IDIAP COMMUNICATION

Quantization and Pruning of Multilayer Perceptrons: Towards Compact Neural Networks

Tomas Lundin Perry Moerland *

IDIAP–Com 97-02

March 97

Dalle Molle Institute for Perceptual Artificial Intelligence • P.O.Box 592 • Martigny • Valais • Switzerland

phone +41 – 27 – 721 77 11
fax +41 – 27 – 721 77 12
e-mail secretariat@idiap.ch
internet http://www.idiap.ch

* e-mail: Perry.Moerland@idiap.ch

Preface

A connectionist system or neural network is a massively parallel network of weighted interconnections, which connect one or more layers of non-linear processing elements (*neurons*). To fully profit from the inherent parallel processing of these networks, development of parallel hardware implementations is essential. However, these hardware implementations often differ in various ways from the ideal mathematical description of a neural network model. It is, for example, required to have quantized network parameters, in both electronic and optical implementations of neural networks. This can be because device operation is quantized or a coarse quantization of network parameters is beneficial for designing *compact networks*.

Most of the standard algorithms for training neural networks are not suitable for quantized networks because they are based on gradient descent and require a high accuracy of the network parameters Several weight discretization techniques have been developed to reduce the required accuracy further without deterioration of network performance. One of the earliest of these techniques [Fiesler-88] is further investigated and improved in this report.

Another way to obtain compact networks is by minimizing their topology for the problem at hand. However, it is impossible to know *a priori* the size of such minimal network topology. An unsuitable topology will increase the training time, lower the generalization performance on unseen test data [Gosh-94], and in some cases even cause non-convergence. One method to lower the importance of choosing the initial network topology and minimizing the network size is pruning, that is, removal of connections or neurons during training. Especially a combination of parameter/weight quantization and network pruning, leading to networks that have a small topology and for which small accuracy is sufficient, is of great importance for hardware implementation of neural networks. Such networks offer a minimization of chip area and computational requirements. Due to their lack of redundancy they are also expected to show a better generalization on unseen patterns (Occam's razor). Such a combination of pruning techniques with weight quantization is studied in the second part of this report.

Five different quantization functions, chapter 3, and six pruning methods, chapter 4, are evaluated in a series of experiments on real-world benchmarks problems. The main goal is to first improve the original weight discretization technique as much as possible to obtain networks with both a small number of discrete weight levels and good generalization performance on unseen test data. Secondly, the results from the first part are combined with the pruning methods to ease the choice of the initial network topology and obtain compact networks.

Contents

1 Multilayer Neural Networks **1**
- 1.1 Model of Neuron . . . 1
- 1.2 Least-Mean-Square Algorithm . . . 2
- 1.3 Back-Propagation Algorithm . . . 4

2 Weight Discretization in Neural Networks **7**
- 2.1 Quantization Effects . . . 7
- 2.2 Weight Discretization Algorithm . . . 8

3 Connectionist Quantization Functions **9**
- 3.1 Weight Discretization Algorithm . . . 9
  - 3.1.1 Quantization Functions . . . 9
- 3.2 Experimental Setup . . . 13
  - 3.2.1 Benchmarks and Parameter Settings . . . 13
  - 3.2.2 Generalization . . . 14
- 3.3 Hardware Implementation Details . . . 15
- 3.4 Discussion of Results for Quantization Functions . . . 16
- 3.5 Conclusion for Quantization Functions . . . 16

4 Pruning of Neural Networks **19**
- 4.1 Pruning Methods . . . 19
- 4.2 Simulations . . . 20
  - 4.2.1 Benchmarks . . . 20
  - 4.2.2 Experimental Setup . . . 22
- 4.3 Discussion of Results for Pruning . . . 23
- 4.4 Conclusions for Pruning . . . 26

5 Conclusions **29**
- 5.1 Future work . . . 29
- 5.2 Acknowledgements . . . 30

A Quantization Results **33**

B Pruning Results **37**

Chapter 1

Multilayer Neural Networks

Multilayer perceptrons trained by backpropagation or one of its variations, are by far the most popular neural network model. They have successfully been applied to a wide area of problems from different domains. This chapter, gives an introduction to the theory of neural networks; firstly, by looking at the basic building block in every neural network, the neuron. Secondly, a description of the multilayer perceptron together with the least mean square algorithm is presented. Finally, the frequently used backpropagation algorithm for training multilayer perceptrons, is described in detail.

1.1 Model of Neuron

The fundamental building block of a neural network model is the *neuron*, here displayed as a spatial filter (figure 1.1). The four basic elements of the neuron model are:

- The inputs are connected to the neuron by a set of synapses, which are characterized by an individual weight or synaptic strength. Each input value x_i is multiplied by its connecting weight w_i.
- An adder, which sums all inputs values weighted by their respective synaptic weights.
- The linearly combined output of the adder is limited to a bounded interval by a non-linear activation function. Frequently used activation functions are the sigmoid function (eq. (1.1)) and the hyperbolic tangent (eq. (1.2)):

$$f(u) = \frac{1}{1+\exp(-u)} \qquad \text{output range [0,1]} \tag{1.1}$$

$$f(u) = \frac{1-\exp(-u)}{1+\exp(-u)} \qquad \text{output range [-1,1]} \tag{1.2}$$

- An external bias, θ, which corresponds to an affine transformation of the hyperplane created by the adder. Often, a fixed input of $+1$ is connected to the bias.

Summarizing, the input-output relation of the neuron model can be described as follows:

$$u = (\sum_{k=1}^{p} w_k x_k) + \theta$$

and the non-linear part:

$$y = f(u),$$

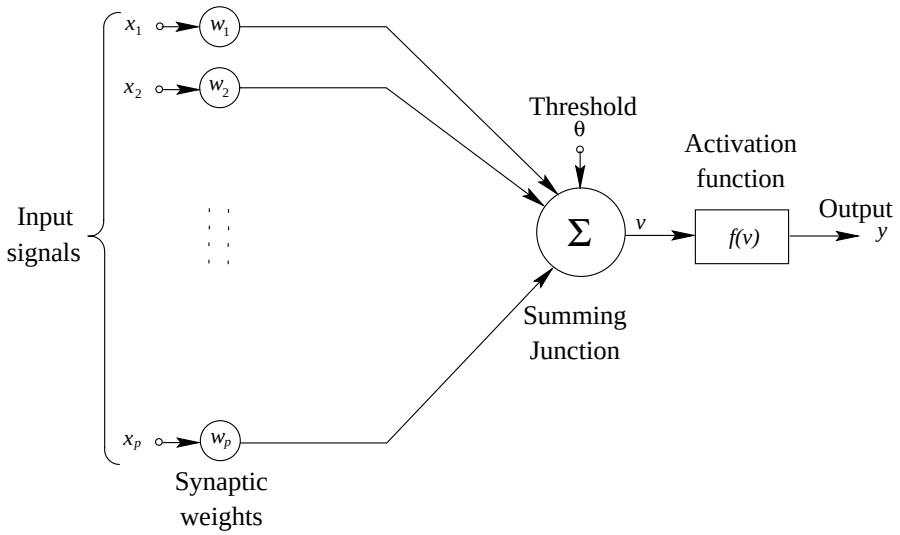

Figure 1.1: Non-linear model of a neuron.

where $x_1, x_2, \ldots, x_p$ are the input values and $w_1, x_2, \ldots, w_p$ are the (synaptic) weights.

This neuron model is used as a basic building block in various neural network models. Examples are recurrent networks and the popular multilayer perceptrons (Figure 1.1), which will be used throughout this report. Typically, an MLP consists of an input layer, receiving the input values, an output layer, furnishing the output values, and one or more hidden layers. Each of the layers consists of a number of neurons and consecutive layers are either fully or partially interconnected. In a fully connected network, a neuron in layer i is connected to all neurons in layer $i+1$. In partially connected networks, some connections are left out which might improve generalization on unseen test data and reduce sensitivity to noise.

1.2 Least-Mean-Square Algorithm

This section gives an introduction to the basic neural network principle of adaptivity, which is illustrated by the Least-Mean-Square (LMS) algorithm for linear neurons (filters). This (linear) neuron model is similar to the one described in section 1.1 but with the identity function as activation function. A fundamental problem, also known as the linear optimum filtering problem, is to determine the optimal weights so that the difference, $e = d - y$, between the desired response d and the system output y is minimized in a mean-square error sense. The mean-squared error cost function is:

$$J = \frac{1}{2} E[e^2], \tag{1.3}$$

where E is the expectation operator. The set of weights that minimizes eq. (1.3) is called the *Wiener Filter* [Haykin-94]. Expanding eq. (1.3) and changing the order of expectation and summation gives:

$$J = \frac{1}{2} E[d^2] - \sum_{k=1}^{p} w_k E[x_k d] + \frac{1}{2} \sum_{j=1}^{p} \sum_{k=1}^{p} w_j w_k E[x_j x_k], \tag{1.4}$$

where the weights are treated as constants and therefore moved outside the expectations.

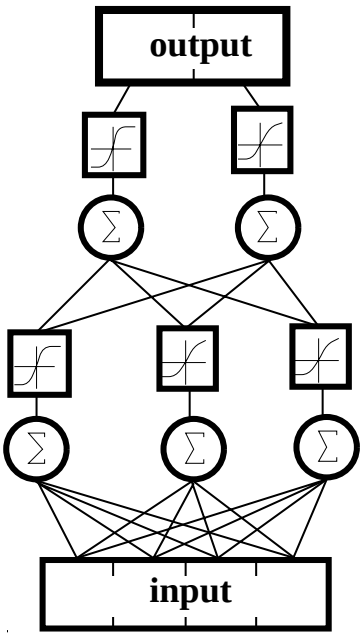

Figure 1.2: A multilayer neural network.

A plot of the cost function in eq. (1.4) versus the weights $w_1, x_2, ..., wp$, yields an *error-performance surface*. This surface has a global minimum in which the optimal weights and the minimum value J_{min} are to be found. These can be determined by differentiating the cost function J with respect to w_k, that is, by calculating the *gradient* of the error surface:

$$\nabla_{w_k} J = \frac{\partial J}{\partial w_k} = 0 \qquad k = 1,2,\ldots,p. \tag{1.5}$$

Differentiating eq. (1.5) with respect to w_k gives:

$$\begin{aligned} \nabla_{w_k} J &= -E[x_k d] + \textstyle\sum_{j=1}^{p} w_j E[x_j x_k] = 0 \\ &\Longrightarrow \textstyle\sum_{j=1}^{p} w_j E[x_j x_k] = E[x_k d] \qquad k = 1,2,\ldots,p. \end{aligned} \tag{1.6}$$

This system of p equations is known as the Wiener-Hopf equations.

To compute the optimum weights w_j for eq. (1.6) it is required to calculate the inverse of a p-by-p matrix. One way to avoid this is to use the *method of steepest descent*, where the weights are adjusted in an iterative fashion moving them towards the optimum solution. The update of the weights should be in the opposite direction to the gradient vector whose elements are defined by $\nabla_{w_k} J$. According to the method of steepest descent, the update of weight $w_k(n)$ at iteration n is defined by:

$$\Delta_{w_k}(n) = -\eta \nabla_{w_k} J(n), \qquad k = 1,2,\ldots,p, \tag{1.7}$$

where η is a positive constant called *learning rate*. The updated values $w_k(n+1)$ can now be expressed in the old value of weight w_k and its adjustment in eq. (1.7):

$$w_k(n+1) = w_k(n) + \Delta_{w_k(n)}(n) = w_k(n) - \eta \nabla_{w_k} J(n), \qquad k = 1,2,\ldots,p. \tag{1.8}$$

Using eqs. (1.6) and (1.8), the updated weight can be expressed as:

$$w_k(n+1) = w_k(n) + \eta\Big[E[x_k d] + \sum_{j=1}^{p} w_j(n) E[x_j x_k]\Big], \qquad k = 1,2,\ldots,p. \tag{1.9}$$

Eq. (1.9) requires knowledge about both the $E[x_k d]$ and $E[x_j x_k]$ expectations. This is not possible when the filter operates in an *unknown environment*, where only estimates are available. The *least-mean-square (LMS) algorithm* includes these estimates to handle an unknown environment:

$$\widehat{E}[x_j x_k] = x_j(n) x_k(n)$$

and

$$\widehat{E}[x_k d] = x_k(n) d(n),$$

The weight update of eq. 1.9 can now be expressed as:

$$\begin{aligned} \widehat{w}_k(n+1) &= \widehat{w}_k(n) + \eta\Big[x_k(n)d(n) - \textstyle\sum_{j=1}^{p} \widehat{w}_j(n) x_j(n) x_k(n)\Big] = \\ &= \widehat{w}_k(n) + \eta\Big[d(n) - \textstyle\sum_{j=1}^{p} \widehat{w}_j(n) x_j(n)\Big] x_k(n) = \\ &= \widehat{w}_k(n) + \eta[d(n) - y(n)] x_k(n) \qquad k = 1,2,\ldots,p. \end{aligned} \tag{1.10}$$

Unlike the method of steepest descent, the LMS algorithm is able to operate in unknown environments. The weight vector $\widehat{\mathbf{w}}(n)$, representing an estimate of $\mathbf{w}(n)$, follows a random trajectory (instead of a precisely defined trajectory along the error surface).

1.3 Back-Propagation Algorithm

Multilayer perceptrons have been applied successfully to a variety of real-world problems. If the network is trained in a supervised manner, the use of the *back-propagation algorithm*, or one of its variations, is by far the most popular. It is based on an error-correction learning rule and an iterative algorithm based on steepest descent to update the weights (as such it might be considered as a generalization of the LMS algorithm to multilayer networks). To improve this algorithm a variety of different training techniques, all based on the original backpropagation algorithm [Rumelhart-86], have been proposed in the literature [Moreira-95]. However, since none of them is showing an improved performance on a wide class of problems, the original on-line backpropagation algorithm is used in this report.

The backpropagation algorithm consists of two different phases: a forward pass and a backward pass. In the forward phase, a pattern from the pattern set is presented to the network. The inputs are propagated through the network producing the system response. During this phase, all the weights in the network are fixed. The system's actual output is compared with the desired output. Based on the outcome of this comparison the algorithm performs a backward phase in which the weights are updated. For a complete derivation of the backpropagation algorithm see, for example, [Rumelhart-86].

Summary of the Back-Propagation Algorithm

The summary consists of five separate steps. As a complement to the definitions in section 1.1 and 1.2, some general notations are defined first:

- The iteration n refers to the nth training example presented to the network.
- The input and output layer are defined as layer $l = 1$ and $l = L$, respectively. If a network has one hidden layer the number of layers is, therefore, equal to three.
- The jth element of the input pattern is denoted by x_j.
- $w_{ij}^{(l)}$, denotes the weight connecting neuron i in layer $l-1$ with neuron j in layer l. Its weight update in the backward phase is denoted by $\Delta w_{ij}^{(l)}$.
- The threshold of neuron j in layer l is denoted by $\theta_j^{(l)}$. For notational convenience it is represented by a weight $w_{0j}^{(l)} = \theta_j^{(l)}$ connected to a fixed input equal to -1.
- The internal activity level of neuron j in layer l is denoted by $v_j^{(l)}$.
- The activation function, which calculates the non-linear output $y_j^{(l)}$ of neuron j, is denoted by f.
- The error signal of output neuron j is defined as:

$$e_j^L = d_j^L - y_j^L,$$

  where d_j is the desired response for neuron j.

