Neural Computation, Vol. 9, No. 1, pp. 185–204

A penalty-function approach for pruning feedforward neural networks

Rudy Setiono

Department of Information Systems and Computer Science

National University of Singapore

Kent Ridge, Singapore 119260

Republic of Singapore

Email:rudys@iscs.nus.sg

Abstract

This paper proposes the use of a penalty function for pruning feedforward neural network by weight elimination. The penalty function proposed consists of two terms; the first term is to discourage the use of unnecessary connections and the second term is to prevent the weights of the connections from taking excessively large values. Simple criteria for eliminating weights from the network are also given. The effectiveness of this penalty function is tested on three well known problems. These test problems are the contiguity problem, the parity problems, and the monks problems. The resulting pruned networks obtained for many of these problems have fewer connections than previously reported in the literature.

1 Introduction

We are concerned in this paper with finding a minimal feedforward backpropagation neural network for solving the problem of distinguishing patterns from two or more sets in n-dimensional space. Backpropagation feedforward neural networks have been gaining acceptance in recent years as the method of choice for this pattern classification problem. One of the difficulty with using feedforward neural networks is the need to determine the optimal number of hidden units before the training process can begin. Too many hidden units may lead to overfitting of the data and poor generalization, while too few hidden units may not give a network that learns the data. Two different approaches have been proposed to overcome the problem of determining the optimal number of hidden units required by an artificial neural network to solve a given problem. The first approach begins with a minimal network and adds more hidden units only when they are needed to improve the learning capability of the network. The second approach begins with an oversized network and then prunes redundant hidden units.

Much research have been done on algorithms that start with a minimal network and dynamically construct the final network. These algorithms include the dynamic node creation (Ash 1989), the cascade correlation algorithm (Fahlman and Lebeire 1990), the tiling algorithm (Mezard and Nadal 1989), the self-organizing neural network (Tenorio and Lee 1990), and the upstart algorithm (Frean 1990). The aim of these algorithms is to find a network with the simplest architecture possible that is capable of correctly classifying all its input patterns.

Interests in algorithms that remove hidden units from an oversized network have also been growing (Chauvin 1989; Chung and Lee 1992; Mozer and Smolensky 1989; Hanson and Pratt 1989). Instead of removing unnecessary hidden units, individual weights (connections) in the network may also be removed to increase the generalization capability of a neural network (Le Cun *et al.* 1990; Hassibi and Stork 1993; Thodberg 1991). Typically, methods for removing weights from the network involve adding a penalty term to the error function (Hertz *et al.* 1991). It is hoped that by adding a penalty term to the error function, unnecessary connections will have small weights, and therefore the complexity of the network can be reduced significantly by pruning. The simplest and most commonly used penalty term is the sum of the squared weights. After the training process has been completed, i.e., it has converged to a local minimum point, a measure of the "saliency" of each connection in the network is computed. The saliency measure gives an indication on the expected increase in the error function after the corresponding connection is eliminated from the network. In the pruning methods Optimal Brain Damage (Le Cun *et al.* 1990) and Optimal Brain Surgeon (Hassibi and Stork 1993), the saliency of each connection is computed using a second-order approximation of the error function near a local minimum. If the saliency of a connection is below a certain threshold, then the connection is removed from the network. The resulting network needs to be retrained if the increase in the error function is larger than a predetermined acceptable increase.

In this paper, a simple scheme for removing redundant weights from a network which has been trained to minimize a penalty function is proposed. The effectiveness of the proposed pruning scheme is tested on several well known problems such as the contiguity problem, the monks problems, and the parity problems. In many cases, the resulting networks contain fewer connections than previously reported in the literature.

The organization of this paper is as follows. In Section 2, we describe our penalty function and network-pruning scheme. Experimental results are reported in Section 3. Finally in Section 4, a brief discussion and final remarks are given.

2 Neural network training and pruning

2.1 Neural network training

Let us consider the fully connected three-layer network depicted in Figure 1. The error function associated with this network is normally defined as the sum of the squared errors:

$$E(w,v) = \frac{1}{2} \sum_{i=1}^{k} \sum_{p=1}^{C} (S_{pi} - t_{pi})^2 \, , \tag{2.1}$$

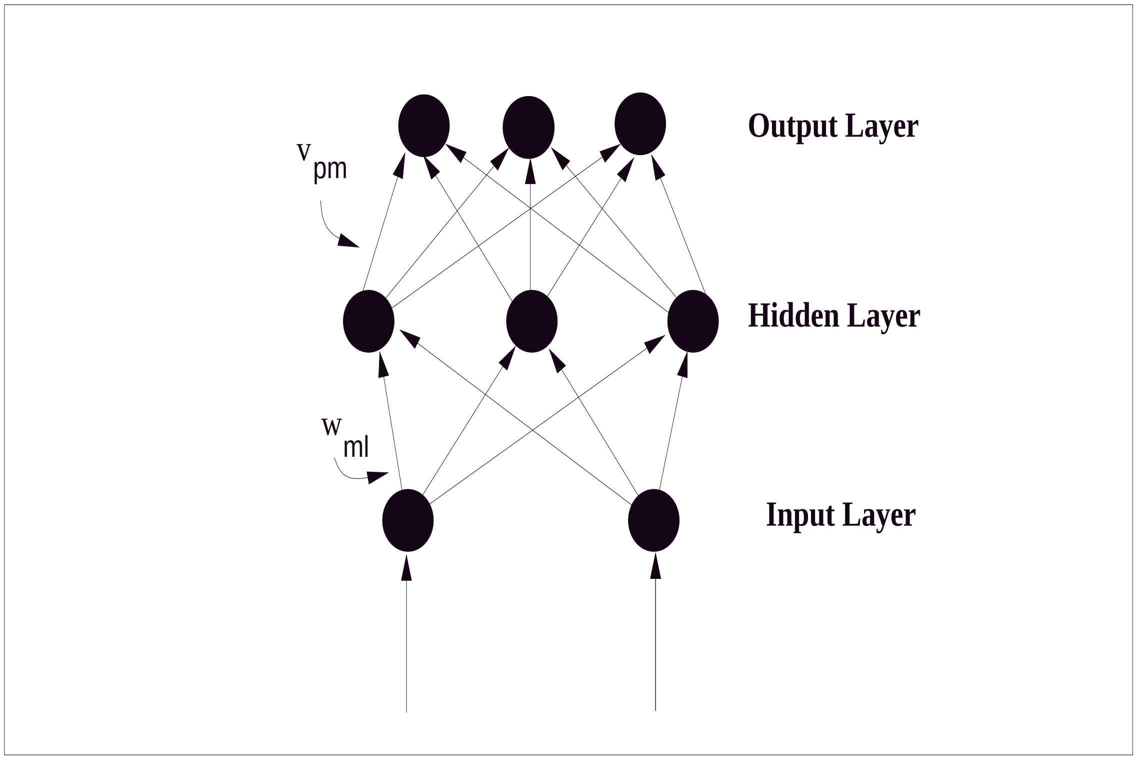

Figure 1: Fully connected feedforward neural network with 3 hidden units and 3 output units.

where

- k is the number of patterns.
- C is the number of output units.
- $t_{pi} = 0$ or 1 is the target value for pattern x_i at output unit p, $p = 1, 2, \ldots, C$.

