Data-Quantization Policies for Power and Area Minimization of Hardware Neural Networks*

J. L. Ayala, M. López-Vallejo
Departamento de Ingeniería Electrónica
Universidad Politécnica de Madrid, Madrid (Spain)
{jayala,marisa}@die.upm.es

Abstract

Hardware implementation of neural networks, like other parallel algorithms, are heavily constrained by chip area and power consumption. In the present work, we propose an experimental methodology and an area and power minimization procedure based on the use of different data quantization policies. An in depth analysis of the effect of data quantization on network convergence has been performed and the hardware implementation of different learning algorithms has been characterized on area and learning time. Finally, a set of design rules on the constrained hardware implementation of neural networks has been devised.

1. Introduction

Hardware implementation of parallel algorithms like neural networks presents serious constraints on chip area and power consumption. These are two of the difficulties that researchers have found on this topic, moving to a software implementation of such algorithmic rules. However, there are applications in which a more efficient operation is needed, and a hardware execution could be required. These applications usually need to operate with Real Time constraints that cannot be satisfied by software. We can find this kind of applications in areas like biotechnology with on-line classification of electrical biosignals[1], Retinal Implant Project from MIT[2], or classification and detection of medical images[3]. In the industrial field, we can refer to complex dynamic systems control or quality control[4]; and detection of particles in the high-energy physics field.

Hardware neural networks, like other data processing algorithms, spend a large area in the data buses due to the necessity of transferring numerical data codified with long words, and due to the arithmetic circuits that handle these operands, which usually exhibit high complexity. These two hardware elements (data buses and arithmetic circuits) with the accesses to the internal and external memories are the main responsible of the power consumption and the chip area in the hardware implementation[5]. This problem is even more serious if a high-reconfigurability is desired (because more than one processing unit[1] is required) or an on-chip learning algorithm is desired (due to the arithmetic circuits that perform the error and partial results calculation)[6].

In the design of Integrated Circuits, the reduction of the data word length results on chip area minimization due to the simplification of the arithmetic circuits and narrower data buses. Moreover, glitch power consumption is reduced in the bit-lines and the energy dissipation due to the combinational logic is also economized.

However, data quantization is a tricky issue because network convergence is very sensitive to saturations and quantization errors in synaptic weights, partial computed results and outputs from the activation function. This sensitivity can be observed in long training periods and, what is more serious, non convergence problems. Several hardware implementations of neural networks can be found in the literature[7]. However, these related works do not perform an accurate study of the effect of data quantization on network convergence, mainly because they lack of on-chip learning capability. Therefore, these implementations do not optimize chip and power consumption, which is the objective of our current research.

The paper is organized as follows: section 2 presents the aims of the experimentation, the experimental methodology that has been followed and the methods and tools that have been employed. Section 3 summarizes the experimental results and a discussion about them is summarized in section 4. Finally, some conclusions are drawn.

2. Approach

As it was said before, the main objective of our present research is to provide a set of design rules that guide the hardware implementation of neural networks with area and power consumption constraints, together with the analysis of the most suitable learning algorithm for being implemented on a hardware platform. Therefore, several experiments have been performed in order to analyze the data quantization effect on the network convergence and learning error.

These experiments will simulate different quantization policies when a learning algorithm is implemented on a hardware platform. In this way, a hardware description language (VHDL) has been employed for describing these hardware designs, and an implementation with standard

*This work is funded by CICYT project TIC2000-0583-C02-02

[1]To provide parallel computation

cells has been selected. However, it is important to remark that the experimental results we present here are not dependent on this selection.

2.1. Experimental Methodology

To achieve the proposed purposes, an experimental methodology has been established. This can be extended in the future to include more learning algorithms, or other design constraints like reconfigurability or applicability.

This experimental methodology can be summarized in the following steps:

1. Selection of the learning algorithm and network topology;
2. Partition of the previous algorithm into independent phases;
3. Selection of the quantization method;
4. Application of the quantization to the learning phases;
5. Hardware description and synthesis of the selected learning algorithm and quantization policy;
6. Experimental data collection and analysis of results;

Concerning the first step, there is a great diversity of learning algorithms that can be applied to the training of neural networks. Most of them have been collected from more general optimization theories (like backpropagation or learning vector quantization), but others have been designed on purpose for neural topologies (Kohonen self-organizing maps or adaptive resonance theory). Some of these algorithms have a very restricted field of activity, being used in specific tasks and problems. However, there are certain training methods that can be applied in most applications and exhibit good convergence time. Moreover, not all the learning algorithms can be easily[2] implemented on hardware platforms due to the complex arithmetic calculations involved. Therefore, the learning algorithm selection phase takes into account these considerations when selecting the method to be checked.