- The mean squared error, E_{av}, is obtained by summing the instantaneous sum of squared errors:

$$E(n) = \frac{1}{2}\sum_j (e_j^L)^2(n),$$

  over all n and then normalizing with respect to the number of patterns N:

$$E_{av}(n) = \frac{1}{N}\sum_{n=1}^{N} E(n).$$

- η is the learning rate and α is the momentum term.

Figure 1.3 gives an overview of the signal-flow and the location of some variables described in this section.
Below follows a detailed description of the on-line backpropagation algorithm for pattern set $\{(x_i, d_i)$ *n = 1,2,...,N*$\}$ that has been used in the rest of this report.

1. **Initialization** Start by initializing all weights. Because the hyperbolic tangent is used as activation function, all weights are randomly chosen from a uniform distribution on the interval *(-0.77, 0.77)* according to [Thimm-96].
2. **Pattern presentation** An arbitrary pattern from the training set is presented to the network.

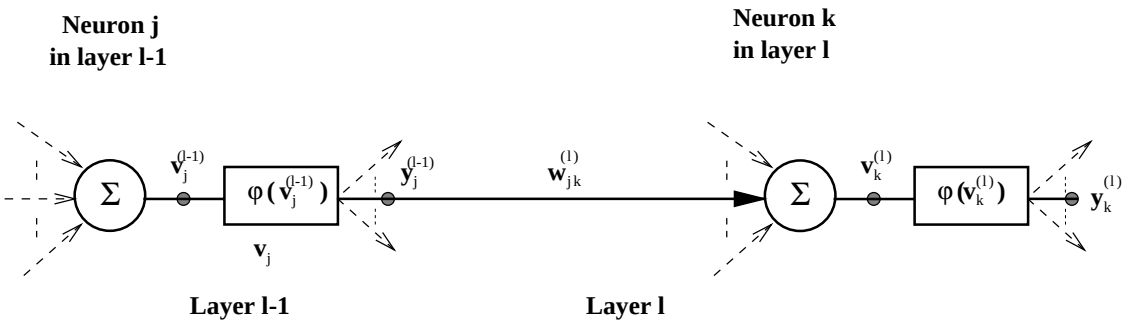

Figure 1.3: Signal-flow and the location of some variables described.

3. Forward propagation The inputs are propagated through the network producing outputs $y_j^{(L)}(n)$. The net activity level $v_j^{(l)}(n)$ for neuron j in layer l is:

$$v_j^{(l)}(n) = \sum_i w_{ij}^{(l)} y_i^{(l-1)}(n).$$

The algorithm used in this report employs a hyperbolic tangent (eq. 1.2) as non-linear activation function:

$$y_j^{(l)}(n) = f(v_j^{(l)}(n)) = \frac{2}{1 + exp(-v_j^{(l)}(n))} - 1.$$

Given these definitions, the error signals are computed according to:

$$e_j(n) = d_j(n) - y_j^{(L)}(n).$$

4. Backward propagation The main feature of the backpropagation algorithm is that the weight updates can be formulated as a backward propagation of the error signals. There are two main cases, depending on whether neuron j belongs to the output layer or not:

$$\delta_j^{(L)}(n) = e_j(n) f'(v_j^{(L)}(n)),$$

if j is an output neuron; otherwise:

$$\delta_j^{(l)}(n) = (\sum_k \delta_k^{(l+1)}(n) w_{jk}^{(l+1)}(n)) f'(v_j^{(l)}(n)).$$

The weight update formula below uses a slightly changed expression from the least-mean-square algorithm:

$$w_{ij}^{(l)}(n+1) = w_{ij}^{(l)}(n) + \eta \delta_j^{(l)}(n) y_i^{(l-1)}(n) + \alpha[w_{ij}^{(l)}(n) - w_{ij}^{(l)}(n-1)].$$

The expression within brackets that is being multiplies with a momentum term α, which has been proven to speed up the rate of learning, yet avoiding the risk of instability in the learning process.

5. Iterate Repeat step step 2-5 until some convergence criterion is satisfied, for example when $E_{av}(n)$ is sufficiently low.

Chapter 2

Weight Discretization in Neural Networks

To be able to profit from the massive parallelism inherent in neural network models, hardware implementations are essential. This has led to a large variety of implementations using digital and analog electronics, optics, and hybrid techniques. Even though these implementations are largely different, a common denominator is the mapping of neural network algorithms onto reliable, compact, and fast hardware. Any hardware implementation has to optimize three main constraints: accuracy, space, and processing speed. The design of hardware implementations is governed by a balancing of these criteria. An analog implementation, for example, is very efficient in terms of chip area and processing speed, but this comes at the price of a limited accuracy of the network components. In general, this amounts to a trade-off between the accuracy of the implementation and the reliability of its performance.

In this report, the quantization of the network weights will be closer investigated; it is required to have quantized weights in hardware when device operation is quantized or because quantization leads to a far more compact implementation reducing VLSI surface area. A detailed review of the effects of quantization in neural networks and the design of weight discretization algorithms that limit the required accuracy is given in [Moerland-97]. In this section, the basic principles and one specific weight discretization algorithm are outlined.

2.1 Quantization Effects

The use of very high precision cannot be matched with the goal of developing fast and compact hardware implementations. While in digital implementations a high numerical precision is too area consuming, it is incompatible with the system noise present in analog implementations. Therefore, hardware implementations of neural networks typically use a representation of the network parameters with a limited accuracy. For example, in Philips' L-Neuro 1.0 architecture, which allows the implementation of feedforward networks and on-chip backpropagation training, 16-bit weights are used during the training process and only 4-bit or 8-bit weights are employed during recall [Mauduit-92]. The need for a further reduction of the accuracy, while retaining a satisfactory network performance, has also led to various *weight discretization* algorithms, especially designed for this purpose.

These approaches can be divided into three categories corresponding to the three different training modes for neural network hardware:

Off-chip learning In this case, the chip is not involved in the training process, which is performed on a computer using high precision. The resulting weights from the training process are discretized and downloaded on the chip. Only the forward propagation pass in the recall phase is performed on-chip which makes it possible to study the quantization effects mathematically. Various studies have

indicated that the accuracy needed in the on-chip forward pass is around 8 bits [Holt-93]. An example using this technique is the application of the analog ANNA chip used for high speed character recognition [Säckinger-92]. A high precision (32-bit floating point) network is mapped on the ANNA chip which uses a 6-bit weight resolution and a 3-bit resolution for the neuron inputs and output. The chips recognition accuracy is just slightly less than the one obtained with floating-point calculations.

Chip-in-the-loop learning Here, the neural network hardware is used during training, but only in the forward propagation pass. The calculation of the new weights is done off-chip using a computer (and floating point calculations). The new weights are then quantized and downloaded on the chip before each forward pass. Several learning algorithms that take advantage of the fact that the limited precision of the weights only plays a role in the forward pass, have been proposed. The weight discretization algorithm that is described in the next section is of the chip-in-the-loop kind and has shown promising results in limiting the weight precision [Fiesler-88].

On-chip learning In this case, the training of the neural network is done entirely on-chip which which offers the possibility of incremental training. The backpropagation algorithm is highly sensitive to the use of limited-precision weights and that the training fails if the accuracy is lower than 16-bits [Asanović-91]. This is mainly because the weight updates are often smaller than the quantization steps which prevents the weights from changing. A successful application of an on-chip learning technique is given by the analog electronic chip proposed by [Leong-95]. This chip has been applied successfully to some classification problems by training it with the combined search algorithm and semi-parallel weight perturbation algorithms using only a 6-bit weight accuracy. Another example is the design of a gradient descent learning rule, implemented on a FPGA (Field Programmable Gate Array) [Girau-96].

2.2 Weight Discretization Algorithm

The original discretization method, see [Fiesler-88] starts by training the network with the backpropagation algorithm using continuous-valued weights. Next, the continuous weights are discretized by mapping them to the closest discretization level using a staircase shaped multiple thresholding function. The so-created quantized weights are then used for the forward propagation pass, see section 1.3, through the network. The errors obtained, which are based on the difference between the actual and desired network outputs, are subsequently used to update the continuous weights during the backward propagation pass, see section 1.3. The discretization method described in [Fiesler-88] uses d discretization levels with $d = 2$, 3, 5, 7, or 9. The discretization levels are symmetric around zero, except for $d = 2$, and are equidistant:

$$\left\{ n - \left\lfloor \frac{d+1}{2} \right\rfloor \;\middle|\; n = 1, 2, ..., d \right\}.$$

This discretization method can be used either in off-chip or chip-in-the-loop learning mode. Chip-in-the-loop learning is especially useful in analog implementations, where it is of crucial importance to incorporate the non-idealities of the system in the forward pass. The weight discretization algorithm combined with off-chip learning results in a final quantized network, the weights of which can be downloaded on the chip. Naturally, the data set used to train the neural network has to be carefully selected and should be representative. To be able to retrain the network if the network performance degrades, it is important to have a flexible hardware construction were the individual weights can be replaced in an updating phase.

Chapter 3

Connectionist Quantization Functions

As explained in the previous chapter, to fully profit from the inherent parallel processing of neural networks, the development of hardware implementations is essential. However, these hardware implementations often differ in various ways from the ideal mathematical description of a neural network model. It is, for example, required to have quantized network parameters, in both electronic and optical implementations of neural networks. For example, a recent analog optical implementation of a multilayer perceptron is characterized by quantized weights represented on a liquid crystal television screen that provides a maximum of 256 grey levels [Saxena-95].

In this chapter, the weight discretization technique described in section 2.2 is further investigated. First, six new threshold quantization functions, and there inclusion in the weight discretization algorithm, are presented. These, quantization functions have been selected to minimize the number of weight levels without deteriorating network performance. Secondly, details about the benchmarks, the experimental setup, and the implemented training scheme are described. Finally, hardware implementation details are discussed together with experimental results and conclusions.

3.1 Weight Discretization Algorithm

The original weight discretization algorithm maps the continuous weights to discrete weight levels, obtained from a quantization function, before each forward pass through the network. The weights are discretized by mapping them to the discrete weight levels to which they are closest.

In order for the quantization functions to be flexible, a parameter d is included that indicates the number of weight discretization levels. The number of levels used in the experiments described in this report are subsequently 2, 3, 5, 7, 15, and 31. These values have been chosen based on the fact that in digital hardware implementations generally 2^d-1 discretization levels are available ($d-1$ bits plus a sign bit). The cases of $d=2$ and $d=5$ have been added to obtain a more precise idea of the effects of having only a small (or even minimal for $d=2$) number of discretization levels.

To minimize the number of weights in the network, it is also desirable to have the zero weight level included in the quantization functions.

3.1.1 Quantization Functions

Six quantization functions have been implemented and tested. Mathematical descriptions and clarifying pictures, based on seven discretization levels, are presented below. First some general notations are defined:

[1] A condensed version of this chapter can be found in [Lundin-96].

- N is the number of discretization levels.
- $level_1$ is the left-most discretization level.
- $level_N$ is the right-most discretization level.
- W_{max} is the maximum absolute weight value of the pre-trained continuous network.
- W_+ is the maximum positive weight value of the pre-trained continuous network.
- W_- is the minimum negative weight value of the pre-trained continuous network.
- LCZ is the index of the discretization level which is closest to zero.
- The mean value of the weights calculated over the entire pre-trained continuous network is denoted by $E(w)$.

The following quantization functions have been implemented and tested:

Symmetrical This is the original quantization function from [Fiesler-88]. The resulting discretization levels are symmetric around zero and equidistant, with a step size of one between the weight levels:

$$level_i = -level_{(N+1)-i} = N(DIV)2 - i + 1 \qquad , \qquad i = 1 \ldots N(DIV)2$$
$$level_{N(DIV)2+1} = 0 \qquad \text{if} \quad N(MOD)2 = 1.$$

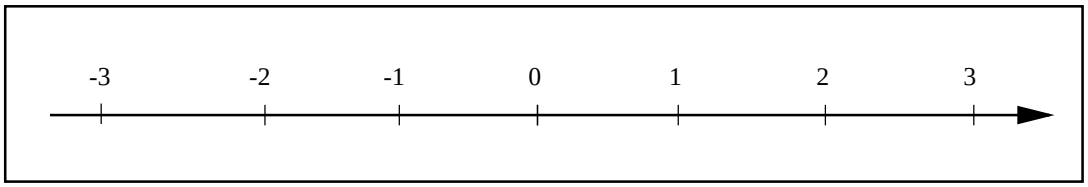

Figure 3.1: Symmetrical.

The intuitive disadvantage of the symmetrical quantization function is that it does not incorporate any knowledge about the weights of the pre-trained continuous network. This information is used in the new quantization functions such that the mapping to discrete weights does not completely perturb the results of continuous pre-training.

W**max This quantization function divides the interval $[-W_{max},+W_{max}]$ in equidistant levels, resulting in weight levels which are symmetric around zero:

$$level_i = \frac{2 \times Wmax}{N-1}(i-1) - W_{max} \qquad , \qquad i = 1 \ldots N.$$

W**max_adapt This is a straightforward modification of the previous quantization function. It consists of three different phases and uses both W_- and W_+. In the first phase, the interval $[W_-,W_+]$ is divided in equidistant levels. Secondly, the index of the level closest to zero, LCZ, is determined and finally the level that corresponds to this index is set to zero:

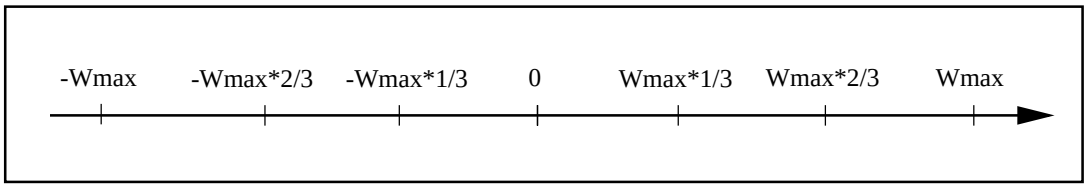

Figure 3.2: W_{max}.

