Published in the Proceedings of the
Fifth International Conference on Microelectronics for Neural Networks and Fuzzy Systems: MicroNeuro'96
Lausanne, Switzerland, February 12–14, 1996.

Hardware-Friendly Learning Algorithms for Neural Networks: an Overview

P. D. Moerland and E. Fiesler
IDIAP
CP 592, CH-1920 Martigny, Switzerland
E-mail: Perry.Moerland@idiap.ch

Abstract

The hardware implementation of artificial neural networks and their learning algorithms is a fascinating area of research with far-reaching applications. However, the mapping from an ideal mathematical model to compact and reliable hardware is far from evident. This paper presents an overview of various methods that simplify the hardware implementation of neural network models. Adaptations that are proper to specific learning rules or network architectures are discussed. These range from the use of perturbation in multilayer feedforward networks and local learning algorithms to quantization effects in self-organizing feature maps. Moreover, in more general terms, the problems of inaccuracy, limited precision, and robustness are treated.

1: Introduction

During the last decade it has been demonstrated that neural networks are capable of providing solutions to many problems in the areas of pattern recognition, signal processing, time series analysis, and many others. While software simulations are very useful for investigating the capabilities of neural network models, they cannot fulfill the need for real-time processing that is necessary for a successful application of neural networks to most real-world problems. To fully profit from the inherent parallelism of neural networks, hardware implementations are essential. However, the mapping of existing neural network algorithms or their resultant networks onto fast, compact, and reliable hardware is a difficult task. Therefore, learning rules which are better suited for hardware implementation have been proposed. These *hardware-friendly learning algorithms* can be divided into two subclasses, namely:

1. Adaptations of existing neural network learning rules that facilitate their hardware implementation and lead to a better exploitation of chip area and parallelism.

2. Learning algorithms that are by their conception suitable for hardware implementation.

An example of the first class are the perturbation algorithms that eliminate the calculation of the derivative of the activation function and the need for separate circuitry for the backward pass of the widely-used backpropagation algorithm [20] [43]. An example of the second class are cellular neural networks that represent a general class of networks, the original definition of which has even been given in terms of analog circuitry [9] and which, due to their local connectivity, are suited for VLSI implementation. In this paper an overview is given of hardware-friendly algorithms for various neural network models. However, before presenting the remedies, some of the typical problems encountered in the hardware implementation of neural networks are outlined.

2: Hardware implementations of neural networks: problems and constraints

Any kind of implementation of neural networks, be it analog electronic, digital electronic, optical, or their hybrid, brings along various constraints:

Accuracy As compared to the ideal neural network models, hardware implementations can only offer a limited accuracy. Examples of this phenomenon are: (i) the representation of weight values with a small number of bits as opposed to the real-valued weights in the model, (ii) non-uniformity of circuit components which are ideally supposed to be identical, (iii) non-linearity effects in components such as multipliers.

Area The design of hardware implementations requires a constant interplay between the accuracy required, the (chip) area available, and the degree of parallelism. Reliable elements are often available but their incorporation comes at the price of an area penalty or a reduction of the degree of parallelism.

One can envisage two different approaches to solve these hardware related problems. Firstly, an improvement of the hardware required for the implementation of neural networks is of crucial importance. For example, the use of pulse modulation techniques which combine the advantages of digital and analog electronics [16] or the design of compact and reliable components. The second approach is to try to overcome them by adapting existing learning algorithms or by designing new hardware-friendly algorithms, which is the focus of this paper.

3: Hardware-friendly learning algorithms

3.1: Multilayer feedforward networks

The most popular algorithm for training multilayer feedforward networks is doubtlessly the backpropagation algorithm, popularized by Rumelhart [35]. Its realization in analog hardware, however, poses some serious problems because of the need of an accurate derivative of the activation function in the calculation of the gradients on the error surface. Another disadvantage is the need for separate circuitry for the backward pass of the algorithm.

Perturbation algorithms The general idea behind perturbation algorithms is to obtain a direct estimate of the gradients by slightly perturbing the parameters of the network and using the forward path to measure the change in the network error; this also implies that the circuitry for the backward pass can be dispensed with. Another advantage is that no a priori knowledge of the non-linearity is used and hence, that the implementation is likely to be more robust for hardware non-linearities.

The first perturbation algorithm was the *Madaline-III* rule [43] and is based on neuron perturbation, that is, each gradient is estimated by perturbing the input

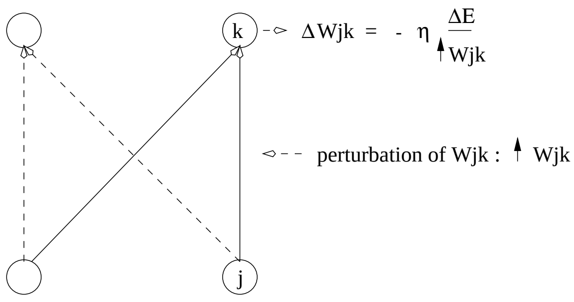

Figure 1. A schematic of the weight perturbation algorithm

value of a neuron. However, since each of the node perturbations has to be done serially the computational complexity increases considerably as compared with standard backpropagation. Moreover, it requires some extra circuitry for the addressing and selection of neurons.

