Generalization in Neural Networks

Carl Edward Rasmussen
Electronics Institute
Technical University of Denmark
Email: ed@eileen.ei.dth.dk

18 August 1993

Abstract

This report is concerned with methods for optimizing the generalization ability of neural networks. The framework is developed to deal with regression type problems, where the networks are trained on a limited amount of noisy data. In this context the problem can be formulated as finding the optimal trade off between data fit and model complexity.

Two paradigms for reducing model complexity are discussed: pruning and weight decay. It is shown by numerical experiments that application of weight decay is essential for obtaining good generalization performance. This is explained by the way in which weight decay confines the space of possible networks to a space of 'reasonable' networks.

Two methods for making statistical estimates of the generalization performance *without* use of validation sets are presented: the Generalization method and the Bayesian method. The advantage of not needing validation sets is that all available data can be utilized in the training phase. This feature is important since the optimal generalization ability of a model is directly related to the amount of available training data.

The Generalization method is an extension of Akaikes FPE estimator, which explicitly takes the application of weight decay into account. In this method the generalization ability is estimated by averaging over the ensemble of possible training sets, which are consistent with the 'true' function. The method allows for optimal setting of weight decay and estimates generalization performance.

The Bayesian framework uses the *evidence* to measure the plausibility of models. This framework embodies *priors* on the network weights, which are shown to be identical to the application of weight decay.

The two methods are tested numerically on the sunspot problem. On linear models for this (limited) problem both methods work well, and marked similarities are found between the two methods. Both methods also exhibit the ability to prune weights, when individual weight decay parameters are applied to each weight in the network.

Preface

The work presented in this report was done in connection with an M.Sc. exam project at the Technical University of Denmark. The work was carried out at the Electronics Institute, with Lars Kai Hansen as supervisor. The project was begun in September 1992 and finished in the middle of June 1993.

I wish to thank the 'Radio Parts' foundation for financial assistance in connection with a visit to the University of Toronto. Further, I wish to thank the people of the neural network group at the Electronics Institute. Specially, I'm in debt to Lars Kai Hansen for his endless patience, inspiration and encouragement. Thanks.

Lyngby, 14. June 1993

Carl Edward Rasmussen

This revised version of the thesis contains corrections to a few mistakes and typos in the first version. These alterations are all of minor importance.

C.E.R.

Contents

List of Symbols

$\langle \ldots \rangle_p$	Average over all training sets of size p
$\langle \ldots \rangle_\nu$	Average over noise contributions
$\langle \ldots \rangle_x$	Average over the input distribution
$\mathbf{A}$	Matrix given by: $\frac{1}{\sigma^2}(\mathbf{B} + \xi \mathbf{I})$
$\mathbf{a}_j$	given by: $\frac{1}{p}\sum_{\alpha=1}^{p} \nu^\alpha \frac{\partial F_{w*}^\alpha}{\partial w_j}$
α	Index to the training examples, runs from $1 \ldots p$
$\mathbf{B}$	Matrix of second order derivative of E_D with respect to the weights
$\mathbf{b}_j$	given by: $-\frac{1}{p}\xi_j w_j^*$
D	Training data base
$\mathbf{D}$	Diagonal matrix of weight decays: $\xi \mathbf{I}$
$\delta_{ii'}$	Kronecker's delta
$\delta \mathbf{w}$	Weight fluctuation: $\mathbf{w} - \mathbf{w}^*$
E	Network cost function
E_D	Network data error: $\frac{1}{2}\sum_{\alpha=1}^{p}(F(\mathbf{x}^\alpha) - t^\alpha)^2$
E_w	Network weight error: $\frac{1}{2}\sum_{i=1}^{N} w_i^2$
$\langle \epsilon_G \rangle_p$	Average generalization error
$\langle \epsilon_T \rangle_p$	Average training error
F_w	The network function
F_{w*}	The teacher function
F_w^α	Short for $F_w(\mathbf{x}^\alpha)$
$G(\ldots)$	Neuron transfer function
γ	Number of well determined parameters
$\tilde{\mathbf{H}}$	Cost Hessian (second order cost derivative with respect to the weights)
$\mathbf{H}$	Cost Hessian divided by p
$\mathcal{H}$	Model (Network architecture)
$\mathbf{I}$	Unit matrix
λ_i	Eigenvalues of $\mathbf{B}$
$\mathbf{J}$	Matrix given by: $\mathbf{H} + \frac{1}{p}\mathbf{D}$
N	Number of parameters (weights and biases) in the network
∇E	Cost gradient, with respect to the weights
ν	Additive noise contribution
Ω	Size of logarithmic interval for a scale parameter
$P(\ldots)$	Probability distribution or probability density
$P(\ldots \mid \ldots)$	Conditional probability distribution or density
p	Number of examples in the training data set
$\mathbf{Q}_{ii'}$	given by: $\sum_{\alpha=1}^{p} \mathbf{x}_i^\alpha \mathbf{x}_{i'}^\alpha + \xi \delta_{ii'}$
$\mathbf{Q}_\alpha$	given by: $\mathbf{Q} - \mathbf{x}^\alpha (\mathbf{x}^\alpha)^T$
$\mathbf{q}_i$	given by: $\sum_{\alpha=1}^{p} \mathbf{x}_i^\alpha t^\alpha$
$\mathbf{q}^\alpha$	given by: $\mathbf{q} - \mathbf{x}^\alpha t^\alpha$

$\mathbf{s}$	Vector containing the scales for a model
σ^2	Noise variance
T	(Super script) denotes vector transpose
$\mathrm{Tr}(\ldots)$	The trace of a matrix
t	The network output target value
$\mathbf{w}$	Vector containing the network weights
$\tilde{\mathbf{w}}$	Weights rotated into the eigenvector basis of $\mathbf{B}$
$\mathbf{w}^*$	Teacher weights
$\mathbf{x}$	Network input vector
ξ	Regularization parameter (weight decay)
y	The network output
ζ	Weight saliency

Overview

This report is divided into six chapters and two appendices. The first two chapters are introductory.

Chapter 1 is a general introduction to artificial feed forward neural networks. The introduction is short and focuses on the problems of immediate relevance to training of networks for regression type problems: the backprop learning rule is presented, and a few hints to speed up convergence are given. The chapter concludes with a presentation of the sunspot problem which is studied extensively throughout the following chapters.

Chapter 2 introduces the problem of confining the network complexity to a level which is reasonable for the given modeling task. The problem of 'overlearning' is demonstrated, and two techniques for constraining model capacity are given: pruning and weight decay. The need for methods to control pruning and the setting of weight decays are emphasized, and the shortcomings of validation sets, and 'Leave One Out' validation are demonstrated.

In chapter 3 Akaike's 'Final Prediction Error' (FPE) estimator is derived. The FPE estimator is subsequently used as a stop criterion in a pruning session utilizing 'Optimal Brain Damage'. It is shown that the performance of the networks obtained is critically dependent on the setting of weight decay parameters. It is also shown that the FPE estimator can not be used to select the optimal weight decay.

In chapter 4 the 'Generalization Method' for optimal setting of weight decay is covered. The generalization ability of a network with weight decay is estimated. Hereby the optimal settings for one or more weight decay parameters can be estimated. The method is tested on a linear model utilizing both a single weight decay and separate weight decays for each parameter in the model. Simulations for nonlinear networks were briefly tried, but further work is needed to evaluate performance for these networks.

Chapter 5 covers the 'Bayesian' method for model optimization. This method combines optimization of weight decays with ranking of different network solutions, which can also be used to make a stop criterion for pruning. A linear model with a single weight decay is pruned and the evidence used as a stop criterion. Results from the Generalization method are compared with those for the Bayesian scheme. No extensive simulations of nonlinear networks was performed.

The conclusions are collected in chapter 6.

In appendix A, a case study of a simple one parameter learning problem is presented. It is a draft for a paper which is coauthored by Lars Kai Hansen, and has been sent to *Neural Computation*. It contains a discussion of the scenario, where regularization leads to deletion of weights.

Appendix B contains a paper entitled 'Designer Networks for Time Series Processing', coauthored by Claus Svarer, Lars Kai Hansen and Jan Larsen. It has been accepted for presentation at the '1993 IEEE-SP Workshop on Neural Networks for signal Processing' in Baltimore. This paper demonstrates the usefulness of 'Optimal Brain Damage' used with the 'Final Prediction Error' estimate as a stop criterion, for two tasks in signal processing.

Chapter 1

Modeling with Feed Forward Neural Networks

The purpose of this chapter is twofold; the concept of (artificial) feed forward neural networks will be introduced, and a learning rule for supervised learning is given. The introduction to neural networks will be confined to the problems of immediate relevance to regression in feed forward networks. For a more general introduction to neural networks see for example (Hertz et al., 1991). The concept of (supervised) learning will be covered, and the back propagation learning algorithm will be introduced. A brief discussion of problems with convergence time is given, and the learning procedure used in the following chapters is specified.

Finally, the sample regression problem of predicting future sunspot activity from the activity of previous years is introduced. This problem will be used to monitor the performance of the methods discussed throughout the report.

1.1 Feed Forward Neural Networks

Neural Networks (NN's) can be thought of simply as a certain class of functions, mapping from one (possibly high dimensional) space to another. The theory of feed forward neural networks falls naturally in a broader group of approximation techniques, such as Fourier series, Chebyshev polynomials, splines etc. The adjustment of the parameters of an NN is an iterative procedure called learning or training.

In statistics, training of an NN is a task which belongs to the class of nonparametric inference. This label refers to the fact that the function which the network implements is not a priori confined to a certain class chosen for example through an understanding of the physics underlying the system which is being modeled (parametric inference). Rather, NN's are universal, ie. they can (with certain restrictions) implement any functional relationship.

1.1.1 The Neuron

The basic unit of an NN is the neuron. In artificial NN's the units are formalized neurons, with very little resemblance to their biological counterparts. This report is solely concerned with artificial NN's, and the formalized neurons will be referred to simply as neurons or units.

The neuron normally has several inputs and a single output. The neuron computes its output by making a weighted sum of the inputs, adding a bias and applying an activation function. For a neuron with N inputs, the relationship is:

$$y = G(\sum_{i=1}^{N} w_i x_i + w_0) \tag{1.1}$$

where x_i are the inputs, w_i are the weights (index i running over all inputs), w_0 is the bias and G is the activation function. Formally the bias term w_0 can be thought of as a weight from an input which is always unity, so in the following 'weights' will refer both to weights and biases. The properties of a neuron with a certain activation function is determined once the weights are given.

Many different activation functions have been studied. These range from deterministic binary or continuous functions, to stochastic units, and units with memory. The first activation function to be studied in the context of computational neuro science was that of the binary neuron (McCulloch and Pitts, 1943) with the activation function given by (the Heaviside function):

$$G(x) = \begin{cases} 1 & x \geq 0 \\ 0 & x < 0 \end{cases} \tag{1.2}$$

In this case, minus the bias term $-w_0$ can be thought of as a threshold parameter. If the weighted sum of inputs exceeds the threshold, the neuron is 'on', otherwise not.

Three examples of continuous activation functions will be given. Firstly we have the linear unit, with transfer function:

$$G(x) = ax \tag{1.3}$$

where a is a constant. The linear unit has the great advantage that it can be studied in detail analytically. On the other hand the performance of networks consisting solely of linear units is limited (Minsky and Papert, 1969).

Secondly we have the non-linear sigmoidal and hyperbolic tangent functions:

$$G_1(x) = \frac{1}{1+\exp(-2x)} \qquad \text{or} \qquad G_2(x) = \tanh(x) \tag{1.4}$$

These functions differ only by a linear transformation of the output ($G_2 = 2G_1 - 1$). As we will soon see, such a linear transformation of the output is of little consequence, since in a network it can be absorbed by the weights and biases of the following units (hence, the linear unit often has $a = 1$).

Because of the learning rules to be considered, we will only allow differentiable activation functions.

1.1.2 The Feed Forward Network

A neural network is a network of connected neurons. The structure of a network can conveniently be visualized by a schematic diagram showing neurons and connections, fig. 1.1. The neurons are naturally divided into three groups: the input units[1], which receive the inputs to the network, the output units which deliver the output, and the hidden units which don't make contact outside the network. The hidden units receive their inputs from other neurons in the network, and their outputs are propagated inside the network.

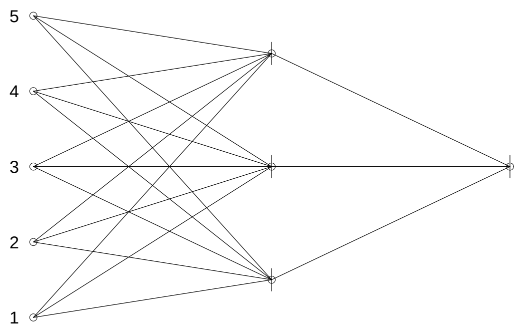

Figure 1.1: A schematic diagram of a fully connected feed forward network. There are 5 inputs, 3 hidden units and a single output unit. The lines represent weights, the circles represent input buffers and the small vertical bars represent the neurons.

In general any topographical arrangement of units and weights can be envisaged. In the following we will constrain ourselves to consider feed forward networks only. In a feed forward net the neurons can be arranged into layers such that the units in each layer feed their output forward to consecutive layers only. Networks that feed information to other units in the same layer or units in earlier layers are known as recurrent networks. These networks are not uninteresting but their formal treatment is very different from that of feed forward nets (since the recurrent connections effectively introduce memory in the network).

Networks where all neurons in each layer have connections to all the neurons in the following layer are called fully connected (as opposed to sparse networks). A fully connected network with one linear output unit, L hidden hyperbolic tangent units, and M inputs implements the function:

[1]The input units are not really neurons, rather they act as buffers which hold the input to the network.

$$y = \sum_{i=1}^{L} w_i \tanh(\sum_{j=1}^{M} w_{ij} x_j + w_{i0}) + w_0 \tag{1.5}$$

It turns out that functions of this kind are quite powerful for modeling data. As examples of networks using this simple two layer structure are: NETtalk (Sejnowski and Rosenberg, 1987), a network which maps written words into phonemes, and the Hyphenation Network (Brunak and Lautrup, 1989), which decides where Danish words can be hyphenated for typographical line breaking. Networks with more layers can perform even more difficult tasks; a four layer network was used (Le Cun et al., 1989) for recognition of hand written postal codes.

The applications cited above are all classification problems, ie. the network output is a label (eg. a 'hyphenate' or 'don't' label). It is well known that many decision processes such as chosing a label represents highly nonlinear mappings, which makes NN's an obvious choice for these problems. There are also many examples of networks trained for regression, ie. the network output is a continuous value. Time series prediction is an example which has been studied extensively, for example in (Svarer et al., 1993; Weigend et al., 1990) and in appendix B. The following treatment will deal only with networks trained for regression[2].

1.2 The Back-Propagation Learning Rule

We have seen that NN's are capable of implementing interesting functions, but as yet no mention has been of how to chose a network architecture and appropriate weights for a given task. Unfortunately it is far from easy to decide what size and architecture is suitable in a given situation. We will let this question lie for the moment (but not for long) and concentrate on chosing the weights once a network architecture is given.

The weights of the network are found using a data base D consisting of p examples:

$$D = \{\mathbf{x}^\alpha, t^\alpha | \alpha = 1 \dots p\} \tag{1.6}$$

where $\mathbf{x}$ are the network inputs and t are the output targets. Both the input and the output can be vectors, but for sake of simplicity the following derivations will deal only with scalar output (generalization to the vectorial case is straight forward[3]). The iterative procedure which leads to the final weights is called learning. The most common learning rule for feed forward networks is called back propagation of errors, or backprop (Rumelhart et al., 1986b; Rumelhart et al., 1986a).

The key to the learning task is the formulation of a cost function, which penalizes the mistakes of the network. The learning procedure is designed to find a minimum in the cost function. A common choice of cost function is the sum of squared errors:

[2] It is believed that the presented framework can be adapted to deal with classification tasks (MacKay, 1992b).

[3] However, care should be taken when deciding how to weight the errors from the components of the output vector against each other in the cost function.

$$E(w) = \frac{1}{2}\sum_{\alpha=1}^{p}(t^{\alpha} - y^{\alpha})^2 \tag{1.7}$$

where t is the target and y is the network output. Implicitly y is a function of the weights, see equation (1.5). The choice of a quadratic cost function is strongly tied to an assumption about the noise in the training data base being additive Gaussian, in which case the cost function can be interpreted as minus the *log likelihood* of the model parameters (the weights). This point will be discussed in further detail in chapters 4 and 5. Suffice it here to say that the cost function should be an increasing function of some measure of distance between the actual an desired network output, and careful considerations are needed to find the 'most appropriate' distance measure for any specific learning task.

In order to find a minimum of the nonlinear cost function in weight space, we employ the method of gradient descent. The network is initialized with random weights. The examples are presented to the network and the weights are updated according to the cost gradient:

$$\Delta\mathbf{w} = -\eta\frac{\partial E(\mathbf{w})}{\partial \mathbf{w}} \tag{1.8}$$

where $\mathbf{w}$ is a vector containing all the weights of the network and η is a (positive) constant called the learning rate. If η is set to an appropriate value (too large learning rates lead to instability) the network will converge to a (local or global) minimum in cost. In this situation the network has learned all the examples in the training set, in a least squares sense.

Lets try it out on the two layer network given by equation (1.5). The update rule for the weights of the output unit is given by:

$$\begin{aligned}
\Delta w_i &= -\eta\frac{\partial E(w)}{\partial w_i} && (1.9)\\
&= \eta\sum_{\alpha=1}^{p}(t^{\alpha} - y^{\alpha})\frac{\partial y^{\alpha}}{\partial w_i} && (1.10)\\
&= \eta\sum_{\alpha=1}^{p}(t^{\alpha} - y^{\alpha})G(\sum_{j=1}^{M} w_{ij}x_j^{\alpha} + w_{i0}) && (1.11)
\end{aligned}$$

where $G(x)$ is the activation function of the hidden units. Similarly the update for the weights of the hidden units is:

$$\begin{aligned}
\Delta w_{ij} &= -\eta\frac{\partial E(w)}{\partial w_{ij}} && (1.12)\\
&= \eta\sum_{\alpha=1}^{p}(t^{\alpha} - y^{\alpha})w_i G'(\sum_{j'=1}^{M} w_{ij'}x_{j'}^{\alpha} + w_{i0})x_j^{\alpha} && (1.13)
\end{aligned}$$

The name back propagation of errors stems from the form of these equations, which is always a product of the error sent backwards through the network and the input at that stage. The rule can be extended to any number of hidden layers.

In the backprop rule as stated above, all examples in the training set must be presented to the network in order to find an update. This is called batch-mode learning. The learning procedure may be changed a little by including only a fraction of the examples in the sum over α (for example when the training data base is very large), in which case the learning is said to be on-line. (Note that the on-line procedure converges only in the $\eta \to 0$ limit.)

1.2.1 Local Minima

There is no guarantee that the backprop learning rule finds the global minimum in cost. If there are local minima (and in many learning problems there are) the learning algorithm can be caught by these. Fortunately, these local minima do not eliminate the utility of backprop. Usually the algorithm finds a local minimum, with a cost which is quite close to the cost of the global minimum. This fact is well known, see for example (Hertz et al., 1991).

When two similar networks are trained on the same data (but with different initial random weights) they will typically find different final network weights, but the difference between the training costs will be small. This discussion shows that one must be cautious about results relying on a single training. Rather a few networks should be trained on each task, to eliminate the effects of an 'unlucky' initialization of weights.

1.2.2 Speeding Up Backprop

The raw backprop rule is notoriously slow in converging on many learning tasks. Several numerical tricks are commonly used to speed up convergence. Since minimization of convergence time is not a primary objective of this study, no overview of methods will be given here, see (Hertz et al., 1991). However, some means are needed to make learning terminate in tolerable time. Two methods will be considered.

If the output[4] unit of the network is linear, we note that the equation:

$$\frac{\partial E(w)}{\partial w_i} = \mathbf{0} \Longleftrightarrow \sum_{\alpha=1}^{p} \left[t^\alpha - \sum_{j=1}^{L} w_j G_j(\mathbf{x}^\alpha) \right] G_i(\mathbf{x}^\alpha) = \mathbf{0} \tag{1.14}$$

where $G_j(\mathbf{x}^\alpha)$ is the output of hidden unit j, is linear in w_j. This means that we can explicitly solve for the weights of this unit which minimizes the cost (Barton, 1990). This speeds up the overall convergence (deeper layers are still trained with gradient descent) of the network, since the quality of the error signal at the lower layers is high.

[4] Usually it will not make much sense to have linear units in deeper layers (since the weights may as well connect directly to the next layer). However some form of orthogonalization between layers might in some cases be useful.

A different approach utilizes second order information in the search for a minimum. To second order the Taylor expansion of the cost in weight space is:

$$E(\mathbf{w}) = E(\mathbf{w}_0) + (\mathbf{w} - \mathbf{w}_0)^T \nabla E(\mathbf{w}_0) + \frac{1}{2}(\mathbf{w} - \mathbf{w}_0)^T \tilde{\mathbf{H}}(\mathbf{w} - \mathbf{w}_0) \tag{1.15}$$

where ∇E is the gradient and $\tilde{\mathbf{H}}$ is the *Hessian* matrix:

$$\nabla E = \left.\frac{\partial E(\mathbf{w})}{\partial \mathbf{w}}\right|_{\mathbf{w}=\mathbf{w_0}} \qquad \tilde{\mathbf{H}}_{ij} = \left.\frac{\partial^2 E(\mathbf{w})}{\partial w_i \partial w_j}\right|_{\mathbf{w}=\mathbf{w}_0} \tag{1.16}$$

Newton's method for finding a minimum of a quadratic surface in one step is:

$$\Delta \mathbf{w} = -\tilde{\mathbf{H}}^{-1} \nabla E(\mathbf{w}_0) \tag{1.17}$$

Utilization of this method has the additional advantage that it eliminates the problem of setting the learning rate η. Unfortunately the size of the Hessian scales poorly with the number of weights in the network, and inversion of the Hessian becomes compute intensive and possibly ill defined (if the Hessian is singular). A poor man's version of the Newton method, the pseudo Newton method, uses a diagonal approximation for the Hessian. In this case the update rule becomes:

$$\Delta w_{ij} = -\frac{\partial E(w)}{\partial w_{ij}} \Bigg/ \frac{\partial^2 E(w)}{\partial w_{ij}^2} \tag{1.18}$$

As it stands, eq. (1.18) is dangerous, since the denominator can be near zero (which would lead to giant steps) or negative[5] (which would lead to steps in the direction of a *maximum* in cost). In a practical implementation eq. (1.18) should therefore be modified to:

$$\Delta w_{ij} = -\frac{\partial E(w)}{\partial w_{ij}} \Bigg/ \left(\left| \frac{\partial^2 E(w)}{\partial w_{ij}^2} \right| + \epsilon \right) \tag{1.19}$$

where ϵ is a small positive constant. At some computational overhead per step these second order methods make learning converge in much fewer iterations. The pseudo Newton method can be viewed as applying individual learning rates, calculated from the cost curvature, for each direction in weight space.