Phase 1: Calculate

$$level_i = \frac{W_+ - W_-}{N-1}(i-1) + W_- \qquad , \qquad i = 1 \ldots N$$

Phase 2: Locate LCZ

$$LCZ = arg\left(min_i |level_i|\right).$$

Phase 3: Adjust

$$level_{LCZ} = 0.$$

In [Bellido-93] it is concluded that the weight distribution in a neural network resembles a Gaussian-

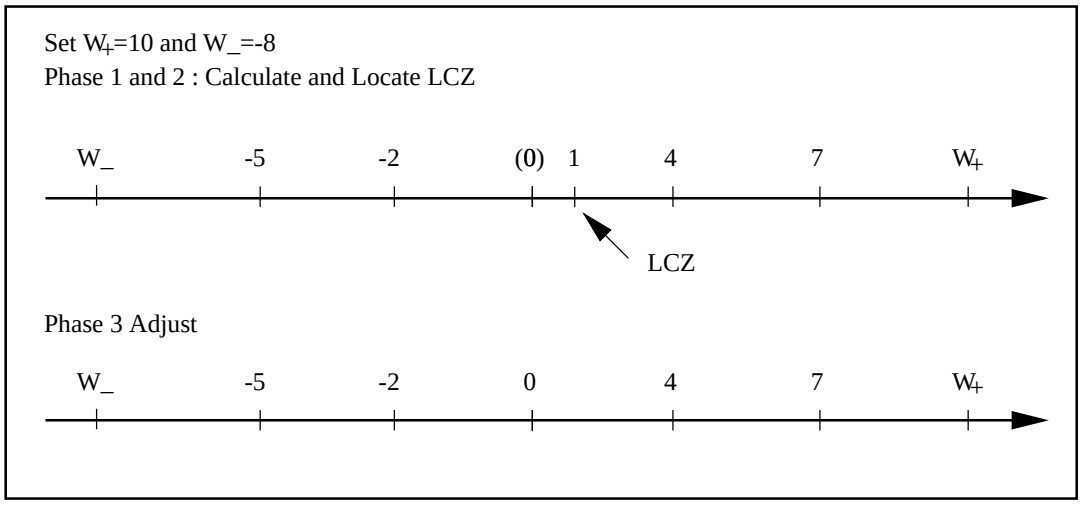

Figure 3.3: W_{max_adapt}.

like distribution. This means that there are many weights with small values and a few weights with large values. It is also shown that these Gaussian distributions can often be skewed or peaked depending on the problem. The following three quantization functions are therefore based on powers-of-two to approximate such a Gaussian distribution.

Power_of_two_W_{max} This quantization function uses the intervals $[-W_{max},0]$ and $[0,+W_{max}]$ which are divided using powers-of-two:

$$level_i = -level_{N+1-i} = W_{max}(\frac{1}{2})^{i-1} \quad , \quad i = 1 \ldots N(DIV)2$$

$$level_{N(DIV)2+1} = 0 \qquad \text{if} \quad N(MOD)2 = 1.$$

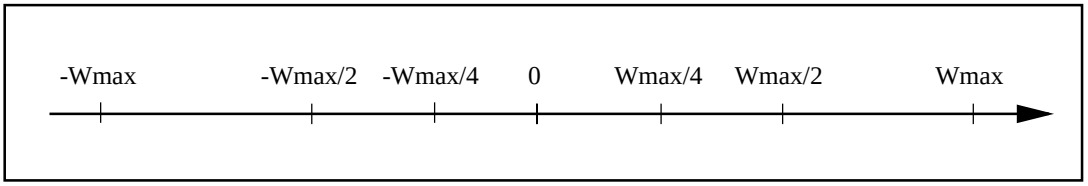

Figure 3.4: Power_of_two_W_{max}.

Since the weight distribution in a neural network also could be skewed, the following two quantization functions use information about the mean value, $E(w)$, of the continuous weights.

Power_of_two Instead of only using the W_{max} value this function divides both intervals, $[-W_{max},E(w)]$ and $[E(w),W_{max}]$, using powers-of-two:

$$\left.\begin{array}{l} level_i = E(w) - (W_{max} + E(w))(\frac{1}{2})^{i-1} \\ level_{N+1-i} = E(w) - (E(w) - W_{max})(\frac{1}{2})^{i-1} \end{array}\right\} \qquad i = 1 \ldots N(DIV)2$$

$$level_{N(DIV)2+1} = E(w) \qquad \text{if} \quad N(MOD)2 = 1.$$

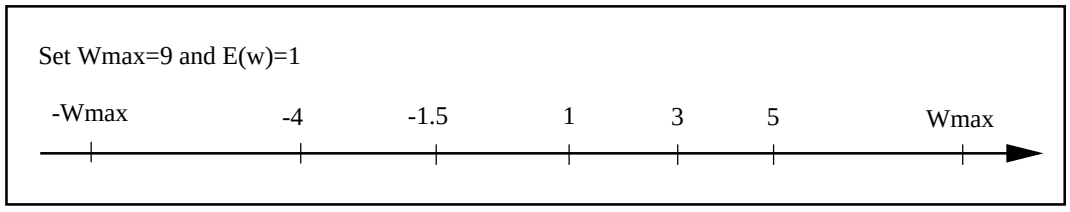

Figure 3.5: Power_of_two.

Power_of_two_adapt This is a modification of the previous quantization function which divides both intervals $[W_-,E(w)]$ and $[E(w),W_+]$ using powers-of-two:

$$\left.\begin{array}{l} level_i = E(w) + (W_- - E(w))(\frac{1}{2})^{i-1} \\ level_{N+1-i} = E(w) - (E(w) - W_+)(\frac{1}{2})^{i-1} \end{array}\right\} \qquad i = 1 \ldots N(DIV)2$$

$$level_{N(DIV)2+1} = E(w) \qquad \text{if} \quad N(MOD)2 = 1.$$

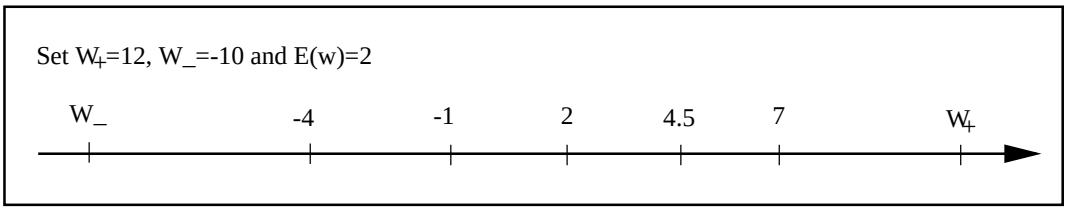

Figure 3.6: Power_of_two_adapt.

Benchmark	Network Topology[1]	Pattern Set Sizes			Number of Runs
		Train.	Val.	Test	
Auto-mpg	7-3-1	196	98	98	10
Sunspot	12-2-1	105	52	52	10
Cancer	9-6-2	350	174	175	10
Wine	13-6-3	89	44	45	10
Diabetes	8-6-2	384	192	192	10
Digit	64-64-10	1000	500	500	3

Table 3.1: Benchmarks characteristics used for experiments run on quantization function.

3.2 Experimental Setup

To get relevant results, only real-world benchmarks have been used for this report. In this section, details about the benchmarks used and the experimental setting are described.

3.2.1 Benchmarks and Parameter Settings

To evaluate the performance of the six quantization functions, a series of experiments on six real-world benchmark problems, two *approximation* problems (Auto-MPG and Sunspot) and four *classification* problems (Wine, Cancer, Diabetes, and Digit) have been performed. The number of runs performed, topology, and pattern set sizes for these benchmarks are listed in Table 3.1.

The input data are scaled to the interval [0,1] if not stated otherwise. Below follows a detailed description of the different benchmark problems.

Approximation problems

Auto-mpg (7,1) concerns city-cycle fuel consumption of cars in miles per gallon, to be predicted in terms of three multi-valued discrete and four continuous attributes [Murphy-94]. Input values have been scaled to the interval [-1,1].

Sunspot (12,1) contains sunspot activity for the years 1700 to 1990 [Murphy-94]. The task is to predict the sun spot activity for one of the years, given the preceding twelve years.

Classification problems

Cancer(9,2) is a diagnosis of breast cancer [Prechelt-94]. The problem is to classify a tumor as either benign or malignant on cell descriptions. Input parameters are for example the cell size and shape, the amount of marginal adhesion and clump thickness.

[1] # of neurons in the input-hidden-output layer.

Wine (13,3) is the results of a chemical analysis of wines grown in the same region in Italy but derived from three different cultivars [Murphy-94]. The analysis determined the quantities of 13 constituents found in each of the three types of wines.

Diabetes (8,2) is a diagnosis of diabetes among Pima Indians [Prechelt-94]. Input parameters are for example the personal data (age, number of times pregnant) and the result of medical examination (blood pressure, body mass index, and results of glucose tolerance test etc).

Digit(64,10) is a subset from handwritten digits of the NIST Special Database 3 [Garris-92]. Each pixel is represented by an eight bit value; the patterns are equally distributed over ten digits and scaled to fit into an image of 8×8.

In fact, to save time, only four quantization functions were tested on the Digit benchmark. These quantization functions were selected on the basis of the outcome of the experiments on the other benchmark problems.

In all experiments a fully interlayer connected multilayer perceptron, with only one hidden layer, was used. The non-linear activation function employed is the hyperbolic tangent, while for approximation problems a linear activation function has been used in the output layer of the network. The desired output values for approximation problems are namely real-valued in the interval [-1,1], while the other benchmarks are classification problems with desired output values of -1 and $+1$. For each run the network was initialized with different random weights in the interval [-0.77,0.77] according to [Thimm-96]. Furthermore, a learning rate of 0.5 was chosen according to [Moreira-95]; only for the Digit benchmark a learning rate of 0.1 has been used. Moreover, a momentum term of 0.9, and a flat-spot constant of 0.1 have been used. In all experiments, the network training was done with the on-line backpropagation algorithm as described in section 1.3. The tables with experimental results for the quantization functions are presented in appendix A.

3.2.2 Generalization

To validate the quality of a network during training and to test the performance afterwards, each of the available pattern sets was partitioned, respecting an equal distribution of the different classes, in three sets: a *training* set with 50% of the total patterns and a *validation* set and a *test* set, each of these with 25% of the patterns. The evaluation of the generalization performance is done by *cross validation* with *early stopping*, where the decision to stop training is based on the error on the validation set. This is a good way to avoid over-fitting of the network to the particular training set used. For the experiments in this section, a description of cross validation with early stopping in [Prechelt-94] has been used as a basis.

The network performance is indicated by the ***squared error percentage*** that is normalized for the number of output neurons and the number of patterns:

$$E = 100 \cdot \frac{1}{N \cdot P} \sum_{p=1}^{P} \sum_{n=1}^{N} (d_{pn} - y_{pn}^{(L)})^2, \tag{3.1}$$

where N is the number of output neurons of the network and P is the number of patterns in the data set considered.

The network is trained using the training set and after every five epochs the validation set is presented to measure the generalization error. Two different measures have been used to decide when to stop training. The first one is the ***training progress*** in a training strip of five at epoch k:

$$Prog(k) = 1000 \cdot \left(\frac{\sum_{t=k-4}^{k} E(t)}{5 \cdot \min_{t \in k-4,\ldots,k} E(t)} - 1 \right), \tag{3.2}$$

where $E(k)$ is the training error at epoch k. $Prog(k)$ gives a measure of the change in evolution of the training over the last five epochs.

The second one is the ***generalization loss*** at epoch k:

$$GL(k) = 100 \cdot \left(\frac{E_{va}(k)}{\min_{t \leq k} E_{va}(t)} - 1 \right), \tag{3.3}$$

$E_{va}(k)$ being the validation error at epoch k. $GL(k)$ corresponds to the relative increase of validation error with respect to its minimal value so far. These two parameters are measured every five epochs, after presenting the validation set.

Training is stopped either when the training progress sinks below 0.1 or when the generalization loss went beyond a threshold of 5.0 in five consecutive measurements (detecting overfitting). This leads to the following ***training scheme***:

```
Training Scheme:
*************
1.  k=0
2.  loss=0
3.  initialize the network, i.e. randomize the weights
4.  Repeat
5.     Train network, according to the backpropagation algorithm,
       for one epoch using all the patterns in the training set
6.     k = k + 1
7.     IF k MOD 5 = 0 THEN
8.        Compute E_va, E_opt, GL(k), and Prog(k)
9.        IF ( GL(k) > 5 ) THEN
10.          loss = loss + 1
11.       ELSE
12.          loss = 0
13.    END {if}
14. UNTIL (loss>5 OR Prog(k)<0.1 OR k>maximum number of iterations)
15. Restore the network parameters according to the point were E_opt
    was achieved.
16. Present the test-set to the network
```

After stopping the training, the test set is presented using the network weights that provided the lowest validation error. The results presented for the benchmarks are all based on the values measured for that optimal network. If weight discretization is to be used, the above algorithm is executed for each discretization level according to 3.1.

3.3 Hardware Implementation Details

Not all of the described quantization functions are equally well suited for hardware implementation. Only the quantization functions which result in weight levels that are symmetric around zero and equidistant, are suitable. These conditions are satisfied by the *Symmetrical*, W_{max}, and *Power_of_two*_W_{max} quantization functions. In this case the discrete weights can namely be normalized to the interval [-1,1] by dividing them by the maximal weight value. The normalized weights can

then be encoded as binary numbers. The scaling of the weights can be compensated for by rescaling the gain (steepness) of the activation function. The *Power_of_two_*W_{max} quantization function has the added advantage that the normalized discrete weight values are restricted to powers-of-two. Therefore, simple shift registers can be employed to substitute the more complex multipliers [Marchesi-93].

The other three quantization functions W_{max_adapt}, *Power_of_two*, and *Power_of_two_adapt* lead to weight levels that are not symmetric around zero and are therefore not suited for hardware implementation; these are of more theoretical interest.

3.4 Discussion of Results for Quantization Functions

To evaluate the experimental results, two different performance measurements have been used. For the approximation problems (Auto-MPG and Sunspot), see tables A.1 and A.2, the normalized mean square error on the test set is most significant. For the classification problems (Cancer, Wine, Diabetes, and Digit), see tables A.3, A.4, A.5, and A.6, the percentage of misclassified test patterns has been used. A pattern is considered correctly classified whenever the highest network output corresponds to the correct class.

Firstly, it was observed that the results depend on the type of benchmark problem used. In fact, the results can be divided into two classes: one for the classification problems and the other for the approximation problems. Therefore, the results of these two classes are discussed separately.

A ranking system was used to evaluate the outcome of the experiments. For each benchmark and discretization level, the test set results of the different quantization functions were ranked from one to six. The ranking scores were then added within each quantization function and class (approximation or classification). This gives an overall performance for all quantization functions in each class.

Starting with classification problems, the *Symmetrical* quantization function performs good using only two or three discretization levels. In fact, for the Cancer and Diabetes benchmarks the same performance is not obtained by any of the other quantization functions until using seven or more discretization levels. For example, with the Diabetes benchmark using $D=3$, the percentage of misclassification for the *Symmetrical* quantization function is 25.10, see table A.5. The *Power_of_two* quantization function is the first to produce a better result with $D=7$. It should be noted that the *Symmetrical* quantization function performs good for all discretization levels on the Digit benchmark. Of the five new quantization functions the *Power_of_two*, *Power_of_two_*W_{max}, and W_{max} quantization functions perform best. For these functions, $D=7$ is sufficient for both power-of-two based quantization functions, while the results for W_{max}, using $D=15$, are comparable with those from the continuous pre-training. Less good performance was obtained for the W_{max_adapt} and *Power_of_two_adapt* quantization functions.

For the approximation problems, the *Symmetrical* and to a less extent the *Power_of_two* quantization functions give an overall poor performance. Note, that this differs from the results for the classification problems. The *Power_of_two_*W_{max},W_{max} and W_{max_adapt} quantization function perform best and results for $D=15$ are comparable with the ones from continuous pre-training. For example, using 15 discretization levels the *Power_of_two_*W_{max} quantization function gives a mean square error percentage of 0.27 on the Auto-MPG benchmark, see table A.1, which is almost as good as in the continuous case. The *Power_of_two_adapt* performs well on the Auto-mpg problem, but less good on the Sunspot problem.