The effect of data quantization on the convergence of the network is analyzed partitioning the learning algorithm into several phases. With this approximation, it is possible to select different quantization policies for different training phases (only if these phases have independent hardware module implementations). More precisely, we have employed the following partition:

Activation function: Quantization of the activation function in a look-up table or quantization in the buses carrying these values if another approximation is employed.

Differentiated activation function: Quantization of the differentiated activation function in a look-up table or quantization in the buses carrying these values if another approximation is employed.

[2]Without an expensive area waste

Forwardpropagation phase: Quantization of the buses carrying numerical data during the forwardpropagation phase in the network.

Backpropagation phase: If can be applied, quantization of the buses carrying numerical data during the backpropagation phase in the network.

Synaptic weights: Quantization of synaptic weights in registers and storing devices.

Arithmetic calculations: Arithmetic calculations performed in the learning and recall phases (error correction terms, weight correction, multiply-and-accumulate operations, and so on).

The quantization policies that have been evaluated are: fixed-point quantization with different bit-size of the integer and fractional part, and floating-point quantization with different bit-size of the mantissa and the exponent. The application of these policies to the phases of the learning algorithm, and the collection and analysis of the experimental results will be described in the following sections.

As it was said before, the hardware implementation of the learning algorithms has been performed with VHDL and their descriptions have been synthesized using the tools provided by *Synopsys*. In this phase of the experimentation, a complete hardware design flow has been followed, from the RTL simulation and description, to the synthesis and post-synthesis verification.

2.2. Methods and Tools

The experimental procedure described in section 2.1 has been followed with the use of a neural network simulator. This simulator, called SNNS (Stuttgart Neural Network Simulator[8]) has been selected because it provides the source code of diverse learning algorithms, that can be improved for performing the experimental setup. More precisely, the changes we made in the source code were required to turn the 32 bits floating point notation employed for the UNIX version of SNNS into the desired one. In this way, fixed-point quantization has been simulated with the following assumptions:

- When an irrational number is codified in fixed-point notation, it has to be quantizated and rounded up;
- A Q_n fixed-point quantization implies a *n-bits* fractional part length into a *l-bits* word length (see figure 1);

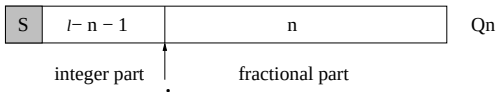

Figure 1. Fixed-point nomenclature

- The Q_n and *l-bits* fixed-point representation of data is:

$$x = (-1)^{b_{l-1}} \cdot \sum_{i=0}^{i=l-2} b_i \cdot 2^{i-n}$$

- The accurate definition of basic arithmetic calculations in fixed-point notation is:

  Addition: Both operands must employ the same notation (Q_n) so precision is not downgraded;

  Multiplication: One operand employing Q_n notation and the other one employing Q_m notation generate a Q_{n+m} result;

Concerning the floating-point quantization, it has been simulated with the following assumptions:

- A floating-point number has four parts: a sign, a mantissa, a radix, and an exponent. The sign is either a 1 or -1. The mantissa, always a positive number, holds the significant digits of the floating-point number. The exponent indicates the positive or negative power of the radix that the mantissa and sign should be multiplied by. The four components are combined as follows to get the floating-point value:

$$sign \cdot mantissa \cdot radix^{exponent}$$

- Floating-point numbers have multiple representations, because the mantissa of any floating-point number can always be multiplied by some power of the radix and change the exponent to get the original number. We have always chosen a representation that ensures the correct codification of the integer part and six decimal numbers;

The previous quantization methods have been applied to the different phases that can be found in the source code of the neural network simulator and they have been summarized in section 2.1. The problems that have been solved during the experiments are complex enough to show a good response when they are solved with a neural network algorithm.