It is possible to give an intuitive explanation for how the second order procedure speeds up convergence. If there is a large spread in the size of curvature in different directions, the learning rate of the simple rule eq. (1.8) will have to be low in order to insure stability in the directions with high curvature. This has the price of making convergence slow in the shallow directions. The second order scheme takes the individual curvatures into consideration.

[5] Utilization of the positive semi-definite Levenberg Marquardt approximation, eq. (2.2) is an alternative to the presented solution.

1.2.3 The learning Procedure

All the (non-linear) networks described in this report are trained using batch mode training. A combination of the methods above are utilized: direct matrix inversion for the weights in the hidden-to-output layer and a modified version of the pseudo Newton method for the weights in the input-to-hidden layer, as in (Svarer et al., 1993).

The modification consists of a quality control, which insures that each step actually leads to a decrease in cost. When a weight change is calculated, it is checked that the new weights lead to a decrease in cost, and the weight change is iteratively diminished by powers of two until a cost decrease is observed, (Svarer et al., 1993). Normally, the quality control intervenes only during the first few epochs of training, when the weights are more or less random. In this situation the quadratic approximation does not hold very well, and a few small steps are needed in order to reach a more regular patch in weight space.

This learning procedure has turned out to work well for the problems studied here.

1.3 The Sunspot Problem

The reason for investigation of the model construction techniques considered is of course their applicability to problems from the real world. Therefore it is important to test the techniques on data from the real world, rather than to study their performance solely on constructed data. Care should be taken when extrapolating performance on constructed data to that of real data, since in many cases important qualitative differencies between these two types of data exist.

The question: 'What characterizes a real data set?' is not particularly easy to answer in a constructive way, rather a discussion of: 'What's wrong with constructed data?' will be given. A fundamental problem is whether the 'correct' model lies in the space of student models. Many constructed data sets are made by use of a teacher network with the same architecture as the student, so the 'correct' network is obviously within student space. In many real world applications the word 'correct' in this context is probably not meaningful. Proofs that for example large enough networks can approximate any continuous function (Cybenko, 1989) are not very useful, since they don't restrict themselves to networks with a reasonable number of weights. So the question of whether a neural network constitutes a good idea for a certain application remains open.

In the cases where noisy data sets are considered, one needs to formulate a noise model. In the case of constructed data one might have a hard time 'forgetting' the attributes of the noise used to corrupt the data. For regression type problems one often choses a Gaussian error model, for reasons of mathematical convenience, due to the law of large numbers or simply from lack of anything better.

There are also problems in evaluating performance in real applications. In the case of constructed data one can easily calculate the error of the model, since the 'true' model is known. In the case of real data one will have to rely on performance estimates obtained from (possibly very limited sized) test sets. In this case it is possible to have chosen lucky or unlucky test sets, so evaluation is always bothered by uncertainty.

In order to be able to enjoy the full spectrum of frustration of working with real data the sunspot series has been adopted for analysis.

1.3.1 The Sunspot Series

As an example of a real world application the problem of predicting the yearly sunspot average from the average of previous years is considered. This *auto regression* problem is well studied in statistics, and has established itself as a benchmark problem in the neural network community (Weigend et al., 1990; Svarer et al., 1993).

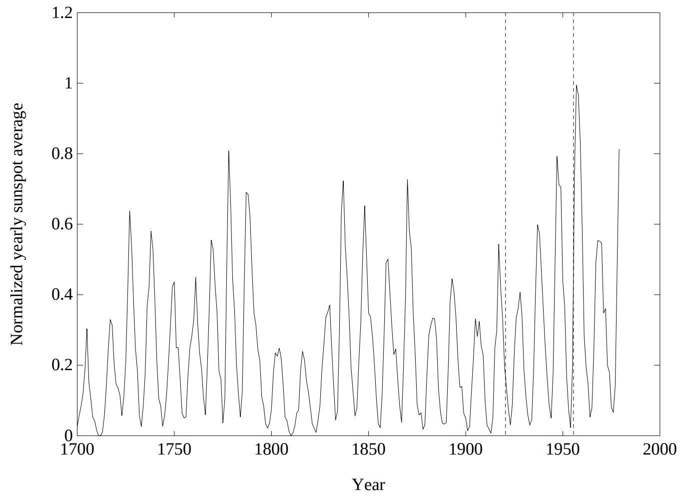

Figure 1.2: The yearly sunspot average from 1700 to 1979. The data is scaled to the 0 to 1 interval. The 'training set' includes data from 1700 to 1920, 'test set 1' ranges from 1921 to 1955 and 'test set 2' from 1956 to 1979.

The sunspot series consist of (normalized) yearly sunspot averages from 1700 to 1979, 280 data points in all. The underlying mechanism for sunspot appearances is not exactly known. A pronounced 11 year period is clearly visible in the data, although with some irregularity (earlier observations suggest that there were virtually no sunspots in the period between 1645 and 1715 (Eddy, 1977)).

The sunspot series is analyzed using a tapped delay line (an auto regressive model), which predicts the sunspot average from the averages of the previous n years:

$$\hat{x}(t) = F(x(t-1), x(t-2), \ldots, x(t-n)) \tag{1.20}$$

Certainly, some of the regularity in the series should be able to be captured by this kind of model. On the other hand the chaotic nature of the series suggests that only a limited accuracy will be obtainable (eg. the depression in the early 1800 is probably hard to predict). In adopting (1.20) we are assuming that the series is stationary, which is clearly not the case (there were virtually no spots between 1645 and 1715), but some of the variance can probably be predicted anyway. So we are clearly working in the regime which does not involve 'correct' models.

Traditionally the sunspot series has been divided into 3 subsets: the training set from 1700 to 1920 and two test sets from 1921 to 1955 and from 1956 to 1979 respectively. Commonly delay lines with 12 taps have been used. In this case the training set comprises 209 examples.

The performance of a trained model can be monitored by the error made on the two test sets. The *average relative variance* performance measure of (Weigend et al., 1990) is used:

$$arv(\mathrm{set}) = \frac{1}{\sigma^2_{\mathrm{set}}} \frac{1}{p_{\mathrm{set}}} \sum_{\alpha \in \mathrm{set}} (x^\alpha - \hat{x}^\alpha)^2 \tag{1.21}$$

where $\sigma^2_{\mathrm{total}}$ is the variance on the whole series[6], 'set' denotes a data set (training or testing) and p_{set} is the number of examples in each set. For the sunspot data $\sigma^2 = 0.0411$. Using the *arv* performance measure removes the dependency on the dynamic range of the data, and normalizes so that a model which simply uses the mean of the set as an estimator achieves $arv = 1.0$.

As already noted, care should be taken when considering the performance on the test sets as a measure of performance. The test sets are quite small, comprising 35 and 24 data points respectively. The statistical significance of a fluctuation in test error may well be questionable.

The sunspot problem will be used extensively in the following chapters to test the performance of the developed techniques. Both linear and nonlinear approaches to the problem will be tried. However, it should be emphasized that the tests using this limited sample problem will not 'prove' the validity of the techniques, rather they will 'hint' to whether a more extensive study might be worth while.

[6]The original article (Weigend et al., 1990) is somewhat ambiguous about whether a single or three separate variance normalization constants should be used.

Chapter 2

Generalization Theory

There are two themes in this section. Firstly, the problems of overfitting are demonstrated, and the utilization of validation sets and 'Leave One Out' cross validation are discussed. Secondly, two strategies for reducing model capacity are given: pruning and weight decay.

The character of the chapter is introductory rather than constructive. The problems with the validation techniques are described and the need for control parameters is emphasized. The control parameters are: stop criteria for pruning and weight decay parameters. The problems are dealt with in a constructive manner in the following chapters.

2.1 Overlearning

One of the most important features of NN's is their ability to generalize. Once the networks have been trained on the training set, we wish their performance on examples outside the training set to be good. In order to expect the network to perform well on the test examples, we imply that the network should not just have learned the individual examples of the training set, but rather it should have learned the *rule* underlying the data.

In the case of noisy data sets, it is however difficult to decide what constitutes the rule, and what should be regarded merely as noise. Lets look at a concrete example. A network with 12 inputs, 8 hidden hyperbolic tangent units and a single linear output (as in eq. (1.5)) is trained on the sunspot data.

In fig. 2.1 a typical learning curve for this problem is shown. The training error is a decreasing function (as indeed expected), but the two test set errors start increasing after some time. This behaviour is termed *overlearning*, and it occurs because the network is too large (contains too many weights)[1] for the problem. In this case the network comprises 113 weights (104 input-to-hidden weights, 8 hidden-to-output weights and 9 biases) and the training set consists of 209 examples, so there is less than two examples per weight (which is evidently too few). The network has learned some of the noise contained in the training set. Since this noise in the training set is uncorrelated with the noise in the test

[1] A similar scenario occurs when fitting polynomials to noisy data: high order polynomials fluctuate wildly to fit the noise.

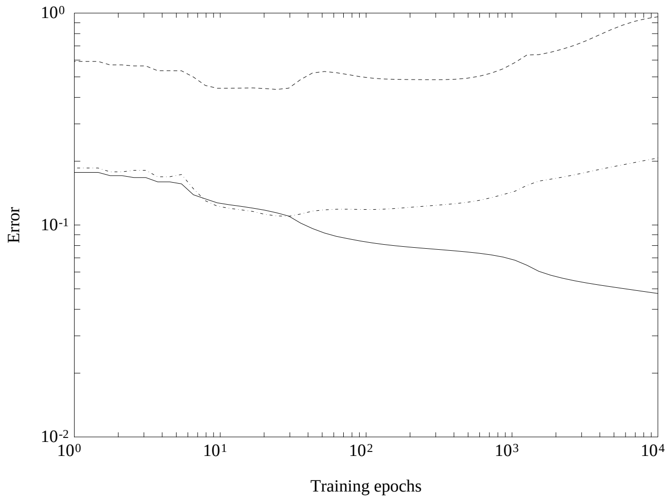

Figure 2.1: Learning curve for a network with 12 taps and 8 hidden units on the sunspot problem. The *arv* error of the training set (solid), test set 1 (dashdot) and test set 2 (dashed) are shown as a function of the number of (batch mode) training epochs. It is seen that a serious problem of overfitting occurs: at around 250 epochs both test set errors start increasing. Note the logarithmic scales.

sets, the performance on the test sets deteriorates.

The above example shows that when training NN's measures are needed to insure that performance on test examples is good. The first step in this direction is the need to obtain some knowledge about the network's performance on examples outside the training set. As we shall see, this information can be retrieved in several different manners. The second step then consist in using this information to optimize the performance on unknown examples.

2.1.1 Validation and Test Sets

Probably the most straight forward way to get information about the performance on examples outside the training set, is to hold back some of the examples when training and utilize these for monitoring performance. This splitting of the data can occur on several levels. It is customary to use three sets: training set, validation set and test set. The training set is used to train the network via the back propagation rule. The validation set can be used to optimize a number of measures associated with the training. The test set is used to obtain knowledge of how well the network can be expected to perform on novel examples.

2.1.2 Early Stopping

A straight forward way to deal with the problem of overlearning suggests itself in fig. 2.1. By holding some of the data in a special *validation* set, we can plot the test error during training and stop training as soon as the test error starts increasing. This technique is called early stopping.

When using early stopping, one gets a network with weights which are a product of a transient phenomenon during the early stages of training. The dynamics of the learning rule and the choice of initial (random) weights, lead to a situation where apparently most of the rule of the data is learned before all of the the noise in the examples. However we do not have very good control over the circumstances which lead to the final model, and in particular, we do not know how far we are from an 'optimal' solution. A better approach would be to deal with the root of the problem: finding the proper complexity for the network.

2.1.3 Leave One Out Cross Validation

It should be noted that there is a fundamental problem associated with division of the training data into several sets. The problem is that the accuracy of both the training of the network and the validation of performance are critically dependent on the size of the data sets. Since the size of the network which can reliably be trained is closely tied to the number of training examples, sacrificing of data for validation sets will lead to deterioration of the models.

It is also a problem to determine how large the validation and test sets should be. If they are very large the problem above will be exaggerated. If on the other hand they are too small, they will give poor estimates of the actual performance, and optimization of a criterion based on a noisy estimator will lead to increased generalization error.

There is a cure for all these problems, called 'leave one out cross validation' (LOO) (Toussaint, 1974). Using this technique, only a single example is used for validation, but the procedure is repeated for all training examples. The final validation error is the average error obtained for all the validation examples. Unfortunately, there is no medication without side effects. The price to be paid for using the LOO technique, is that a large number of models must be trained.

In the case of nonlinear NN's, even training of a single model can be computationally costly, so a full LOO scheme might well be prohibitively costly. A compromise can be obtained by using a modified 'Leave n Out' validation procedure. For linear models there is a short cut, by which the LOO estimate can be computed cheaply (covered in section 2.3.1).

Using validation techniques, it is (in principle) possible to search for networks with architectures that are well suited for the learning problem in question. However, due to computational limits only a rough idea of the optimal architecture can be achieved in this way. In the following two sections adaptive methods are considered, which will eliminate the need to train many networks with different architectures.

2.2 Reducing Capacity by Pruning

Learning schemes which adapt the network architecture during learning are attractive, since they can reduce the number of network trainings needed for a certain task. Two forms of these schemes exist: pruning algorithms and growth algorithms.

Growth algorithms start off with a very simplistic network (perhaps a single neuron) and build units (or weights) into the network as training proceeds. There are several examples of growth algorithms, for example the 'Cascade Correlation' algorithm of (Fahlman and Lebiere, 1990), for an overview see (Hertz et al., 1991). From a computational point of view growth algorithms are favorable, since one does not waste computation on training the (too) large initial networks. Despite their merits growth algorithms will not be considered further here[2].

A pruning algorithm is invoked to reduce the number of units and/or weights in a network, which is already trained for a task. Essentially pruning algorithms rely on a ranking of the weights in order of importance, and a subsequent deletion of the least important weights. Pruning schemes are typically applied through the following steps:

1. Train a 'sufficiently large' fully connected network.
2. Rank the weights, and remove the least important one(s).
3. Retrain the network, and return to 2) if further pruning is desired.

An obvious problem with all pruning schemes is the need for a criterion for stopping the pruning (Svarer et al., 1993). Here again we must turn to the use of validation sets or methods relying on statistical estimates, developed in chapter 3. Several levels of refinement have been put into algorithms for ranking weights. The crudest method is simply assuming that the smallest weights are probably the least important. Now two more enlightened algorithms for ranking of weights will briefly be described.

2.2.1 Optimal Brain Damage

Optimal Brain Damage (OBD) (Le Cun et al., 1990) is a technique for pruning networks. Using this technique one estimates the *saliency* of the weights, whereby the weights can be ranked. The saliency of a weight is the defined as the increase in the network training error introduced by removing a weight.

The estimate of the saliency is calculated from a second order Taylor expansion of the training error in weight space:

$$E(\mathbf{w}) = E(\mathbf{w}_0) + (\mathbf{w} - \mathbf{w}_0)^T \nabla E(\mathbf{w}_0) + \frac{1}{2}(\mathbf{w} - \mathbf{w}_0)^T \tilde{\mathbf{H}}(\mathbf{w} - \mathbf{w}_0) \tag{2.1}$$

[2] A growth based scheme has been tried, but problems arose since the procedure repeatedly found exactly the same nets (architectures and weights). The scheme was abandoned since it didn't seem to be able to search the space of networks very well.

The estimation of the saliency in OBD relies on three approximations. Firstly, a diagonal approximation of the Hessian is assumed. This implies that the deletion of one parameter does not influence the saliency of other parameters. Secondly, it is assumed that the network is well trained before OBD is invoked, so that the first order derivatives are zero ($\nabla E(\mathbf{w}_0) = 0$). Thirdly, the Levenberg Marquardt approximation for the second order derivatives of E is adopted:

$$\frac{\partial^2 E}{\partial w^2} = \sum_{\alpha=1}^{p} \left(\frac{\partial F(\mathbf{x}^\alpha)}{\partial w} \right)^2 + \sum_{\alpha=1}^{p} (F(\mathbf{x}^\alpha) - t^\alpha) \frac{\partial^2 F(\mathbf{x}^\alpha)}{\partial w^2} \simeq \sum_{\alpha=1}^{p} \left(\frac{\partial F(\mathbf{x}^\alpha)}{\partial w} \right)^2 \tag{2.2}$$

This approximation is well justified if the differences between the outputs and the targets are small, and further the neglected sum is a product of uncorrelated factors (the derivative and the output error) which should be very small if the network output is unbiased. The saliency of weight w_{ij} is thus calculated as:

$$\zeta_{ij} = \frac{1}{2} \frac{\partial^2 E}{\partial w_{ij}^2} w_{ij}^2 \tag{2.3}$$

In later sections the OBD algorithm, together with a stop criterion, will be utilized for network pruning. The method will be implemented as stated above, except that the Levenberg Marquardt approximation will not be used. Rather the full second derivative of E will be calculated.

The assumption that the saliencies are uncoupled is probably not always good. If two parameters are closely coupled, the effects of pruning of one of them may to a large degree be absorbed by the other weight during retraining. So the saliency computed by OBD is actually an upper limit to the effect of pruning and retraining (Gorodkin et al., 1993).

2.2.2 Optimal Brain Surgeon

The Optimal Brain Surgeon (OBS) is a technique for pruning, which is very similar to OBD, but slightly more elaborate. Basically the refinement of the algorithm consists in using the whole inverse Hessian, rather than a diagonal approximation. The OBS paper (Hassibi et al., 1992) contains a recipe for computing the inverse Hessian. In this case the saliency of weight i is given by:

$$\zeta_i = \frac{1}{2} \frac{w_i^2}{(\tilde{\mathbf{H}}^{-1})_{ii}} \tag{2.4}$$

Using OBS the estimated saliency of weights are calculated, *including* the effects of retraining the remaining weights (in the quadratic approximation). So the computational overhead associated with computation of the inverse Hessian is regained by the lack of need for retraining between the weight deletions, as is the case with OBD.

At present it is unclear which of the methods OBD and OBS is the most effective. Certainly OBD is extremely simple to implement, and there might be problems with the OBS in the case of small data sets, since many of the terms in the Hessian may be ill defined, the basic problem being that the Hessian comprises N^2 terms, which are to be estimated from p examples. In general we wish to train networks in the limit where $N < p$ is adequately honoured, but even in this limit we might be on shaking ground when trying to estimate the (inverse) Hessian[3].

2.3 Reducing capacity by Weight Decay

A different way to reduce model capacity is by constraining the values of the weights by regularization. This paradigm is well known when solving underdetermined equations. In its simplest form it is implemented by addition of a weight decay term in the training cost function (Hertz et al., 1991). The most popular choice for this term is a weight decay constant multiplied by the sum of squared weights:

$$E = \frac{1}{2}\sum_{\alpha=1}^{p}(t^{\alpha} - F(\mathbf{x}^{\alpha}))^2 + \frac{\xi}{2}\sum_i w_i^2 \tag{2.5}$$

where ξ is the weight decay parameter, controlling the strength of weight decay. Addition of this weight decay term has the effect of dragging the weights towards zero. Weights which are poorly determined by the data will be pulled towards zero, whereas weights that are well determined by the training data will (to a large degree) be able to resist the drag of weight decay, since the curvature of the data cost is large. A graphical representation of this effect is given in fig. 5.1. A different form for the weight decay term was suggested in (Rumelhart, 1988):

$$E = \frac{1}{2}\sum_{\alpha=1}^{p}(t^{\alpha} - F(\mathbf{x}^{\alpha}))^2 + \frac{\xi}{2}\sum_i \frac{w_i^2}{1 + w_i^2} \tag{2.6}$$

which works in the same way as ordinary weight decay, with the exception that the weight decay term becomes constant for very large weights, rather than growing. It should be noted that, in fact, this form of weight decay has two (potentially) adaptive constants, since the 1 in the denominator of eq. (2.6) could also be adapted.

By introduction of weight decay it is hoped that the influence of parameters whose values are largely determined by noise will be reduced. On the other hand, the weight decay should be gentle enough, to ensure that the weights which are really important only deviate marginally from their former value. Obviously, this creates an optimization problem for ξ.

In figure 2.2 the 'leave one out' procedure was used to optimize the regularization parameter for a linear model on the sun spot series. A minimum is clearly seen in the error on both test sets. Note however, that the improvement is very small, since the model contains only 13 parameters (12 weights and one bias), and there were 209 training examples. The parameters of this model are therefore relatively well determined by the data alone, and only a marginal improvement can be gained by the use of weight decay.

[3] There are reasons to believe that the diagonal elements of the Hessian are relatively more well defined than the off diagonal elements, since they consist of a sum of p positive contributions.