3.5 Conclusion for Quantization Functions

The performance of six different quantization functions, in training multilayer perceptrons with a small number of discrete weight levels, has been evaluated. Two of these quantization functions, W_{max} and *Power_of_two_*W_{max}, show generally good results in a series of experiments including both classification and approximation problems. For these quantization functions, only 15 discrete weight

levels are sufficient to obtain nearly the same performance as for a network with continuous weight values.

Both quantization functions are also well suited for hardware implementation, for example by using only shift registers when $Power\_of\_two\_W_{max}$ is implemented. The results also indicate that for classification problems the *Symmetrical* quantization function performs good when using only ternary weights $\{-1, 0, 1\}$ or even binary weights $\{-1, 1\}$ [Bartlett-96].

Chapter 4

Pruning of Neural Networks

As described in the introduction, a major drawback for multilayer neural networks is the choice of a suitable network *topology*. An unsuitable topology will increase training time, lower generalization performance on unseen test data [Gosh-94], and in some cases even cause non-convergence. Furthermore, when developing hardware implementations, small networks will reduce the required chip size. Several methods for which the choice of initial network topology is less critical have been proposed in the literature. Examples are early stopping methods [Finnoff-93], explicit regularization [Bishop-95], network growing [Kwok-95], and network pruning [Reed-93]. The latter will be considered in this chapter.

The basic idea of pruning is to start training a network that is considered sufficiently big to ensure convergence. When convergence is reached certain weights/neurons are removed, and the network is retrained. This procedure is repeated until the best possible network is found. The reduced network is likely to perform better on unseen data. It is important to make an evaluation of the numerous existing pruning algorithms since none of them is guaranteed to find the minimal topology for a given problem. However, there is always a trade-off between training time, generalization performance, and network size.

The optimal networks found by pruning are then further trained with the weight discretization algorithm and the best performing quantization functions of chapter 3. Also in this stage, pruning is performed to obtain compact multilayer perceptrons with good generalization performance.

In this chapter, the results from chapter 3 are combined with pruning methods to find small networks with good generalization performance. Firstly, six pruning methods are presented together with details about the additional benchmarks used in this chapter. Secondly, the experimental settings and the implemented general pruning scheme are described. Finally, an elaborate discussion of the results and conclusions is presented.

4.1 Pruning Methods

For this report, a total of six pruning methods have been implemented:

1. **Randomized pruning** removes a fixed percentage[1] of connections randomly chosen from the network. Results of this pruning strategy should serve as a basis for comparison.

2. **Smallest weight pruning** is based on the concept that connections with a small weight value are less important [Gosh-94]. Therefore, it removes a fixed percentage[1] of the smallest weights in the network.

[1] In this report 10 per cent is used.

3. **Autoprune**. W. Finnoff *et al.* [Finnoff-93] define a *test statistic* for testing the significance of the deviation of a weight from zero:

$$T(w_i) = log\left(\frac{\left|\sum_p w_i - \eta(\partial E/\partial w_i)_p\right|}{\eta\sqrt{\sum_p((\partial E/\partial w_i)_p - (\overline{\partial E/\partial w_i}))^2}}\right). \tag{4.1}$$

   Based on this T-statistics, a connection is removed if the probability that it becomes zero is high. According to [Finnoff-93], 35 per cent of the connections are pruned in the first pruning step and 10 per cent in all the consecutive pruning steps.

4. **Lambdaprune** is based on the same test statistics of equation 4.1 but [Prechelt-95] proposes in this method to adapt the pruning strength. Weights satisfying $T(w_i) < \lambda * \mu_T$, where μ_T is the mean of the test statistics, are pruned. λ is defined accordingly:

$$\lambda = \lambda(GL) = 100 * \lambda_{max}\left(1 - \frac{1}{1 + \frac{GL}{\alpha}}\right), \tag{4.2}$$

   where $\lambda_{max} = 2/3$ and $\alpha = 2$ according to [Prechelt-95].

5. **Smallest variance pruning**. J. Sietsma and R. J. F. Dow [Sietsma-91] proposed to remove a fixed percentage[1] of the connections with the smallest contribution (that is, $w_{ij}^{(l)} y_i^{(l-1)}$) variance on the training set. After pruning, the mean contribution of the removed connection is added to the corresponding bias.

6. **Node pruning** [Sietsma-91]. The hidden neuron with the smallest variance of its activation value on the training set is removed. The mean activation value of the removed hidden neuron times each of the outgoing weights, is added the corresponding bias of a neuron in the next layer.

4.2 Simulations

4.2.1 Benchmarks

To evaluate the performance of the six pruning methods, a series of experiments on six real-world benchmark problems, three *approximation* problems (Auto-MPG, Sunspot, and Leman) and three *classification* problems (Cancer, Wine, and Heart), have been performed. The number of runs, initial network topology, pattern set sizes, and number of connections before pruning for these benchmarks are presented in Table 4.1. Deliberately oversized topologies have been selected to be able to evaluate the pruning methods.

For *Node* pruning, experiments were only performed for the continuous-valued network on all the benchmarks except the heart benchmark. The tables with experimental results for the pruning methods are in appendix B. The same activation function, initial network parameter setting, and performance measurements as described in section 3.2.1, were used. The input data are scaled to the interval [0,1] if not stated otherwise. Below follows a complementary description of the new benchmark problems used in this chapter. A description of the other benchmarks can be found in section 3.2.1

Approximation problems

Leman (2,1) data set describes the cadmium distribution in Lake Geneva by mapping co-ordinates onto (μg cadmium)/(g water) (in the range of $0.08\mu g/g$ to $3.29\mu g/g$).

Benchmark	Network Topology	Pattern Set Sizes			Number of Runs	Total # of Connections Before Pruning	Error % test set[2]
		Train.	Val.	Test			
Auto-mpg	7-6-1	196	98	98	10	48	0.24
Sunspot	12-4-1	105	52	52	10	52	0.29
Leman	2-8-1	101	49	51	10	24	0.65
							% of miscl.test set[3]
Cancer	9-8-2	350	174	175	10	88	1.15
Wine	13-8-3	89	44	45	10	128	2.27
Heart	35-16-2	152	76	75	5	592	5.13

Table 4.1: Benchmarks characteristics used for experiments on pruning methods.

Classification problems

Heart (35,2) predicts heart disease [Prechelt-94]. The problem is to decide whether at least one of four major vessels is reduced in diameter by more than 50 %. The decision is made from input data as age, sex, habits, subjective patient pain description, and results from medical examination.

4.2.2 Experimental Setup

A complete pruning strategy has to decide *when to prune*, *which weight(s) or neuron(s)* to prune, and *when to stop* training. If the algorithm prunes too early (i.e. the data set is not satisfactorily learned), a generalization improvement might be overlooked. However, if the algorithm prunes too late, overfitting to the training data might already be too large. If too many connections are removed in one pruning step, the network performance can be unnecessarily destroyed. On the other hand, if only a few weights are removed in each pruning step, training time will increase dramatically.

The pruning methods outlined in section 4.1 decide how many connections to remove at a certain pruning step. *Randomize*, *Variance*, *Weight*, and *Node* pruning remove 10% of the connections in each pruning step, while *Autoprune* removes 35% of the connections in the first pruning step, and 10% in all later pruning steps. *Lambdaprune* has an adaptive pruning strength and removes in a pruning step all weights satisfying: $T(w_i) < \lambda * \mu_T$.

There is a big diversity in the literature on pruning methods in the decision when to prune and when to stop training. To deal with this problem, an overall pruning scheme is presented in this report, providing a fair comparison between the different pruning methods. Using eqs.(3.2) and (3.3) as described in 3.2.2 , the following ***overall pruning scheme*** was implemented:

```
Overall Pruning Scheme:
***********************
0.  Restore optimal network resulting from training scheme
1.  k=0
2.  GL_max=MAXIMUM_INTEGER
3.  loss=0
4.  Repeat
5.      Train network, according to the on-line backpropagation
        algorithm, for one epoch presenting all the patterns
        from the training set in random order to the network
6.      IF Autoprune OR Lambdaprune THEN
```

[2] From the experiments in Tables A.1-A.2
[3] From the experiments in Tables A.3-A.6

```
7.        compute the T(w_i) values
8.     IF Node pruning THEN
9.        compute the activation value variance
10.    IF Smallest Variance pruning THEN
11.       compute the contribution variance
12.    k = k + 1
13.    IF k MOD 5 = 0 THEN
14.       Compute E_va, E_opt, GL(k), and Prog(k)
15.    END
16.    IF (GL(k) > GL_max) THEN
17.       loss = loss + 1
18.    ELSE
19.       loss = 0
20.    GL_max=GL(k)
21.    IF ((k>=25) AND (loss >=2)) OR k>(maximum_number
       _of_ iterations) THEN
22.       Apply the pruning method selected and remove
          weights from the network
23.        k=0
24.        GL_max=MAXIMUM_INTEGER
25.        loss=0
26.    END {if}
27. UNTIL (global_iterations > (10*max_number_of_iterations))
          OR ((k >= 25) AND ((GL(k) >= 500)
          OR (Prog (k) <= 0.1))
28. Restore the network parameters according to the point
    were E_opt was achieved
29. Present the test set to the network
```

In this pruning scheme the decision when to prune is triggered by an increase in the error on the validation set in two successive strips (of five epochs). This criterion is based on the view that pruning should occur whenever the generalization behaviour (measured on the validation set) starts deteriorating. This criterion is chosen independent of E_{opt}, because a pruning step often results in a intermediate increase of GL which should not lead to yet another pruning step. The overall pruning scheme is preceded by the early stopping scheme on the initial network topology described in section 3.2.2.

An illustrative behaviour of the error curve during pruning is displayed in figure 4.1. The stars marked on the x-axis indicate the point where the validation set error of the network is improving. The circles on the x-axis indicate pruning steps.

4.3 Discussion of Results for Pruning

As in section 3.4, the results can be divided into two classes; namely the class of approximation problems (Auto-MPG, Sunspot, and Leman), see table B.1, B.2, and B.3, and the class of classification problems (Cancer, Wine, and Heart), see table B.4, B.5, and B.6. The results of the simulations of those two classes are discussed separately. Within each class, the pruning methods are evaluated by looking at the number of connections in the optimal network, i.e the network with the smallest mean square error on the validation set, and the generalization performance obtained on the test set. Finally, for each benchmark and discretization level, the test set results of the different quantization functions were compared. For an easy comparison of the different pruning methods, the benchmarks problems will be discussed separately, or pairwise, within each class.

Firstly, the results for classification problems, are discussed. For the Cancer and Heart benchmarks,

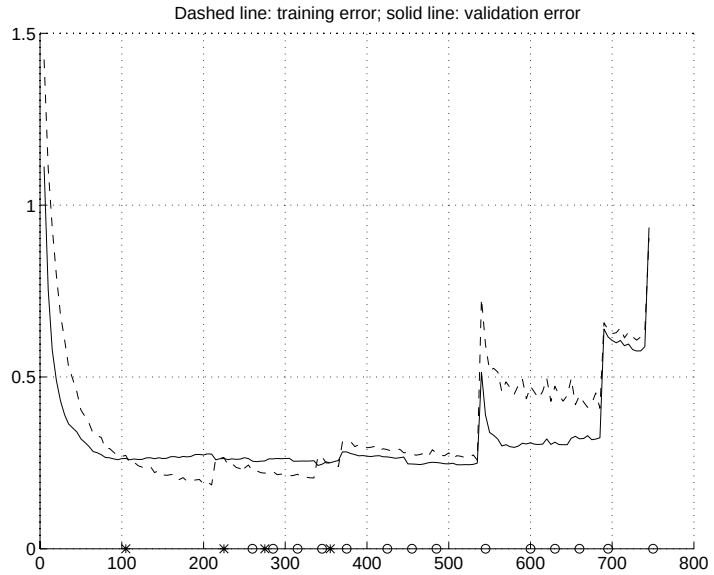

Figure 4.1: Error curves for on simulation on the Sunspot benchmark.[4]

it can be stated that all pruning methods perform worse than *Randomize* pruning, see B.4 and B.6. For *Lambdaprune* and *Autoprune*, the optimal network is the initial network. Due to heavy oscillations in the training and validation set error, see figure 4.2, *Lambdaprune* prunes too heavily, sometimes even 60% of the connections, in the initial pruning steps. *Autoprune* also prunes too heavily, mainly because that remove 35% of the connections in the first pruning step is too high. This perturbs the network to such a great extent that its generalization ability is destroyed. The pruning scheme in 4.2.2 requires two consecutive increments in generalization loss for pruning to take place. Therefore, the oscillatory behaviour also makes it difficult to decide *when* to prune. This has a big impact on the other three pruning methods: preventing pruning to take place or causing the generalization loss GL at the moment of pruning to be high.

For the Wine benchmark, the training and validation set error approach nearly asymptotically zero. The validation error at the optimal point will therefore be extremely low. A high relative increase in the validation set error then causes the pruning algorithm to stop if $GL \geq 500$, even when the absolute increase is low. A consequence is that all the pruning methods, except for $Variance$ pruning, perform bad.

The percentage of misclassification obtained by all the pruning methods for continuous networks on the Cancer and Heart benchmarks, is overall good. For example, for the Cancer benchmark, $Node$ pruning obtains 1.02 percentage of misclassification, a value even lower than the result obtained in Table A.3 (see also table 4.1). For Wine the generalization performance is less good, mainly due to the overfitting occurring because of the big network size.

Moreover, there is often lack of correlation between the mean square output error and the percentage of misclassification. For example, in table 4.2, the optimal mean square error percentage is 36.31% after 280 iterations, for a network with 292 connections. However, the lowest percentage of misclassification (on the validation set) is 19.73% reached after 605 iterations, for a network with only 140 connections. This indicates that minimizing a sum-of-squares error function is not optimal for classification problems. A more suitable choice would be the cross-entropy error function [Bishop-95].

Because the pruning results for classification problems are generally poor, a complete evaluation of

[4] Number of epochs on the x-axis and mean square error percentage on the y-axis.

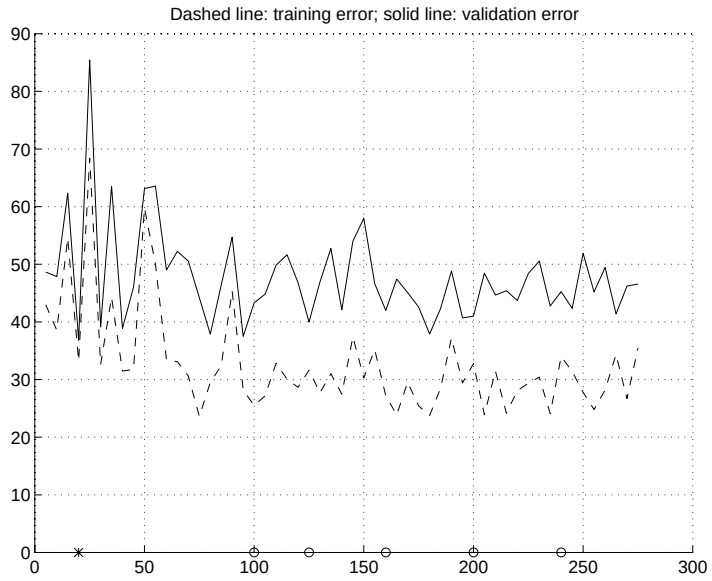

Figure 4.2: Error curves for one simulation on the Heart benchmark.