- S_{pi} is the output of the network at output unit p.

$$S_{pi} = f\left(\sum_{j=1}^{h} f\left((x_i)^T w_j\right) v_{pj}\right) \tag{2.2}$$

- h is the number of hidden units in the network.

- x_i is an n-dimensional input pattern, $i = 1, 2, \ldots, k$.

- w_m is an n-dimensional vector of weights for the arcs connecting the input layer and the m-th hidden unit, $m = 1, 2 \ldots, h$. The weight of the connection from the ℓ-th input unit to the m-th hidden unit is denoted by $w_{m\ell}$.

- v_m is a C-dimensional vector of weights for the arcs connecting the m-th hidden unit and the output layer. The weight of the connection from the m-th hidden unit to the p-th output unit is denoted by v_{pm}.

The activation function $f(y)$ is usually the sigmoid function

$$\sigma(y) = 1/(1 + e^{-y})$$

or the hyperbolic tangent function

$$\delta(y) = (e^{y} - e^{-y})/(e^{y} + e^{-y}).$$

The main difference between these two functions is their range. The range of $\sigma(y)$ is $[0, 1]$, while the range of $\delta(y)$ is $[-1, 1]$. Since we will be using the cross-entropy error function which involves taking the logarithm of the predicted network outputs, the sigmoid function is applied at the output units. Karnin (1990) used the hyperbolic tangent function as the activation function at the hidden units. We have also used this function to obtain all the results reported in this paper.

It has been suggested by several authors (Lang and Witbrock 1988; van Ooyen and Nienhuis 1992) that the cross-entropy error function

$$F(w,v) = -\sum_{i=1}^{k}\sum_{p=1}^{C} t_{pi}\log S_{pi} + (1-t_{pi})\log(1-S_{pi}) \tag{2.3}$$

improves the convergence of the training process. The components of the gradient of this function are

$$\begin{aligned}
\frac{\partial F(w,v)}{\partial w_{m\ell}} &= \frac{\partial F(w,v)}{\partial S_{pi}} \times \frac{\partial S_{pi}}{\partial w_{m\ell}} \\
&= \sum_{i=1}^{k}\sum_{p=1}^{C}\left[e_{pi} \times v_{pm} \times \left(1 - \delta(x_i w_m)^2\right) \times x_{i\ell}\right] \\
\frac{\partial F(w,v)}{\partial v_{pm}} &= \frac{\partial F(w,v)}{\partial S_{pi}} \times \frac{\partial S_{pi}}{\partial v_{pm}} \\
&= \sum_{i=1}^{k}\left[e_{pi} \times \delta(x_i w_m)\right]
\end{aligned}$$

for all $m = 1,2,\ldots,h$, $\ell = 1,2,\ldots,n$, and $p = 1,2,\ldots,C$, with the error for each pattern at output unit p defined as follows

$$e_{pi} = S_{pi} - t_{pi},\ i = 1,2,\ldots,k.$$

The number of output units C depends on the number of different classes in the data. When there are only 2 classes, a single output unit is sufficient. Each pattern can be given a target value equals to 0 or 1 depending on its class label. When there are more than 2 classes, C is set to the number of classes. Each pattern x_i in class $\mathcal{C}_c$, $c = 1,2,\ldots,C$ is given a C-dimensional target vector t_i such that $t_{pi} = 1$ if $p = c$, and $t_{pi} = 0$ otherwise.

The difference between the gradient of the functions $E(w,v)$ and $F(w,v)$ is the presence of the product $S_{pi}(1-S_{pi})$ in the gradient of the sum of the squared error function $E(w,v)$. As pointed out in van Ooyen and Nienhuis (1992), when the values of S_{pi} are converging to either 0 or 1, the magnitude of the gradient of $E(w,v)$ becomes small. This leads to a slow down in the convergence of any minimization algorithm that utilizes gradient informations to move from one iterate to the next.

2.2 Choice of penalty function and criteria for weight elimination

When a network is to be pruned, it is a common practice to add a penalty term to the error function during training. Usually, the penalty term is the sum of the squared weights

$$P_1(w,v) = \epsilon/2\left(\sum_{m=1}^{h}\sum_{\ell=1}^{n} w_{m\ell}^2 + \sum_{m=1}^{h}\sum_{p=1}^{C} v_{pm}^2\right), \tag{2.4}$$

where ϵ is a small positive weight decay constant (Hinton 1989). This quadratic penalty term is used to discourage the weights from taking large values. If second-order approximations of the error function are employed to find a local minimum of the error function, the addition of this quadratic term also contributes to the stability of the training process. The penalty term 2.4 adds ϵ to the diagonal of the second derivative matrix of the error function. With this perturbation, it is more likely for the matrix to be positive definite, and hence, a descent direction can be obtained.

There are some drawbacks of using this penalty term. When the backpropagation method is used to train the neural network, the addition of this quadratic penalty term will cause all the weights to decay exponentially to zero at the same rate (Hanson and Pratt 1989). The quadratic penalty also disproportionately penalizes large weights.