3. Experimental Results

This section presents the most descriptive results obtained when solving different problems with their appropriate neural network topology and with different quantization approaches. The learning algorithms and network topologies that have been employed are[9]:

MLP trained with Backpropagation: It is simply a gradient descent method to minimize the total square error of the output computed by the net. The very general nature of the training method means that a MLP[3] trained with backpropagation can be used to solve problems in many areas. Also, the arithmetic calculations involved during the training phase are simple enough to be implemented in hardware without a large area waste.

MLP trained with Batch backpropagation: A modification of the previous algorithm. In this learning method, weight correction terms are accumulated for several patterns (or an entire epoch) before doing a single weight adjustment, rather than updating the weights after each pattern is presented. This procedure has a smoothing effect on the correction terms that could increase the chances of convergence to a local minimum. There are no modifications in the hardware implementation with respect to the original backpropagation.

RBF trained with Gradient Descent: These networks can be used for approximating functions and recognizing patterns. Network topologies are designed with only a hidden layer, and a learning algorithm based on simple gradient descent can be used. Therefore, the hardware realization of the learning algorithm does not present higher complexity than the backpropagation one but it needs up to three times more area for the implementation of three multipliers and three adders in order to perform in parallel the updating of synaptic weights and the two parameters of the gradient descent learning algorithm (μ and σ).

For the RBF (*Radial Basis Function*) network, a letter recognition example has been used[8]. The network inputs are 5x7 binary input matrix. The network has 10 hidden units in one hidden layer. Each of the 26 output units represents a capital letter and shows an output of 1 if the input pattern belongs to the proper class, otherwise 0. For the backpropagation and batch backpropagation algorithms, the XOR-problem has been used[8].

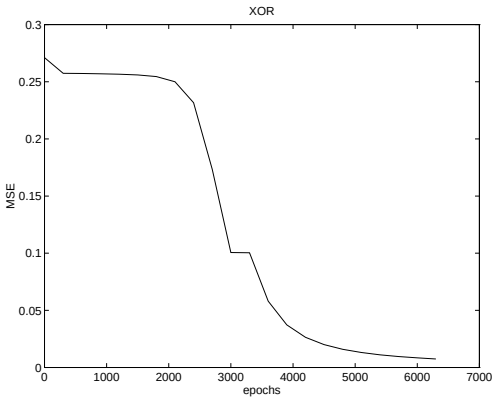

Figure 2. Backpropagation simulation (reference)

3.1. Backpropagation experiments

We have selected the most meaningful graphs of our experimental results corresponding to the XOR-problem and the backpropagation learning algorithm, to extract some interesting conclusions.

Figure 2 shows the mean square error (MSE) of the training phase when a 32 bits floating-point simulation[4] is performed; in the *x-axis*, the number of training phases

[3] Multi-Layer Perceptron

[4] Default precision of SNNS

(epochs) is considered. This graph will be the reference for the following experiments.

Figure 3 shows a 24 bits (16 bits for the integer part and 8 bits for the fractional one) fixed-point quantization of the activation function (direct and differentiated forms) in continuous line, together with a 10 bits (6 bits for the integer part and 4 bits for the fractional one) fixed-point quantization of the direct form of the activation function and a 24 bits (16 bits for the mantissa and 8 bits for the exponent) floating-point quantization of the differentiated form in crossed line.

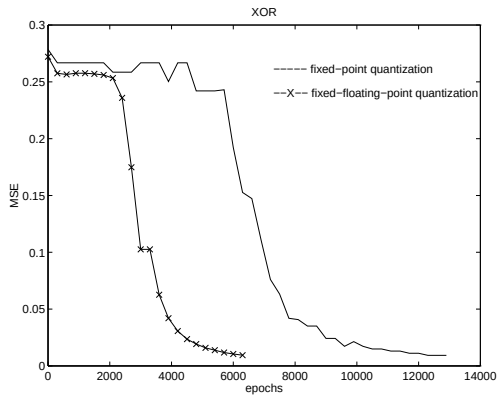

Figure 3. Backpropagation simulation (quantization of the activation function)

Finally, in figure 4 we present the results related to the 32 bits (16 bits for the integer part and 16 bits for the fractional one) fixed-point quantization of the complete architecture in continuous line, while the results of a 24 bits fixed-point quantization of the synaptic weights and a floating-point quantization of remaining phases are shown in crossed line.