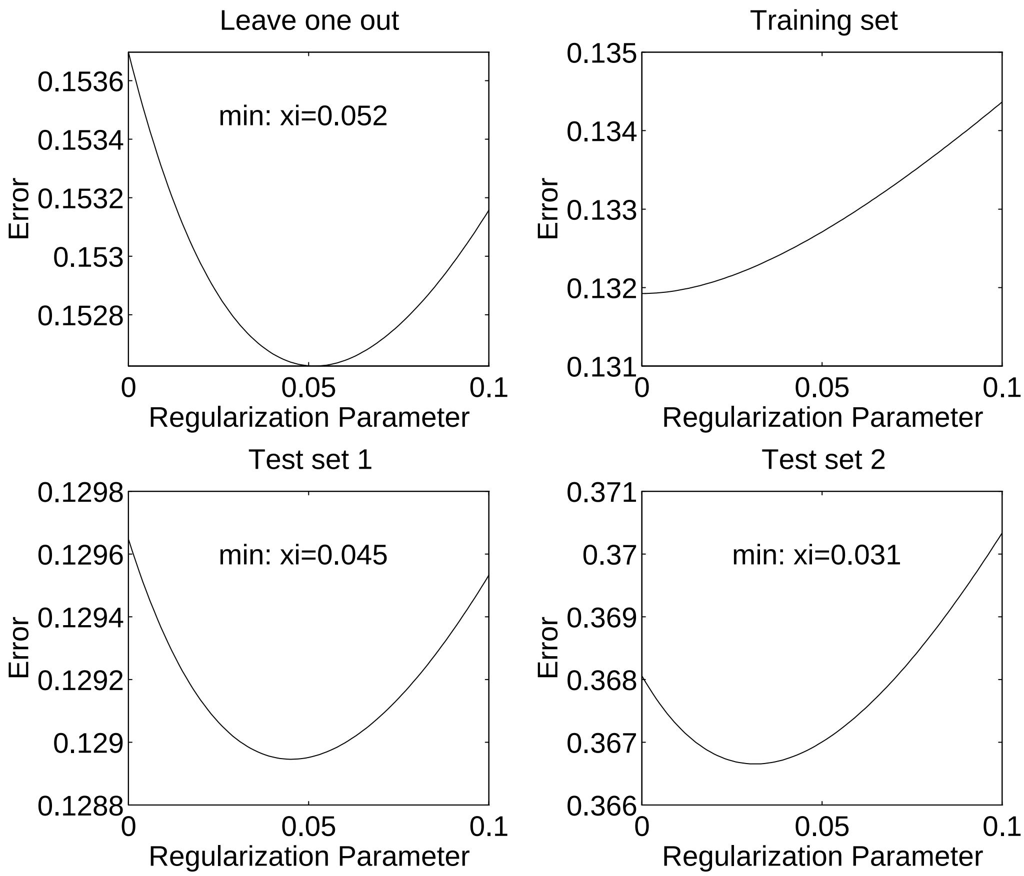

Figure 2.2: The leave one out cross validation technique used on a linear model for the sunspot prediction task. The first plot shows the 'leave one out' validation arv error on the training set; the regularization parameter is seen to have an optimal value. To the right the raw training error is seen to be an increasing function of ξ. Below both test sets are seen to have a minimum in error at values of ξ, which are not far from the optimum obtained from the first curve. Note the different scales.

It should be noted that quite a large number of computations are needed to obtain a plot as figure 2.2. The model was trained for 100 different values of the weight decay parameter, each training consisting of 209 linear models (corresponding to leaving out each of the 209 examples in turn). Naturally, if one is only concerned about finding the optimal value for lambda, it would be more profitable to use some kind of line search algorithm (ie. bisection).

2.3.1 Leave One Out Cross Validation for Linear Models

In the case of linear models one can utilize the fact that the models which are trained are nearly the same. Training a linear model (with weight decay) amounts to solving the following equation:

$$\mathbf{w} = \mathbf{Q}^{-1}\mathbf{q} \qquad \mathbf{Q}_{ii'} = \sum_{\alpha=1}^{p} \mathbf{x}_i^\alpha \mathbf{x}_{i'}^\alpha + \xi\delta_{ii'} \qquad \mathbf{q}_i = \sum_{\alpha=1}^{p} \mathbf{x}_i^\alpha t^\alpha \tag{2.7}$$

So it appears that the leave one out procedure calls for p matrix inversions. However the by use of a standard inversion rule, individual examples can be extracted from the matrix inversion, whereby the work can be reduced to a single matrix inversion (and some multiplications):

$$\mathbf{Q}_\alpha = \mathbf{Q} - \mathbf{x}^\alpha(\mathbf{x}^\alpha)^T \qquad \mathbf{Q}_\alpha^{-1} = \mathbf{Q}^{-1} + \frac{1}{1-(\mathbf{x}^\alpha)^T\mathbf{Q}^{-1}\mathbf{x}^\alpha}\mathbf{Q}^{-1}\mathbf{x}^\alpha(\mathbf{x}^\alpha)^T\mathbf{Q}^{-1} \tag{2.8}$$

where the corresponding p weight vectors are given by:

$$\mathbf{w}^\alpha = \mathbf{Q}_\alpha^{-1}\mathbf{q}^\alpha \qquad \mathbf{q}^\alpha = \mathbf{q} - \mathbf{x}^\alpha t^\alpha \tag{2.9}$$

This reduction will lead to a marked speedup if the matrix dimension is high (since inversion takes of the order N^3). However, no such tricks exist for the general case of a nonlinear network, so an entire retraining of the network is needed, each with a different sample left out. Also one may consider networks with several weight decay parameters, in which case the search becomes increasingly computationally costly. In conclusion, it is attractive and feasible to use leave one out cross validation for linear models; for nonlinear models the computational cost of this approach will often be prohibitive.

In the chapters to come, methods which do not utilize validation sets will be investigated. The primary advantage of these methods should be that the computationally costly LOO cross validation can be avoided.

Chapter 3

Akaike's FPE Estimate

A common feature of the techniques considered under in this chapter (and the following) is that they utilize statistical estimates of the generalization error, rather than a separate validation set. The statistical estimates rely on the training error and knowledge about the network architecture (the number of weights). The primary advantage of these methods is their ability to utilize the entire data set in the training phase, so the methods should be especially well suited for cases where there is a shortage of training data.

3.1 Derivation of Akaike's FPE

One of the fundamental relationships between the training and generalization error is Akaike's *Final Prediction Error* (FPE) estimate (Akaike, 1969). Under a certain set of conditions Akaike's FPE gives the following simple relationship between (average) generalization and training errors:

$$\langle \epsilon_G \rangle_p = \frac{p+N}{p-N} \langle \epsilon_T \rangle_p \tag{3.1}$$

where $\langle \ldots \rangle_p$ denotes the average over all training sets of size p. We are not so much interested in the errors for a particular training set, rather we want to compute the expected error for a typical case. So we want to average over the both the noise distribution and the input distribution.

3.1.1 Underlying Assumptions

The derivation of Akaike's FPE relies on a set of assumptions. It is not entirely clear to what extent these assumptions are fulfilled in a particular application. The problems regarding the assumptions will be discussed later, in the light of the performance of the FPE estimate on the sunspot modeling problem. As for now the assumptions will be listed, together with a brief discussion; in no particular order:

- The teacher must lie in student space, ie. the network must be able to implement the target function. Since neural networks (with at least one hidden layer) have been shown to be universal (Cybenko, 1989), this might be thought of as not being a problem. But when small networks are trained with a limited amount of training data, problems might arise.
- It is assumed that the network implements a function which is close to the target function, so that an expansion of the network function in the weights is reasonably accurate. We hope that the only function(s) which can account for all the points in the training set must be of the right shape.
- It is assumed that the distribution of inputs is representative of the inputs which will occur in the samples which will be used for prediction. It is a natural requirement that the training data set should cover the input space from which test examples are drawn reasonably well.
- The noise on the training examples should be independent, additive, zero mean noise. The independence of the noise contributions is essential and may not always be fulfilled in cases such as time series.
- It is assumed that there are many examples. Terms that are of order $1/p^2$ or higher will be neglected.

In a general case these assumptions will only be partially honoured, so testing of the model will be required.

3.1.2 The Nitty Gritty

The following derivation will apply to unbiased estimation, ie. a network trained without weight decay. The derivations which take the effect of weight decay into account are covered in chapter 4. The derivation is adapted from (Hansen, 1993).

Since the teacher lies in student space, there must exist a set of weights which will reproduce the teacher function. These weights are termed the *teacher weights*, $\mathbf{w}^*$. The network function is called F_w, and the teacher function F_{w*}. The training data base contains p examples from the teacher F_{w*} contaminated by additive, independent, zero mean noise:

$$D = \{\mathbf{x}^\alpha, t^\alpha | \alpha = 1, \ldots, p\} \qquad t^\alpha = F_{w*}(\mathbf{x}^\alpha) + \nu^\alpha \qquad \nu^\alpha \in \mathcal{N}(0, \sigma^2) \tag{3.2}$$

The training cost function is the usual quadratic[1]:

$$E = \frac{1}{2}\sum_{\alpha=1}^{p}(t^\alpha - F_w^\alpha)^2 = \frac{1}{2}\sum_{\alpha=1}^{p}(F_{w*}^\alpha + \nu_\alpha - F_w^\alpha)^2 \tag{3.3}$$

We are interested in deriving expressions for the average training and the average generalization errors, defined by:

[1]In the following F_w^α will be used as short hand for $F_w(\mathbf{x}^\alpha)$.

$$\langle \epsilon_T \rangle_p = \left\langle \frac{1}{p} \sum_{\alpha=1}^{p} (F_{w*}^\alpha - F_w^\alpha + \nu^\alpha)^2 \right\rangle_p \tag{3.4}$$

$$\langle \epsilon_G \rangle_p = \int \left\langle (F_{w*} - F_w + \nu)^2 \right\rangle_p P(x) P(\nu) d\mathbf{x} d\nu \tag{3.5}$$

where $\langle \ldots \rangle_p$ denotes the average over all training sets of size p. These definitions are quite straight forward simply the expected squared error for the training set and for a test example. Since the network function is assumed to be close to the teacher function, we can expand the student to first order around the true teacher, in weight fluctuations δw_j:

$$F_w^\alpha \simeq F_{w*}^\alpha + \sum_j \frac{\partial F_{w*}^\alpha}{\partial w_j} \delta w_j \tag{3.6}$$

Introducing this in eq. (3.3) yields:

$$E \simeq \frac{1}{2} \sum_{\alpha=1}^{p} \left(\sum_j \frac{\partial F_{w*}^\alpha}{\partial w_j} \delta w_j - \nu^\alpha \right)^2 \tag{3.7}$$

The trained network is at a (local or global) minimum in cost, so the weight fluctuations obey:

$$\frac{\partial E}{\partial \delta w_j} = 0 \Longrightarrow \sum_{\alpha=1}^{p} \left(\sum_{j'} \frac{\partial F_{w*}^\alpha}{\partial w_{j'}} \delta w_{j'} - \nu^\alpha \right) \frac{\partial F_{w*}^\alpha}{\partial w_j} = 0 \Longrightarrow \delta \mathbf{w} = \mathbf{H}^{-1} \mathbf{a} \tag{3.8}$$

where we have defined:

$$\mathbf{H}_{jj'} = \frac{1}{p} \sum_{\alpha=1}^{p} \frac{\partial F_{w*}^\alpha}{\partial w_j} \frac{\partial F_{w*}^\alpha}{\partial w_{j'}} \qquad \mathbf{a}_j = \frac{1}{p} \sum_{\alpha=1}^{p} \nu^\alpha \frac{\partial F_{w*}^\alpha}{\partial w_j} \tag{3.9}$$

In the following a mixture of matrix- and tensor-notation will be used in order to achieve maximum clarity. As a preliminary result we need to calculate:

$$\begin{aligned}
\langle \delta w_j \delta w_{j'} \rangle_p &= \langle \mathbf{H}^{-1} \mathbf{a} (\mathbf{H}^{-1} \mathbf{a})^T \rangle_p = \langle \mathbf{H}^{-1} \mathbf{a} \mathbf{a}^T \mathbf{H}^{-1} \rangle_p && (3.10) \\
&= \left\langle \left\langle \mathbf{H}^{-1} \left(\frac{1}{p^2} \sum_{\alpha\alpha'} \nu^\alpha \nu^{\alpha'} \frac{\partial F_{w*}^\alpha}{\partial w_j} \frac{\partial F_{w*}^{\alpha'}}{\partial w_{j'}} \right) \mathbf{H}^{-1} \right\rangle_\nu \right\rangle_x && (3.11) \\
&= \left\langle \mathbf{H}^{-1} \left(\frac{\sigma^2}{p^2} \sum_{\alpha\alpha'} \delta_{\alpha\alpha'} \frac{\partial F_{w*}^\alpha}{\partial w_j} \frac{\partial F_{w*}^{\alpha'}}{\partial w_{j'}} \right) \mathbf{H}^{-1} \right\rangle_x && (3.12) \\
&= \left\langle \mathbf{H}^{-1} \frac{\sigma^2}{p} \mathbf{H} \mathbf{H}^{-1} \right\rangle_x = \frac{\sigma^2}{p} \langle (\mathbf{H}^{-1})_{jj'} \rangle_x && (3.13)
\end{aligned}$$

where $\langle \ldots \rangle_x$ denotes the average over the input distribution and $\langle \ldots \rangle_\nu$ denotes the average over noise contributions. In eq. (3.10) the symmetry of $\mathbf{H}$ was used, in eq. (3.11) $\mathbf{a}$ from eq. (3.9) was introduced and in eq. (3.12) the independence of the noise contributions gave raise to a 'Kronechers delta' $\delta_{\alpha\alpha'}$. The average training error from eq. (3.4) becomes:

$$\langle \epsilon_T \rangle_p \tag{3.14}$$

$$= \left\langle \frac{1}{p}\sum_{\alpha=1}^{p} (F_{w*}^{\alpha} - F_{w}^{\alpha} + \nu^{\alpha})^2 \right\rangle_p = \left\langle \frac{1}{p}\sum_{\alpha=1}^{p} \left(-\sum_j \frac{\partial F_{w*}^{\alpha}}{\partial w_j}\delta w_j + \nu^{\alpha} \right)^2 \right\rangle_p \tag{3.15}$$

$$= \left\langle \frac{1}{p}\sum_{\alpha=1}^{p} \left(\sum_{jj'} \frac{\partial F_{w*}^{\alpha}}{\partial w_j}\frac{\partial F_{w*}^{\alpha}}{\partial w_{j'}}\delta w_j \delta w_{j'} + (\nu^{\alpha})^2 - 2\sum_j \nu^{\alpha}\frac{\partial F_{w*}^{\alpha}}{\partial w_j}\delta w_j \right) \right\rangle_p \tag{3.16}$$

$$= \left\langle \sum_{jj'} \left(\frac{1}{p}\sum_{\alpha=1}^{p} \frac{\partial F_{w*}^{\alpha}}{\partial w_j}\frac{\partial F_{w*}^{\alpha}}{\partial w_{j'}} \right) \delta w_j \delta w_{j'} \right\rangle_p + \sigma^2 - \left\langle \sum_j \left(\frac{2}{p}\sum_{\alpha=1}^{p} \nu^{\alpha}\frac{\partial F_{w*}^{\alpha}}{\partial w_j} \right) \delta w_j \right\rangle_p \tag{3.17}$$

$$= \sum_{jj'} \langle\langle \mathbf{H}_{jj'}\delta w_j \delta w_{j'}\rangle_\nu\rangle_x + \sigma^2 - \left\langle \left\langle \sum_j \frac{2}{p}\sum_{\alpha=1}^{p} \nu^{\alpha}\frac{\partial F_{w*}^{\alpha}}{\partial w_j} \sum_{j'} (\mathbf{H}^{-1})_{jj'} \frac{1}{p}\sum_{\alpha'} \nu^{\alpha'}\frac{\partial F_{w*}^{\alpha'}}{\partial w_{j'}} \right\rangle_\nu \right\rangle_x$$

$$= \frac{\sigma^2}{p}\sum_{jj'} \langle \mathbf{H}_{jj'}(\mathbf{H}^{-1})_{jj'}\rangle_x + \sigma^2 - \left\langle \sum_{jj'} \frac{2\sigma^2}{p^2}\sum_{\alpha\alpha'} \delta_{\alpha\alpha'}(\mathbf{H}^{-1})_{jj'}\frac{\partial F_{w*}^{\alpha}}{\partial w_j}\frac{\partial F_{w*}^{\alpha'}}{\partial w_{j'}} \right\rangle_x \tag{3.18}$$

$$= N\frac{\sigma^2}{p} + \sigma^2 - \frac{2\sigma^2}{p}\sum_{jj'} \langle \mathbf{H}_{jj'}(\mathbf{H}^{-1})_{jj'}\rangle_x = \sigma^2(1 - \frac{N}{p}) \tag{3.19}$$

The average generalization error from eq. (3.5) becomes:

$$\begin{aligned}
\langle \epsilon_g \rangle_p &= \int \left\langle (F_{w*} - F_w + \nu)^2 \right\rangle_p P(\mathbf{x})P(\nu)d\mathbf{x}d\nu && (3.20)\\
&= \int \left\langle (F_{w*} - F_w)^2 \right\rangle_p P(\mathbf{x})d\mathbf{x} + \sigma^2 && (3.21)\\
&= \int \left\langle \sum_{jj'} \frac{\partial F_{w*}}{\partial w_j}\frac{\partial F_{w*}}{\partial w_{j'}}\delta w_j \delta w_{j'} \right\rangle_p P(\mathbf{x})d\mathbf{x} + \sigma^2 && (3.22)\\
&= \int \sum_{jj'} \frac{\partial F_{w*}}{\partial w_j}\frac{\partial F_{w*}}{\partial w_{j'}} \langle \delta w_j \delta w_{j'}\rangle_p P(\mathbf{x})d\mathbf{x} + \sigma^2 && (3.23)\\
&= \frac{\sigma^2}{p}\sum_{jj'} (\mathbf{H}_\infty)_{jj'}\langle(\mathbf{H}^{-1})_{jj'}\rangle_x + \sigma^2 && (3.24)\\
&\simeq N\frac{\sigma^2}{p} + \sigma^2 = \sigma^2(1 + \frac{N}{p}) && (3.25)
\end{aligned}$$

where $\mathbf{H}_\infty$ denotes the $\mathbf{H}$ matrix computed from the entire input distribution (ie. infinitely many examples). In eq. (3.22) the independence of the noise and the input was used and in eq. (3.25) it was used that $\langle(\mathbf{H}^{-1})\rangle_x = \mathbf{H}_\infty^{-1} + corr.$, where the correction is of order $1/p$ or higher[2], and can consequently be neglected. Combining the equations for the average training and test error (3.19) and (3.25), we obtain Akaike's formula:

[2]This requires that $\mathbf{H}$ is not singular and in fact there are a number of additional requirements. The formal proof is beyond the scope of the present study and will be neglected.

$$\langle \epsilon_g \rangle_p = \frac{p+N}{p-N} \langle \epsilon_t \rangle_p \tag{3.26}$$

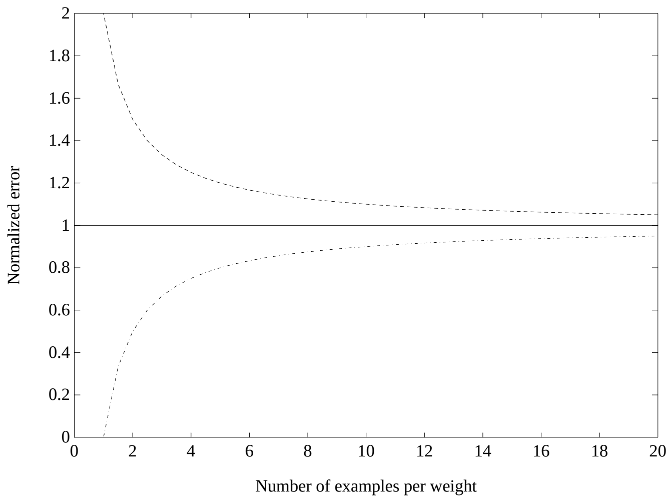

Figure 3.1: Average training (dashdot) and test (dashed) error as a function of the number of training examples per weight: p/N. The plots are obtained from Akaike's FPE estimator. The errors are normalized with the true noise (solid). Note that the system 'blows up' when N approaches p.

3.2 Using Akaike's FPE as a Stop Criterion for Pruning

Akaike's FPE estimate can be used as a stop criterion for pruning. When a weight is pruned (set to zero) we expect the training error to increase by a slight amount, while Akaike's correction factor $(p+N)/(p-N)$ will cause the estimated test error to decrease. The network will reach an optimal size, where the increase in error obtained from deletion of a single weight is only just compensated by the decrease in $(p+N)/(p-N)$.

At first sight this procedure may seem to cure many of our troubles: we have a stop criterion which indicates when the network has the optimal size, so we can forget about weight decay parameters and the tedious work of finding their optimal value! As will be shown in the following this is not so. In fact it will be shown that it is unwise to train networks without weight decay, even with a stop criterion such as Akaike's.

The procedure for applying Akaike's criterion is as follows:

1. Train a sufficiently large fully connected network.
2. Calculate Akaike's FPE estimator.
3. Rank the weight by saliency and eliminate the least significant weight(s).
4. Retrain the pruned network.

5. Go back to 2) unless the network has become hopelessly small.

6. Select the network with the lowest FPE estimate.

3.2.1 Application to the Sunspot Problem

The pruning procedure was applied to the sunspot problem. To get a reasonable estimate of the performance of the pruning technique, 49 pruning sessions were completed, with 7 different values of weight decay.

The pruning sessions followed the following recipe: Initially a fully connected network with 12 inputs, 4 hidden units and a single output (57 weights in all) was trained for 1000 (batch mode) epochs[3]. The initial random weights were uniformly distributed in the ± 1 interval. Akaike's FPE estimate was evaluated. The weights in the input to hidden layer were ranked with the OBD algorithm[4]. The weight with the smallest saliency[5] was clamped to zero. The network was retrained for 1000 epochs and the FPE estimate was evaluated. If the new FPE estimate was lower than the lowest previously seen, the network weights were stored. The pruning procedure was repeated until only 6 weights remained (at this point performance had deteriorated substantially).

It is noted that the hidden to output weights do not participate in the pruning scheme. However, in many cases it was found that some (typically one) of the output weights in the final networks were tuned to zero. This effect was caused by the hidden unit loosing all its inputs (by pruning), and its output weight subsequently being driven to zero by the weight decay.

In fig. 3.2 the training and test errors for the 49 networks selected by the FPE stop criterion are plotted as a function of the weight decay. The raw training error is seen to be an increasing function of ξ. The test error is seen to fluctuate wildly for small weight decay constants. For moderate values of ξ the test error has a flat minimum and for large values of ξ the test error increases slightly. The spread of test errors is seen to narrow down as weight decay is increased, all nets being identical for the highest value of ξ. This is probably due to the linearizing effect of weight decay; networks with small weights are more linear, so there are fewer local minima to distract the search for optimal weights.

The networks which have very little weight decay are seen to have a poor performance on the test examples. These networks have large weights (fig. 3.3) and the networks implement wildly fluctuating functions which fit the data noise to a very high degree. To see why the size of the weights are important when considering how nonlinear the network function is, we compute the magnitude of the curvature of the network function:

[3] Training for 1000 epochs is *a lot*. It insures that we are not doing 'early stopping'. In a practical application fewer epochs will probably suffice.

[4] The saliencies were computed using the entire cost function (including the weight decay terms). It could be argued that it would be more correct to exclude the weight decay terms from the saliency estimation (since it is the *training* error which appears in the FPE estimate), but since the level of weight decay in the present simulations was small, the difference will be negligible.

[5] In (Svarer et al., 1993) several weights could be pruned between retrainings. The current approach is thought to be more in the line of OBD, but also slower.