To remedy this problem, the following penalty function has been suggested (Weigend *et al.* 1990)

$$P_2(w,v) = \epsilon/2\left(\sum_{m=1}^{h}\sum_{\ell=1}^{n} \frac{w_{m\ell}^2}{1+w_{m\ell}^2} + \sum_{m=1}^{h}\sum_{p=1}^{C} \frac{v_{pm}^2}{1+v_{pm}^2}\right). \tag{2.5}$$

Since the value of the function $f(w) = w^2/(1+w^2)$ is small when w is close to zero and approaches 1 as w becomes large, the function 2.5 can be considered as a measure of the total number of nonzero weights in the network. The derivative function $f'(w) = 2w/(1+w^2)^2$

indicates that the backpropagation training will be very little affected with the addition of the penalty function 2.5 for weights having large values. Selecting suitable values for the backpropagation learning rate and ϵ will cause small weights to decay at a higher rate than large weights. A disadvantage of this penalty function is that it does not make any distinction between large and very large weights; that is, there exists a sufficiently large w_1 such that for all $|w| \geq w_1, 1 - \nu \leq f(w) \leq 1$ for any small positive scalar ν. The reason why we would like to have weights that are not too large is linked to the criteria for weight elimination we will develop below.

Recall that given an input pattern x_i, the output of the network at unit p is given according to the equation

$$S_{pi} = \sigma\left(\sum_{j=1}^{h} \delta\left((x_i)^T w_j\right) v_{pj}\right). \tag{2.6}$$

The derivatives of S_{pi} with respect to the weights of the network are as follows

$$\frac{\partial S_{pi}}{\partial w_{m\ell}} = S_{pi}(1 - S_{pi}) v_{pm} x_{i\ell}\left(1 - \delta(x_i w_m)^2\right) \tag{2.7}$$

$$\frac{\partial S_{pi}}{\partial v_{qm}} = \begin{cases} S_{pi}(1 - S_{pi})\delta(x_i w_m) & \text{if } p = q \\ 0 & \text{otherwise.} \end{cases} \tag{2.8}$$

For the moment, let us consider S_{pi} as a function of a single variable corresponding to the connection between the ℓth input unit and the mth hidden unit. From the mean value theorem (Grossman 1995, page 173), we have that

$$S_{pi}(w) = S_{pi}(w_{m\ell}) + \frac{\partial S_{pi}(w_{m\ell} + \lambda(w - w_{m\ell}))}{\partial w_{m\ell}}(w - w_{m\ell}), \tag{2.9}$$

where $0 < \lambda < 1$. The maximum value of the product $S_{pi}(1 - S_{pi})$ is 0.25 and the maximum value of $(1 - \delta(x_i w_m)^2)$ is 1. Assuming that $x_{i\ell} \in [0, 1]$, from equation 2.7 we obtain the bound

$$\left|\frac{\partial S_{pi}(w)}{\partial w_{m\ell}}\right| \leq |v_{pm}|/4, \ \forall w \in \mathbb{R}. \tag{2.10}$$

Combining equations 2.9 and 2.10 and letting w equal to zero, we obtain

$$|S_{pi}(0) - S_{pi}(w_{m\ell})| \leq |v_{pm} w_{m\ell}|/4. \tag{2.11}$$

The above equation gives an upper bound on the change in the output of the network when the weight $w_{m\ell}$ is eliminated.

Similarly, by considering S_{pi} as a function of a single variable v_{pm} that corresponds to the connection between the mth hidden unit and the pth output unit, from equation 2.8 we have the bound

$$\left| \frac{\partial S_{pi}(v)}{\partial v_{pm}} \right| \leq 1/4, \ \forall v \in \mathbb{R}. \tag{2.12}$$

Hence, the change in the pth output of the network after the weight v_{pm} has been eliminated is bounded by

$$|S_{pi}(0) - S_{pi}(v_{pm})| \leq |v_{pm}|/4. \tag{2.13}$$

A pattern is correctly classified if the following condition is satisfied

$$|e_{pi}| = |S_{pi} - t_{pi}| \leq \eta_1, \ p = 1, 2, \ldots, C \tag{2.14}$$

where $\eta_1 \in [0, 0.5)$. Suppose now that with the original fully connected network, we are able to meet a prespecified accuracy requirement. The bound 2.11 shows that we can set $w_{m\ell}$ to zero without deteriorating the classification rate of the network if the product $|v_{pm} w_{m\ell}|$ is sufficiently small. Similarly, the bound 2.13 suggests that v_{pm} can be set to zero if its magnitude is sufficiently small. We have the following two propositions.

Proposition 1 *Let (w, v) be a set of weights of a network with an accuracy rate of $\mathcal{R}$, with condition 2.14 satisfied by each correctly classified input pattern. Let* $\max_p |v_{pm} w_{m\ell}| \leq 4\eta_2$*, and let $\tilde{S}_{pi}$ be the output of the network with $w_{m\ell}$ set to zero, then for each pattern x_i that is correctly classified by the original network, the following holds*

$$|\tilde{e}_{pi}| = |\tilde{S}_{pi} - t_{pi}| \leq \eta_1 + \eta_2, \ p = 1, 2, \ldots, C.$$

Proposition 2 *Let (w, v) be a set of weights of a network with an accuracy rate of $\mathcal{R}$, with condition 2.14 satisfied by each correctly classified input pattern. Let $|v_{pm}| \leq 4\eta_2$, and let $\tilde{S}_{pi}$ be the output of the network with v_{pm} set to zero, then for each pattern x_i that is correctly classified by the original network, the following holds*

$$|\tilde{e}_{pi}| = |\tilde{S}_{pi} - t_{pi}| \leq \eta_1 + \eta_2, \; p = 1, 2, \ldots, C.$$

As long as the sum $\eta_1 + \eta_2$ is less than 0.5, the above propositions ensure that the pruned network still achieves an accuracy rate of $\mathcal{R}$. After $w_{m\ell}$ or v_{pm} is set to zero, the absolute errors of each correctly classified pattern can be made arbitrarily small by increasing the weights from the hidden units to the output units as the following proposition suggests.

Proposition 3 *Let (w, v) be a set of weights of a network with an accuracy rate of $\mathcal{R}$ such that for each correctly classified pattern x_i, the following condition holds*

$$|e_{pi}| = |S_{pi} - t_{pi}| \leq \eta, \; p = 1, 2, \ldots, C$$

for some $\eta \in [0, 0.5)$. For any $\rho \in (0, 1)$, there exists a sufficiently large scalar $\lambda > 0$ such that

$$|\hat{e}_{pi}| = |\hat{S}_{pi} - t_{pi}| \leq \rho\eta,$$

where $\hat{S}_{pi}$ is the output of a new network with weights $(w, \lambda v)$.

Proposition 3 guarantees the existence of a new set of weights which still gives an accuracy rate of $\mathcal{R}$ with the same error tolerance 2.14. It will be prudent to retrain the network to find this new set so that the weights of connections between the hidden units and output units v will not become too large. Simply multiplying v by some large λ will hinder the process of removing subsequent weights from the network.