Mean Square Error Percentage			Percentage of Misclassification			Number of	Total # of	Improving or
Train.	Val.	Test	Train.	Val.	Test	Connections	Iterations	Degrading
33.55	41.11	13.85	17.10	21.05	6.66	592	75	+
25.98	37.66	14.88	14.47	19.73	8.00	592	125	+
32.12	40.30	13.35	17.76	20.42	6.66	400	150	-
28.46	41.57	7.67	15.13	22.36	2.66	360	195	-
35.28	37.60	12.94	17.76	19.73	6.66	324	235	+
34.64	36.31	12.39	17.76	22.36	5.33	292	280	+
26.95	43.12	12.95	12.50	23.68	5.33	263	305	-
31.49	40.71	16.95	16.44	23.68	8.00	237	415	-
26.79	37.45	23.32	12.50	19.73	10.66	213	470	-
26.35	40.78	14.71	13.15	19.73	6.66	192	505	-
29.99	40.74	16.26	11.84	22.36	5.33	173	540	-
23.70	40.10	16.62	13.15	25.00	6.66	156	570	-
40.43	49.69	35.40	11.84	19.73	6.66	140	605	-

Table 4.2: One simulation on the Heart benchmark with a continuous network.

the combination of discretization and pruning is a little ambitious; however, some remarks are in order. If weight removal was successful in the continuous network, no significant further pruning is achieved using discrete weights. For the Cancer and Heart benchmarks the $Wmax$ and $Power\_of\_two\_W_{max}$ quantization functions perform good for $D = 15$ and $D = 31$. Good performance implies that the misclassification on the test set for the optimal continuous network is comparable with the one obtained by the discrete network. It should be emphasized that the *Symmetrical* quantization function performs good using only ternary weights {-1,0 1}, $D = 3$, or even binary weights {-1,1}, $D = 2$. For the Wine benchmark the misclassification rate is overall high using continuous networks, and often an improvement in the percentage of misclassification is obtained when using discrete weights. There's no marked difference between the different pruning algorithms for the results with the discrete networks.

Secondly, the results for approximation problems, are discussed. For the Sunspot benchmark, *Lambdaprune* and *Autoprune* perform good, resulting in networks with about 20 connections, see table B.2. This benchmark is apparently robust for heavy weight removal in the initial pruning steps. The other pruning methods do not show a significantly improved performance compared with *Randomize*. However, the trade-off between a small network topology and good generalization is accentuated here. If the validation set error would be allowed to be 5% higher compared to the optimal network, *Variance*

and *Weight* pruning would reduce the topology to a similar or even lower level as for *Lambdaprune* and *Autoprune*. For example in table 4.3, the resulting network would have 18 connections, with an validation error performance of 0.2739, instead of 47 connections, with an validation error performance of 0.2704.

Mean Square Error Percentage			Number of	Total # of	Improving or
Train.	Val.	Test	Connections	Iterations	Degrading
0.2465	0.2704	0.3169	52	260	+
0.2272	0.2706	0.3071	52	290	-
0.2241	0.2704	0.3182	47	335	+
0.1926	0.2799	0.3119	42	365	-
0.1864	0.2787	0.2979	38	395	-
0.1812	0.2836	0.3040	34	425	-
0.1805	0.2820	0.3037	31	455	-
0.1814	0.2938	0.3234	28	575	-
0.1780	0.2854	0.3086	25	605	-
0.1894	0.2743	0.2764	22	635	-
0.1828	0.2879	0.2850	20	680	-
0.1897	0.2739	0.3007	18	710	-
0.1798	0.3306	0.2863	16	755	-

Table 4.3: One simulation on the Sunspot benchmark with continuous network.

For the Auto-MPG and Leman benchmarks, *Variance* and *Weight* pruning perform good, succeeding to reduce the network topology with 50% or even 70%. For *Lambdaprune* and *Autoprune*, the results are less good with about the same performance as *Randomize*. These data sets are apparently sensitive to heavy weight removal in initial pruning steps.

For approximation problems, the mean square error obtained for continuous networks by the pruning methods is overall good. For example, with the Sunspot benchmark, *Variance* pruning obtains a mean square error percentage of 0.28, comparable with previous results (see table 4.1). The results for the Leman benchmark (table B.3) are also promising. For example, *Weight* pruning obtains a mean square error percentage of 0.56, which is about 15% lower than the previous results in table 4.1.

Regarding discretization, if pruning was successful in the continuous network, further pruning is not improving network performance using discrete weights. For the Auto-MPG and Leman benchmarks, the *Wmax* and *Power_of_two*$\_W_{max}$ quantization function perform good employing $D \geq 7$. For example, with the Leman benchmark using the W_{max} quantization function and $D = 7$, *Weight* pruning obtains a mean square error percentage of 0.59 , a value comparable with the continuous network. For the Sunspot benchmark, both quantization functions perform good using $D = 15$ and $D = 31$.

4.4 Conclusions for Pruning

The objective of this chapter was to evaluate the performance of six pruning methods. Experiments on six benchmark problems were performed, using both continuous and discrete weights. The quantization functions of the previous chapter were employed. A general pruning scheme, indicating when to prune and when to stop pruning has been presented, providing a basis for the comparison of the different pruning methods. The goal was to find networks with a small topology, and see how this correlates with their generalization performance.

Firstly it can be stated that the results are highly class-dependent; in fact, even within the classes of classification and approximation problem, there are large differences in performance for the various pruning methods.

The conclusions obtained in the previous are substantiated in this chapter. The quantization functions *Wmax* and *Power_of_two*$\_W_{max}$ perform good on both classes, using $D = 15$ and $D =$

31. For approximation problems, even 7 discretization levels can be employed. The *Symmetrical* quantization function performs well on classification problems using only ternary weights {-1,0 1} or even binary weights {-1,1}. Furthermore, $Wmax$ performs slightly better than $Power\_of\_two\_W_{max}$ on approximation problems. If significant pruning was performed on the continuous network, the network size will not decrease further in the discrete case, without deteriorating network performance.

With respect to pruning several problems were encountered. Firstly, for classification problems, two main drawbacks of the general pruning scheme, can be distinguished: sensitivity to large oscillations of the training and validation set error, and to a high relative increase in the validation set error. Both problems can be diminished by basing the criteria, when to stop training and when to prune, on a fixed mean square error on the training set; this ensures that weight removal will take place. Initial experiments shows promising results using this technique. For example, with the Wine benchmark (table B.7), $Variance$ pruning obtains a mean square error percentage of 3.18 with an average network topology of only 33.2 connections. Moreover, due to the lack of correlation between the mean-square error and the percentage of misclassification for classification problems, the decision when to prune should be based on the percentage of misclassification or the cross-entropy error function should be used.

Secondly, one major problem can be distinguished for approximation problems: the trade-off between network size and generalization performance. An alternative is to allow a certain deviation from the optimal validation set error. This guarantees that smaller topologies with an insignificantly higher validation set error, and sometimes even with a better test error performance can be achieved.

Finally, initial experiments indicate that *Lambdaprune*, the pruning strength of which is based on generalization loss, performs better if λ_{max} (section 4.1) is decreased to 1/3 or even 1/4, avoiding extremely large pruning steps.

Summarizing, this chapter gives insight into problems that occur when pruning neural networks. Since no pruning method showed an overall good performance, some suggestions how to improve the general pruning scheme have been presented. It is clear that to design a successful pruning method, not only the question of which and how many weights to remove is crucial, but also the decision when to prune and when to stop pruning. These issues have been illuminated in this chapter, providing useful information for future work.

Chapter 5

Conclusions

In order to reduce the accuracy of the parameters in multilayer perceptrons without deterioration of network performance, a weight discretization algorithm and six different quantization functions have been evaluated in this report. Two of these quantization functions, W_{max} and $Power\_of\_two\_W_{max}$, give nearly the same performance as for a network with continuous weight values, when using only 15 discrete weight levels. Moreover, the *Symmetrical* quantization function performs good for classification problems using only ternary weights $\{-1, 0, 1\}$ or even binary weights $\{-1, 1\}$. These three quantization functions result in networks that are well-suited for hardware implementation and can considerably reduce the VLSI surface area required.

Another way to obtain compact networks is by minimizing the network topology for the problem at hand. Since it is impossible to know *a priori* the size of the minimal network topology, different pruning techniques, together with a general pruning scheme, have been evaluated. Moreover, to further minimize chip area and computational requirements, a combination of these pruning techniques with weight quantization has also been investigated in this report.

It has been established that a general pruning scheme based on the error on a validation set is not robust enough to be useful. Especially, for classification problems the pruning scheme fails to decide properly when to prune and does not considerably reduce the network topology without degrading generalization performance. For approximation problems, the pruning scheme shows better results; the choice of the optimal pruning technique, however, depends on the problem at hand. Initial experiments with a pruning scheme based on the error on the training set show promising results and for some pruning methods leads to a considerable reduction of the network size, while retaining generalization performance.

The combination of pruning and weight quantization confirms the results for the three quantization functions mentioned above, even when the network topology has been significantly pruned.

5.1 Future work

The design of *compact networks* which have a small topology and for which a small accuracy is sufficient, is indeed of great importance both for the hardware implementation of neural networks and for the field of neural networks in general. The results on quantization and pruning presented in this report are a valuable first step towards this goal. However, the methods described in this report are still open for improvement and several promising future research directions are outlined here:

- The weight discretization algorithm showed good results for classification problems using the *Symmetrical* quantization function with binary $\{-1, 1\}$ and ternary $\{-1, 0, 1\}$ weights. This might be further improved by adding a random factor to the weight updates to avoid the discrete network getting trapped on an error plateau and to allow a better exploration of the discrete

solution space [Leong-95]. Another option is to use a meta-strategy for optimization problems, such as Tabu Search [Glover-89], to guide the discrete search.

- Initial experiments with a pruning scheme based on a fixed training set error-criteria showed promising results and should be further explored. Moreover, the large majority of the pruning algorithms investigated in this report are performing connection removal and lead to an irregular network structure which is not suitable for hardware implementation. Therefore, pruning methods that remove neurons in stead of connections and maintain the regularity of the network are more hardware-friendly and should be further investigated.
- It is well-known that there is a reciprocal relationship between the number of hidden neurons in a MLP and the number of bits per discrete weight [Fiesler-88][Brause-93]. The weight discretization technique should therefore be combined with a neural network growing method adding hidden neurons until satisfactory performance is reached. This growing stage can of course still be followed by a pruning stage to remove neurons and optimize generalization performance.

5.2 Acknowledgements

The support of the Swiss National Science Foundation (Grant SNSF 21-45621.95) is gratefully acknowledged.

Bibliography

[Asanović-91] K. Asanović and N. Morgan. Experimental Determination of Precision Requirements for Back-Propagation Training of Artificial Neural Networks. *Proceedings of the Second International Conference MicroNeuro'91*, pp. 9–15, (U. Ramacher, U. Rückert, and J. A. Nossek, eds.), 1991.

[Bartlett-96] P. Bartlett. The Sample Complexity of Pattern Classification with Neural Networks: the Size of the Weights Is More Important Than the Size of the Network. Technical report of the Department of Systems Engineering at the Australian National University, Canberra, Australia, ftp://syseng.anu.edu.au/pub/peter/TR96d.ps.Z.

[Bellido-93] I. Bellido and E. Fiesler. Do Backpropagation Trained Neural Networks have Normal Weight Distributions? *Proc. of the Int. Conf. on Artificial Neural Networks*, pp. 772–775, Springer, London, 1993.

[Bishop-95] C. M. Bishop. *Neural Networks for Pattern Recognition*. Oxford University Press, Oxford, 1995.

[Brause-93] R. W. Brause. The error-bound descriptional complexity of approximation networks. *Neural Networks*, vol. 6, no. 2, pp. 177–187, 1993.

[Fiesler-88] E. Fiesler, A. Choudry, and H. J. Caulfield. Weight Discretization in Backward Error Propagation Neural Networks. *Neural Networks*, special supp. "Abstracts of the First Annual INNS Meeting", vol. 1, p. 380, 1988.

[Finnoff-93] W. Finnoff, F. Hergert, and H. G. Zimmermann. Improving Model Selection by Nonconvergent Methods, *Neural Networks*, vol. 6, pp. 771–783, 1993.

[Garris-92] M. D. Garris and R. A. Wilkinson. NIST Special Database 3. National Institute of Standards and Technology, Advanced System Division, Image Recognition Group (1992).

[Girau-96] B. Girau and A. Tisserand. On-line Arithmetic-Based Reprogrammable Hardware Implementation of Multilayer Perceptron Back-Propagation. *Proceedings of the Fifth International Conference on Microelectronics for Neural Networks and Fuzzy Systems*, pp. 168–175, IEEE Computer Society Press, 1996.

[Glover-89] F. Glover. Tabu Search - Part 1. *ORSA Journal on Computing*, vol. 1, no. 3, pp. 190–206, 1989.

[Gosh-94] J. Ghosh and K. Tumer. Structural Adaptation and Generalization in Supervised Feed-Forward Networks. *Journal of Artificial Neural Networks*, vol. 1, no. 4, pp. 431–458, 1994.

[Haykin-94] Simon Haykin. Neural Networks: A Comprehensive Foundation. Macmillan College Publishing Company, Englewood Cliffs, ISBN 0-02-352761-7, 1994.

[Holt-93] J. L. Holt and J.-N. Hwang. Finite Error Precision Analysis of Neural Network Hardware Implementations. *IEEE Transactions on Computers*, vol. 42, no. 3, pp. 1380–1389, March 1993.

[Kwok-95] T. -Y. Kwok and D. -Y. Yeung. Constructive Feedforward Neural Networks for Regression Problems: A Survey. Technical report HKUST-CS95-43, Department of Computer Science, Hong Kong University of Science and Technology, 1995.

[Leong-95] P. H. W. Leong and M. A . Jabri. A Low-Power VLSI Arrhythmia Classifier. *IEEE Transactions on Neural Networks*, vol. 6, no. 6, pp. 1435–1445, November 1995.

[Lundin-96] T. Lundin, E. Fiesler, and P. Moerland. Connectionist Quantization Functions. *Proceedings of the '96 SIPAR-Workshop on Parallel and Distributed Computing*, Scientific and Parallel Computing Group, University of Genève, Genève, Switzerland, 1996.

[Marchesi-93] M. Marchesi, G. Orlandi, F. Piazza, and A. Uncini. Fast Neural Networks Without Multipliers. *IEEE Transactions on Neural Networks*, vol. 4, no. 1, pp. 53–62, January 1993.

[Mauduit-92] N. Mauduit, M. Duranton, J. Gobert, and J. -A. Sirat. Lneuro 1.0: A Piece of Hardware LEGO for Building Neural Network Systems. *IEEE Transactions on Neural Networks*, vol. 3, no. 3, pp. 414–422, May 1992.

[Moerland-97] P. D. Moerland and E. Fiesler. Neural Network Adaptations to Hardware Implementations. *Handbook of Neural Computation*, E. Fiesler and R. Beale, eds., chap. E1.2, pp. 1–13, IOP Publishing and Oxford University Press, 1997.

