge.

Schwartz, D.B. , Samalam, V.K. , Solla, S.A. , and Denker, J.S. (1990). Exhaustive learning. *Neural Computation*, 2:371–382.

Svarer, C. , Hansen, L.K. , and Larsen, J. (1993a). On design and evaluation of tapped-delay neural network architectures. In Berenji, H.R. et al., editor, *icnn*, volume 1, pages 46–51, Piscataway, NJ. (San Francisco 1993), IEEE.

Svarer, C. , Hansen, L.K. , Larsen, J. , and Rasmussen, C.E. (1993b). Designer networks for time series processing. In Kamm, C.A. et al., editor, *Neural Networks for Signal Processing III*, pages 78–87, Piscataway, NJ. (Baltimore 1993), IEEE.

Thodberg, H.H. (1991). Improving generalization of neural networks through pruning. *Journal of Neural Systems*, 1:317–326.

Tong, H. and Lim, K.S. (1980). Threshold autoregression, limit cycles and cyclical data. *Journ. Roy. Stat. Soc. B*, 42:245.

Weigend, A.S. and Gershenfeld, editors (1994). *Time Series Prediction: Forecasting the Future and Understanding the Past.* Addison-Wesley Publishing Company, Reading, MA.

Weigend, A.S. , Huberman, B.A. , and Rumelhart, D.E. (1990). Predicting the future: a connectionist approach. *International Journal of Neural Systems*, 1:193–209.

Wellhouse, W. (1993). *Classification of Cell Shapes by Means of a Neural Network with Fourier Preprocessing.* Master Thesis, Department of Mathematical Science, San Diego State University, San Diego.

Wellhouse, W. and Salamon, P. (1992). *Description of cell shape data.* Private Communication, Department of Mathematical Science, San Diego State University, San Diego.

Widrow, B. (1962). Generalization and information storage in networks of adaline "neurons". In Yovits, M.C. , Jacobi, G.T. , and Goldstein, G.D. , editors, *Self-Organizing Systems 1962*, pages 435–461, Washington. (Chicago 1962), Spartan.

Widrow, B. and Hoff, M.E. (1960). Adaptive switching circuits. In *1960 IRE WESCON Convention Record*, volume 4, pages 96–104. IRE, New York.

Wu, H.H. and Schowengerdt, R.A. (1991). Shape discrimination using invariant fourier representation and a neural network classifier. *Stochastic and Neural Methods in Signal Processing, Image Processing, and Computer Vision*, 1569:147–154.