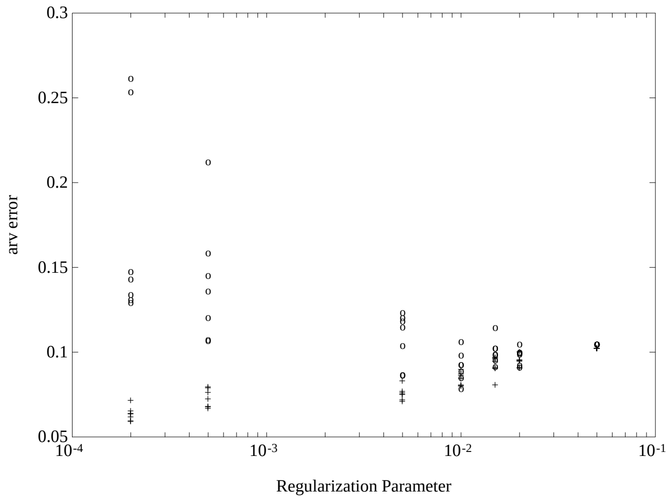

Figure 3.2: The training and test errors for the sunspot problem, as a function of the weight decay parameter ξ. The training errors are indicated by '+' and the test errors for test set 1 are indicated by 'o'. The networks were selected from a pruning session using the FPE stop criterion. Seven nets were trained for each value of ξ. Note that the seven nets with the highest weight decay have identical performances.

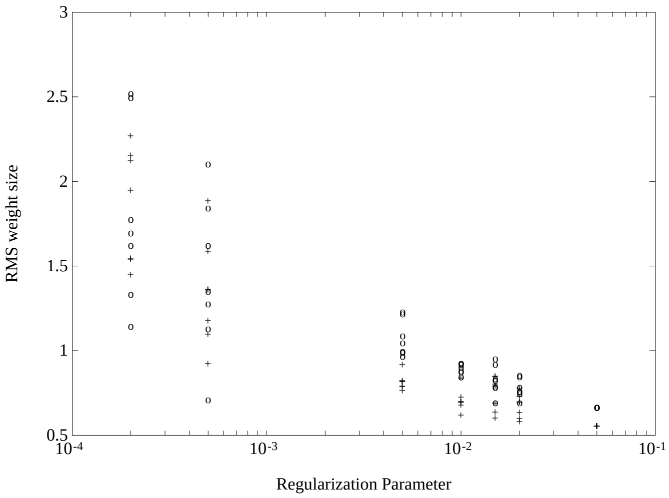

Figure 3.3: The Root Mean Square size of the weights in the pruned networks selected by the FPE stop criterion. The poorly regularized networks have large weights. The output weights (o) are seen to be larger than the input weights (+), at least for the well regularized networks.

$$\left|\frac{\partial^2 F(x)}{\partial x_i^2}\right| = 2\sum_j \left|w_j \tanh(\sum_{i'} w_{i'j}x_{i'})\right| [1-\tanh^2(\sum_{i'} w_{i'j}x_{i'})]w_{ij}^2 \tag{3.27}$$

This heuristic measure shows that there are two requirements for strong curvature to appear: the weights must be large and the argument of the $\tanh^2(\ldots)$ must be small. It is in the region where the combination of inputs and weights make the argument of

$\tanh^2(\ldots)$ small, that the network actually does the interesting computation (outside this region the hidden units are saturated). So, in the region of computation, a large curvature requires large weights.

As seen in fig. 3.4 these poorly regularized networks are large, containing in the order of 40 weights. Thus these networks are strongly nonlinear and overfit the data severely. These networks implement functions which are not reasonable, in the light of the data.

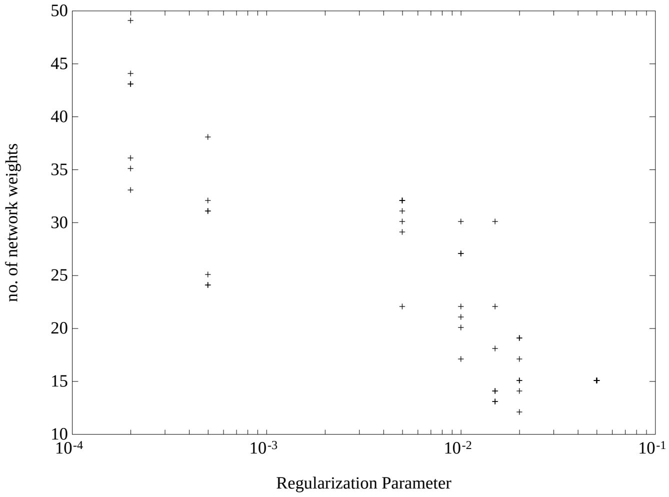

Figure 3.4: The size of the networks that were selected by FPE, as a function of the weight decay parameter ξ, for the same networks as in fig. 3.2. For small weight decays the networks are too large.

The application of weight decay puts a constraint on the network function, so that only reasonably smooth functions are considered. Weight decay confines the space which is searched to a region of sensible functions. But we are left with the problem of finding the optimal value for the weight decay parameter.

It would be nice if the FPE estimator could point out the networks with optimal weight decay. To examine whether this is the case, the FPE estimate for the networks are plotted as a function of the weight decay parameter in fig. 3.5. It is seen that the estimated test error is an increasing function of ξ. So the FPE estimator on its own, can not be used to select the optimal networks, among the networks found by this pruning scheme.

From fig. 3.2 and fig. 3.4 it is concluded that Akaike's criterion is not useful as a stop criterion unless weight decay is utilized. So the problem of finding a suitable weight decay parameter remains.

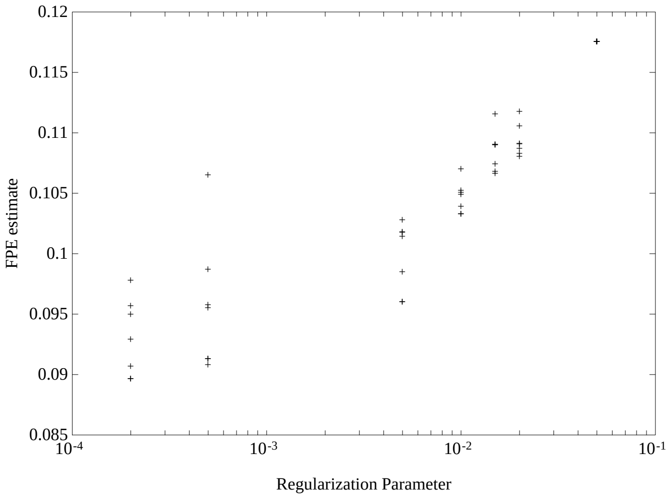

Figure 3.5: The final FPE estimate obtained using different weight decay parameters. The curve is an increasing function of ξ. The plot shows the the FPE value can not be used to select an optimal value for ξ for the networks considered here.

Chapter 4

The Generalization Method

This chapter presents the Generalization method. This method develops an estimate of the generalization error which is very close to Akaike's FPE, except that the bias introduced by the use of weight decay is explicitly accounted for in the following. Once an estimate of the generalization error is obtained it can be used to find an optimal value for the weight decay parameter.

4.1 Derivation of the Generalization Method

The following derivations are adapted from (Hansen, 1993). We will allow for the case of multiple weight decay parameters, by assigning an individual weight decay constant to each weight in the network. At a later point the values of these parameters may be fixed in groups, or to a single value. In this case the network training function is given by:

$$E = \frac{1}{2}\sum_{\alpha=1}^{p}(t^{\alpha} - F_w(\mathbf{x}^{\alpha}))^2 + \frac{1}{2}\sum_j \xi_j w_j^2 \tag{4.1}$$

As before we assume that the teacher lies in student space, and we call the teacher weights $\mathbf{w}^*$. The training examples are contaminated by zero mean additive independent noise with variance σ^2. Assuming that our current guess of the teacher is close to its actual value, we expand the network function in weight fluctuations:

$$F_w(\mathbf{x}^{\alpha}) \simeq F_{w*}(\mathbf{x}^{\alpha}) + \sum_j \frac{\partial F_{w*}(\mathbf{x}^{\alpha})}{\partial w_j}\delta w_j \tag{4.2}$$

Introducing the expansion in the cost function, we obtain[1]:

$$E = \frac{1}{2}\sum_{\alpha=1}^{p}\left[(t^{\alpha} - F_{w^*}^{\alpha})^2 + \sum_{jj'}\frac{\partial F_{w*}^{\alpha}}{\partial w_j}\frac{\partial F_{w*}^{\alpha}}{\partial w_{j'}}\delta w_j \delta w_{j'} - 2(t^{\alpha} - F_{w^*}^{\alpha})\sum_j \frac{\partial F_{w*}^{\alpha}}{\partial w_j}\delta w_j\right] \tag{4.3}$$

[1]For notational conveniency F_w^{α} will be used for $F_w(\mathbf{x}^{\alpha})$.

$$+\frac{1}{2}\sum_j \xi_j \left[(w_j^*)^2 + (\delta w_j)^2 + 2w_j^*\delta w_j\right]$$

Rewriting this in matrix notation:

$$E = \frac{1}{2}\sum_{\alpha=1}^{p}(t^\alpha - F_{w^*}^\alpha)^2 + \frac{p}{2}\delta\mathbf{w}^T\mathbf{H}\delta\mathbf{w} - p\mathbf{a}^T\delta\mathbf{w} + \frac{1}{2}(\mathbf{w}^*)^T\mathbf{D}\mathbf{w}^* + \frac{1}{2}\delta\mathbf{w}^T\mathbf{D}\delta\mathbf{w} + \delta\mathbf{w}^T\mathbf{D}\mathbf{w}^* \quad (4.4)$$

where we have defined:

$$\mathbf{H}_{jj'} = \frac{1}{p}\sum_{\alpha=1}^{p}\frac{\partial F_{w*}^\alpha}{\partial w_j}\frac{\partial F_{w*}^\alpha}{\partial w_{j'}} \qquad \mathbf{a_j} = \frac{1}{p}\sum_{\alpha=1}^{p}\nu^\alpha\frac{\partial F_{w*}^\alpha}{\partial w_j} \qquad \mathbf{D}_{jj'} = \xi_j\delta_{jj'} \quad (4.5)$$

During training the cost function is optimized, so:

$$\frac{\partial E}{\partial\delta\mathbf{w}} = \mathbf{0} \implies p\mathbf{H}\delta\mathbf{w} - p\mathbf{a} + \mathbf{D}\delta\mathbf{w} + \mathbf{D}\mathbf{w}^* = \mathbf{0} \quad (4.6)$$

$$\implies \delta\mathbf{w} = \mathbf{J}^{-1}(\mathbf{a}+\mathbf{b}) \quad (4.7)$$

where $\mathbf{J}$ and $\mathbf{b}$ are given by:

$$\mathbf{J}_{jj'} = \mathbf{H}_{jj'} + \frac{1}{p}\xi_j\delta_{jj'} \qquad b_j = -\frac{1}{p}\xi_j w_j^* \quad (4.8)$$

The average weight fluctuation, the *bias*, introduced by the weight decay term is given by:

$$\langle\delta\mathbf{w}\rangle_p = \langle\langle\mathbf{J}^{-1}(\mathbf{a}+\mathbf{b})\rangle_\nu\rangle_x = \langle\mathbf{J}^{-1}\rangle_x\mathbf{b} \quad (4.9)$$

since $(\mathbf{J}^{-1}\mathbf{a})$ averaged over ν is zero.

We are now ready to evaluate the average training error:

$$\langle\epsilon_T\rangle_p = \left\langle\frac{1}{p}\sum_{\alpha=1}^{p}(t^\alpha - F_w^\alpha)^2\right\rangle_p = \left\langle\frac{1}{p}\sum_{\alpha=1}^{p}\left(\sum_j\frac{\partial F_{w*}^\alpha}{\partial w_j}\delta w_j - \nu^\alpha\right)^2\right\rangle_p \quad (4.10)$$

$$= \left\langle\frac{1}{p}\sum_{\alpha=1}^{p}(\nu^\alpha)^2 + \frac{1}{p}\sum_{\alpha=1}^{p}\sum_{jj'}\frac{\partial F_{w*}^\alpha}{\partial w_j}\frac{\partial F_{w*}^\alpha}{\partial w_{j'}}\delta w_j\delta w_{j'} - \frac{2}{p}\sum_{\alpha=1}^{p}\sum_j\frac{\partial F_{w*}^\alpha}{\partial w_j}\delta w_j\nu^\alpha\right\rangle_p$$

$$= \sigma^2 + \sum_{jj'}\langle\mathbf{H}_{jj'}\delta w_j\delta w_{j'}\rangle_p - 2\sum_j\left\langle\frac{1}{p}\sum_{\alpha=1}^{p}\frac{\partial F_{w*}^\alpha}{\partial w_j}\delta w_j\nu^\alpha\right\rangle_p \quad (4.11)$$

The two remaining averages of eq. (4.11) will be evaluated separately. The first average is computed by introducing eq. (4.7):

$$\sum_{jj'} \langle \mathbf{H}_{jj'} \delta w_j \delta w_{j'} \rangle_p \tag{4.12}$$

$$= \sum_{jj'} \left\langle \mathbf{H}_{jj'} \left(\sum_k \mathbf{J}^{-1}_{jk}(a_k + b_k) \right) \left(\sum_{k'} \mathbf{J}^{-1}_{j'k'}(a_{k'} + b_{k'}) \right) \right\rangle_p \tag{4.13}$$

$$= \sum_{jj'} \sum_{kk'} \langle \langle \mathbf{H}_{jj'} \mathbf{J}^{-1}_{jk} a_k \mathbf{J}^{-1}_{j'k'} a_{k'} \rangle_\nu \rangle_x + \langle \langle \mathbf{H}_{jj'} \mathbf{J}^{-1}_{jk} b_k \mathbf{J}^{-1}_{j'k'} b_{k'} \rangle_\nu \rangle_x \tag{4.14}$$

$$= \frac{\sigma^2}{p} \sum_{jj'} \sum_{kk'} \langle \mathbf{H}_{jj'} \mathbf{J}^{-1}_{jk} \mathbf{H}_{kk'} \mathbf{J}^{-1}_{j'k'} \rangle_x + \sum_{jj'} \sum_{kk'} \langle \mathbf{H}_{jj'} \mathbf{J}^{-1}_{jk} b_k \mathbf{J}^{-1}_{j'k'} b_{k'} \rangle_x \tag{4.15}$$

$$= \frac{\sigma^2}{p} \left\langle \mathrm{Tr}(\mathbf{HJ}^{-1}\mathbf{HJ}^{-1}) \right\rangle_x + \frac{1}{p^2} \left\langle (\mathbf{w}^*)^T \mathbf{DJ}^{-1}\mathbf{HJ}^{-1}\mathbf{Dw}^* \right\rangle_x \tag{4.16}$$

The second average of eq. (4.11) is computed by again introducing eq. (4.7):

$$2 \sum_j \left\langle \frac{1}{p} \sum_{\alpha=1}^{p} \frac{\partial F^\alpha_{w*}}{\partial w_j} \delta w_j \nu^\alpha \right\rangle_p \tag{4.17}$$

$$= 2 \sum_j \left\langle \left\langle \frac{1}{p} \sum_{\alpha=1}^{p} \frac{\partial F^\alpha_{w*}}{\partial w_j} \nu^\alpha \left(\sum_k \mathbf{J}^{-1}_{jk}(a_k + b_k) \right) \right\rangle_\nu \right\rangle_x \tag{4.18}$$

$$= 2 \frac{\sigma^2}{p} \sum_{jk} \langle \mathbf{H}_{jk} \mathbf{J}^{-1}_{jk} \rangle_x \tag{4.19}$$

$$= 2 \frac{\sigma^2}{p} \left\langle \mathrm{Tr}(\mathbf{HJ}^{-1}) \right\rangle_x \tag{4.20}$$

Summing up, the average training error thus becomes:

$$\langle \epsilon_T \rangle_p = \sigma^2 - \frac{\sigma^2}{p} \left\langle \mathrm{Tr}(\mathbf{HJ}^{-1}(2\mathbf{I} - \mathbf{HJ}^{-1})) \right\rangle_x + \frac{1}{p^2} \left\langle (\mathbf{w}^*)^T \mathbf{DJ}^{-1}\mathbf{HJ}^{-1}\mathbf{Dw}^* \right\rangle_x \tag{4.21}$$

In the limit $\xi \to \mathbf{0}$ we have $\mathbf{HJ}^{-1} \to \mathbf{I}$ so the above equation reduces to the expression found from Akaike, eq. (3.19).

The average generalization error becomes:

$$\langle \epsilon_G \rangle_p = \int \left\langle (t - F_w(\mathbf{x}))^2 \right\rangle_p P(\mathbf{x}) P(\nu) d\mathbf{x} d\nu \tag{4.22}$$

$$= \int \left\langle \left(t - F_{w*} - \sum_j \frac{\partial F_{w*}}{\partial w_j} \mathbf{J}^{-1}(a_j + b_j) \right)^2 \right\rangle_p P(\mathbf{x}) P(\nu) d\mathbf{x} d\nu \tag{4.23}$$

$$= \sigma^2 + \sum_{jj'} \frac{\partial F_{w*}}{\partial w_j} \frac{\partial F_{w*}}{\partial w_{j'}} \left[\langle \mathbf{J}^{-1} a_j \mathbf{J}^{-1} a_{j'} \rangle_p + \langle \mathbf{J}^{-1} b_j \mathbf{J}^{-1} b_{j'} \rangle_p \right] \tag{4.24}$$

$$= \sigma^2 + \frac{\sigma^2}{p} \mathrm{Tr}(\mathbf{H}_\infty \langle \mathbf{J}^{-1}\mathbf{HJ}^{-1} \rangle_x) + \frac{1}{p^2} (\mathbf{w}^*)^T \mathbf{D} \langle \mathbf{J}^{-1} \rangle_x \mathbf{H}_\infty \langle \mathbf{J}^{-1} \rangle_x \mathbf{Dw}^* \tag{4.25}$$

When utilizing the framework for a particular learning task, we do not have access to $\mathbf{H}_\infty$, $\mathbf{w}^*$ or the averages over the input distribution $\langle \ldots \rangle_x$. In order to proceed in a particular application we therefore replace the averages by the actual values, replace $\mathbf{H}_\infty$ by $\mathbf{H}$ and the teacher weights $\mathbf{w}^*$ by the current weights $\mathbf{w}$. We have thus again used the assumptions about the inputs being representative and the student function being close to the teacher function.

Under these conditions we are able to optimize the weight decay parameter to achieve the best generalization performance. Assume that we have only a single weight decay, ie. $\xi_j = \xi$ for all j. Assume further that the value of the weight decay parameter is small so we are justified in the following expansion of $\mathbf{J}^{-1}$:

$$\mathbf{J}^{-1} = (\mathbf{H} + \frac{1}{p}\mathbf{D})^{-1} \simeq \mathbf{H}^{-1} - \mathbf{H}^{-1}\frac{1}{p}\mathbf{D}\mathbf{H}^{-1} \tag{4.26}$$

Substituting this in the generalization error we obtain:

$$\begin{aligned}
\langle \epsilon_G \rangle_p &= \sigma^2 + \frac{\sigma^2}{p}\mathrm{Tr}\left[\mathbf{H}(\mathbf{H}^{-1} - \mathbf{H}^{-1}\frac{1}{p}\mathbf{D}\mathbf{H}^{-1})\mathbf{H}(\mathbf{H}^{-1} - \mathbf{H}^{-1}\frac{1}{p}\mathbf{D}\mathbf{H}^{-1})\right] && (4.27)\\
&\quad + \frac{1}{p^2}\mathbf{w}^T\mathbf{D}(\mathbf{H}^{-1} - \mathbf{H}^{-1}\frac{1}{p}\mathbf{D}\mathbf{H}^{-1})\mathbf{H}(\mathbf{H}^{-1} - \mathbf{H}^{-1}\frac{1}{p}\mathbf{D}\mathbf{H}^{-1})\mathbf{D}\mathbf{w}^T \\
&= \sigma^2 + \frac{\sigma^2}{p}\mathrm{Tr}\left[(\mathbf{I} - \frac{1}{p}\mathbf{D}\mathbf{H}^{-1})(\mathbf{I} - \frac{1}{p}\mathbf{D}\mathbf{H}^{-1})\right] + \frac{\xi^2}{p^2}\mathbf{w}^T\mathbf{H}^{-1}\mathbf{w} && (4.28)\\
&= \sigma^2 + \frac{N\sigma^2}{p} - 2\frac{\xi\sigma^2}{p^2}\mathrm{Tr}(\mathbf{H}^{-1}) + \frac{\xi^2}{p^2}\mathbf{w}^T\mathbf{H}^{-1}\mathbf{w} && (4.29)
\end{aligned}$$

Finding the optimal value for the weight decay parameter:

$$\begin{aligned}
\frac{\partial}{\partial \xi}\langle \epsilon_G \rangle_p = 0 \quad &\Longrightarrow \quad -2\frac{\sigma^2}{p^2}\mathrm{Tr}(\mathbf{H}^{-1}) + 2\frac{\xi}{p^2}\mathbf{w}^T\mathbf{H}^{-1}\mathbf{w} = 0 && (4.30)\\
&\Longrightarrow \quad \xi = \frac{\sigma^2\mathrm{Tr}(\mathbf{H}^{-1})}{\mathbf{w}^T\mathbf{H}^{-1}\mathbf{w}} && (4.31)
\end{aligned}$$

which is valid for small weight decays.

4.2 Application to the Sunspot problem

We can test the utility of eq. (4.31) by applying it to the linear model for the sunspot prediction task. An iterative procedure is adopted for finding the optimal weight decay. The values of ξ and $\mathbf{w}$ are alternately reevaluated using eq. (4.31) and the equation for $\mathbf{w}$:

$$\frac{\partial E}{\partial \mathbf{w}} = \mathbf{0} \Longrightarrow \mathbf{w} = (p\mathbf{H} + \xi\mathbf{I})^{-1}\sum_{\alpha=1}^{p}\mathbf{x}^\alpha t^\alpha \tag{4.32}$$

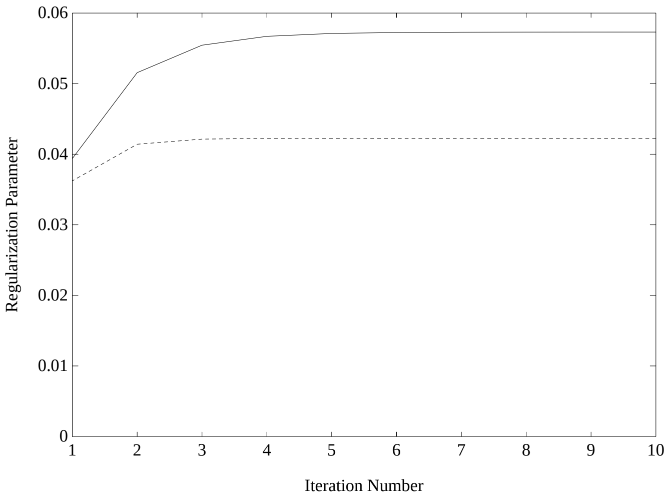

Figure 4.1: The estimated value of the regularization parameter during iteration. Full line is obtained using the full $\mathbf{H}$ in eq. (4.31) and dashed line is obtained using a diagonal approximation for $\mathbf{H}$. The system settles in a few iterations to reasonable values, cf. fig. 2.2.