[Moreira-95] M. Moreira and E. Fiesler. Neural Networks with Adaptive Learning Rate and Momentum Terms. Technical report 95-04, IDIAP, Martigny, Switzerland, available via anonymous ftp: ftp://ftp.idiap.ch/pub/techreports/95-04.ps.Z.

[Murphy-94] Data made available in 1994 by librarians P. M. Murphy and D. W. Aha from the UCI Repository of Machine Learning Databases, a machine-readable data repository accessible via anonymous-ftp: ftp://ftp.ics.uci.edu/pub/machine-learning-databases.

[Prechelt-94] L. Prechelt, "PROBEN1 — A Set of Benchmarks and Benchmarking Rules for Neural Network Training Algorithms," Technical report 21/94, Fakultät für Informatik, Universität Karlsruhe, D-76128 Karlsruhe, Germany (1994). The benchmark set is available at ftp://ftp.ira.uka.de/pub/neuron/proben1.tar.gz.

[Prechelt-95] L. Prechelt. Adaptive Parameter Pruning in Neural Networks. ICSI technical report TR-95-009, ICSI, Berkeley, USA, 1995.

[Reed-93] R. Reed. Pruning Algorithms: a Survey. *IEEE Transactions on Neural Networks*, vol. 4, no. 5, pp. 740–746, September 1993.

[Rumelhart-86] D. Rumelhart, G. Hinton, and R. Williams, "Learning Internal Representations by Error Propagation," in *Parallel Distributed Processing: Explorations in the Microstructure of Cognition*, vol. 1: Foundations, chap. 8, pp. 318–362, MIT Press, Cambridge, Massachusetts (1986).

[Saxena-95] I. Saxena and E. Fiesler. Adaptive Multilayer Optical Neural Network with Optical Thresholding. *Optical Engineering*, special on Optics in Switzerland (P. Rastogi, editor), vol. 34, no. 8, pp. 2435–2440, 1995.

[Sietsma-91] J. Sietsma and R. F. J. Dow. Creating Artificial Neural Networks That Generalize. *Neural Networks*, vol. 4, no. 1, pp. 67–69, 1991.

[Säckinger-92] E. Säckinger, B. E. Boser, J. Bromley, Y. LeCun, and L. D. Jackel. Application of the ANNA Neural Network Chip to High-Speed Character Recognition. *IEEE Transactions on Neural Networks*, vol. 3, no. 3, pp. 498–505, May 1992.

[Thimm-96] G. Thimm and E. Fiesler. Weight Initialization in Higher Order and Multi-Layer Perceptrons. Accepted for publication in *IEEE Transactions on Neural Networks*, 1997.

Appendix A

Quantization Results

Discretization function	D	# of epochs (mean)	square error % (mean)		
			train.	val.	test
	C [1]	171.5	0.36	0.31	0.24
Symmetrical	2	5.0	8.08	8.61	7.04
	3	5.0	8.08	8.61	7.04
	5	8.5	6.55	6.98	5.66
	7	10.0	5.30	5.65	4.55
	15	12.0	2.06	2.07	1.71
	31	9.0	0.46	0.36	0.39
W_{max}	2	9.0	1.69	1.49	1.64
	3	35.5	1.18	1.10	1.21
	5	48.5	0.58	0.45	0.45
	7	8.0	0.43	0.39	0.33
	15	77.5	0.44	0.34	0.32
	31	108.5	0.38	0.31	0.27
W_{max_adapt}	2	9.0	2.22	2.26	2.27
	3	29.5	0.99	0.90	0.96
	5	61.0	0.63	0.48	0.54
	7	53.0	0.49	0.42	0.39
	15	63.5	0.42	0.32	0.31
	31	137.0	0.38	0.30	0.27
Power_of_two $\_W_{max}$	2	9.0	1.69	1.49	1.64
	3	35.5	1.18	1.10	1.21
	5	48.5	0.58	0.45	0.45
	7	36.0	0.50	0.38	0.40
	15	33.5	0.40	0.33	0.27
	31	37.0	0.39	0.33	0.27
Power_of_two	2	9.0	1.69	1.49	1.64
	3	19.5	1.14	1.03	1.07
	5	30.5	0.72	0.57	0.63
	7	45.0	0.51	0.38	0.39
	15	27.0	0.44	0.35	0.32
	31	32.5	0.46	0.35	0.35
Power_of_two *_adapt*	2	21.0	1.62	1.46	1.52
	3	44.0	1.01	0.85	0.94
	5	35.5	0.62	0.47	0.49
	7	40.0	0.51	0.40	0.39
	15	29.0	0.43	0.36	0.31
	31	23.5	0.43	0.36	0.33

Table A.1: Discretization results for the *Auto-mpg* benchmark.

Discretization function	D	# of epochs (mean)	square error % (mean)		
			train.	val.	test
	C	257.00	0.32	0.28	0.29
Symmetrical	2	5.00	3.76	3.20	3.90
	3	5.00	3.76	3.20	3.90
	5	10.00	3.10	2.60	3.21
	7	10.00	2.59	2.13	2.65
	15	21.00	1.30	0.91	1.28
	31	49.50	0.47	0.35	0.47
W_{max}	2	12.00	1.64	1.48	1.58
	3	5.50	1.26	0.96	1.28
	5	22.00	1.38	1.22	1.36
	7	20.50	0.62	0.51	0.75
	15	13.50	0.38	0.30	0.34
	31	64.00	0.36	0.30	0.32
W_{max_adapt}	2	6.00	1.27	0.96	1.29
	3	6.00	1.80	1.63	1.87
	5	7.50	0.64	0.51	0.63
	7	13.50	0.56	0.41	0.54
	15	24.00	0.40	0.31	0.37
	31	112.50	0.34	0.29	0.30
Power_of_two $\_W_{max}$	2	12.00	1.64	1.48	1.58
	3	5.50	1.26	0.96	1.28
	5	22.00	1.38	1.22	1.36
	7	64.50	0.51	0.43	0.48
	15	30.50	0.41	0.40	0.35
	31	47.00	0.41	0.40	0.35
Power_of_two	2	12.00	1.64	1.48	1.58
	3	16.50	1.54	1.26	1.57
	5	10.50	0.80	0.67	0.84
	7	33.50	0.72	0.55	0.72
	15	24.00	0.47	0.37	0.45
	31	23.00	0.48	0.37	0.47
Power_of_two *_adapt*	2	29.00	1.74	1.59	1.71
	3	32.50	1.32	1.18	1.42
	5	18.50	0.82	0.75	0.93
	7	22.00	0.49	0.40	0.46
	15	36.00	0.40	0.34	0.37
	31	55.50	0.40	0.35	0.38

Table A.2: Discretization results for the *Sun-spot* benchmark.

[1] 'C' denotes continuous pre-training.

Discretization function	D	# of epochs (mean)	% of misclassification (mean)		
			train.	val.	test
	C	45.00	3.97	2.23	1.15
Symmetrical	2	152.00	4.80	2.80	1.78
	3	183.00	4.49	2.74	1.61
	5	48.00	3.97	2.80	3.39
	7	45.00	4.34	2.97	3.39
	15	50.00	4.74	2.97	2.59
	31	67.50	4.20	2.63	2.64
W_{max}	2	63.00	7.37	2.74	4.54
	3	60.50	26.86	24.00	28.05
	5	6.50	6.34	4.06	3.45
	7	10.50	4.89	3.26	2.47
	15	18.50	3.66	2.11	1.21
	31	35.00	3.89	2.40	1.78
W_{max_adapt}	2	8.50	34.57	31.43	37.36
	3	9.00	19.34	16.51	18.91
	5	7.50	6.20	4.91	4.37
	7	11.50	4.31	3.03	2.13
	15	46.50	3.51	2.46	1.44
	31	49.50	3.63	2.23	1.15
Power_of_two $\_W_{max}$	2	63.00	7.37	2.74	4.54
	3	60.50	26.86	24.00	28.05
	5	6.50	6.29	4.06	3.39
	7	11.00	3.86	2.51	1.44
	15	44.50	3.71	2.29	1.44
	31	66.00	4.00	2.46	1.61
Power_of_two	2	63.00	7.37	2.74	4.54
	3	13.50	28.57	25.54	30.52
	5	8.50	4.40	2.97	1.78
	7	22.00	3.83	2.29	1.38
	15	38.00	3.80	2.34	1.26
	31	51.00	3.80	2.06	1.67
Power_of_two	2	26.50	6.66	4.00	4.14
	3	33.50	17.23	14.51	16.61
	5	9.50	5.86	3.89	4.25
	7	16.50	3.80	2.46	1.26
	15	63.00	3.83	2.29	1.72
	31	61.50	3.74	2.06	1.44

Table A.3: Discretization results for the *Cancer* benchmark.

Discretization function	D	# of epochs (mean)	% of misclassification (mean)		
			train.	val.	test
	C	529.00	0.79	0.44	2.27
Symmetrical	2	53.50	6.63	4.00	6.59
	3	120.00	5.73	3.11	6.14
	5	72.00	4.49	2.89	5.00
	7	55.50	5.06	3.33	7.05
	15	35.50	4.49	2.22	7.05
	31	39.00	1.91	1.11	4.55
W_{max}	2	35.50	10.56	8.89	12.50
	3	41.50	3.26	2.67	7.50
	5	40.50	2.02	0.67	4.09
	7	32.00	0.90	0.44	3.18
	15	109.00	0.34	0.44	2.95
	31	118.00	1.24	0.00	3.18
W_{max_adapt}	2	24.00	62.92	62.67	62.73
	3	47.50	4.04	2.22	7.05
	5	29.00	0.90	0.22	5.00
	7	23.50	1.80	0.22	4.55
	15	23.50	1.12	0.00	3.64
	31	18.00	0.45	0.00	3.18
Power_of_two $\_W_{max}$	2	35.50	10.56	8.89	12.50
	3	41.50	3.26	2.67	7.50
	5	40.50	2.02	0.67	4.09
	7	110.50	1.01	0.44	3.64
	15	205.00	0.79	0.22	3.86
	31	87.50	0.56	0.67	4.09
Power_of_two	2	35.50	10.56	8.89	12.50
	3	58.00	4.16	2.44	7.27
	5	42.50	1.69	0.67	5.00
	7	411.50	0.34	0.44	3.18
	15	511.50	0.11	1.56	3.86
	31	327.00	1.35	0.67	3.18
Power_of_two *_adapt*	2	45.00	8.99	8.44	12.05
	3	58.00	3.48	2.44	7.50
	5	24.00	2.25	0.67	4.32
	7	28.00	0.56	0.22	3.86
	15	321.50	0.56	0.00	4.09
	31	252.00	0.11	0.00	2.95

Table A.4: Discretization results for the *Wine* benchmark.

Discretization function	D	# of epochs (mean)	% of misclassification (mean)		
			train.	val.	test
	C	107.5	19.97	25.89	23.49
Symmetrical	2	216.0	24.66	30.78	25.26
	3	172.5	24.51	30.16	25.10
	5	57.5	25.91	29.79	27.14
	7	49.0	25.89	29.22	27.45
	15	54.5	23.28	28.39	26.25
	31	82.0	23.20	27.66	25.99
W_{max}	2	56.5	28.07	29.43	30.16
	3	57.5	26.43	32.08	29.64
	5	64.5	22.94	28.02	25.26
	7	67.5	22.81	27.29	27.03
	15	98.0	20.00	26.30	24.84
	31	136.5	20.76	26.41	24.90
W_{max_adapt}	2	5.0	33.59	36.46	35.94
	3	37.5	31.69	35.62	34.74
	5	93.5	25.62	29.69	29.06
	7	94.5	22.99	27.92	25.83
	15	98.0	20.62	25.89	25.26
	31	174.0	21.17	26.82	25.05
Power_of_two $\_W_{max}$	2	56.5	28.07	29.43	30.16
	3	57.5	26.43	32.08	29.64
	5	64.5	22.94	28.02	25.26
	7	54.5	21.90	27.76	26.35
	15	137.0	20.52	25.68	24.22
	31	82.0	21.33	26.56	26.04
Power_of_two	2	56.5	28.07	29.43	30.16
	3	59.0	33.26	35.31	34.53
	5	62.0	24.48	27.81	26.98
	7	79.5	20.86	26.15	24.01
	15	73.5	21.20	27.24	25.21
	31	75.0	21.82	26.09	24.74
Power_of_two *_adapt*	2	86.5	27.01	29.01	28.80
	3	80.0	30.49	33.07	32.60
	5	61.0	24.01	27.97	27.45
	7	81.5	20.96	26.30	24.48
	15	100.0	21.22	26.46	25.68
	31	104.5	21.51	26.77	25.52

Table A.5: Discretization results for the *Diabetes* benchmark.

Discretization function	D	# of epochs (mean)	% of misclassification (mean)		
			train.	val.	test
	C	58.33	0.07	10.53	7.20
Symmetrical	2	50.00	3.80	13.07	8.93
	3	88.33	2.27	11.60	8.67
	5	65.00	1.60	10.67	7.87
	7	65.00	1.87	10.53	8.80
	15	50.00	0.33	11.60	8.00
	31	18.33	0.80	11.47	7.60
W_{max}	2	45.00	12.16	22.56	19.44
	3	76.00	4.64	15.44	14.48
	5	92.00	1.68	13.12	9.84
	7	105.00	0.64	11.68	10.24
	15	52.00	0.24	11.76	8.08
	31	36.00	0.60	10.56	7.36
Power_of_two $\_W_{max}$	2	40.00	12.60	22.67	18.93
	3	103.33	4.07	16.40	14.40
	5	96.67	1.87	12.27	9.60
	7	158.33	0.07	11.47	8.67
	15	55.00	0.67	11.60	7.33
	31	55.00	0.47	10.67	9.20
Power_of_two	2	45.00	12.16	22.56	19.44
	3	124.00	4.12	15.60	12.72
	5	81.00	1.60	12.64	9.68
	7	80.00	0.64	11.60	8.64
	15	35.00	0.72	11.0	7.76
	31	61.00	0.08	11.44	8.40

Table A.6: Discretization results for the *Digit* benchmark.