We summarize our pruning algorithm below.

Algorithm N2P2F: Neural Network Pruning with Penalty Function

1. Let η_1 and η_2 be positive scalars such that $\eta_1 + \eta_2 < 0.5$. (η_1 is the error tolerance, η_2 is a threshold that determines if a weight can be removed).

2. Pick a fully connected network, and train this network to meet a prespecified accuracy level with condition 2.14 satisfied by all correctly classified input patterns. Let (w, v) be the weights of this network.

3. For each $w_{m\ell}$ in the network, if

$$\max_p |v_{pm} w_{m\ell}| \leq 4\eta_2, \tag{2.15}$$

   then remove $w_{m\ell}$ from the network.

4. For each v_{pm} in the network, if

$$|v_{pm}| \leq 4\eta_2, \tag{2.16}$$

   then remove v_{pm} from the network.

5. If no weight satisfies condition 2.15 or condition 2.16, then for each $w_{m\ell}$ in the network, compute

$$\omega_{m\ell} = \max_p |v_{pm} w_{m\ell}|.$$

   Remove $w_{m\ell}$ with the smallest $\omega_{m\ell}$.

6. Retrain the network. If classification rate of the network falls below the specified level, then stop and use the previous setting of network weights. Otherwise, go to Step 3.

To reduce the retraining time, we remove all the weights that satisfy condition 2.15 or condition 2.16 in Steps 3 and 4 of the algorithm. Although it can no longer be guaranteed that the accuracy of the retrained network will not drop, our experiments show that many weights can be eliminated simultaneously without deteriorating the performance of the network. This is especially true when the starting fully connected network has an excessive number of redundant hidden units.

Since the two conditions 2.15 and 2.16 for pruning depend on the magnitude of the weights for connections between the input units and the hidden units (2.15) and between the hidden units and the output units (2.16), it is imperative that during training these weights be prevented from getting too large. At the same time, small weights should be encouraged to decay rapidly to zero. To achieve these goals, we make use of both penalty functions 2.4 and 2.5 to obtain the following function:

$$P(w,v) = \epsilon_1 \sum_{m=1}^{h} \left(\sum_{\ell=1}^{n} \frac{\beta w_{m\ell}^2}{1+\beta w_{m\ell}^2} + \sum_{p=1}^{C} \frac{\beta v_{pm}^2}{1+\beta v_{pm}^2} \right) + \epsilon_2 \sum_{m=1}^{h} \left(\sum_{\ell=1}^{n} w_{m\ell}^2 + \sum_{p=1}^{C} v_{pm}^2 \right). \quad (2.17)$$

Hence, the complete error function to be minimized during the training process is

$$\theta(w,v) = P(w,v) - \sum_{i=1}^{k} \sum_{p=1}^{C} \left(t_{pi} \log S_{pi} + (1-t_{pi}) \log(1-S_{pi}) \right). \quad (2.18)$$

The values for the weight-decay parameters $\epsilon_1, \epsilon_2 > 0$ must be chosen to reflect the relative importance of the accuracy of the network versus its complexity. These parameters also determine the range of values where the penalty for each weight in the network is approximately equal to ϵ_1. Consider the plot of the function

$$f(w) = \epsilon_1 \beta w^2/(1+\beta w^2) + \epsilon_2 w^2,$$

where $\epsilon_1 = 0.1, \epsilon_2 = 10^{-5}$, and $\beta = 10$ in Figure 2a. This function intersects the horizontal line $f = \epsilon_1$ at $w^* \approx \pm 5.62$. If ν is a small positive scalar, then we can find the values of

$w_L > 0$ and $w_U > 0$ such that for all values of w, $|w| \in (w_L, w_U)$, $\epsilon_1 - \nu \leq f(w) \leq \epsilon_1 + \nu$. This is the penalty corresponding to a nonzero connection in the network. In this interval, almost no distinction is made between the penalty for smaller or larger weights. For example, when $\nu = 10^{-2}$ the penalty for any weight with magnitude in the interval $(w_L, w_U) = (0.95, 31.64)$ is within 10 % of ϵ_1. By decreasing the value of ϵ_2, the interval over which the penalty value is approximately equal to ϵ_1 can be made wider as shown in Figure 2c, where $\epsilon_2 = 10^{-6}$. A weight is prevented from taking a value that is too large, since the quadratic term becomes dominant for values of w that are greater than w_U. The value of w_L can be made arbitrarily small by increasing the value of the parameter β. The derivative of the function $f(w)$ near zero is relatively large (Figures 2b and 2d). This will give a small weight w stronger tendency to decay to zero.