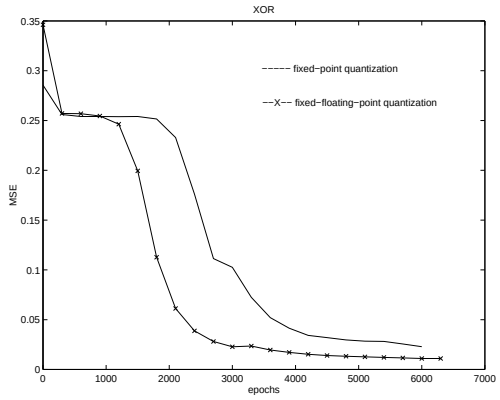

Figure 4. Backpropagation simulation (quantization of the architecture)

The analysis of these results will be postponed to section 4 where the three learning algorithms and the quantization methods will be compared. Nevertheless, in the previous figures we can notice how the backpropagation algorithm is very sensitive to the quantization of the activation function, specifically the differentiated form of this. On the contrary, synaptic weights can be quantizated with shorter words without a high decrease in performance.

3.2. Batch backpropagation experiments

As it was shown in section 3.1, figure 5 shows the reference simulation, figure 6 depicts the fixed-point quantization of the complete activation function and the direct form of this in continuous and crossed line, respectively. On the other hand, figure 7 shows the fixed-point quantization of the synaptic weights and presents a reduced fixed-point quantization (10b-4b) of the forwardpropagation phase and synaptic weights.

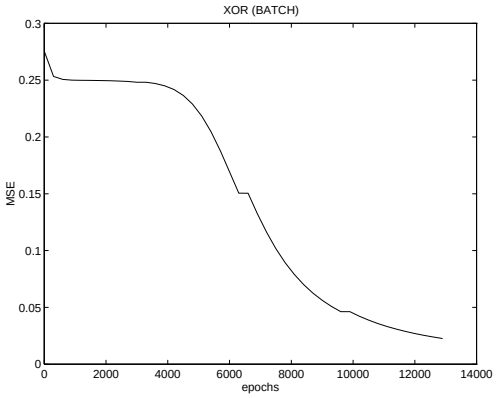

Figure 5. Batch backpropagation simulation (reference)

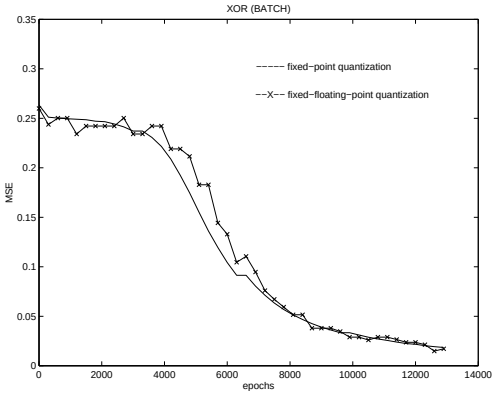

Figure 6. Batch backpropagation simulation (quantization of the activation function)

In this case (batch-backpropagation learning algorithm), the sensitivity of the network convergence to the quantization policies is clearly lower than the one obtained in the backpropagation case. The learning rate does not experiment a heavy increase when the quantization method is more restrictive. However, these rates are always much longer than the ones obtained by traditional backpropagation.

3.3. Radial Basis Function experiments

Relating to the experiments with a Radial Basis Function network, the results are presented in the following

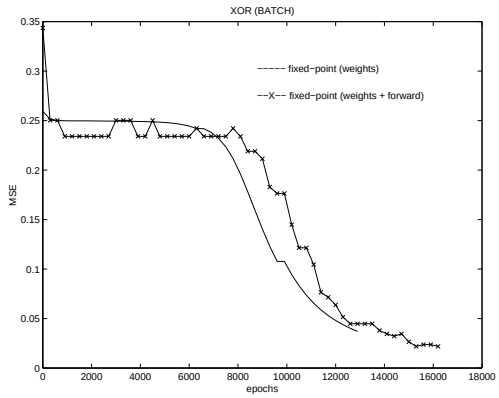

Figure 7. Batch backpropagation simulation (quantization of the architecture)

graphs. Figure 8 shows the reference simulation as usually, the fixed-point quantization of the activation function is shown in figure 9, a reduced fixed-point quantization (8b-3b) of synaptic weights and a fixed-point quantization of the complete architecture (buses and arithmetic circuits) appear in figure 10.