Appendix B

Pruning Results

Discretization function	Pruning method	D	# of epochs (mean)	square error % (mean) train.	val.	test	Size	Discretization function	# of epochs (mean)	square error % (mean) train.	val.	test	Size
W_{max}	*Randomize*	C	546.00	0.30	0.24	0.21	36.9	*Power_of_two* _W_{max}	598.50	0.31	0.24	0.21	36.4
		2	32.50	0.97	0.87	0.85	36.9		85.00	1.19	1.00	1.10	33.9
		3	181.00	0.82	0.67	0.68	30.3		74.00	0.89	0.77	0.86	35.6
		5	175.50	0.46	0.39	0.35	33.3		72.50	0.62	0.51	0.55	35.3
		7	99.00	0.41	0.34	0.33	33.8		149.50	0.43	0.31	0.32	33.9
		15	74.50	0.33	0.25	0.23	36.2		69.00	0.35	0.28	0.26	35.3
		31	171.00	0.31	0.24	0.22	33.6		85.50	0.36	0.27	0.26	34.1
	Variance	C	654.50	0.31	0.25	0.21	27.6		654.50	0.31	0.25	0.21	27.6
		2	68.50	4.69	4.32	4.40	26.4		76.00	4.69	4.32	4.40	26.4
		3	53.50	1.85	1.69	1.92	19.3		53.50	1.85	1.69	1.92	19.3
		5	68.00	0.98	0.80	0.90	25.5		66.50	0.98	0.80	0.90	25.5
		7	86.00	0.57	0.48	0.46	21.7		99.50	0.51	0.40	0.40	21.0
		15	94.00	0.41	0.32	0.31	26,0		64.50	0.38	0.28	0.26	27.1
		31	64.50	0.34	0.26	0.22	27.6		75.00	0.38	0.28	0.25	27.1
	Weight	C	613.50	0.30	0.25	0.21	32.5		613.50	0.30	0.25	0.21	32.5
		2	74.50	1.33	1.25	1.30	31.1		74.50	1.33	1.25	1.30	31.1
		3	84.00	0.94	0.81	0.88	29.4		84.00	0.94	0.81	0.88	29.4
		5	111.50	0.48	0.38	0.38	29.3		111.50	0.48	0.38	0.38	29.3
		7	167.50	0.40	0.31	0.29	26.0		141.50	0.41	0.29	0.29	27.2
		15	155.00	0.32	0.25	0.23	27.2		144.00	0.38	0.28	0.27	26.5
		31	112.00	0.31	0.24	0.23	29.7		148.00	0.39	0.29	0.29	27.8
	Auto	C	291.00	0.30	0.26	0.20	46.3		291.00	0.30	0.26	0.20	46.3
		2	58.50	0.86	0.76	0.71	45.3		58.50	0.86	0.76	0.71	45.3
		3	174.00	0.60	0.52	0.48	40.5		174.00	0.60	0.52	0.48	40.5
		5	227.50	0.41	0.31	0.31	36.9		227.50	0.41	0.31	0.31	36.9
		7	248.00	0.38	0.28	0.27	36.0		197.00	0.35	0.26	0.24	35.3
		15	198.00	0.33	0.25	0.23	37.8		174.00	0.35	0.27	0.25	36.5
		31	158.50	0.30	0.25	0.21	41.4		175.50	0.34	0.27	0.25	39.4
	Lambda	C	324.00	0.30	0.25	0.20	43.5		324.00	0.30	0.25	0.20	43.5
		2	37.50	1.03	0.92	0.85	43.5		28.50	1.08	0.98	0.95	43.5
		3	105.50	0.91	0.75	0.81	34.8		104.50	0.93	0.74	0.82	35.8
		5	109.50	0.54	0.42	0.43	37.0		114.50	0.53	0.42	0.41	39.9
		7	86.50	0.41	0.32	0.29	37.5		81.50	0.38	0.31	0.28	36.9
		15	63.50	0.34	0.26	0.23	41.6		61.00	0.34	0.28	0.24	41.4
		31	95.50	0.31	0.25	0.22	40.2		70.50	0.35	0.29	0.25	43.5
	Node	C	334.00	0.30	0.26	0.21	41.6						

Table B.1: Pruning results for the *Auto-MPG* benchmark.

Discretization function	Pruning method	D	# of epochs (mean)	square error % (mean) train.	val.	test	Size	Discretization function	# of epochs (mean)	square error % (mean) train.	val.	test	Size
W_{max}	*Randomize*	C	567.00	0.27	0.26	0.31	42.7	*Power_of_two* _W_{max}	338.00	0.25	0.25	0.29	43.3
		2	87.00	1.60	1.15	1.60	40.9		46.50	1.16	0.95	0.95	43.3
		3	43.00	1.27	0.88	1.27	42.3		58.50	1.11	0.77	1.09	40.3
		5	149.00	0.66	0.41	0.64	40.8		116.50	0.55	0.39	0.55	38.4
		7	109.00	0.47	0.34	0.47	41.0		101.00	0.42	0.31	0.37	40.0
		15	75.50	0.31	0.28	0.32	41.7		214.00	0.30	0.25	0.32	35.8
		31	63.00	0.28	0.26	0.31	41.9		162.00	0.30	0.25	0.31	39.5
	Variance	C	336.50	0.22	0.26	0.28	38.6		336.50	0.22	0.26	0.28	38.6
		2	76.00	1.43	1.19	1.53	38.6		76.00	1.43	1.19	1.53	38.6
		3	16.50	1.24	0.91	1.28	38.6		16.50	1.24	0.91	1.28	38.6
		5	179.50	0.70	0.47	0.70	23.7		227.00	0.72	0.48	0.72	23.2
		7	212.50	0.41	0.30	0.40	29.6		190.50	0.39	0.29	0.36	25.9
		15	195.50	0.30	0.25	0.32	27.8		121.50	0.29	0.26	0.32	34.3
		31	183.00	0.24	0.25	0.29	28.1		121.50	0.29	0.26	0.33	32.7
	Weight	C	361.50	0.22	0.26	0.28	37.6		361.50	0.22	0.26	0.28	37.6
		2	38.00	1.25	1.11	1.18	36.8		38.00	1.25	1.11	1.18	36.8
		3	29.50	1.17	0.91	1.20	37.1		29.50	1.17	0.91	1.20	37.1
		5	128.00	0.59	0.36	0.57	33.2		128.00	0.59	0.36	0.57	33.2
		7	177.50	0.42	0.28	0.42	32.8		162.50	0.35	0.27	0.34	30.6
		15	219.50	0.26	0.26	0.30	30.8		273.50	0.28	0.25	0.30	24.6
		31	267.50	0.21	0.26	0.27	26		390.00	0.30	0.25	0.31	22.2
	Auto	C	844.00	0.27	0.22	0.27	19.6		844.00	0.27	0.22	0.27	19.6
		2	15.50	1.47	1.35	1.46	19.3		15.50	1.47	1.35	1.46	19.3
		3	29.00	1.40	1.00	1.41	19.3		29.00	1.40	1.00	1.41	19.3
		5	112.50	0.81	0.65	0.75	17.9		112.50	0.81	0.65	0.75	17.9
		7	61.00	0.51	0.40	0.48	19.3		75.00	0.44	0.35	0.38	18.9
		15	91.50	0.32	0.26	0.29	19.6		86.50	0.34	0.26	0.32	18.2
		31	62.50	0.29	0.23	0.28	18.1		88.00	0.36	0.27	0.32	17.9
	Lambda	C	622.00	0.29	0.23	0.27	20.9		622.00	0.29	0.23	0.27	20.9
		2	47.00	1.52	1.44	1.48	20.9		47.00	1.52	1.44	1.48	20.9
		3	15.50	1.23	0.93	1.22	20.9		15.50	1.23	0.93	1.22	20.9
		5	56.50	0.77	0.58	0.76	20.9		56.50	0.77	0.58	0.76	20.9
		7	59.00	0.48	0.41	0.43	20.9		49.50	0.43	0.36	0.37	20.9
		15	41.50	0.33	0.27	0.29	20.9		63.00	0.35	0.26	0.33	19
		31	53.50	0.29	0.24	0.28	20.9		26.00	0.34	0.26	0.32	20.9
	Node	C	177.50	0.25	0.26	0.30	49.4						

Table B.2: Pruning results for the *Sunspot* benchmark.

Discretization function	Pruning method	D	# of epochs (mean)	square error % (mean) train.	val.	test	Size	Discretization function	# of epochs (mean)	square error % (mean) train.	val.	test	Size
W_{max}	*Randomize*	C	704.00	0.98	0.70	0.55	13.9	*Power_of_two* $\_W_{max}$	654.50	1.01	0.71	0.55	14.8
		2	54.00	2.18	1.26	1.24	13.6		69.00	1.96	1.34	1.24	14.2
		3	20.50	1.40	0.78	0.69	13.9		14.00	1.41	0.76	0.72	14.8
		5	67.00	1.40	0.71	0.66	13.5		97.00	1.38	0.72	0.63	14.3
		7	63.50	1.27	0.68	0.62	13.4		84.00	1.29	0.65	0.63	14.0
		15	100.50	1.15	0.65	0.60	13.5		64.50	1.27	0.67	0.61	14.6
		31	115.00	1.01	0.68	0.57	13.4		105.50	1.32	0.65	0.63	14.1
	Variance	C	956.00	0.93	0.70	0.56	7.1		956.00	0.93	0.70	0.56	7.1
		2	77.00	2.47	2.01	1.99	6.2		77.00	2.47	2.01	1.99	6.2
		3	9.00	1.84	1.51	1.28	7.1		9.00	1.84	1.51	1.28	7.1
		5	39.50	1.58	0.75	0.79	6.9		33.00	1.58	0.75	0.79	6.8
		7	75.00	1.25	0.74	0.66	6.0		33.00	1.35	0.70	0.68	7.0
		15	105.50	1.18	0.64	0.62	6.1		34.00	1.28	0.69	0.66	7.1
		31	78.50	1.08	0.66	0.60	6.9		47.50	1.25	0.67	0.63	6.9
	Weight	C	839.00	0.96	0.70	0.56	8.6		839.00	0.96	0.70	0.56	8.6
		2	9.50	2.55	1.92	1.90	8.6		9.50	2.55	1.92	1.90	8.6
		3	19.50	1.42	0.78	0.70	8.6		19.50	1.42	0.78	0.70	8.6
		5	36.00	1.31	0.72	0.67	8.5		36.00	1.31	0.72	0.67	8.5
		7	15.50	1.14	0.70	0.59	8.6		24.00	1.30	0.67	0.65	8.6
		15	59.00	1.12	0.66	0.60	8.3		28.00	1.35	0.68	0.67	8.6
		31	36.50	0.98	0.71	0.58	8.6		39.00	1.34	0.67	0.64	8.6
	Auto	C	317.00	1.05	0.73	0.55	13.3		317.00	1.05	0.73	0.55	13.3
		2	18.50	1.43	0.88	0.78	13.3		18.50	1.43	0.88	0.78	13.3
		3	22.50	1.66	0.82	0.80	13.3		22.50	1.66	0.82	0.80	13.3
		5	40.50	1.37	0.70	0.65	13.3		40.50	1.37	0.70	0.65	13.3
		7	61.50	1.29	0.67	0.61	13.1		79.00	1.40	0.67	0.64	12.7
		15	57.50	1.20	0.68	0.60	13.0		75.50	1.42	0.67	0.65	13.1
		31	48.00	1.15	0.70	0.58	13.0		57.00	1.41	0.67	0.64	13.1
	Lambda	C	136.50	1.32	0.77	0.60	16.5		136.50	1.32	0.77	0.60	16.5
		2	18.00	1.60	0.85	0.77	16.5		18.00	1.60	0.85	0.77	16.5
		3	19.50	1.39	0.74	0.62	16.5		19.50	1.39	0.74	0.62	16.5
		5	70.00	1.36	0.71	0.61	16.5		70.00	1.36	0.71	0.61	16.5
		7	69.00	1.36	0.72	0.60	15.4		88.50	1.36	0.72	0.60	15.5
		15	72.50	1.28	0.74	0.58	16.1		98.00	1.38	0.71	0.61	13.5
		31	45.50	1.27	0.76	0.58	16.5		81.50	1.35	0.72	0.59	15.7
	Node	C	587.00	0.98	0.71	0.53	6.6						

Table B.3: Pruning results for the *Leman* benchmark.

Discretization function	Pruning method	D	# of epochs	% of misclassification (mean)			Size	Discretization function	# of epochs	% of misclassification (mean)			Size
			(mean)	train.	val.	test			(mean)	train.	val.	test	
W_{max}	*Randomize*	C	265.50	3.91	1.89	1.84	60.4	*Power_of_two* _W_{max}	162.00	3.71	2.06	1.21	72.6
		2	95.50	5.60	2.97	3.28	56.4		59.50	6.89	3.49	4.08	70.6
		3	116.00	8.80	6.29	7.07	56.2		42.00	8.83	6.23	7.24	71.3
		5	55.50	5.09	2.57	2.76	58.8		80.50	5.20	2.74	2.53	68.8
		7	47.00	4.74	2.51	2.13	59.5		40.00	4.06	2.57	1.67	71.6
		15	81.00	4.00	2.11	1.90	56.0		67.00	4.14	2.06	1.95	68.8
		31	133.50	4.26	1.66	2.13	50.0		115.00	3.80	2.17	1.72	63.5
	Variance	C	111.50	3.89	2.29	1.67	80.7		111.50	3.89	2.29	1.67	80.7
		2	71.00	6.63	3.26	4.14	76.2		71.00	6.63	3.26	4.14	76.2
		3	29.00	8.57	7.26	7.82	78.1		29.00	8.57	7.26	7.82	78.1
		5	30.50	4.31	2.74	2.47	80.7		30.50	4.31	2.74	2.47	80.7
		7	27.50	3.91	2.91	1.61	80.7		50.00	4.03	2.46	1.49	77.4
		15	59.50	3.74	2.40	1.61	79.8		72.50	3.94	2.17	1.61	77.5
		31	88.50	3.69	2.23	1.21	75.7		65.50	3.69	2.29	1.09	78.2
	Weight	C	103.50	3.83	2.23	1.26	81.9		103.50	3.83	2.23	1.26	81.9
		2	60.50	7.46	4.29	4.89	81.2		70.00	9.94	6.46	7.99	80.4
		3	37.00	12.00	9.54	11.09	81.9		21.50	8.69	6.80	7.76	81.9
		5	15.50	5.26	3.14	3.22	81.9		24.50	5.11	2.97	2.99	81.9
		7	27.50	4.86	3.31	2.53	81.9		24.50	4.03	2.17	1.21	81.9
		15	43.00	3.94	2.34	1.49	81.9		66.50	3.86	2.29	1.26	79.2
		31	68.00	3.51	2.34	1.09	77.6		71.50	3.83	2.23	1.38	78.7
	Auto	C	58.50	4.00	2.29	1.21	88.0		58.50	4.00	2.29	1.21	88.0
		2	47.00	7.37	3.31	5.29	87.1		47.00	7.37	3.31	5.29	87.1
		3	17.50	12.51	9.31	10.98	88.0		17.50	12.51	9.31	10.98	88.0
		5	10.50	4.94	2.74	2.53	88.0		10.50	4.94	2.74	2.53	88.0
		7	15.50	4.26	2.57	2.13	88.0		34.50	3.74	2.17	1.44	88.0
		15	48.50	3.74	2.11	1.55	88.0		63.00	3.69	2.29	1.55	84.4
		31	61.50	3.60	2.34	1.67	85.3		41.50	3.63	2.29	1.26	88.0
	Lambda	C	58.50	4.00	2.29	1.21	88.0		58.50	4.00	2.29	1.21	88.0
		2	39.50	7.40	3.37	5.34	88.0		39.50	7.40	3.37	5.34	88.0
		3	17.50	12.49	9.37	11.03	88.0		17.50	12.49	9.37	11.03	88.0
		5	15.50	5.34	3.03	2.64	88.0		15.50	5.29	3.03	2.64	88.0
		7	23.50	4.20	2.69	2.24	88.0		30.50	3.69	2.23	1.44	88.0
		15	60.50	3.80	2.23	1.55	88.0		44.50	3.71	2.34	1.21	88.0
		31	41.00	3.66	2.34	1.26	88.0		34.50	3.66	2.34	1.26	88.0
	Node	C	172.50	3.89	2.31	1.02	60.5						
Symmetrical	*Randomize*	C	135.50	3.71	2.23	1.55	75.6						
		2	69.00	4.23	2.86	1.78	74.3						
		3	175.50	4.26	2.74	1.72	73.5						
		5	170.50	4.00	2.34	1.90	61.5						
		7	238.50	4.20	2.40	1.90	54.1						
		15	131.50	4.17	2.34	2.30	64.3						
		31	131.50	3.94	2.51	2.01	66.8						
	Variance	C	111.50	3.89	2.29	1.67	80.7						
		2	160.50	4.31	2.17	1.90	63.9						
		3	213.00	4.54	2.11	1.78	59.0						
		5	294.50	4.34	2.11	2.24	46.2						
		7	226.00	3.97	2.17	2.07	58.4						
		15	169.00	4.06	2.34	2.30	63.7						
		31	148.50	4.26	2.34	2.24	67.6						
	Weight	C	103.50	3.83	2.23	1.26	81.9						
		2	130.50	4.60	2.86	1.67	73.9						
		3	180.50	4.69	2.74	1.95	74.2						
		5	141.00	4.26	2.63	2.59	71.7						
		7	90.50	4.26	2.91	2.99	78.4						
		15	108.00	4.20	2.80	2.18	75.9						
		31	50.50	4.51	2.46	2.47	79.2						
	Auto	C	58.50	4.00	2.29	1.21	88.0						
		2	186.50	3.77	2.29	1.49	72.9						
		3	203.50	4.29	2.51	1.38	69.0						
		5	272.50	3.54	2.51	1.90	61.3						
		7	195.50	3.91	2.63	2.47	69.0						
		15	166.00	4.20	2.74	2.99	71.5						
		31	106.50	4.06	2.63	2.36	79.9						
	Lambda	C	58.50	4.00	2.29	1.21	88.0						
		2	55.00	4.03	2.34	1.90	88.0						
		3	66.00	4.00	2.34	1.67	88.0						
		5	34.50	4.40	2.80	3.22	88.0						
		7	68.00	4.40	2.97	3.05	88.0						
		15	52.00	4.20	2.97	2.93	88.0						
		31	52.50	3.94	2.46	2.36	88.0						

Table B.4: Pruning results for the *Cancer* benchmark.