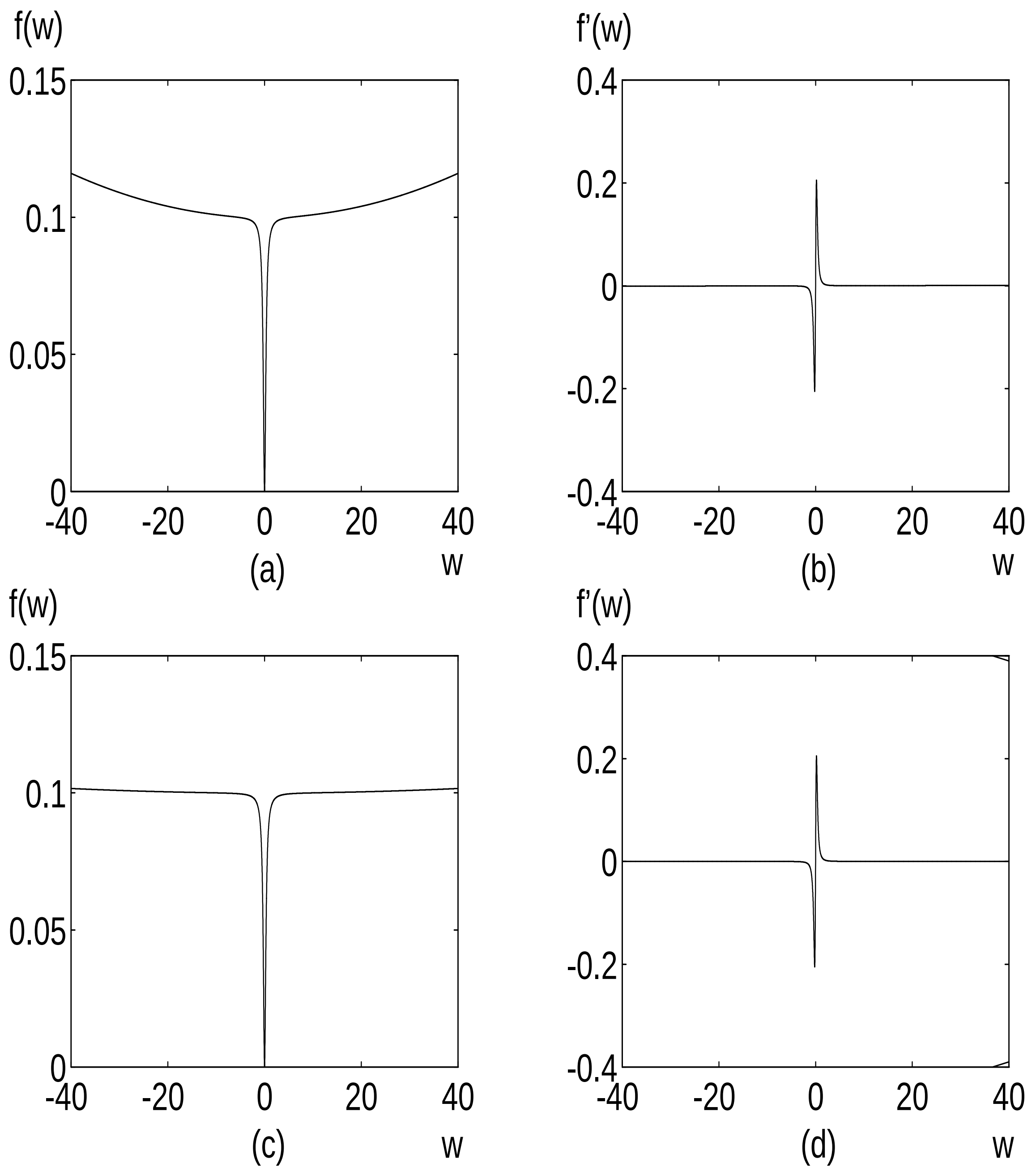

Figure 2: Plots of the function $f(w) = \epsilon_1 \beta w^2/(1+\beta w^2) + \epsilon_2 w^2$ and its derivative. Figure (a,b): $\epsilon_1 = 0.1, \epsilon_2 = 10^{-5}$, and $\beta = 10$. Figure (c,d): $\epsilon_1 = 0.1, \epsilon_2 = 10^{-6}$, and $\beta = 10$.

3 Experimental results

We have selected well known problems to test the pruning algorithm described in the previous section. These problems were the contiguity problem, the 4-bit and 5-bit parity problems, and the monks problems. Since we were interested in finding the simplest network architecture that could solve these problems, all available input data were used for training and pruning. An exception was the monks problems. For the three monks problems, the division of the data sets into training and testing sets has been fixed (Thrun *et al.* 1991).

There was one output unit in the networks. For all problems, thresholds or biases in the hidden units were incorporated into the network by appending a 1 to each input pattern. The threshold value of the output unit was set to zero. This was done to simplify the implementation of the training and pruning algorithms. Only two different sets of parameters were involved in the error function. They corresponded to the weights of the connections between the input layer and the hidden layer, and between the hidden layer and the output layer. During training, the gradient of the error function was computed by taking partial derivatives of the function with respect to these parameters only. During pruning, conditions were checked to determine whether a connection could be removed. There was no special condition for checking whether a hidden unit bias could be removed. Note that if after pruning, there exists a hidden unit connected only to the input unit with a constant input value of 1 for all patterns and the output unit, then the weights of two arcs connected to this hidden unit determine a nonzero threshold value at the output unit.

The function $\theta(w, v)$ (cf. equation 2.18) was minimized by a variant of the quasi-Newton algorithm, the BFGS method. It has been shown that quasi-Newton algorithms, such as the BFGS method can speed up the training process significantly (Watrous 1987). At each iteration of the algorithm, a positive definite matrix that is an approximation of the inverse of the Hessian of the function to be minimized is computed. The positive definiteness of this matrix ensures that a descent direction can be generated. Given a descent direction, a step size is computed via an inexact line search algorithm. Details of this algorithm can be found in Dennis and Schnabel (1983).

For all the experiments reported here, we used the same values for the parameters involved in the function $\theta(w,v)$. These were $\epsilon_1 = 0.1, \epsilon_2 = 10^{-5}$ and $\beta = 10$. During the training process, the BFGS algorithm was terminated when the following condition was satisfied

$$\|\nabla\theta(w,v)\| \leq 10^{-8}\ \max\{1, \|w,v\|\},$$

where $\|\nabla\theta(w,v)\|$ is the 2-norm of the gradient of $\theta(w,v)$. At this point the accuracy of the network was checked. The value of η_1 (cf. equation 2.14) was 0.35. If the classification rate of the network met the prespecified required accuracy, then the pruning process was initiated. The required accuracy of the original fully connected network was 100 % for all problems, except for the Monks 3 problem. For the Monks 3 problem, the acceptable accuracy rate was 95 %. We did not attempt to get 100 % accuracy rate for this problem due to the presence of 6 incorrect classifications (out of 122 patterns) in the training set.

The parameter η_2 used during the pruning process was set at 0.10. Typically, at the start of the pruning process many weights would be eliminated because they satisfied condition 2.15. Subsequently, weights were removed one at a time based on the magnitude of the product $|v_{pm}w_{m\ell}|$. Before the actual pruning and retraining was done, the current network was saved. Should pruning additional weight(s) and retraining the network fail to give a new set of weights that met the prespecified accuracy requirement, the saved network would be considered as the smallest network for this particular run of the experiment.

No. of starting hidden units	**6**	**9**
No. of connections before pruning	72	108
Ave. no. of connections after pruning	28.44 (1.33)	29.18 (1.32)
Ave. no of hidden units after pruning	5.98 (0.14)	7.62 (0.75)

Table 1: The average number of connections and hidden units in 50 networks trained to solve the contiguity problem after pruning. Figures in parentheses indicate standard deviations.

3.1 The contiguity problem

This problem has been the subject of several study on network pruning (Thodberg 1991; Gorodkin *et al.* 1993). The input patterns were binary strings of length 10. Of all possible 1024 patterns, only those with 2 or 3 clumps were used. A clump was defined as a block of 1's in the string separated by at least one 0. The target value t_i for each of the 360 patterns with 2 clumps was 0, and the target value for each of the 432 patterns with 3 clumps was 1. A sparse symmetric network architecture with 9 hidden units and a total of 37 weights has been suggested for this problem (Gorodkin *et al.* 1993).