Figure 4.1 shows the very encouraging result that the weight decays obtained using the estimate from the Generalization method correspond very closely to the results using the 'Leave One Out' procedure depicted is fig. 2.2. Even in the case where a diagonal approximation for $\mathbf{H}$ is used in eq. (4.31) the results are still good. This approximation will be useful when applying eq. (4.31) to the general case of nonlinear networks.

4.2.1 Individual Weight Decays

It is not always beneficial to apply the same weight decay to all the parameters in an entire network. In the case of the linear model considered here for the sunspot prediction problem there is no obvious way to divide the weights into classes (although it could be done according to their maximum likelihood magnitudes). Instead, it will be tried to take the number of weight decays to the very extreme of a separate weight decay for each parameter.

Returning to eq. (4.27) and redoing the calculations with separate weight decays, we obtain:

$$\langle \epsilon_G \rangle_p = \sigma^2 + \frac{N\sigma^2}{p} - 2\frac{\sigma^2}{p^2}\mathrm{Tr}(\mathbf{D}\mathbf{H}^{-1}) + \frac{1}{p^2}\mathbf{w}^T\mathbf{D}\mathbf{H}^{-1}\mathbf{D}\mathbf{w} \tag{4.33}$$

Using a diagonal approximation for $\mathbf{H}$ and maximizing the expected generalization error:

$$\frac{\partial}{\partial \xi_j}\langle \epsilon_G \rangle_p = 0 \Longrightarrow -2\frac{\sigma^2}{p^2}(\mathbf{H}^{-1})_{jj} + 2\frac{\xi_j}{p^2}w_j^2(\mathbf{H}^{-1})_{jj} = 0 \Longrightarrow \xi_j = \frac{\sigma^2}{w_j^2} \tag{4.34}$$

This equation is derived under the condition that the weight decay parameters are small. As we will see in a moment some of the weight decay parameters diverge to very large

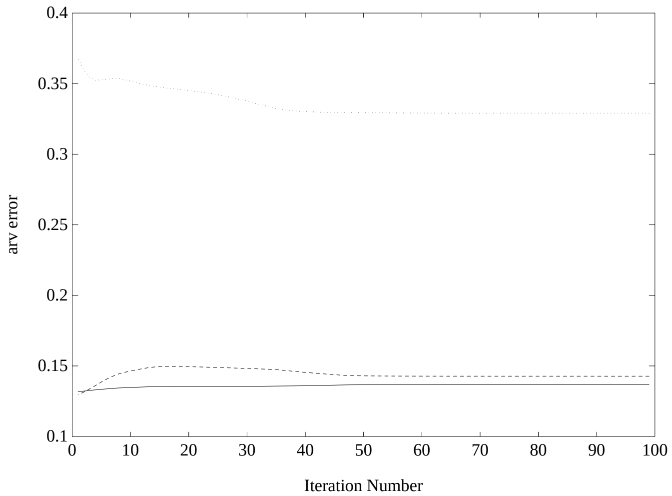

Figure 4.2: The training (solid) and test errors (dashed, test set 1 and dotted, test set 2) for a linear model with separate weight decays for each parameter trained on the sunspot problem. The system settles to situation where the performance on test set 1 is worse that without regularization, but where the performance on test set two has increased considerably.

values; in this situation eq. (4.34) holds only for the subspace of weights with small weight decays. The parameters which have diverging weight decays are effectively eliminated (pruned) from the problem, thus at any time only the subspace of weights with small weight decays is considered.

Equation (4.34) can similarly be used iteratively together with eq. (4.32) to train a linear model. Two modifications were added to this simple scheme; firstly a tiny constant was added in the denominator of eq. (4.34) to insure numerical stability when weights become small[2]. Secondly, a low pass filter was applied to the update of weight decay, to ensure that the model could settle gently to the desired point. Thus, the following equations was used to update ξ:

$$\xi_j(t) = 0.8 \times \xi_j(t-1) + \frac{0.2\sigma^2}{w_j^2 + 10^{-6}} \tag{4.35}$$

where $\xi(t)$ indicates ξ at iteration step t. The system is started with zero weight decay and trained for 100 iterations.

In fig. 4.2 the performance of the model is shown throughout the iteration scheme. By comparing with fig. 2.2 we note that performance on test set 1 deteriorates, while a substantial improvement is seen for test set 2. This discrepancy is probably due to the very limited size of the two test sets. It should be noted that the final model only comprises 4 weights which are different from zero. The weight decays for the 9 remaining weights have grown so large that the weights have effectively been pruned from the network.

[2] In a more well-programmed solution, the very small weights should instead be explicitly removed from the problem. The addition of a small constant should be regarded as a 'quick and dirty' solution.

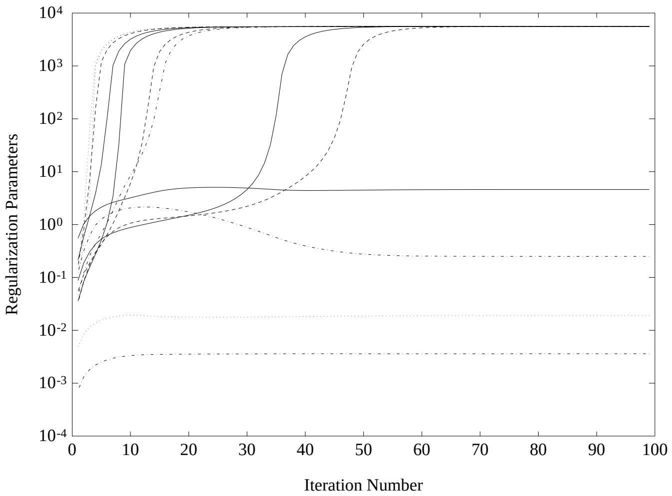

Figure 4.3: The development of the weight decay parameters during iteration on the linear sunspot problem. Only 4 parameters survive, for the rest weight decay increases to very high levels (only limited artificially). Note the log scale.

In fig. 4.3 it is seen how 9 of the 13 weight decays grow to very high values, effectively pruning their weights from the network; only 4 weights survive. It should be noted that the same final network is in fact obtained, when using the scheme without the low pass filter on the equation for ξ. In this case the system behaviour is very fast, but still settles to the same solution. In general it is thought advisable to keep the changes to the weight decays on a slower time scale to insure that the system settles properly. If a weight because of some transient phenomenon momentarily becomes very small, it can be caught there by its weight decay if these are allowed to adapt at high speed.

It is concluded that individual weight decays may be an effective means of deciding which parameters should be included in a model. However, a more thorough numerical testing of this hypothesis is certainly needed. The data from the sunspot series are inconclusive, mainly because of the extremely limited size of the test sets.

4.2.2 Application to Nonlinear Networks

A simulation which iteratively updated a single weight decay using eq. (4.31) was set up for a pruning session similar to that described in chapter 3. However, some difficulties were encountered in these simulations. A diagonal approximation for $\mathbf{H}$ was used in eq. (4.31). The problems arose when one of the output weights achieved a value close to zero. In this case the curvature of the cost in directions in weight space defined by the input weights to that hidden unit becomes very shallow, ie. the diagonal element of $\mathbf{H}$ becomes small. The result is that the corresponding input weights become very small, while the inverse diagonal element of $\mathbf{H}$ becomes huge, thus ξ increases dramatically, which in turn destroys the network.

It was seen however that once all the input weights to the troublesome hidden unit were pruned, the weight decay returned to normal values and reasonable networks reappeared.

Also it should be noted that the weight decay values which where found (when the algorithm was not 'out of bounds') where close to the optimal values found elsewhere (Svarer et al., 1993). These values started off at ~ 0.03 and decreased monotonically through the pruning session to about ~ 0.005. In (Svarer et al., 1993) constant weight decays of 0.01 and 0.02 where used, and the final network where retrained without weight decay.

It should be emphasized that the problems which were encountered here have easy solutions. Firstly, a more advanced pruning algorithm which can remove more than a single weight per step could alleviate the problems by simply removing the troublesome (and insignificant) weights before they havoc the entire net. Secondly, a schedule where separate weight decays are used for each hidden unit, would decouple the the weight decays, such that a problem of the above kind would cause the removal of the troublesome unit only (as desired). The implementation of these ideas require a little extra programming, but they will be pursued in the near future.

Chapter 5

Bayesian Back Propagation

This chapter discusses the Bayesian framework for back propagation which MacKay introduced to the neural network community (MacKay, 1992a; MacKay, 1992c). The two main contributions of these papers are: the possibility of setting weight decay parameters without using a validation set and the possibility of ranking of networks with different architectures (and other models) by the use of *evidence*.

Bayesian methods make it possible to introduce *priors* in the modeling process, which turns out to be extremely important for the nonparametric inference involved in training neural networks. Only a small degree of subjectiveness is required to set the priors, since examination of the evidence is used to tune the priors.

It would be unfair to give the impression that Bayesian analysis is universally accepted among statisticians (Pallesen, 1979). The interpretation of the priors is controversial, since the *classical* statistician (as opposed to the *Bayesian*) uses a definition of the probability concept, which is closely tied to a frequency interpretation. The priors used for Bayesian inference are not always associated with a corresponding sampling frequency, rather they reflect (subjective) degrees of *belief*. The theoretical foundations of the definitions of the probability concept will not be discussed further here. A pragmatic attitude towards these problems is adopted; it will be assumed that the priors which are used make sense (at some level), and evaluation of the methods will rely on how well they actually work.

5.1 Bayesian Interpolation

The ultimate goal of the following analysis is evaluating the probability of a model, conditioned on a set of data: $P(\mathcal{H}|D)$, where the data set is given by:

$$D = \{\mathbf{x}^\alpha, t^\alpha | \alpha = 1, \ldots, p\} \tag{5.1}$$

Here $\mathcal{H}$ denotes a particular model, eg. a fully connected neural network, with one linear output and one layer of 8 hidden hyperbolic tangent units and 12 inputs. Actually, what we will find is not a properly normalized probability as stated above, but rather a non-normalized measure, given on an arbitrary scale. Normalization of these probabilities would require 'the space of models' to be defined, which is intractable. Since our primary objective is the comparison of models, no absolute scale is needed.

The evaluation of the above quantity involves several levels of inference. As an example, we shall consider how to infer the weights of a neural network. We use Bayes' rule to combine the *likelihood* with the *prior*, to obtain the *posterior* weights:

$$P(\mathbf{w}|D,\mathcal{H}) = \frac{P(D|\mathbf{w},\mathcal{H})P(\mathbf{w}|\mathcal{H})}{P(D|\mathcal{H})} \tag{5.2}$$

where $\mathbf{w}$ are the network weights. Maximizing the posterior yields the *maximum aposteriori* (MAP) estimation of $\mathbf{w}$. The normalization term in the denominator of eq. (5.2) is called the *evidence* for $\mathcal{H}$.

At the next[1] level of inference different models are compared. Again we use Bayes' rule, this time to infer the posterior for the model:

$$P(\mathcal{H}|D) = \frac{P(D|\mathcal{H})P(\mathcal{H})}{P(D)} \tag{5.3}$$

We note that the evidence from before appears together with a prior on models, $P(\mathcal{H})$. When comparing posteriors for different models, the normalization constant $P(D)$ is omited, cf. the discussion above.

5.1.1 Why the Framework Embodies Ockham's Razor

Ockham's Razor is a fundamental principle in science, stating that simpler models which explain some phenomenon, should be preferred over more elaborate ones. This touches precisely on the question which is interesting when doing nonparametric inference; how should the data fit be traded off against model simplicity?

This trade off is embodied by the Bayesian framework. To see how this comes about, consider again the evidence from eq. (5.2). It acts as a normalization (over weight space) for the posterior, which can be used in its evaluation:

$$\begin{aligned} P(D|\mathcal{H}) &= \int P(D|\mathbf{w},\mathcal{H})P(\mathbf{w}|\mathcal{H})d\mathbf{w} && (5.4) \\ &\simeq P(D|\mathbf{w}_{MP},\mathcal{H})\frac{\Delta\mathbf{w}_{posterior}}{\Delta\mathbf{w}_{prior}} && (5.5) \end{aligned}$$

The weights are said to be marginalized from the model. Looking at the equation, we note that the more diffuse the prior (ie. the prior spreading over large intervals in many dimensions), the lower the evidence, since the prior itself is also normalizable. In this way, over elaborate models with too many poorly regularized parameters are discouraged. It costs an Ockham factor to marginalize a parameter.

[1]The more advanced inference in the following includes an intermediate level.

5.2 Bayesian Back Prop

In this section the theory for the Bayesian framework for feed forward networks will be derived. Attention will be paid to formulating the problem in a proper Bayesian way and making the deviations from this proper path (due to the desire for analytical and computational simplicity) explicit.

5.2.1 The Weight Likelihood

Formulation of probabilistic cost function, minus the log likelihood, is required in order to do the Bayesian inference. Assume that the p examples which constitute the data base D, are contaminated by independent zero mean additive Gaussian noise on the dependent variables.[2] In this case the likelihood of the parameters is given by[3]:

$$P(D|\mathbf{w}, \sigma^2, \mathcal{H}) = \left(\frac{1}{\sqrt{2\pi\sigma^2}}\right)^p \exp -\frac{1}{\sigma^2} E_D \tag{5.6}$$

where σ^2 is the variance of the noise, which at this stage is unknown, and E_D is the well known quadratic 'data error' term:

$$E_D = \frac{1}{2} \sum_{\alpha=1}^{p} (F(\mathbf{x}^\alpha) - t^\alpha)^2 \tag{5.7}$$

5.2.2 The Weight Prior

The prior on the weights should state that the magnitude of the weights is probably small. There are several arguments as to why such a prior is expected to be useful. Firstly, we know that large weights are needed in order to boost the non linearities in the network. So by using a prior which states that the weights should be small, we are demanding a high price in improved data fit, for the acceptance of non linearities. Secondly, by confining the weights to a limited interval, they give rise to a smaller Ockham factor (an interval around zero is assumed, reflecting that we have no prior expectation about the sign of the weights).

In this section we will (for the sake of simplicity) use a prior characterized by a single parameter. In general, a framework which allows for multiple weight decays should be utilized; when training a network with a hidden layer, the magnitude of the input weights should not be coupled to the magnitude of the output weights, since only the input weights scale with the (arbitrary) size of the inputs.

[2] In the case of auto regression, this assumption is obviously false, since the dependent and independent variables are from the same distribution. However if the model is not too non linear, the noise contributions can be included in the output signal.

[3] By $P(D|\ldots)$ is meant: $P(t^1, \ldots, t^p|\mathbf{x}^1, \ldots, \mathbf{x}^p, \ldots)$. This convenient abuse of notation will be used extensively in the following.

The exact form of the prior is usually chosen to be Gaussian, since this makes the subsequent analysis simple, and leads to the well known weight decay:

$$P(\mathbf{w}|\xi,\sigma^2,\mathcal{H}) = \left(\sqrt{\frac{\xi}{2\pi\sigma^2}}\right)^N \exp -\frac{\xi}{\sigma^2}E_w \tag{5.8}$$

where E_w is the 'weight error' term:

$$E_w = \frac{1}{2}\sum_i^N w_i^2 \tag{5.9}$$

where the sum extends over all weights (and biases). The prior is parametrized by the parameter ξ/σ^2. This parametrization was suggested by (Thodberg, 1993). The value of ξ is not known at this stage. In the following ξ and σ^2 will be called the *scales* of the model. They are lumped in the vector $\mathbf{s}$.

5.2.3 The Maximum Aposteriori Weights

The posterior weights are obtained from Bayes' rule:

$$P(\mathbf{w}|\xi,\sigma^2,D,\mathcal{H}) = \frac{P(D|\mathbf{w},\xi,\sigma^2,\mathcal{H})P(\mathbf{w}|\xi,\sigma^2,\mathcal{H})}{P(D|\xi,\sigma^2,\mathcal{H})} \tag{5.10}$$

The maximum aposteriori weights are found by maximizing the posterior with respect to the weights. During this process the normalization constant of eq. (5.10) can be neglected, since it is independent of $\mathbf{w}$. Using eq. (5.6) and (5.8):

$$\frac{\partial}{\partial\mathbf{w}}P(\mathbf{w}|\xi,\sigma^2,D,\mathcal{H}) = \mathbf{0} \tag{5.11}$$

$$\Longrightarrow \quad \frac{\partial}{\partial\mathbf{w}}\left[\left(\frac{1}{\sqrt{2\pi\sigma^2}}\right)^p \left(\sqrt{\frac{\xi}{2\pi\sigma^2}}\right)^N \exp\left(-\frac{\xi}{\sigma^2}E_w - \frac{1}{\sigma^2}E_D\right)\right] = \mathbf{0} \tag{5.12}$$

$$\Longrightarrow \quad \frac{\partial}{\partial\mathbf{w}}(E_D + \xi E_w) = \mathbf{0} \tag{5.13}$$

When ξ is given, the maximum aposteriori weights, w_{MP}, are found simply by training the network with a modified cost function, which is seen to be identical to training a network with the usual weight decay.

5.2.4 The Evidence for s and $\mathcal{H}$

The next step is to evaluate the evidence for $\mathbf{s}$ and $\mathcal{H}$, obtained from the normalization condition of eq. (5.10):

$$P(D|\xi,\sigma^2,\mathcal{H}) = \int P(D|\mathbf{w},\xi,\sigma^2,\mathcal{H})P(\mathbf{w}|\xi,\sigma^2,\mathcal{H})d\mathbf{w} \tag{5.14}$$

Unfortunately, this integral can not be solved exactly, since it contains the non linear network function $F(\mathbf{x})$. MacKay suggested that we approximate the log probability with a quadratic function in $\mathbf{w}$, ie., the probability by a Gaussian:

$$\begin{aligned} P(D|\xi,\sigma^2,\mathcal{H}) \quad \simeq \quad & \left(\frac{1}{\sqrt{2\pi\sigma^2}}\right)^p \left(\sqrt{\frac{\xi}{2\pi\sigma^2}}\right)^N \exp\left(-\frac{1}{\sigma^2}E_D^{MP} - \frac{\xi}{\sigma^2}E_w^{MP}\right) \times \\ & \int \exp -\frac{1}{2}(\mathbf{w}-\mathbf{w}_{MP})^T\mathbf{A}(\mathbf{w}-\mathbf{w}_{MP})d^N\mathbf{w} \end{aligned} \tag{5.15}$$

where

$$\mathbf{A} = \frac{1}{\sigma^2}(\mathbf{B}+\xi\mathbf{I}) \qquad \mathbf{B}_{ij} = \left.\frac{\partial^2 E_D}{\partial\mathbf{w}_i\partial\mathbf{w}_j}\right|_{\mathbf{w}=\mathbf{w}_{MP}} \tag{5.16}$$

and the connection between the maximum likelihood and maximum posterior weights is given by:

$$\mathbf{w}_{MP} = \frac{1}{\sigma^2}\mathbf{A}^{-1}\mathbf{B}\mathbf{w}_{ML} \tag{5.17}$$

Doing the Gaussian integral[4] we obtain:

$$\begin{aligned} &\log P(D|\xi,\sigma^2,\mathcal{H}) = \\ &\quad -\frac{1}{\sigma^2}(\xi E_w^{MP} + E_D^{MP}) - \frac{1}{2}\log\det(\mathbf{B}+\xi\mathbf{I}) - \frac{p}{2}\log(2\pi\sigma^2) + \frac{N}{2}\log\xi \end{aligned} \tag{5.18}$$

5.2.5 The Posterior Distribution of Weights

We can now combine eq. (5.10) with eq. (5.18) to obtain an expression for the posterior distribution of the weights:

$$\begin{aligned} &P(\mathbf{w}|\xi,\sigma^2,D,\mathcal{H}) = \\ &\quad \left(\frac{1}{\sqrt{2\pi\sigma^2}}\right)^N \sqrt{\det(\mathbf{B}+\xi\mathbf{I})}\exp\frac{1}{\sigma^2}\left(E_D^{MP} - E_D + \xi E_w^{MP} - \xi E_w\right) \end{aligned} \tag{5.19}$$

[4] Using: $\int_{-\infty}^{\infty} \exp -\frac{1}{2}\mathbf{w}^T\mathbf{A}\mathbf{w}d^N\mathbf{w} = (2\pi)^{N/2}\det^{-1/2}\mathbf{A}$

5.2.6 The Maximum Aposteriori Scales

At the second level of inference we want to determine the most probable values for the scales. At this level Bayes' rule combines the evidence for **s** and $\mathcal{H}$ with a prior for the scales to obtain the posterior for the scales:

$$P(\xi, \sigma^2 | D, \mathcal{H}) = \frac{P(D|\xi, \sigma^2, \mathcal{H}) P(\xi, \sigma^2 | \mathcal{H})}{P(D|\mathcal{H})} \tag{5.20}$$

where we can omit the normalization term $P(D|\mathcal{H})$ when searching for the maximum posterior ξ and σ^2, since it is independent on these parameters.

5.2.7 Priors on ξ and σ^2

Now we have to chose a prior for σ^2 and ξ. Since (in general) we don't have strong expectations about these values, we want to specify priors with little influence on the posterior. One obvious way to make a prior which does not influence the posterior probability is by chosing the prior to be independent of the parameter in question, ie. a constant. In this way the posterior becomes equal to the likelihood, so the prior has no influence at all. Before a more detailed discussion it should be emphasized, that it is probably not very fruitful to try to find the 'right' prior in any particular sense, but rather to try a few (or a single) reasonable one(s), and let the evidence decide which is the best. This holds in particularly for the case where we have multiple priors.

There are arguments, which suggest that in some cases a constant prior may be inappropriate. Both the parameters ξ and σ^2 describe the relation between two model quantities. The regularization parameter ξ quantifies the relation between the data fit and the smoothness. The noise parameter quantifies the relationship between the noise and the (arbitrary) scale of the targets. For such relational parameters, uniform priors are inappropriate.