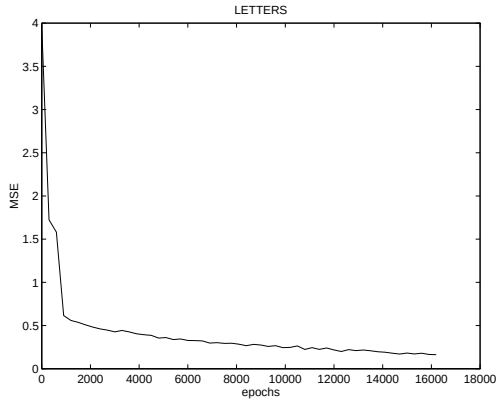

Figure 8. RBF simulation (reference)

With no doubt, RBF training presents the best behavior when a restricted quantization method is employed. Learning rates are slightly sensitive to shorter length words, but the training periods are much longer than the previous ones.

4. Discussion

As it has been shown with the experimental results collected in sections 3.1, 3.2 and 3.3, there are several issues that have to be taken into account when implementing a neural network algorithm (specifically, the learning algorithm) on a hardware platform not downgrading the convergence properties.

Summarizing these design rules, it has been proved how the gradient descent-based learning algorithms are very sensitive to data quantization of the differentiated form of the activation function. This can be explained paying attention to the expressions that regulate the learning

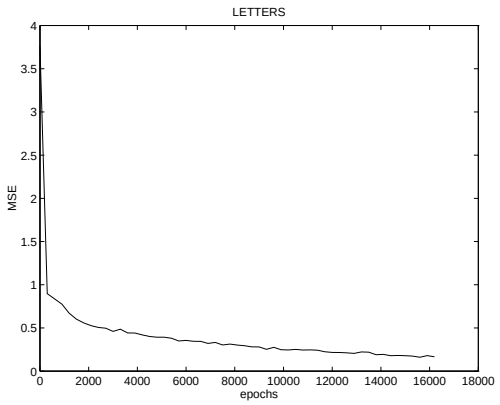

Figure 9. RBF simulation (quantization of the activation function)

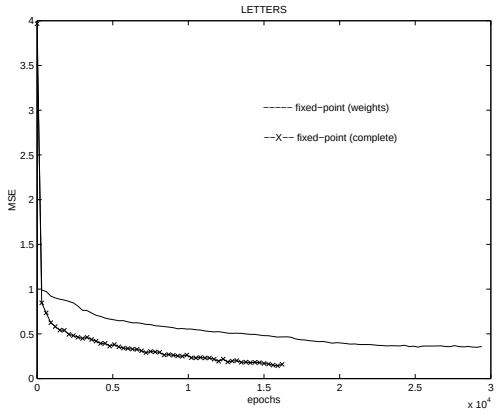

Figure 10. RBF simulation (quantization of the architecture)

phase (where w_{ij} and v_{jk} are the link weights, μ_j are the centers of the basis functions and σ_k are the bias of all output neurons):

$$\Delta w_{ij} = -\eta_w \frac{\partial E}{\partial w_{jk}} \quad \Delta v_{jk} = -\eta_v \frac{\partial E}{\partial v_{jk}} \tag{1}$$

$$\Delta \mu_j = -\eta_\mu \frac{\partial E}{\partial \mu_j} \tag{2}$$

$$\Delta \sigma_k = -\eta_\sigma \frac{\partial E}{\partial \sigma_k} \tag{3}$$

where equation 1 is applied to the three learning algorithms considered, but equations 2 and 3 only apply to the RBF network. An inaccurate quantization of the values involved in these computations can cause a longer training period or even the non convergence of the network. The effect of quantization on the learning algorithm is clearly shown on the differentiated form of the activation function (the differentiated term of the equations), and so it has to be coded with a higher precision. In RBF algorithms this effect is partially reduced.

The reduction in the number of bits employed for the quantization of the activation function produces smaller *Look-Up Tables* (if these are used) and arithmetic circuits

(if a combinational approach is devised for the approximated activation function). The minimization of the memory size is the main objective of a large variety of power reduction policies due to the heavy energy consumption of these devices when they are read or written[5].

Moreover, fixed-point quantization of the synaptic weights can be employed for shortening the word-length and simplifying the data buses. For the implementation of the arithmetic circuits, a floating-point quantization is desirable for improving the data precision and the codification of the weight correction terms (that can reach much more different values during the evolution of the algorithm). In this case, the size of the data buses can be reduced and consequently important savings are obtained in power consumption due to the data transfers.