Two experiments were carried out using this data set as input. In the first experiment, 50 neural networks having 6 hidden units were used as the starting networks. In the second experiment, the same number of networks, each with 9 hidden units, were used. The accuracy rate of all these network, which were trained starting from different random initial weights was 100 %. The number of connections and hidden units left in the networks after pruning are tabulated in Table 1.

Thodberg (1991) trained 38 fully connected networks with 15 hidden units using 100 patterns as training data. The trained networks were then pruned by a brute-force scheme and the average number of connections left in the pruned network was 32. This number, however, did not include the biases in the networks. Employing the Optimal Brain Damage pruning scheme, Gorodkin et al. (1993) obtained pruned networks with a number of weights ranging from 36 to more than 90. These networks were trained on 140 examples. In contrast, the maximum number of connections left in our networks was 33 and only one of the pruned networks had this many connections. The results of the experiments with different initial number of hidden units in the networks show the robustness of our pruning algorithm. Starting with 6 or 9 hidden units, the networks were pruned until there were on average 29 connections left.

3.2 The parity problem

The parity problem is a well known difficult problem that has often been used for testing the performance of a neural network training algorithm. The input set consists of 2^n patterns in n-dimensional space and each pattern is an n-bit binary vector. The target value t_i is equal to 1 if the number of one's in the pattern is odd and it is 0 otherwise. We used the 4-bit and 5-bit parity problems to test our pruning algorithm.

For the 4-bit parity problem, three sets of experiments were conducted. In the first set, 50 networks each with 4 hidden units were used as the starting networks. In the second and third sets of experiments, the initial number of hidden units were 5 and 6. Similarly, three sets of experiments were conducted for the 5-bit parity problem. The initial number of hidden units in 50 starting networks for each set of experiment were 5, 6 and 7 respectively.

The average number of weights and hidden units of the networks after pruning are shown in Table 2. In this table, we also show the average number of hidden units left after pruning. Many experiments using backpropagation network assume that n hidden units are needed

Parity 4			
No. of starting hidden units	**4**	**5**	**6**
No. of connections before pruning	24	30	36
Ave. no. of connections after pruning	17.40 (1.55)	18.12 (1.60)	18.76 (2.30)
Ave. no of hidden units after pruning	3.42 (0.50)	3.64 (0.66)	3.98 (0.77)
Parity 5			
No. of starting hidden units	**5**	**6**	**7**
No. of connections before pruning	35	42	49
Ave. no. of connections after pruning	21.80 (3.14)	21.84 (3.02)	22.34 (3.17)
Ave. no of hidden units after pruning	3.58 (0.81)	3.68 (0.84)	3.82 (0.83)

Table 2: The average number of connections and hidden units obtained from 50 networks for the 4-bit and 5-bit parity problems after pruning.

to solve the n-bit parity problem. Previously published neural network pruning algorithms have also failed to obtain networks with less than n hidden units (Chung and Lee 1992; Hanson and Pratt 1989). In fact, 3 hidden units are sufficient for both the 4-bit and 5-bit parity problems.

3.3 The monks problems

The monks problems (Thrun *et al.* 1991) are an artificial robot domain, in which robots are described by six different attributes:

A_1:	head_shape	$\in$	round, square, octagon;	A_2:	body_shape	$\in$	round, square, octagon;
A_3:	is_smiling	$\in$	yes, no;	A_4:	holding	$\in$	sword, balloon, flag;
A_5:	jacket_color	$\in$	red, yellow, green, blue;	A_6:	has_tie	$\in$	yes, no.

The learning tasks of the three monks problems are of binary classification, each of them is given by the following logical description of a class.

- Problem Monks 1: (head_shape = body_shape) or (jacket_color = red). From 432 possible samples, 124 were randomly selected for the training set.
- Problem Monks 2: Exactly two of the six attributes have their first value. From 432 samples, 169 were selected randomly for the training set.
- Problem Monks 3: (Jacket_color is green and holding a sword) or (jacket_color is not blue and body_shape is no octagon). From 432 samples, 122 were selected randomly for training and among them there were 5% misclassifications, i.e., noise in the training set.

The testing set for the three problems consisted of all 432 samples. Two sets of experiments were conducted on the Monks 1 and Monks 2 problems. In the first set, 50 networks each with 3 hidden units were used as the starting networks for pruning. In the second set, an equal number of networks with 5 hidden units were used. These networks correctly classified all patterns in the training set. For the Monks 3 problem, in addition to the two sets of experiments, 50 networks with just 1 hidden unit were also trained as the starting networks. All starting networks for Monks 3 had an accuracy rate of at least 95 % on the training data.

The required accuracy of the pruned networks for Monks 1 and Monks 2 problems on the training patterns was 100 %. For the Monks 3 problem, it was 95 %. The results from the experiments for Monks 1 and Monks 2 problems are summarized in Table 3. The results for Monks 3 problem are tabulated in Table 4. The P-values in these tables had been computed to test whether the accuracy of the networks on the testing set increased significantly after pruning. With the exception of the networks with 1 hidden unit for the Monks 3 problem, the P-values clearly show that pruning did increase the predictive accuracy of the networks. There was not much difference in the average number of connections left after pruning between networks that originally had 3 hidden units and those with 5 hidden units. For the Monks 3 problem, when the starting networks had only 1 hidden unit, the average number of connections left was 9.92. This figure was significantly less than the averages obtained from networks with 3 and 5 starting hidden units. Part of this difference can be accounted for by the number of hidden units still left in the pruned networks. Most of the networks with 3 or 5 starting hidden units still had 3 hidden units after pruning. The connections from these hidden units to the output unit added on average 2 more connections to the total.

The smallest pruned networks with 100 % accuracy on the training and the testing sets for the Monks 1 and Monks 2 problems had 11 and 15 connections, respectively. For the Monks 3 problem, the pruning algorithm found a network with only 6 connections. This network was able to identify the 6 noise cases in the training set. The predicted outputs for these noise patterns were the opposite of the target outputs, and hence an accuracy rate of 95.08 % was obtained on the training set. Since there was no noise in the testing data, the predictive accuracy rate was 100 %. The network is depicted in Figure 3. For comparison, the backpropagation with weight decay and the Cascade-Correlation algorithms both achieved 97.2 % accuracy on the testing set (Thrun *et al.* 1991).