To see why this is the case, we should look at the next level of inference, where the scale parameters are marginalized:

$$P(D|\mathcal{H}) = \int P(D|\xi, \sigma^2, \mathcal{H}) P(\xi, \sigma^2 | \mathcal{H}) d\xi d\sigma^2 \tag{5.21}$$

Now consider an example where the only values of ξ which gives non-negligible contributions to the integral lie in the interval 2 to 3. In this case we integrate over an interval of width 1. Now consider a case where we instead of ξ had chosen the reciprocal parameter $\xi' = 1/\xi$. Since none of these parametrizations can be said to be more correct than the other, it is natural to require the value of the integral for $P(D|\mathcal{H})$ to be independent of this simple reparametrization. But, using a constant prior, the integral in ξ' from the above example will only extend over an interval of width $1/2 - 1/3 = 1/6$, yielding a different result for the integral.

The solution to this apparent problem is to use a prior instead which is uniform in log[5] space, rather than in linear space. On the log axis the two intervals in the example

[5] All log's are natural log's

above have the same width, namely $\log \frac{3}{2}$. We conclude that the natural measure for the relational parameters is the log of the parameters.

Thus, the prior for ξ should be uniform on an interval of width Ω_ξ in log space containing the most probable value, ξ_{MP}:

$$P(\log \xi|\mathcal{H}) = \frac{1}{\log \Omega_\xi} \tag{5.22}$$

where the connection between $P(\xi)$ and $P(\log \xi)$ is given by:

$$P(\xi|\mathcal{H})d\xi = P(\log \xi|\mathcal{H})d(\log \xi) = \frac{1}{\xi}P(\log \xi|\mathcal{H})d\xi \tag{5.23}$$

Combining the equations above we achieve the prior for ξ and similarly for σ^2:

$$P(\xi|\mathcal{H}) = \frac{1}{\xi \log \Omega_\xi} \qquad P(\sigma^2|\mathcal{H}) = \frac{1}{\sigma^2 \log \Omega_{\sigma^2}} \tag{5.24}$$

The sizes of the intervals Ω_ξ and Ω_{σ^2} must be set by a subjective estimate of our prior ignorance of the scales. Typically a value of a few orders of magnitude may be utilized. This introduction of subjectiveness might be seen as hampering the entire framework, but in a practical situation, the solution found by this Bayesian approach is in fact very insensitive to these values (only the $\log \Omega$ occurs in the equations). In the following $\log \Omega = \log \Omega_\xi \log \Omega_{\sigma^2}$.

Returning now to eq. (5.20)[6], we can evaluate the maximum posterior ξ and σ^2, by neglecting the denominator (since it's independent of ξ and σ^2) and setting the derivative of $P(\log \xi, \log \sigma^2|D,\mathcal{H})$ with respect to $\log \xi$ and $\log \sigma^2$ to zero. For the noise parameter this gives:

$$\frac{\partial}{\partial \log \sigma^2}P(\log \xi, \log \sigma^2|D,\mathcal{H}) = 0 \Longleftrightarrow \frac{\partial}{\partial \sigma^2}P(D|\xi,\sigma^2,\mathcal{H}) = 0 \tag{5.25}$$

$$\Longrightarrow \quad \frac{\partial}{\partial \sigma^2}\log P(D|\xi,\sigma^2,\mathcal{H}) = \frac{1}{\sigma^4_{MP}}(\xi_{MP}E_w^{MP} + E_D^{MP}) - \frac{p}{2\sigma^2_{MP}} = 0 \tag{5.26}$$

$$\Longrightarrow \quad \sigma^2_{MP} = \frac{2}{p}(\xi_{MP}E_w^{MP} + E_D^{MP}) \tag{5.27}$$

Before finding the maximum posterior regularization parameter, the number of well determined parameters γ will be introduced. Referring to eq. (5.17) we note that the maximum posterior weights are determined by combining the prior with the data fit. In the eigenvector basis of $\mathbf{B}$ the weights are called $\tilde{\mathbf{w}}$. The degree to which the maximum posterior weights are determined by the data fit (the maximum likelihood weights) is (see fig. 5.1):

$$\tilde{w}_{MP,i} = \frac{\lambda_i}{\lambda_i + \xi_{MP}}\tilde{w}_{ML,i} = \gamma_i \tilde{w}_{ML,i} \tag{5.28}$$

[6] which in the log domain is: $P(\log \xi, \log \sigma^2|D,\mathcal{H}) = \dfrac{P(D|\xi,\sigma^2,\mathcal{H})P(\log \xi, \log \sigma^2|\mathcal{H})}{P(D|\mathcal{H})}$

where λ_i are the eigenvalues of $\mathbf{B}$. Here γ_i has a value between 0 and 1, which determines to what degree the posterior was determined by the data fit; it is called the well determined'ness of $\mathbf{w}_{MP,i}$. The total number of well determined weights is:

$$\gamma = \sum_{i=1}^{N} \frac{\lambda_i}{\lambda_i + \xi_{MP}} = N - \sum_{i=1}^{N} \frac{\xi_{MP}}{\lambda_i + \xi_{MP}} = N - \xi_{MP}\mathrm{Tr}(\mathbf{A}^{-1}) \tag{5.29}$$

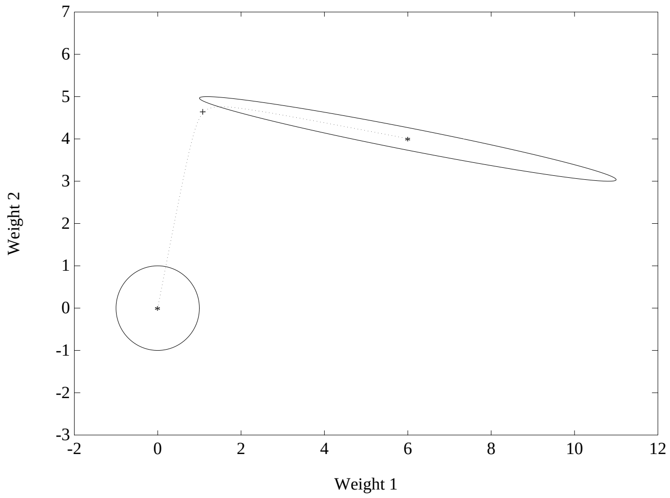

Figure 5.1: A contour plot from a two dimensional weight space. The prior is visualized by a circle centered on the origin. The likelihood is an ellipsis centered on the maximum likelihood value '$*$'. The circle and the ellipsis are iso-probability curves, the prior being drawn for $\xi = 1$. The likelihood has one well determined and one ill determined parameter. Note that the basis of the likelihood is tilted compared to the usual weight basis. The maximum posterior point follows a trajectory (dotted) when the weight decay is increased from 0 (maximum likelihood) to ∞ (the origin). With the relative strengths of prior and likelihood as indicated in the figure ($\xi = 1$) the maximum posterior point lies at '$+$'. Note that in the well determined direction the likelihood strongly fixes the posterior, while in the perpendicular, ill determined direction the prior quickly takes control.

The plot in fig. 5.1 was obtained using the following values for $\mathbf{B}$:

$$\mathbf{B} = \begin{bmatrix} 0.559 & 2.693 \\ 2.693 & 13.975 \end{bmatrix} \tag{5.30}$$

With $\xi = 1$ this gives (using eq. (5.29)) a number of well determined parameters of $\gamma = 1.01$. If we introduce a diagonal approximation for $\mathbf{A}$ instead of using eq. (5.29) as it stands, (ie. assume that all off diagonal elements are zero) we obtain $\gamma = 1.31$. This shows that the diagonal approximation for $\mathbf{A}$ is not always good; in order to make a good estimate of the number of well determined parameters we must include information about the couplings between the parameters. In fig. 5.1 there are quite strong couplings between the parameters, which is seen by the tiltedness of the likelihood ellipsis. The diagonal approximation gives an upper limit to the number of well determined parameters, since there may be ill determined directions that are not axis-parallel, which go undetected in the diagonal approximation.

We can now evaluate the maximum posterior value of the regularization parameter by differentiation[7] of $P(\log\xi, \log\sigma^2|D,\mathcal{H})$ with respect to $\log\xi$, and introducing eq. (5.27):

$$\frac{\partial}{\partial\log\xi}P(\log\xi,\log\sigma^2|D,\mathcal{H}) = 0 \Longleftrightarrow \frac{\partial}{\partial\xi}P(D|\xi,\sigma^2,\mathcal{H}) = 0 \tag{5.31}$$

$$\Longrightarrow \frac{\partial}{\partial\xi}\log P(D|\xi,\sigma^2,\mathcal{H}) = -\frac{E_w^{MP}}{\sigma_{MP}^2} - \frac{1}{2}\mathrm{Tr}(\mathbf{A}^{-1}) + \frac{N}{2\xi_{MP}} = 0 \tag{5.32}$$

$$\Longrightarrow \frac{E_w^{MP}}{\sigma_{MP}^2} = \frac{\gamma}{2\xi_{MP}} \Longleftrightarrow \xi_{MP} = \frac{E_D^{MP}\gamma}{E_w^{MP}(p-\gamma)} \tag{5.33}$$

We note that the maximum posterior values of the two parameters correspond to their maximum likelihood (obtained from $P(D|\xi,\sigma^2,\mathcal{H})$). This is in fact expected, since we used a prior which was flat (in log space), and therefore had no effect on the maximum posterior value.

5.2.8 Marginalization of the Scales

Now we can return to evaluating the evidence for the model, as given by eq. (5.21). In order to evaluate the integral we assume that the posterior for the scales can be approximated by a Gaussian:

$$P(D|\mathbf{s},\mathcal{H}) \simeq P(D|\mathbf{s}_{MP},\mathcal{H})\exp -\left(\frac{1}{2}(\log\mathbf{s}-\log\mathbf{s}_{MP})^T\mathbf{S}(\log\mathbf{s}-\log\mathbf{s}_{MP})\right) \tag{5.34}$$

where:

$$\mathbf{S}_{ij} = \left.\frac{\partial^2\log P(D|\mathbf{s},\mathcal{H})}{\partial\log\mathbf{s}_i\partial\log\mathbf{s}_j}\right|_{\mathbf{s}=\mathbf{s}_{MP}} \tag{5.35}$$

In this case the integral over $\mathbf{s}$ becomes:

$$P(D|\mathcal{H}) = \int P(D|\mathbf{s},\mathcal{H})P(\log\mathbf{s}|\mathcal{H})d\log\mathbf{s} = \tag{5.36}$$
$$\frac{1}{\log\Omega}P(D|\mathbf{s}_{MP},\mathcal{H})\int\exp -\left(\frac{1}{2}(\log\mathbf{s}-\log\mathbf{s}_{MP})^T\mathbf{S}(\log\mathbf{s}-\log\mathbf{s}_{MP})\right)d\log\mathbf{s}$$

Doing the integral, we obtain:

$$P(D|\mathcal{H}) = P(D|\mathbf{s}_{MP},\mathcal{H})\frac{2\pi}{\log\Omega}\frac{1}{\sqrt{\det\mathbf{S}}} \tag{5.37}$$

[7] It is used that: $\frac{\partial}{\partial\xi}\log\det(\mathbf{B}+\xi\mathbf{I}) = \frac{\partial}{\partial\xi}\log\prod_i(\lambda_i+\xi) = \frac{\partial}{\partial\xi}\sum_i\log(\lambda_i+\xi) = \sum_i\frac{1}{\lambda_i+\xi} = \mathrm{Tr}(\mathbf{A}^{-1})$, and assumed that $\frac{\partial}{\partial\xi}\mathbf{B}\simeq 0$ which is exact only for linear networks (where $\mathbf{B}$ is independent of $\mathbf{w}$).

where the tricky point is the evaluation of $\det \mathbf{S}$. The elements of $\mathbf{S}$ are derived below:

$$\frac{\partial}{\partial \log \sigma^2} \log P(D|\xi, \sigma^2, \mathcal{H}) = \frac{1}{\sigma^2}(\xi E_w^{MP} + E_D^{MP}) - \frac{p}{2} \tag{5.38}$$

Using that σ^2_{MP} corresponds to a maximum, eq. (5.26) we obtain:

$$\frac{\partial^2}{(\partial \log \sigma^2)^2} \log P(D|\xi, \sigma^2, \mathcal{H}) = -\frac{1}{\sigma^2}(\xi E_w^{MP} + E_D^{MP}) = -\frac{p}{2} \tag{5.39}$$

And, for the regularization parameter:

$$\frac{\partial}{\partial \log \xi} \log P(D|\xi, \sigma^2, \mathcal{H}) = -\frac{\xi}{\sigma^2} E_w^{MP} - \frac{\xi}{2}\mathrm{Tr}(\mathbf{A}^{-1}) + \frac{N}{2} \tag{5.40}$$

Using eq. (5.32):

$$\begin{aligned}&\frac{\partial^2}{(\partial \log \xi)^2} \log P(D|\xi, \sigma^2, \mathcal{H}) = \\ &\quad -\frac{\xi}{\sigma^2} E_w^{MP} + \frac{1}{2}\sum_{i=1}^{N} \frac{\xi^2}{(\lambda_i + \xi)^2} - \frac{\xi}{2}\mathrm{Tr}(\mathbf{A}^{-1}) = -\frac{N}{2} + \frac{1}{2}\sum_{i=1}^{N} \frac{\xi^2}{(\xi + \lambda_i)^2}\end{aligned} \tag{5.41}$$

and the cross term:

$$\frac{\partial^2}{\partial \log \xi \partial \log \sigma^2} \log P(D|\xi, \sigma^2, \mathcal{H}) = \frac{\xi}{\sigma^2} E_w^{MP} = \frac{\gamma}{2} \tag{5.42}$$

Using the above, we arrive at:

$$\det \mathbf{S} = \frac{pN}{4} - \frac{p}{4}\sum_{i=1}^{N} \frac{\xi^2}{(\xi + \lambda_i)^2} - \frac{\gamma^2}{4} \simeq \frac{Np - \gamma^2}{4} \tag{5.43}$$

This last approximation is valid if most of the parameters are well determined[8]. In this case the sum over eigenvalues above vanishes in comparison to $pN/4$. For reasonable networks (well pruned) it is expected that the number of well determined parameters is close to the number of parameters in the model, so $\xi \ll \lambda_i$ can be assumed to hold for most λ_i. Neglecting the sum over eigenvalues and reintroducing in eq. (5.37):

$$P(D|\mathcal{H}) = P(D|\mathbf{s}_{MP}, \mathcal{H}) \frac{2\pi}{\log \Omega} \sqrt{\frac{4}{pN - \gamma^2}} \tag{5.44}$$

Finally we reintroduce the maximum posterior scales from eq. (5.27) and (5.33) to obtain:

$$\log P(D|\mathcal{H}) = -\frac{p}{2} - \frac{p}{2}\log\frac{4\pi E_D^{MP}}{p - \gamma} + \frac{1}{2}\sum_{i=1}^{N} \log\frac{\xi_{MP}}{\lambda_i + \xi_{MP}} + \log\left(\frac{1}{\log \Omega}\frac{4\pi}{\sqrt{pN - \gamma^2}}\right) \tag{5.45}$$

[8]In the limit $N \simeq \gamma$ this corresponds to the results in (MacKay, 1992a; Thodberg, 1993) who have $\det \mathbf{S} \simeq (p\gamma - \gamma^2)/4$. The true value lies between these two estimates.

5.2.9 Symmetry Factors

At the first level of inference in the derivation of the Bayesian scheme, we calculated an Ockham factor for the weights, given by:

$$\mathrm{Ock}(\mathbf{w}) = \frac{\Delta \mathbf{w}_{posterior}}{\Delta \mathbf{w}_{prior}} \tag{5.46}$$

During the subsequent analysis we assumed the posterior to be well approximated by a (multivariate) Gaussian. In a fully connected neural network there are two types of symmetries, which make the assumption about the posterior being a single Gaussian invalid. Let's look at a fully connected network with a single hidden layer consisting of h units. In this network the signs of the incoming and outgoing weights can be swapped for each hidden unit, without altering the network function. Also the hidden units can be permuted without any change to the network function. So there are[9]:

$$h!2^h \tag{5.47}$$

possible implementations of the network function in question. In order to account for the degeneration we could multiply the posterior volume by the symmetry factor when calculating the Ockham factor for the weights. There are however problems with this approach.

When a fully connected network gets pruned of some of its weights, some of the symmetries in the network are lost. This would lead to a marked decrease in the evidence when pruning one weight of a fully connected network. This seems intuitively questionable. Also it is unclear whether a network which is pruned of a single weight belongs to the same architecture $\mathcal{H}$ as all the networks which can be permuted into this architecture. Further, it could be argued that pruned networks come in many more varieties than fully connected networks (ie. a missing hidden weight can be permuted among hidden units), so a *prior on architectures* $P(\mathcal{H})$ could be introduced. This prior should penalize sparse networks for having many possible realizations.

All in all, it is unclear to me at the present time how these issues should be treated. Firstly there are the fundamental theoretical problems of understanding what the abstract quantities such as 'evidence' and 'priors on architectures' really mean. Secondly, there are the issues of how to calculate the symmetry factors for sparse networks (and also for networks with weights close to zero, where the symmetrical Gaussians may to a large degree overlap).

However, it should be emphasized that the primary reason that we are interested in evaluating the evidence, is that it supposedly correlates well with the generalization ability of a network. In this light it could be tested numerically in what form the symmetries should be accounted for to give the best estimates of the generalization performance.

The present study has mostly focused on linear networks without hidden units, and the problems of symmetries have not been resolved.

[9] assuming that none of the hidden units have exactly the same weights

5.2.10 Using the Network for Predictions

When using the network to make predictions on examples outside the training data base, the proper Bayesian approach requires that an ensemble of networks, corresponding to the entire posterior weight distribution:

$$y^* = \int F(\mathbf{x}^*, \mathbf{w}) P(\mathbf{w}|\xi, \sigma^2, D, \mathcal{H}) P(\xi, \sigma^2|D, \mathcal{H}) d\xi d\sigma^2 d\mathbf{w} \tag{5.48}$$

where $\mathbf{x}^*$ denotes an input for which we want to predict the output $\mathbf{y}^*$. In the general case of a non linear network this integral is not easily evaluated. One approach to this has been suggested by Neal (Neal, 1993), who samples the integrand using Monte Carlo Methods. Here the integral will not be evaluated, rather the network comprising the maximum posterior weights and scales will be used directly for prediction. This procedure is sometimes termed 'penalized maximum likelihood':

$$y^* \simeq F(\mathbf{x}^*, \mathbf{w}_{\mathbf{MP}}) \tag{5.49}$$

In the case where the posterior distributions of eq. (5.48) are approximated by Gaussians, eq. (5.48) (for reasons of symmetry) becomes equal to eq. (5.49).

5.3 Application to the Sunspot Problem

The Bayesian estimate of the regularization parameter for a linear model was calculated for the sunspot problem.

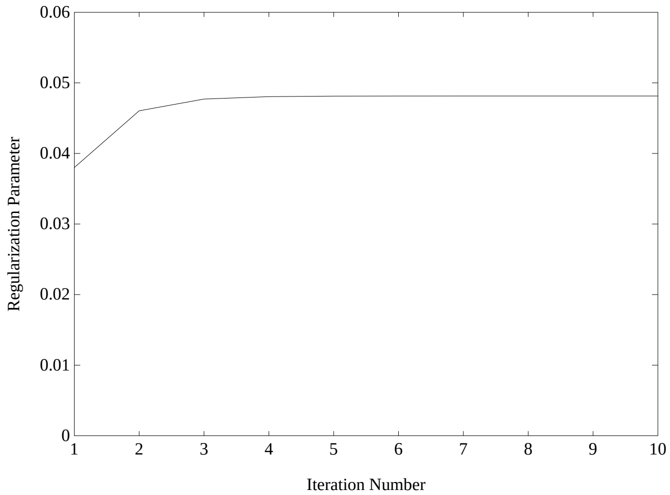

Figure 5.2: Using an iterative procedure to find the optimal setting for the weight decay parameter. The system settles within a few iterations.

The Bayesian framework was applied to the linear model for the sunspot prediction task. A single weight decay parameter was employed. The weights and the regularization parameters were alternately updated. The weights were found according to eq. (5.13), by:

$$w_j = \sum_{j'} \left[(\mathbf{B} + \xi \mathbf{I})^{-1}\right]_{jj'} \sum_{\alpha} x_{j'}^{\alpha} t^{\alpha} \tag{5.50}$$

The regularization parameter was found by iterating equations for the noise and the weight decay. According to eq. (5.27) and (5.33), these are given by:

$$\sigma^2(t) = \frac{2}{p}(\xi(t-1)E_w + E_D) \qquad\qquad \xi(t) = \frac{N\sigma^2(t)}{2E_w + \sigma^2 \mathrm{Tr}(\mathbf{A}^{-1})} \tag{5.51}$$

where t indicates the iteration number. These equations were iterated 10 times to find good values. The first time the iteration scheme was invoked a starting value of $\xi = 0$ was used. On subsequent occasions the previous ξ was used as initial value. The performance of the model was monitored through calculation of the following: FPE estimate, LOO estimate, training error, test errors and the evidence.

The LOO estimate was obtained using the method described in section 2.3.1. The evidence was not calculated on the absolute scale, since no comparison to other models was intended. Rather the following expression for the evidence on an arbitrary scale was adopted from eq. (5.45):

$$\log P(D|\mathcal{H}) = -\frac{p}{2}\log\frac{E_D}{p-\gamma} + \frac{1}{2}\sum_{i=1}^{N}\log\frac{\xi}{\lambda_i + \xi} - \frac{1}{2}\log(pN - \gamma^2) \tag{5.52}$$

where γ is given by eq. (5.29) and λ_i are the eigenvalues of $\mathbf{B}$. Then the least significant weight was located with the OBD algorithm and pruned. The system was then retrained as above and the whole procedure repeated until only a single weight remained.

5.3.1 Individual Weight Decays

The framework so far developed, does not allow for settings of several weight decay parameters. However, the case where there is a separate weight decay parameter for each weight in the network is particularly simple. Referring to eq. (5.32), we calculate the optimal size for individual weight decays as[10]:

$$\begin{aligned} &\frac{\partial}{\partial \xi_j}\log P(D|\xi, \sigma^2, \mathcal{H}) \\ &= \quad -\frac{w_j^2}{2\sigma^2} - \frac{1}{2}(\mathbf{A}^{-1})_{jj} + \frac{1}{2\xi_j} = 0 \Longrightarrow \xi_j = \frac{\sigma^2}{w_j^2 + \sigma^2(\mathbf{A}^{-1})_{jj}} \end{aligned} \tag{5.53}$$

We note the striking similarity with eq. (4.34). A similar experiment to that described in section 4.2.1, was made for the Bayesian scheme. The low pass filter and small constant for numerical stability was applied, eq. (4.35).