Discretization function	Pruning method	D	# of epochs (mean)	% of misclassification (mean) train.	val.	test	Size	Discretization function	# of epochs (mean)	% of misclassification (mean) train.	val.	test	Size
W_{max}	*Randomize*	C	1896.50	0.11	0.00	4.55	126.8	*Power_of_two* _W_{max}	1746.50	0.11	0.00	4.77	128.0
		2	142.50	7.42	6.00	7.50	112.2		144.00	8.76	5.33	10.91	112.5
		3	205.50	3.48	2.67	7.05	125.6		109.00	3.71	2.89	6.82	122.4
		5	45.50	1.12	0.22	4.09	126.8		195.00	2.02	0.00	4.55	126.8
		7	54.00	1.24	0.00	4.09	125.6		36.50	0.90	0.00	4.32	128.0
		15	38.00	0.34	0.00	3.64	126.8		23.50	0.22	0.00	4.55	128.0
		31	21.50	0.34	0.00	4.32	126.8		29.50	0.34	0.00	3.64	128.0
	Variance	C	3563.50	0.22	0.00	4.55	100.7		3673.50	0.11	0.00	4.77	111.8
		2	111.50	6.29	5.11	10.00	92.0		103.50	7.75	7.33	10.91	101.7
		3	348.50	2.92	2.00	6.14	100.7		59.00	5.06	2.44	6.59	108.1
		5	484.00	1.12	0.22	4.55	94.2		484.50	0.56	0.00	3.64	105.3
		7	35.50	1.12	0.22	4.09	99.3		29.50	1.24	0.22	3.86	111.8
		15	25.50	0.45	0.00	3.64	100.7		25.50	0.34	0.22	5.23	111.8
		31	18.50	0.00	0.00	4.77	100.7		19.50	0.56	0.67	3.86	111.8
	Weight	C	2654.50	0.11	0.00	5.00	117.9		2654.50	0.11	0.00	5.00	117.9
		2	128.50	8.31	6.22	11.59	106.7		128.50	8.31	6.22	11.59	106.7
		3	109.50	3.71	2.22	6.59	107.7		109.50	3.71	2.22	6.59	107.7
		5	495.50	1.69	0.00	3.86	114.4		495.50	1.69	0.00	3.86	114.4
		7	189.00	0.45	0.00	3.64	117.9		38.00	0.11	0.67	3.64	115.4
		15	18.50	0.34	0.00	3.64	117.9		40.50	0.22	0.00	4.32	116.6
		31	28.00	0.00	0.00	5.00	117.9		32.00	0.34	0.00	3.41	117.9
	Auto	C	1746.50	0.11	0.00	4.77	128.0		1746.50	0.11	0.00	4.77	128.0
		2	133.50	6.40	5.33	7.27	111.5		133.50	6.40	5.33	7.27	111.5
		3	239.50	2.36	2.00	6.14	120.6		239.50	2.36	2.00	6.14	120.6
		5	489.50	1.24	0.22	4.77	125.5		489.50	1.24	0.22	4.77	125.5
		7	39.50	1.24	0.00	3.41	128.0		26.50	1.46	0.00	3.41	128.0
		15	39.00	0.22	0.00	3.41	128.0		34.00	0.22	0.22	4.55	128.0
		31	20.00	0.11	0.00	3.64	128.0		34.00	0.22	0.22	2.50	126.7
	Lambda	C	5200.50	0.11	0.00	4.77	126.6		5200.50	0.11	0.00	4.77	126.6
		2	53.00	8.31	6.00	10.00	126.6		53.00	8.31	6.00	10.00	126.6
		3	74.50	3.93	2.44	7.27	109.2		74.50	3.93	2.44	7.27	109.2
		5	45.00	1.91	0.89	6.82	126.6		45.00	1.91	0.89	6.82	126.6
		7	34.50	0.90	0.00	4.09	126.6		28.50	1.57	0.22	4.09	126.6
		15	30.00	0.22	0.00	3.18	126.6		27.50	1.46	0.00	3.64	126.6
		31	20.00	0.45	0.22	4.09	126.6		38.00	0.11	0.22	3.86	126.6
	Node	C	2762.50	0.11	0.00	4.55	108.8						
Symmetrical	*Randomize*	C	1896.50	0.11	0.00	4.55	126.8						
		2	51.50	7.53	2.67	7.50	126.8						
		3	44.00	5.39	1.33	5.00	125.6						
		5	78.00	3.71	2.00	4.77	121.5						
		7	84.00	4.16	2.00	7.05	118.6						
		15	69.00	3.37	2.00	5.68	124.4						
		31	53.50	1.35	0.67	4.55	125.6						
	Variance	C	3673.50	0.11	0.00	4.77	111.8						
		2	65.50	5.84	2.22	7.50	111.8						
		3	76.50	4.83	2.67	5.45	111.8						
		5	68.00	4.83	2.22	7.27	110.2						
		7	81.00	4.38	2.89	5.68	107.7						
		15	70.00	2.25	1.33	5.00	106.9						
		31	50.50	0.45	0.44	5.45	110.5						
	Weight	C	2654.50	0.11	0.00	5.00	117.9						
		2	47.00	6.40	2.00	7.05	117.9						
		3	80.50	6.18	2.22	6.14	117.9						
		5	84.00	3.48	2.00	5.45	114.3						
		7	67.50	3.15	3.11	6.14	112.1						
		15	65.00	1.91	1.33	7.05	114.4						
		31	74.00	1.01	0.67	5.45	115.6						
	Auto	C	1746.50	0.11	0.00	4.77	128.0						
		2	187.50	3.82	0.89	4.32	102.4						
		3	179.50	3.15	0.67	4.09	103.7						
		5	157.00	1.57	0.67	4.55	101.9						
		7	107.50	2.70	0.44	4.77	111.8						
		15	99.00	1.24	1.11	5.91	117.9						
		31	33.00	1.91	0.44	5.45	128.0						
	Lambda	C	5200.50	0.11	0.00	4.77	126.6						
		2	59.00	5.62	2.89	6.36	126.6						
		3	94.50	6.07	2.67	6.59	126.6						
		5	58.00	3.71	2.22	6.82	126.6						
		7	52.00	4.16	2.44	6.14	126.6						
		15	50.50	4.27	2.89	6.59	126.6						
		31	39.00	1.57	0.22	4.55	126.6						

Table B.5: Pruning results for the *Wine* benchmark.

Discretization function	Pruning method	D	# of epochs (mean)	% of misclassification (mean) train.	val.	test	Size	Discretization function	# of epochs (mean)	% of misclassification (mean) train.	val.	test	Size
W_{max}	*Randomize*	C	303.00	15.39	18.16	5.33	408.2	*Power_of_two* $\_W_{max}$	313.00	18.29	17.63	6.93	383.0
		2	388.00	25.00	22.37	9.33	218.4		245.00	21.58	21.32	13.33	303.6
		3	174.00	21.84	18.95	9.60	331.2		190.00	25.00	24.21	13.07	320.8
		5	147.00	24.47	21.32	9.87	352.8		304.00	24.21	21.84	12.80	300.8
		7	147.00	20.13	20.00	5.87	353.8		534.00	21.71	20.79	10.40	167.2
		15	320.00	20.66	21.05	9.60	255.4		113.00	18.42	18.68	6.93	318.2
		31	190.00	15.79	20.00	7.47	345.6		143.00	18.55	17.37	8.27	318.2
	Variance	C	222.00	15.39	18.95	5.60	436.2		222.00	15.39	18.95	5.60	436.2
		2	289.00	27.11	23.68	16.80	212.8		289.00	27.11	23.68	16.80	212.8
		3	214.00	21.05	20.26	6.13	264.4		214.00	21.05	20.26	6.13	264.4
		5	172.00	20.13	20.53	5.33	346.8		103.00	19.47	20.00	5.07	366.8
		7	99.00	19.61	22.11	8.00	370.6		145.00	16.84	19.74	5.60	293.4
		15	144.00	21.97	24.21	13.33	313.8		223.00	14.21	20.79	6.40	282.2
		31	202.00	22.50	23.42	15.20	236		213.00	14.47	20.26	5.60	284.8
	Weight	C	245.00	13.42	20.79	6.67	443.8		245.00	13.42	20.79	6.67	443.8
		2	196.00	29.34	28.42	22.40	364.8		196.00	29.34	28.42	22.40	364.8
		3	83.00	20.39	20.53	6.93	434.0		83.00	20.39	20.53	6.93	434.0
		5	213.00	24.74	25.79	15.47	390.4		213.00	24.74	25.79	15.47	390.4
		7	170.00	23.68	24.47	15.20	359.4		166.00	29.87	28.68	22.93	361.6
		15	123.00	22.37	23.95	14.93	387.6		190.00	15.00	20.26	5.87	364
		31	121.00	27.24	28.42	23.20	392.4		139.00	15.39	21.05	4.80	392.6
	Auto	C	44.00	16.71	17.11	5.60	592.0		44.00	16.71	17.11	5.60	592.0
		2	177.00	26.58	22.89	15.73	523.6		177.00	26.58	22.89	15.73	523.6
		3	101.00	25.26	23.68	14.67	580.2		101.00	25.26	23.68	14.67	580.2
		5	54.00	19.61	20.00	5.33	580.2		97.00	20.26	19.47	6.67	557.8
		7	107.00	18.42	21.32	4.80	547.2		65.00	18.16	19.74	6.40	569.6
		15	55.00	17.24	20.00	4.80	592		51.00	15.79	20.79	4.53	592.0
		31	60.00	16.32	19.21	5.07	580.2		78.00	18.82	20.00	5.87	592.0
	Lambda	C	44.00	16.71	17.11	5.60	592.0		44.00	16.71	17.11	5.60	592.0
		2	67.00	19.34	20.53	7.20	592.0		67.00	19.34	20.53	7.20	592.0
		3	55.00	20.00	19.47	6.13	592.0		55.00	20.00	19.47	6.13	592.0
		5	78.00	19.61	21.84	5.33	592.0		78.00	19.61	21.84	5.33	592.0
		7	66.00	18.16	18.16	5.33	592.0		38.00	18.03	18.95	5.07	592.0
		15	35.00	17.63	20.53	4.80	592.0		58.00	15.39	20.26	4.00	592.0
		31	51.00	16.05	19.47	5.33	592.0		82.00	19.08	19.74	6.13	592.0
Symmetrical	*Randomize*	C	61.00	15.66	17.63	5.33	580.2						
		2	469.00	21.58	19.21	6.40	283.6						
		3	315.00	20.53	19.74	5.60	352.4						
		5	250.00	19.87	17.89	7.20	423.4						
		7	590.00	23.55	19.47	10.40	256.8						
		15	327.00	22.63	16.84	8.27	384.2						
		31	228.00	19.61	17.89	5.33	458.8						
	Variance	C	222.00	15.39	18.95	5.60	436.2						
		2	209.00	25.92	22.11	13.87	296.0						
		3	165.00	21.58	21.32	6.93	320.4						
		5	276.00	20.53	17.89	6.40	250.4						
		7	107.00	20.39	19.21	7.20	319.4						
		15	422.00	24.87	20.53	9.33	213.8						
		31	505.00	19.61	18.68	7.47	128.4						
	Weight	C	245.00	13.42	20.79	6.67	443.8						
		2	327.00	30.79	28.16	22.13	249.0						
		3	623.00	19.47	20.53	7.47	177.2						
		5	451.00	22.50	21.84	9.87	266.0						
		7	581.00	20.79	22.11	9.07	244.8						
		15	245.00	18.42	21.58	7.73	298.8						
		31	262.00	19.08	20.26	6.67	298.8						
	Auto	C	44.00	16.71	17.11	5.60	592.0						
		2	203.00	18.55	21.05	5.60	460.8						
		3	178.00	18.68	19.74	5.60	513.0						
		5	189.00	19.08	20.26	6.40	511.8						
		7	153.00	18.95	21.05	5.60	544.8						
		15	99.00	20.39	18.16	6.40	580.2						
		31	51.00	19.47	19.21	8.27	580.2						
	Lambda	C	44.00	16.71	17.11	5.60	592.0						
		2	49.00	21.05	20.00	6.93	592.0						
		3	79.00	19.21	19.21	6.67	592.0						
		5	61.00	19.21	20.79	8.00	592.0						
		7	60.00	20.79	20.00	7.47	592.0						
		15	78.00	20.00	18.95	6.93	592.0						
		31	46.00	19.08	18.95	7.20	592.0						

Table B.6: Pruning results for the *Heart* benchmark.

Discretization function	Benchmark	D	% of misclassification test set	Size	Benchmark	% of misclassification test set	Size
W_{max}	*Wine*	C	3.18	33.2	*Cancer*	2.19	47
		2	7.73	98.6		4,25	82
		3	6.82	73.2		4,14	73
		5	5.45	45.0		11,15	73,2
		7	6.82	34.6		2,76	82
		15	4.09	27.8		1,49	53,4
		31	2.73	33.2		2,07	34,4

Table B.7: Results for the Wine and Cancer benchmarks (average over five simulations) with *Variance* pruning. Pruning condition for the Wine benchmark: mean square error on training set ≤ 3.0. Pruning condition for the Cancer benchmark: mean square error on training set ≤ 9.0.