Some interesting observations on the relationship between the number of hidden units and the generalization capability of a network were made by Weigend (1993). One of these observations is that large networks perform better than smaller networks provided that the training of the networks are stopped early. To determine when the training should be terminated, a second data set –the cross-validation set– is used. Our results on the monks problems are mixed. For the Monks 1 problem, networks with 5 hidden units have higher predictive accuracy than those with 3 hidden units. For the Monks 2 problem, the reverse is true. For the Monks 3 problem, where there are noise in the data, there is a trend that larger networks have poorer predictive accuracy rates than smaller ones. This is true even after the networks have been pruned. Note that the statistics were obtained from networks that had been trained to reach a local minimum of the augmented error function. One clear conclusion that can be drawn from the figures in Tables 3 and 4 is that, pruned networks have better generalization capability than the fully connected networks.

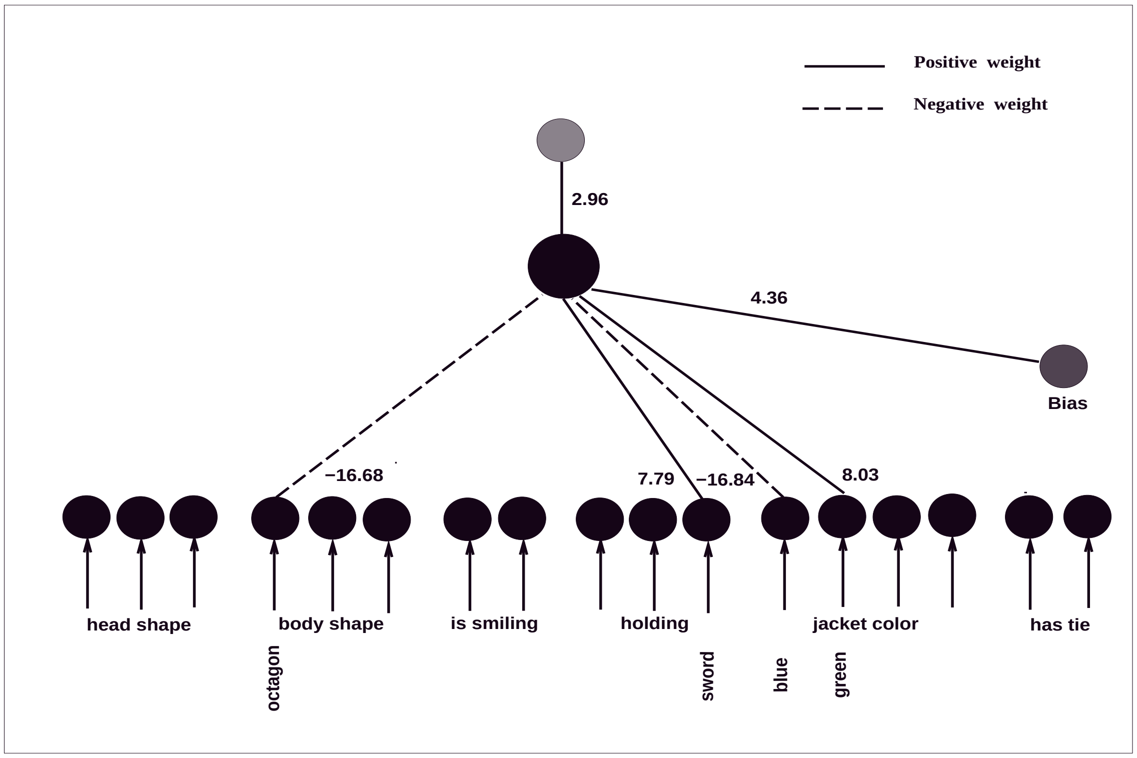

Figure 3: A network with 6 weights for the Monks 3 problem. The number next to a connection shows the weight for that connection. The accuracy rates on the training set and the testing set are 95.08 and 100 %, respectively.

4 Discussion and final remarks

We presented a penalty function approach for feedforward neural network pruning. The penalty function consists of two components. The first component is to discourage the use of unnecessary connections and the second component is to prevent the weights of these connections from taking excessively large values. Simple criteria for eliminating network connections are also given. The two components of the penalty function have been used individually in the past. However, applying our approach that combines the penalty function and the magnitude-based weight elimination criteria, we are able to get smaller networks than those reported in the literature. We have also experimented pruning with one of the two penalty parameters (ϵ_1 and ϵ_2) set to zero; the results were not as good as those reported in Section 3.

The approach described in this paper works for networks with one hidden layer. However, it is possible to extend the algorithm so that it works for a network with $N > 1$ hidden layers. Let us label the input layer $\mathcal{L}_0$, the output layer $\mathcal{L}_{N+1}$, and the hidden layers $\mathcal{L}_1, \mathcal{L}_2, \ldots \mathcal{L}_N$. Connections are present only between units in layers $\mathcal{L}_i$ and $\mathcal{L}_{i+1}$. A criterion for removal of a connection between unit H_1 in layer $\mathcal{L}_i$ and unit H_2 in layer $\mathcal{L}_{i+1}$ can be developed to make sure that the changes in the predicted outputs are not more than η_2. The effect of such a removal will be propagated layer by layer from H_2 to the output units in $\mathcal{L}_{N+1}$. Hence, the criterion for a removal will involve the weights from unit H_2 to layer $\mathcal{L}_{i+2}$ and all the weights from layer $\mathcal{L}_{i+2}$ onward.

The effectiveness of the pruning algorithm using our proposed penalty function has been demonstrated by its performance on three well known problems. Using the same set of penalty parameters, networks that solve three test problems – the contiguity problem, the parity problems and the monk problems – with few connections have been obtained. The

Monks 1		
No. of starting hidden units	**3**	**5**
No. of connections before pruning	57	95
Ave. accuracy on testing set (%)	99.10 (2.66)	99.52 (1.44)
Ave. no. of connections after pruning	12.96 (1.63)	12.54 (1.05)
Ave. accuracy on testing set (%)	99.82 (0.74)	100.00 (0.00)
P-value	0.033	0.009
Monks 2		
No. of starting hidden units	**3**	**5**
No. of connections before pruning	57	95
Ave. accuracy on testing set (%)	98.91 (1.47)	98.13 (2.32)
Ave. no. of connections after pruning	16.76 (1.81)	16.86 (1.40)
Ave. accuracy on testing set (%)	99.44 (0.78)	99.39 (0.70)
P-value	0.012	0.001

Table 3: The average number of connections and predictive accuracy obtained from 50 networks for the Monks 1 and Monks 2 problems after pruning. P-values are computed for testing if the predictive accuracy rates of the networks increase significantly after pruning.