The similarity of the equations are reflected in the way the system behaves. Comparing fig. 5.6 with fig. 4.3 we see only negligible difference. The same 4 weights survive, and their final values are actually the same within 3 decimal places.

[10] Using $\frac{\partial}{\partial \xi_i}\log\det(\mathbf{A} + \xi\mathbf{I}) = [(\mathbf{A} + \xi\mathbf{I})^{-1}]_{ii}$

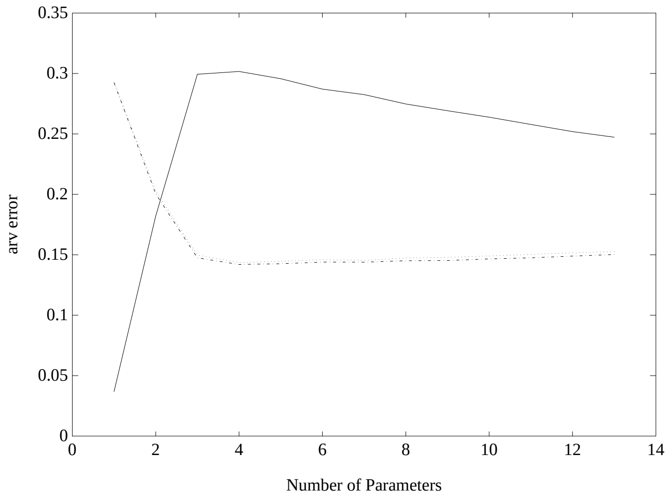

Figure 5.3: The log evidence (on an arbitrary scale) (solid) together with the FPE estimate (dashed) and a LOO estimate (dotted). The network initially had 13 weights and was pruned (right to left) down to a single weight. All three estimators point to the model containing 4 parameters a being optimal. Note how extremely close the FPE and LOO estimates are. Also, note the qualitative similarity between the evidence and the FPE estimate.

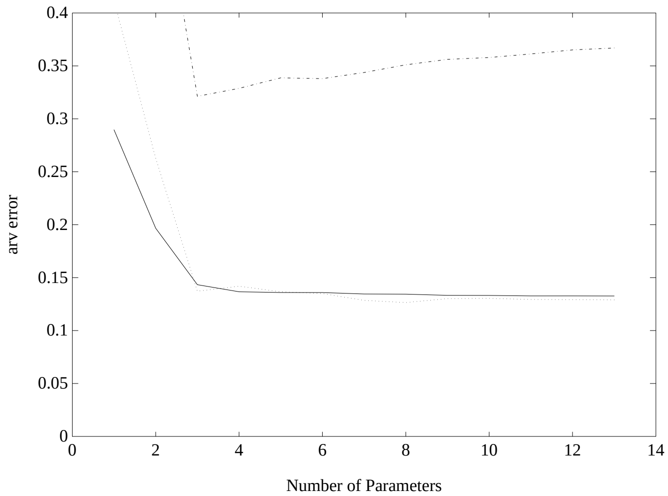

Figure 5.4: The training (solid) and test set errors, test set 1 (dotted) and test set 2 (dash dotted) for the same network as in the previous figure. The test errors have a minimum around 3 or 4 weights. Note, that the error on test set 1 fluctuates quite a bit. The error for test set 2 decreases markedly.

5.3.2 Application to Nonlinear Networks

A simple approximative approach to application of the Bayesian framework to a nonlinear network was tried. An approximation to the evidence was calculated from eq. (5.45), where the eigenvalues of $\mathbf{B}$ for computational convenience, were substituted by the diagonal elements. This approximation for the evidence was monitored during a pruning session

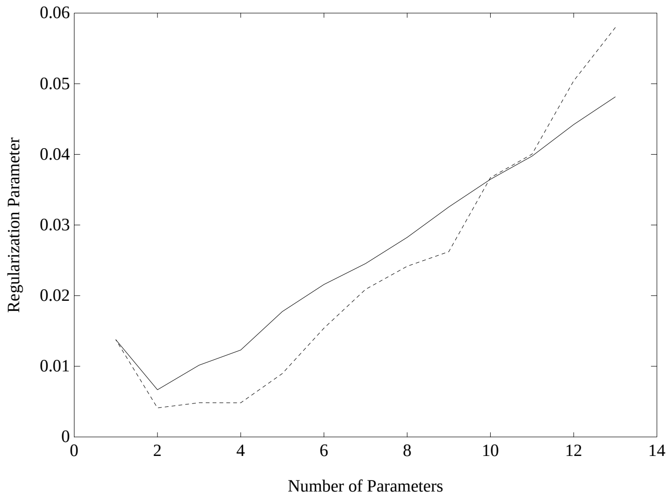

Figure 5.5: The size of the regularization parameters throughout the pruning session. The full line corresponds to Bayes' estimate and the dashed line was obtained with the Generalization method. Note how close the two estimates are.

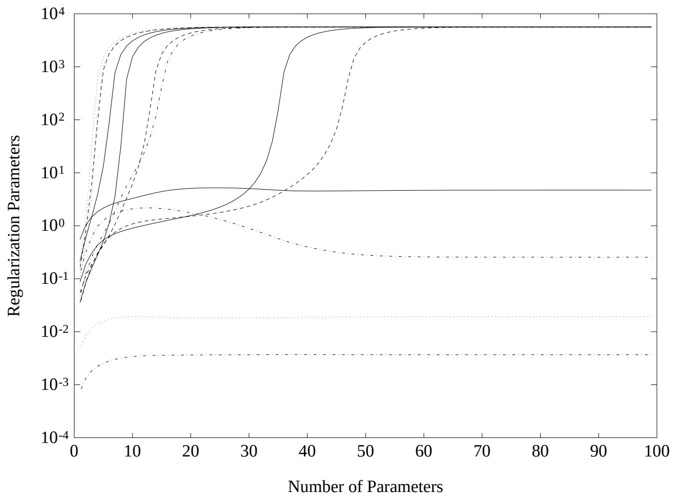

Figure 5.6: The development of the 13 individual weight decays for a linear model for the sunspot problem. Only 4 weights survive.

similar to that described in chapter 3. Unfortunately the evidence pointed to the smallest network (6 parameters) as the best. This network was in fact very poor.

This result is of a very preliminary nature and probably the poor behaviour is due to the very poor estimation of the eigenvalues of $\mathbf{B}$, resulting in severe over estimation of the evidence for large networks. The results from this 'quick and dirty' approach are disregarded. It was only due to time constraints that the application of the Bayesian framework was left at this stage. It will be taken up in the very near future.

All in all the results of the Bayesian scheme are thought to be very encouraging and further work in these lines will most certainly be fruitful.

Chapter 6

Conclusions

It has been shown that several measures are useful in combating overfitting when training feed forward neural network for regression on limited, noisy training sets. These include pruning and the application of weight decay, and in particular the combination of these two has proven specially powerful.

The role of weight decay has been clarified on several points. It has empirically been shown that weight decay is effective in avoiding 'bad' networks. A heuristic explanation of why this is to be expected is given. Further the form and role of weight decay has been derived through a Bayesian approach to modeling.

It has been shown that 'Leave One Out' cross validation is computationally prohibitively costly for nonliner networks. It is shown that adaptation of network architecture during training is feasible, thus reducing the number of necessary network trainings. Further, it is shown that the application of individual weight decays can itself induce pruning of unimportant parameters.

Numerical tests on the sunspot problem have confirmed the utility of the two methods for linear models. For nonlinear models further simulations are needed. The frameworks should be applied to a variety of problems in order to make a statistically sound evaluation of their performance.

It was seen that the solutions obtained be the two models were on many points strikingly similar. This strongly suggests that, in least in the Gaussian approximation used here, there exists a connection between the FPE estimate and the evidence. At this stage it is not entierly clear exactly what this connection is.

The numerical simulations suggest that both the methods presented here constitute effective and efficient means by which models with good generalization can be trained and selected.

Appendix A

Pruning from Adaptive Regularization

This appendix contains the draft for a paper entitled: 'Pruning from Adaptive Regularization', by L.K. Hansen and C.E. Rasmussen. It has been sent to *Neural Computation*.

Pruning from Adaptive Regularization

Lars Kai Hansen and Carl Edward Rasmussen
CONNECT, Electronics Institute
Technical University of Denmark, B 349
DK-2800 Lyngby. Denmark
Emails: lars,ed@eiffel.ei.dth.dk

August 12, 1993

Abstract

Inspired by the recent upsurge of interest in Bayesian methods we consider adaptive regularization. A generalization based scheme for adaptation of regularization parameters is introduced and compared to Bayesian regularization. We show that pruning arises naturally within both adaptive regularization schemes. As model example we have chosen the simplest possible: estimating the mean of a random variable with known variance. Marked similarities are found between the two methods in that they both involve a "noise limit", below which they regularize with infinite weight decay, *i.e.* they *prune*. However, pruning is not always beneficial. We show explicitly that both methods in some cases may increase the generalization error. This corresponds to situations where the underlying assumptions of the regularizer are poorly matched to the environment.

1 Introduction

We believe in Ockham's Razor: the generalization (or test) error of estimated parameters is decreased by constraining capacity to a minimum needed for capturing the rule, see *e.g.* (Thodberg, 1991; Solla, 1992). However, the *proviso* may be hard to implement, in particular for non-linear noisy systems where the rule is ill-defined.

Pruning is a popular tool for reducing model capacity and pruning schemes have been successfully applied to layered neural networks (Le Cun et al., 1990; Thodberg, 1991; Svarer et al., 1993). While pruning is a discrete decision process, regularization introduces *soft* constraints such as weight decay (Moody, 1991). A common feature of these techniques is the need for control parameters: stop criteria for pruning and weight decays for regularization. In (Svarer et al., 1993) a statistical stop criterion was developed for pruning of networks for regression problems.

Recently, MacKay reviewed a Bayesian approach to adaptive regularization in the context of neural networks, demonstrating that the *evidence* based method can improve the generalization properties and he compared it to cross validation (MacKay, 1992a; MacKay, 1992b). Cross validation is known to be rather noisy, hence methods based on statistical arguments are recommended, see *e.g.* (Akaike, 1969; Moody, 1991; Hansen, 1993). In this presentation we define such an alternative approach for adaptive regularization, and we test it on a case of ultimate simplicity, namely that of estimating the mean of a Gaussian variable of known variance, a case for which detailed insight can be obtained.

In the course of comparing the two schemes, we have discovered a new feature of adaptive regularization: we find that both approaches involve a "noise limit", below which they regularize with infinite weight decay, *i.e.* they *prune*. This finding unifies the pruning and regularization approaches to capacity control.

2 Specification of the toy problem

The problem is defined as follows: consider a student-teacher setup based on a *teacher* (with parameter $\tilde{w}$) providing N examples of the form:

$$y_m = \tilde{w} + \nu_m \qquad \nu_m \in \mathcal{N}(0, \sigma^2) \qquad m = 1, \ldots, N \tag{1}$$

where the normal noise contributions are independent, have zero mean and common known variance σ^2. Based on the training data set, $D = \{y_m | m = 1, \ldots, N\}$, the task of the *student* (with parameter w) is to infer the mean: $\tilde{w}$. The measure of success for the student is the generalization error:

$$E_{\mathrm{G}}(w) = \int d\nu P(\nu)(\tilde{w} + \nu - w)^2 = (\tilde{w} - w)^2 + \sigma^2 \tag{2}$$

This is the error on a random new test example for the specific student weight as estimated on the given training set. In evaluating an estimation scheme our measure will be the *average generalization error*, obtained by averaging over all possible training sets of the given size N.

In the next three sections we consider three such estimators: the usual "empirical mean" obtained from maximum likelihood, an estimator based on MacKay's maximum evidence, and a novel estimator based on explicit minimization of the expected generalization error. In section 6 we compute the average generalization error of each of the schemes, as a function of the teacher parameter.

3 Maximum likelihood estimation

The likelihood of the student parameter associated with the training set, D, is:

$$P(D|w) = \prod_{m=1}^{N} \frac{1}{\sqrt{2\pi\sigma^2}} \exp\left(-\frac{1}{2\sigma^2}(y_m - w)^2\right) \tag{3}$$

Maximizing this w.r.t. w, produces the standard maximum likelihood estimator, the *empirical mean*:

$$w_{\mathrm{ML}}(D) = \frac{1}{N}\sum_{m=1}^{N} y_m \tag{4}$$

This estimator is *unbiased*, *i.e.* the average value, averaging over all possible training sets, is the true mean $\tilde{w}$.

For a specific training set the generalization error is obtained using (2). However, as mentioned above, we are not so much interested in the value for the particular training set. Rather we are interested in the expected error, obtained by averaging over all possible training sets of size N. To compute this quantity we note that $w_{\mathrm{ML}} \in \mathcal{N}(\tilde{w}, \sigma^2/N)$, hence the average generalization error is:

$$\langle E_{\mathrm{G}}(w_{\mathrm{ML}})\rangle_N = \langle(\tilde{w} - w_{\mathrm{ML}})^2\rangle + \sigma^2 = \frac{\sigma^2}{N} + \sigma^2 \tag{5}$$

The first term is the average *excess error* made by the student due to the finite training set. The second term is due to the imperfectness of the data; there is noise in the test examples used for grading the student. The minimal error approached for large training sets is simply the noise level.

4 Bayesian regularization

A Bayesian scheme, recently suggested to the neural net community by MacKay, adopts two levels of inference. The first level of inference consists of estimating the teacher parameters from the data, conditioned on certain parameterized assumptions (*i.e.* the *prior* distribution of the teacher mean value). The second level of inference, consists of estimating the parameters of the prior, with the purpose of maximizing the *evidence* of the data.

The first level starts from the likelihood. Evidently, it would be intuitively more appealing to use the parameter which maximizes the direct probability $P(w|D)$ of the parameter conditioned on the data. This distribution can be obtained from Bayes' rule:

$$P(w|D) = \frac{P(D|w)P(w)}{P(D)} \tag{6}$$

We follow MacKay and employ a parameterized prior: $P(w) \equiv P(w|\alpha)$, with a parameter α playing the role of a "weight decay", to be determined at the second level of inference. The prior takes the form:

$$P(w|\alpha) = \sqrt{\frac{\alpha}{2\pi}} \exp\left(-\frac{1}{2}\alpha w^2\right) \tag{7}$$

and using the likehood from eq. (3), we arrive at the *posterior* distribution:

$$P(w|D,\alpha) = P(D|\alpha)^{-1} \sqrt{\frac{\alpha}{2\pi}} \exp\left(-\frac{1}{2}\alpha w^2\right) \prod_{m=1}^{N} \frac{1}{\sqrt{2\pi\sigma^2}} \exp\left(-\frac{1}{2\sigma^2}(y_m - w)^2\right) \tag{8}$$

To find the maximum posterior estimate of the mean, we maximize $P(w|D,\alpha)$ w.r.t. w, to get:

$$w_{\mathrm{MP}}(D,\alpha) = \frac{N}{N + \alpha\sigma^2} w_{\mathrm{ML}}(D) \tag{9}$$

In the following we will suppress the explicit dependence on the training data.

The second level of inference is to estimate the regularization parameter, and we do this by again invoking Bayes' rule:

$$P(\alpha|D) = \frac{P(D|\alpha)P(\alpha)}{P(D)} \tag{10}$$

We assume the prior on α to be flat, *i.e.* independent of α. Hence, the most probable regularization is obtained by maximizing the likelihood of it. MacKay dubbed this quantity the *evidence*:

$$\begin{aligned} P(D|\alpha) &= \int dw P(D|w,\alpha)P(w|\alpha) & (11)\\ &= \int_{-\infty}^{\infty} dw \sqrt{\frac{\alpha}{2\pi}} \exp\left(-\frac{1}{2}\alpha w^2\right) \prod_{m=1}^{N} \frac{1}{\sqrt{2\pi\sigma^2}} \exp\left(-\frac{1}{2\sigma^2}(y_m - w)^2\right) & \\ \Longrightarrow \log P(D|\alpha) &= \frac{1}{2}\log(\alpha/2\pi) - \frac{1}{2}\log\left(\frac{N + \alpha\sigma^2}{2\sigma^2}\right) + \frac{(N w_{\mathrm{ML}})^2}{2\sigma^2(N + \alpha\sigma^2)} + \text{const.} & (12) \end{aligned}$$

where we have lumped all terms not depending on α in the constant. Maximizing the evidence w.r.t. α,

$$\alpha = \begin{cases} \dfrac{1}{w_{\mathrm{ML}}^2 - \sigma^2/N} & w_{\mathrm{ML}}^2 > \sigma^2/N \\ \infty & w_{\mathrm{ML}}^2 \leq \sigma^2/N \end{cases} \tag{13}$$

which may be substituted into the expression for w_{MP} to find:

$$w_{\mathrm{MP}} = \begin{cases} (1 - \dfrac{\sigma^2}{N w_{\mathrm{ML}}^2}) w_{\mathrm{ML}} & w_{\mathrm{ML}}^2 > \sigma^2/N \\ 0 & w_{\mathrm{ML}}^2 \leq \sigma^2/N \end{cases} \tag{14}$$

Hence there is a sharp "noise limit", below which the scheme tells us to "prune", that is to regularize with infinite strength. The noise limit has a straight forward interpretation: prune the parameter if the signal to noise ratio is less than unity. Note that the estimator is biased: it does not have the correct mean value. It is *consistent* since $w_{\mathrm{MP}} \rightarrow \tilde{w}$, for large samples, while the noise limit shrinks to zero. To illustrate the power of the method we compute in section 6 the (training set) ensemble average of the generalization error.

The above derivation uses the *evidence approximation* as in (MacKay, 1992a). In this approximation a single value for α is used to infer the value of w. In an "ideal Bayesian" scheme, we should instead integrate over the posterior distribution of α (Neal, 1993). The ideal Bayesian scheme will not be considered here, but it also shows the pruning effect[1].

5 Generalization error minimization

While the evidence itself is an interesting statistical quantity, it is not obvious what the relation between maximizing the evidence and minimizing test error is (MacKay, 1992a). Since the latter often is the basic modeling objective, we propose here to base the optimization of α on the expected generalization error, cf. eq. (2).

In order to be able to compare directly with the above, we use a regularized *least squares* procedure to find the student weight:

$$E_{\mathrm{T}} = \frac{1}{2\sigma^2} \sum_{m=1}^{N} (y_m - w)^2 + \frac{1}{2} \alpha w^2 \tag{15}$$

Minimizing w.r.t. w, we recover (9):

[1]Since the distribution of $P(\alpha|D)$ for small w_{ML} aquires a significant "tail", the resulting $w_{\mathrm{MP}} \rightarrow 0$ at a finite value of w_{ML}.

$$w_{\mathrm{G}} = \frac{N}{N+\alpha\sigma^2} w_{\mathrm{ML}} \tag{16}$$

Our aim is to minimize the average generalization error (averaged again w.r.t. training sets). To this end we note that the distribution of $w_{\mathrm{G}}(D,\alpha)$ can be computed using simple manipulations of random variables. Noting again that $w_{\mathrm{ML}} \in \mathcal{N}(\tilde{w}, \sigma^2/N)$ we have $w_{\mathrm{G}} \in \mathcal{N}\left(\frac{N}{N+\alpha\sigma^2}\tilde{w}, \frac{N}{(N+\alpha\sigma^2)^2}\sigma^2\right)$[2]. Consequently the averaged generalization error becomes:

$$\langle E_{\mathrm{G}}(w_{\mathrm{G}})\rangle_N = \left(\frac{\alpha\sigma^2}{N+\alpha\sigma^2}\tilde{w}\right)^2 + \left(1 + \frac{N}{(N+\alpha\sigma^2)^2}\right)\sigma^2 \tag{17}$$

You might be uncomfortable with the appearance of the unknown teacher parameter $\tilde{w}$, however, it will shortly be replaced by an estimate based on the data. Minimizing the generalization error (17) w.r.t. α, we find the simple expression:

$$\alpha = \frac{1}{\tilde{w}^2} \tag{18}$$

which is inserted in eq. (16), to obtain:

$$w_{\mathrm{G}} = \frac{\tilde{w}^2}{\tilde{w}^2 + \sigma^2/N} w_{\mathrm{ML}} \tag{19}$$

To proceed we need to insert an estimate of the teacher parameter $\tilde{w}$; we will consider two cases. Firstly we try setting $\tilde{w} = w_{\mathrm{ML}}$ in eq. (19), hence:

$$w_{\mathrm{G1}} = \frac{w_{\mathrm{ML}}^2}{w_{\mathrm{ML}}^2 + \sigma^2/N} w_{\mathrm{ML}} \tag{20}$$

This estimator is biased, but consistent and it doesn't call for pruning even if the data are very noisy.

Secondly, being more brave, we could argue that the best estimate we have of $\tilde{w}$ is w_{G} and accordingly set $\tilde{w} = w_{\mathrm{G2}}$ in eq. (19), to obtain a self consistent equation[3] for w_{G2}:

[2] Using $(\xi = a + b\psi) \wedge (\psi \in \mathcal{N}(c, d^2)) \Rightarrow \xi \in \mathcal{N}(a + bc, b^2 d^2)$

[3] Interestingly, this equation is recovered from the Bayesian approach if we optimize w and α simultaneously (ie. if we don't separate inference in two distinct levels). In this case Bayes' rule becomes:

$$P(w,\alpha|D) = \frac{P(D|w,\alpha)P(w|\alpha)P(\alpha)}{P(D)}$$

$$w_{\mathrm{G2}} = \frac{w_{\mathrm{G2}}^2}{w_{\mathrm{G2}}^2 + \sigma^2/N} w_{\mathrm{ML}} \tag{21}$$

By substituting $\tilde{w} = w_{\mathrm{G2}}$, the student is not operating with the true cost function (17), but with a modified, self consistent, cost function depicted in figure 1. We note that by this substitution, a potentially dangerous global minimum is created at $w_{\mathrm{G2}} = 0$.