Finally, it can be observed how Radial Basis Function networks need shorter word-length quantizations and they can be applied to problems with censored data (where data can be codified with a lower number of bits).

The previous learning algorithms and quantization policies have been synthesized with a technological library of 0.35 μm provided by *Mietec* (MTC45000). The following graphs show the area improvement due to the proposed quantizations, and the learning time penalty related to the reference architecture. Figure 11 presents a bar graph with the reference architecture and the two discussed quantizations of the activation function. The line traversing the bars reflects the time penalty due to the quantization policies in relation to the reference architecture. This line represents the percentual improvement or penalty on the learning time when a quantization policy is selected.

On the other hand, figure 12 shows a bar graph with the reference architecture and the two discussed quantizations of the remaining architecture; the line in the graph represents the same that the line of figure 11. All the area values are normalized to the reference of the learning algorithm, as soon as the learning time. Therefore, figures 11 and 12 allow us analyze the quantization effect once the learning algorithm has been selected.

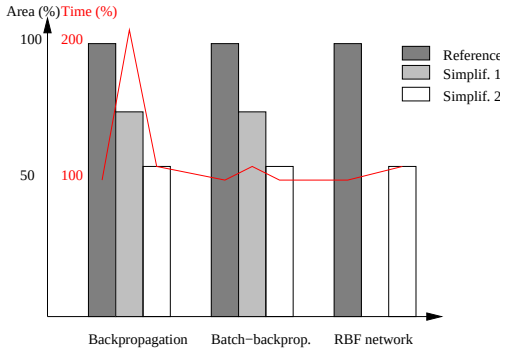

Figure 11. Quantization of the activation function

5. Conclusions

In the this paper, an experimental work oriented to the area and power minimization on hardware implementation

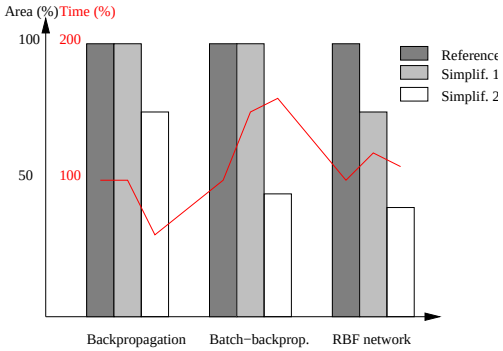

Figure 12. Quantization of the remaining architecture

of neural networks has been presented. A method of experimentation has been proposed, selecting the analysis of three versatile learning algorithms, but enabling the future extension to other ones.

Different quantization policies have been studied as an area and power reduction procedure, and the quantization effect on network convergence has been examined for the first time. Hardware implementations of the learning algorithms have been performed and these realizations have been characterized on chip area and learning time when different data quantization methods are employed.

Finally, a set of design rules on the hardware implementation of neural networks has been devised, providing the researcher with an heuristical and useful background on this issue.

6. References

[1] M. Maglaveras, "An adaptive backpropagation neural network for real time ischemia episodes detection: development and perfomance analysis using the European ST-T database", *IEEE Transactions on Biomedical Engineering*, vol. 45, pp. 805–813, Julio 1998.

[2] MIT, "The Retinal Implant Project", http://rleweb.mit.edu/retina.

[3] R. Manoharan E. B. Hanlon, "Prospects for in vivo Raman spectroscopy", *Phys. Med. Biol.*, , n. 45, 2000.

[4] P. J. G. Lisboa, "Industrial use of safety-related artificial neural networks", Technical report, Liverpool John Moores University, 2001.

[5] E. De Greef F. Catthoor, S. Wuytack, *Custom Memory Management Methodology : Exploration of Memory Organisation for Embedded Multimedia System Design*, Kluwer Academic Publishers, 1998.

[6] N. Sundararajan, *Parallel architectures for artificial neural networks*, IEEE Computer Society, 1998.

[7] P. Ienne, "Digital hardware architectures for neural networks", *SPEEDUP Journal*, vol. 1, n. 9, June 1995.

[8] University of Stuttgart, *SNNS Stuttgart Neural Network Simulator. User manual, version 4.2*, 1998.

[9] S. Haykin, *Neural Networks: A Comprehensive Foundation*, Prentice Hall, 1999.