Monks 3			
No. of starting hidden units	**1**	**3**	**5**
No. of connections before pruning	19	57	95
Ave. accuracy on training set (%)	95.85 (0.69)	99.02 (1.09)	99.92 (0.38)
Ave. accuracy on testing set (%)	96.07 (2.43)	93.43 (1.91)	92.82 (1.53)
Ave. no. of connections after pruning	9.92 (2.06)	14.64 (2.79)	14.86 (2.19)
Ave. accuracy on training set (%)	95.82 (0.69)	96.00 (1.07)	96.46 (1.12)
Ave. accuracy on testing set (%)	96.11 (2.55)	94.12 (2.95)	93.85 (2.21)
P-value	0.468	0.082	0.003

Table 4: The average number of connections and predictive accuracy obtained from 50 networks for the Monks 3 problem after pruning. P-values are computed for testing if the predictive accuracy rates of the networks increase significantly after pruning.

small number of connections in the pruned networks allow us to develop an algorithm to extract compact rules from networks that have been trained to solve real-world classification problems (Setiono 1995).

References

Ash, T. 1989. Dynamic node creation in backpropagation networks. *Connection Science*, **1(4)**, 365–375.

Chauvin, Y. 1989. A back-propagation algorithm with optimal use of hidden units. In *Advances in Neural Information Processing Systems*, Vol. 1, 519–526. Morgan Kaufmann, San Mateo, CA.

Chung, F.L., and Lee, T. 1992. A node pruning algorithm for backpropagation networks. *Int. J. of Neural Systems*, **3(3)**, 301–314.

Dennis, J.E., and Schnabel, R.B. 1983. In *Numerical methods for unconstrained optimization and nonlinear equations* Prentice Halls, Englewood Cliffs, NJ.

Fahlman, S.E. and Lebiere, C. 1990. The cascade-correlation learning architecture. In *Advances in Neural Information Processing Systems*, Vol. 2, 524–532. Morgan Kaufmann, San Mateo, CA.

Frean, M. 1990. The upstart algorithm: a method for contructing and training feedforward neural networks. *Neural Computation*, **2(2)**, 198–209.

Gorodkin, J., Hansen, L.K., Krogh, A., and Winther, O. 1993. A quantitative study of pruning by optimal brain damage. *Int. J. of Neural Systems*, **4(2)**, 159–169.

Grossman, S.I. 1995. Multivariable Calculus, Linear Algebra, and Differential Equations. Saunders College Publishing, Orlando, FL.

Hanson, S.J., and Pratt, L.Y. 1989. Comparing biases for minimal network construction with back-propagation. In *Advances in Neural Information Processing Systems*, Vol. 1, 177–185. Morgan Kaufmann, San Mateo, CA.

Hassibi, B., and Stork, D.G. 1993. Second order derivatives for network pruning: optimal brain surgeon. In *Advances in Neural Information Processing Systems*, Vol. 5, 164–171. Morgan Kaufmann, San Mateo, CA.

Hertz, J., Krogh, A., and Palmer, R.G. 1991. In *Introduction to the theory of neural computation*, Addison Wesley, Redwood City, CA.

Hinton, G.E. 1989. Connectionist learning procedure. *Artificial Intelligence*, **40**, 185–234.

Hirose, Y., Yamashita, K., and Hijiya, S. Backpropagation algorithm which varies the number of hidden units. *Neural Networks*, **4**, 61–66.

Karnin, E.D. 1990. A simple procedure for pruning back-propagation trained neural networks. *IEEE Trans. Neural Networks* **1(2)**, 239–242.

Lang, K.J., and Witbrock, M.J. 1988. Learning to tell two spirals apart. In *Proc. of the 1988 Connectionist Summer School*, 52–59. Morgan Kaufmann, San Mateo, CA.

Le Cun, Y., Denker, J.S., and Solla, S.A. (1990) Optimal brain damage. In *Advances in Neural Information Processing Systems*, Vol. 2, 598–605. Morgan Kaufmann, San Mateo, CA.

Mezard, M., and , Nadal, J.P. 1989. learning in feedforward layered networks: the tiling algorithm. *Journal of Physics A*, **22(12)**, 2191–2203.

Mozer, M.C., and Smolensky, P. 1989. Skeletonization: a technique for trimming the fat from a network via relevance assestment. In *Advances in Neural Information Processing Systems*, Vol. 1, 107–115. Morgan Kaufmann, San Mateo, CA.

Setiono, R. 1995. Extracting rules from neural networks by pruning and hidden unit splitting. Submitted to *Neural Computation.*

Shanno, D.F., and Phua, K.H. 1976. Algorithm 500: Minimization of unconstrained multivariate functions. *ACM Trans. on Mathematical Software* **2(1)**, 87–94.

Tenorio, M.F., and Lee, W. 1990. Self-orgainizing network for optimum supervised learning. *IEEE Trans. Neural Networks* **1(1)**, 100–110.

Thodberg, H.H. 1991. Improving generalization of neural networks through pruning. *Int. J. of Neural Systems*, **1(4)**, 317–326.

Thrun, S.B., et al. 1991. The MONK's problems - a performance comparison of different learning algorithm. Preprint CMU-CS-91-197, Carnegie Mellon University, Pittsburgh, PA.

van Ooyen, A., and Nienhuis, B. 1992. Improving the convergence of the backpropagation algorithm. *Neural Networks*, **5** 465–471.

Watrous, R. 1987. Learning algorithms for connectionist networks: Applied gradient methods of nonlinear optimization *Proc. IEEE First International Conference on Neural Networks*, IEEE Press, New York, 619–627.

Weigend, A.S. 1993. On overfitting and the effective number of hidden units. In *Proc. of the 1993 Connectionist Models Summer School*, 335–342. Lawrence Erlbaum, Hillsdale, N.J.

Weigend, A.S., Rumelhart, D.E., and Huberman, B.A. Back-propagation, weight-elimination and time series prediction. In *Proc. of the 1988 Connectionist Models Summer School*, 105–116. Morgan Kaufmann, San Mateo, CA.