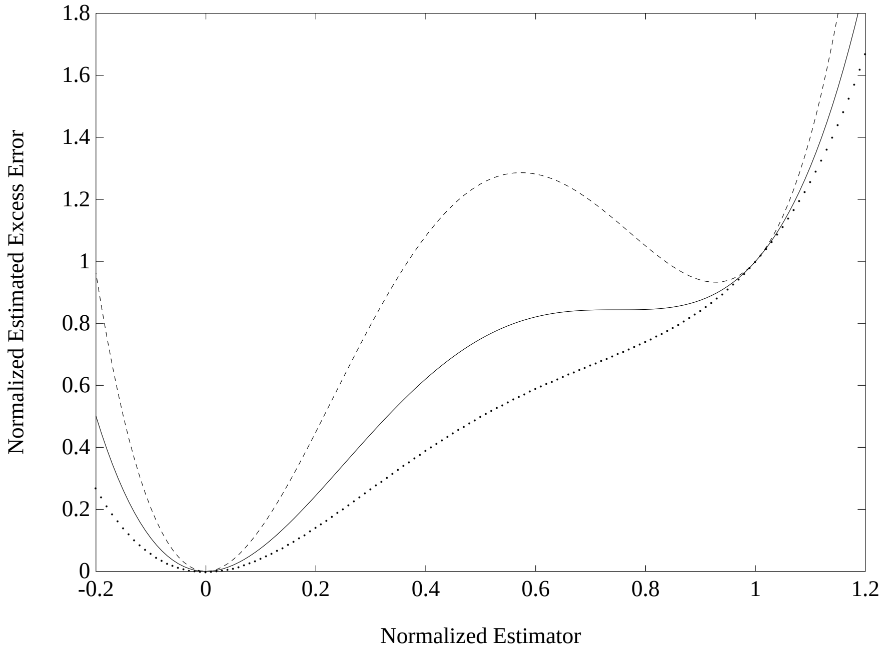

Figure 1: The estimated average excess generalization error as function of normalized estimator. The graphs are obtained by introducing (16) in (17), and setting $\tilde{w} = w_{\mathrm{G}}$. The estimator is scaled by w_{ML}, and the estimated excess error is scaled by σ^2/N. The three graphs correspond to different noise levels (dashed: low, solid: critical & dotted: high). Note that if the noise is low, a local minimum is found, otherwise pruning takes place.

Like MacKay, we could envision eq. (21) solved by iteration, *e.g.* starting from $w_{\mathrm{G2}}^{(0)} = w_{\mathrm{ML}}$. Iterating the function on the right hand side we see that, besides the solution $w_{\mathrm{G2}} = 0$, there may be two more fixed points (or solutions) depending on the parameters. Analyzing the iteration scheme for this simple case is straightforward, one of the possible fixed points is stable, the other unstable. If it exists, the stable fixed point (corresponding to the local minimum in figure 1) is found by the iteration scheme, and it is given by:

$$w_{\mathrm{G2}} = \begin{cases} \dfrac{w_{\mathrm{ML}}}{2}\left(1 + \sqrt{1 - 4\sigma^2/N w_{\mathrm{ML}}^2}\right) & w_{\mathrm{ML}}^2 > 4\sigma^2/N \\ 0 & w_{\mathrm{ML}}^2 \leq 4\sigma^2/N \end{cases} \tag{22}$$

Note that the noise limits are different for the generalization error method and Bayes' method. The generalization error method is more conservative than Bayes' (if we associate conservatism with setting $w = 0$). The noise limit is a factor 4 larger, meaning that pruning or super-regularization will happen more often. Note also that this estimator, like the Bayesian, is biased but consistent.

6 Comparing the regularization schemes

As discussed above, the quantity to compare is the average (w.r.t. training sets) generalization error of the estimators. Since each estimator is a function of the empirical mean, w_{ML}, only, the average involves averaging w.r.t. the distribution of this quantity and we already noted that $w_{\mathrm{ML}} \in \mathcal{N}(\tilde{w}, \sigma^2/N)$. The generalization error of the maximum likelihood estimator itself was given in (5), it is independent of the teacher magnitude.

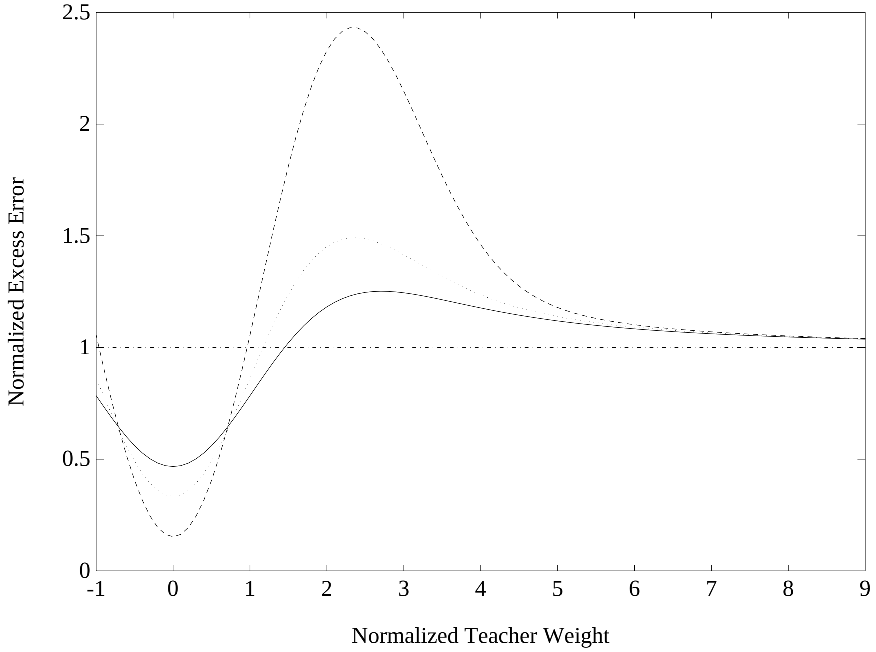

Figure 2: Average excess generalization error as function of teacher magnitude for the two adaptive regularization schemes used to estimate the mean of a Gaussian variable (Dash-dotted: Maximum Likelihood, Dotted: Bayes, solid: Generalization method 1 & dashed: Generalization method 2). The teacher weight has been scaled by $\sqrt{\sigma^2/N}$, and the excess error, $\Delta E = E - \sigma^2$, is scaled by the excess error of the maximum likelihood estimator (σ^2/N).

Since the regularized estimators only depend on the training set value of w_{ML}, the average excess generalization errors ($\Delta E(\tilde{w}) = E(\tilde{w}) - \sigma^2$) are functions of $\tilde{w}/\sqrt{\sigma^2/N}$:

$$\Delta E(\tilde{w}) = \frac{1}{\sqrt{2\pi\sigma^2/N}} \int_{-\infty}^{\infty} dw_{\mathrm{ML}} \exp\left(-\frac{(w_{\mathrm{ML}} - \tilde{w})^2}{2\sigma^2/N}\right) (\tilde{w} - w_{\mathrm{E}}(w_{\mathrm{ML}}))^2 \tag{23}$$

where w_{E} is either of the three biased estimators. The integral is easily evaluated numerically. In figure 2 we picture these functions, and we note that adaptation of the regularization parameter leads to a decreased error *only if the teacher is small.* If the teacher is very large the adaptive schemes lead to a negligible increase in error as they indeed should. However, if the teacher is of *intermediate* size all schemes produce seriously increased errors. The reason for this increased error is simply that the adaptive schemes are too conservative, there is a certain probability even for a unit magnitude teacher that the scheme will prune, hence introduce a large error.

We conclude that the benefit of the adaptive schemes depends critically on the distribution of teachers, *i.e.* the extent to which the domain lives up to the assumptions of the regularizer. In fact distributions of teachers can be given for which each of the estimation schemes would be preferred. We note that for a teacher distribution that has significant mass at small teachers like the $P_0(\tilde{w}) \sim 1/|\tilde{w}|$ distribution corresponding to the prior of the Bayesian scheme (Buntine and Weigend, 1991) the *brave generalization student* is superior for the present problem. This fact is most easily appreciated by noting that integrating the functions in figure 2 w.r.t. $P_0(\tilde{w})$ corresponds to integrating the functions on a logarithmic ordinate axis with a uniform measure, hence stretching the region around $|\tilde{w}| \sim 0$ to infinity. We iterate: this is no big surprise since that student precisely minimizes the generalization error with the correct (implicit) prior.

Our experience with adaptive regularization is generally positive. A detailed account of multivariate applications, where pruning is achieved by utilizing individual weight regularization will be given elsewhere (Rasmussen and Hansen, 1993).

In conclusion we have shown that the use of adaptive regularization for the simple task of estimating the mean of a Gaussian variable of known variance, leads to non-trivial decision processes. We have defined a competitive scheme for identification of the regularization parameter, based on minimalization of the expected generalization error. We have shown that pruning can be viewed as infinite regularization, and that this is a natural consequence of adaptive regularization.

7 Acknowledgements

We thank the participants in the 1992 CONNECT Journal Club for early discussions on Bayesian methods. We thank Anders Krogh, Jan Larsen, David MacKay, Claus Svarer, and Hans Henrik Thodberg for valuable discussions on this work. This research is supported by the Danish Research Councils for the Natural and Technical Sciences through the Danish Computational Neural Network Center (CONNECT).

References

Akaike, H. (1969). Fitting autoregressive models for prediction. *Ann. Inst. Stat. Mat.*, 21:243–247.

Buntine, W. and Weigend, A. (1991). Bayesian back-propagation. *Complex Systems*, 5:603–643.

Hansen, L. (1993). Stochastic linear learning: Exact test and training error averages. *Neural Networks*, 6:393–396.

Le Cun, Y., Denker, J., and Solla, S. (1990). Optimal brain damage. In Touretzky, D., editor, *Advances in Neural Information Processing Systems*, volume 2, pages 598–605, San Mateo. (Denver 1989), Morgan Kaufmann.

MacKay, D. (1992a). Bayesian interpolation. *Neural Computation*, 4:415–447.

MacKay, D. (1992b). A practical framework for backpropagation networks. *Neural Computation*, 4:448–472.

Moody, J. (1991). Note on generalization, regularization and architecture selection in nonlinear systems. In Kung, S., Juang, B., and Kamm, C., editors, *IEEE International Conference on Neural Networks*, pages 1–10. IEEE.

Neal, R. (1993). Bayesian learning via stochastic dynamics. In Giles, C., Hanson, S., and Cowan, J., editors, *Neural Information Processing Systems 5*, San Mateo. Morgan Kaufmann.

Rasmussen, C. and Hansen, L. (1993). *In preparation.*

Solla, S. (1992). Capacity control in classsifiers for pattern recognizers. In Kung, S.Y. et al., editor, *IEEE International Conference on Neural Networks*, pages 255–266, Piscataway, NJ. IEEE.

Svarer, C., Hansen, L., and Larsen, J. (1993). On design and evaluation of tapped-delay neural network architectures. In Berenji, H.R. et al., editor, *IEEE International Conference on Neural Networks*, volume 1, pages 46–51, Piscataway, NJ. (San Francisco 1993), IEEE.

Thodberg, H. (1991). Improving generalization on neural networks through pruning. *International Journal of Neural Systems*, 1:317–326.

Appendix B

Designer Networks for Time Series Processing

This appendix contains a paper entitled: 'Designer Networks for Time Series Processing', by C. Svarer, L.K. Hansen, J. Larsen and C.E. Rasmussen. It has been accepted for presentation at the 1993 IEEE-SP Workshop on Neural Networks for Signal Processing, September 7–9 1993, Baltimore, Maryland, USA.

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

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

where p_{set} is the number of examples in the data (train or test) set in question, and σ_{total}^2 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.

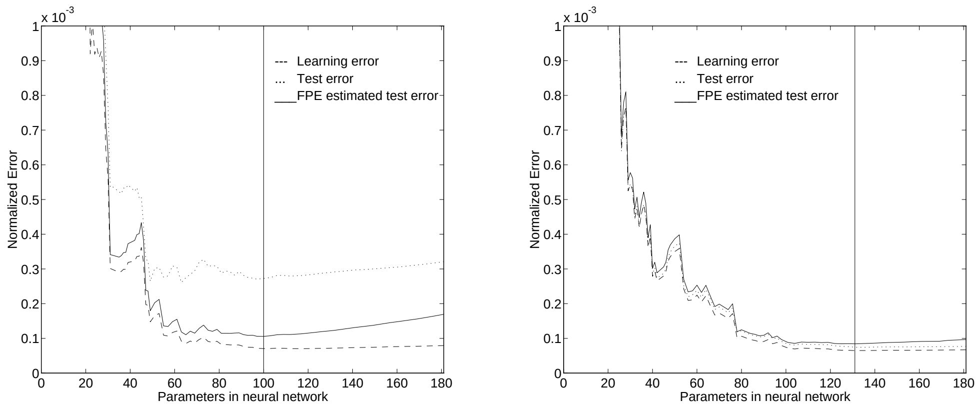

Figure 1: The evolution of training and test errors during pruning for the Mackey-Glass time series (left) training set size 500, (right) training set size 1000. The FPE estimate of the test error is based on eq. (9). The vertical line indicates the network for which the *estimated* test error is minimal. 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

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

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

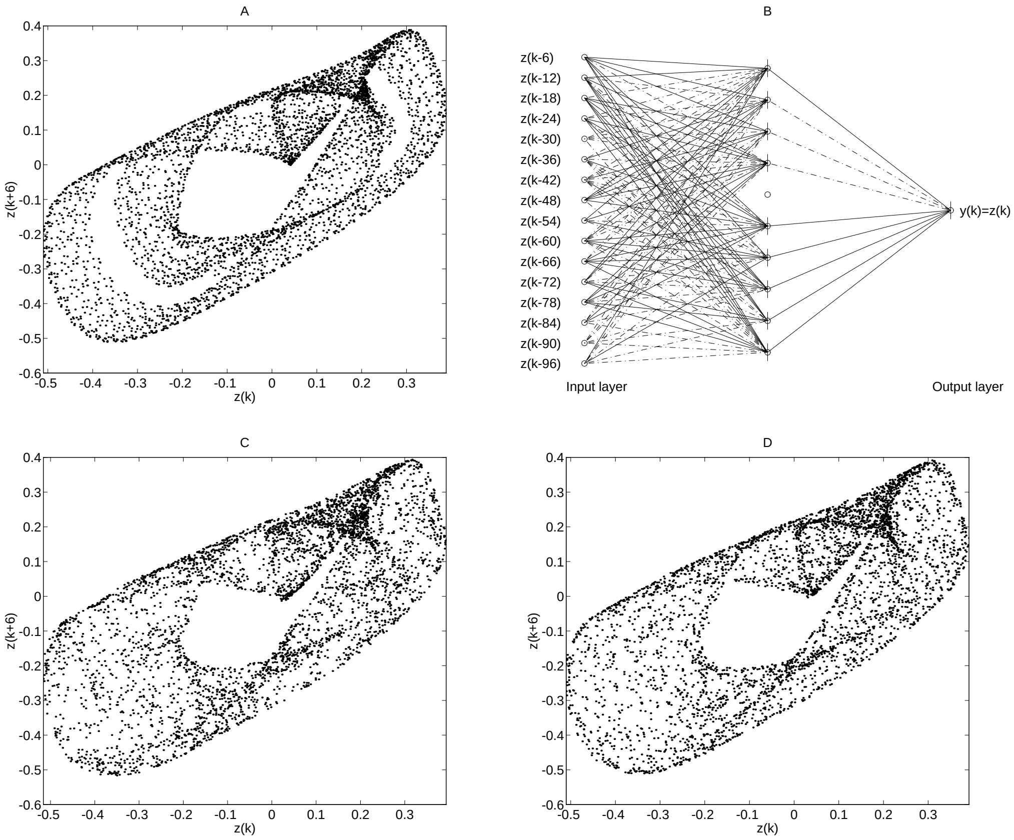

Figure 2: Phase-state plot ($z(k)$ versus $z(k+6)$) of a) The Mackey-Glass attractor. The correlation coefficient for two independent samples of 8850 points is 0.991. b) The pruned network obtained with a training set of 1000 examples. c) The estimated attractor for fully connected network trained on 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

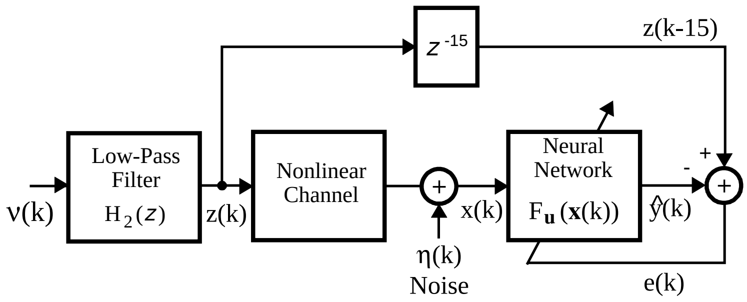

Figure 3: Channel equalization. The non-linear channel is driven by a low-pass filtered white noise signal $z(k)$. Output from the channel is denoted $x(k)$. The neural network is trained to reproduce the channel input $z(k)$ based on a sample of the noisy output $x(k)$. The variance of the channel Gaussian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

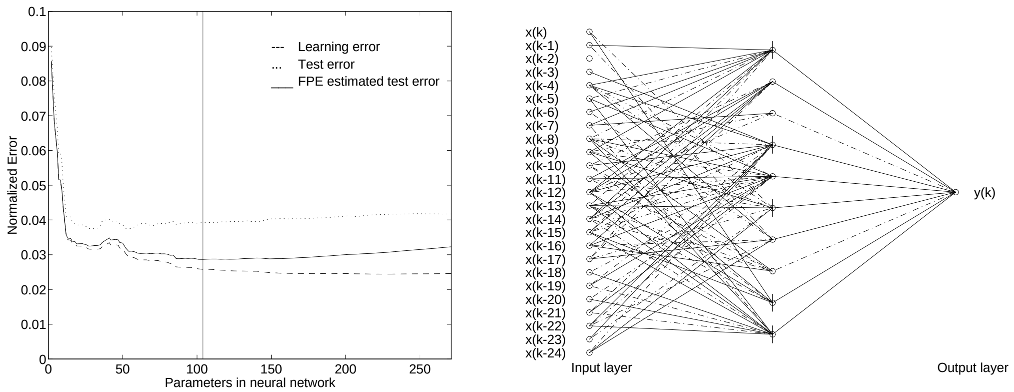

Figure 4: The left panel shows the evolution of training and test errors during pruning for channel equalization. The FPE estimate of the test error is based on eq. (9). The vertical line indicates the network for which the *estimated* test error is minimal. In the right panel we show the resulting, 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.

Bibliography

Akaike, H. (1969). Fitting autoregressive models for prediction. *Ann. Inst. Stat. Mat.*, 21:243–247.

Anderson, J. and Rosenfeld, E., editors (1988). *Neurocomputing: Foundations of Research.* MIT Press, Cambridge.

Barton, S. (1990). A matrix method for optimizing a neural network. *Neural Computation*, 3:450–459.

Brunak, S. and Lautrup, B. (1989). Liniedeling med et neuralt neværk. *Skrifter for Anvendt Matematik og Lingvistik*, 14:55–74.

Cybenko, G. (1989). Approximation by superpositions of a sigmoidal function. *Mathematics of Control, Signals, and Systems*, 2:303–314.

Eddy, J. (1977). The case of the missing sunspots. *Scientific American*, 236:80–92.

Fahlman, S. and Lebiere, C. (1990). The cascade-correlation learning architecture. In Touretzky, D., editor, *Advances in Neural Information Processing Systems*, volume 2, pages 524–532, San Mateo. (Denver 1989), Morgan Kaufmann.

Gorodkin, J., Hansen, L., Krogh, A., Svarer, C., and Winther, O. (1993). A quantitative study of pruning by optimal brain damage. *Preprint, Electronics Institute.*

Hansen, L. (1993). Notes on generalization. *Unpublished handwritten notes.*

Hassibi, B., Stork, D., and Wolff, G. (1992). Optimal brain surgeon and general network pruning. *Neural Computation*, 4:1–8.

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

Le Cun, Y., Boser, B., Denker, J., Henderson, D., Howard, R., Hubbard, W., and Jackel, L. (1989). Backpropagation applied to handwritten zip code recognition. *Neural Computation*, 1:541–551.

Le Cun, Y., Denker, J., and Solla, S. (1990). Optimal brain damage. In Touretzky, D., editor, *Advances in Neural Information Processing Systems*, volume 2, pages 598–605, San Mateo. (Denver 1989), Morgan Kaufmann.

MacKay, D. (1992a). Bayesian interpolation. *Neural Computation*, 4:415–447.

MacKay, D. (1992b). The evidence framework applied to classification. *Neural Computation*, 4:720–736.

MacKay, D. (1992c). A practical framework for backpropagation networks. *Neural Computation*, 4:448–472.

McCulloch, W. and Pitts, W. (1943). A logical calculus of ideas immanent in nervous activity. *Bulletin of Mathematical Biophysics*, 5. Reprinted in (Anderson and Rosenfeld, 1988).

Minsky, M. and Papert, S. (1969). *Perceptrons.* MIT Press, Cambridge. Partially reprinted in (Anderson and Rosenfeld, 1988).

Neal, R. (1993). Bayesian learning via stochastic dynamics. In Giles, C., Hanson, S., and Cowan, J., editors, *Neural Information Processing Systems 5*, San Mateo. Morgan Kaufmann.

Pallesen, L. (1979). *Bayesiansk Inferens.* IMSOR, DtH, Lyngby.

Rumelhart, D. (1988). Learning and generalization. In *IEEE International Conference on Neural Networks.* (San Diego 1988), IEEE.

Rumelhart, D., Hinton, G., and Williams, R. (1986a). Learning internal representations by error propagation. In Rumelhart, D. and McClelland, J., editors, *Parallel Distributed Processing*, volume 1, chapter 8, pages 318–362. MIT Press, Cambridge. Reprinted in (Anderson and Rosenfeld, 1988).

Rumelhart, D., Hinton, G., and Williams, R. (1986b). Learning representations by back-propagating errors. *Nature*, 323:533–536. Reprinted in (Anderson and Rosenfeld, 1988).

Sejnowski, T. and Rosenberg, C. (1987). Parallel networks that learn to pronounce english text. *Complex Systems*, 1:145–168.

Svarer, C., Hansen, L., and Larsen, J. (1993). On design and evaluation of tapped-delay neural network architectures. In Berenji, H.R. et al., editor, *IEEE International Conference on Neural Networks*, volume 1, pages 46–51, Piscataway, NJ. (San Francisco 1993), IEEE.

Thodberg, H. (1993). Ace of bayes: Application of neural networks with pruning. *In preparation.*

Toussaint, G. (1974). Bibliography on estimation of misclassification. *IEEE trans. Inform. Theory.*, IT-20:472–479.

Weigend, A., Huberman, B., and Rumelhart, D. (1990). Predicting the future: a connectionist approach. *International Journal of Neural Systems*, 1:193–209.