Perturbation of the weights (see figure 1) eliminates some of this extra circuitry needed to implement node perturbation. It also performs better when limited precision weights are used [20]. This comes at the price of an even higher computational complexity which stems from the fact that all weights (except the weights to the output layer) have to perturbed serially.

The complexity of the weight perturbation algorithm is addressed and reduced viewing a perturbation of the weights incoming to a neuron as a summed weight perturbation of that neuron. The result is a weight perturbation method that improves upon the Madeline-III rule, since it does not require access to hidden neurons and has the same computational complexity [13]. This scheme has been actually implemented in hardware and shows good behaviour on some small benchmarks.

The loss of parallelism in the weight perturbation scheme can also be overcome by perturbing all weights simultaneously and using the resulting error to update the weights [2]. For a reliable estimate of a gradient, multiple perturbations should be performed, but this number is normally quite small compared to the total number of weights in the network. A similar approach has in fact been pursued by Cauwenberghs [8] studying in some more detail its convergence properties.

Chain-rule perturbation [18] also addresses the complexity of standard weight perturbation and employs a chain rule approximation of the gradient that enables all weights going out of a neuron to be perturbed in parallel. It improves upon the Madeline-III rule and summed weight perturbation [13] because it does not make any assumptions about the multiplication, allowing non-linear synapses which are typical for many analog implementations.

Local learning algorithms In [32] an anti-Hebbian local learning algorithm is described in which the weight update for a certain layer only depends on the input and output of that layer and a global error signal. This local learning rule circumvents the backpropagation of error signals that complicates the hardware implementation of the backpropagation algorithm. Although it is not a gradient descent rule, it is still guaranteed that the synaptic weights are updated in the error descent direction. Brandt and Lin [6] have also developed an algorithm that requires no explicit backpropagation of errors and uses information locally available at a neuron, most importantly the rates of change of the outgoing weights. One of their local algorithms is equivalent to the standard backpropagation algorithm. Their algorithm and especially the measuring of the rates of change of the weights might, however, still be hard to implement.

Training without derivatives The necessity of the derivative of the activation function in the backpropagation algorithm can be circumvented by an approximation, which only needs the non-linearity itself in the backward pass. This is, for example, established by a well-chosen Taylor expansion that offers a close approximation to the original algorithm [17].

A completely different approach to exclude derivatives from learning algorithms has been taken by Battiti and Tecchiolli. Their reactive tabu search is a heuristic method that can solve combinatorial optimization problems [5]. It can be applied to the training of neural networks by transforming the continuous space of the weights into a discrete one by a Gray encoding of the weight values. The heuristic that is used to obtain a new set of weights is to choose the configuration with the smallest error value, that differs only in one bit from the current configuration; because of the Gray encoding this method performs in fact a discretized form of steepest descent. In order to avoid cycles when changing the weight configuration and not to be confined to a limited part of the search space, some additional constraints are included in the heuristic. Another advantage of the reactive tabu search is the limited precision of the weights that is needed. These characteristics make it suitable for hardware implementation, as is illustrated by the TOTEM chip [4].

Complex backpropagation In some applications one would like a hardware implementation of an NN that accepts sinusoidal (complex-valued) signals. In [15] backpropagation is therefore extended to the complex domain allowing complex-valued inputs, weights, activation functions, and outputs. It carefully solves the problem of the design of a suitable complex activation function that is bounded, non-linear, differentiable, and easily implementable. These properties exclude for example the complex extension of the standard sigmoid function, which is unbounded.

Threshold networks The design of a compact digital neural network can be simplified considerably by using hard-limiting threshold gates as activation functions instead of a differentiable (sigmoidal) non-linearity. While training algorithms for two-layer threshold networks, that is perceptrons, abound, they are limited to solving linearly separable problems only. This constraint can be resolved by allowing more layers of neurons, but most algorithms for training these multilayer networks are based on gradient descent and require a differentiable activation function. The development of training algorithms for multilayer networks with a threshold as activation function is therefore an important issue for NN hardware implementation.

The Madaline-II rule [43] is closely related to the neuron perturbation of the Madaline-III. However, the discontinuities in a threshold network exclude the direct estimation of a gradient. Therefore, the error is minimized in the following way: a small neuron input in the second layer is perturbed to see whether an inversion of the activation value of this neuron reduces the Hamming error on the output neurons. If this is the case, the incoming weights of this neuron are adapted with a perceptron algorithm to reinforce this inversion. If not, the same procedure is repeated until the output layer is reached, the weights of which are directly updated by a perceptron algorithm.

Other approaches have been trying to use standard backpropagation to obtain threshold networks. In [10] the steepness of the sigmoidal non-linearity is gradually increased during training of the network to obtain a final network with only thresholds, using the fact that the sigmoid function approaches a threshold when the steepness parameter approaches infinity. This algorithm can be useful when off-line training of the network is appropriate.

There is a host of so-called constructive algorithms that are gradually building a threshold network by adding neurons and weights [37]. One recent example is the use of a geometrical approach to construct a multilayer threshold network [23]. The goal is to find a set of separating hyperplanes (hidden layer neurons) in the input pattern space with the property that inputs located between two neighbouring hyperplanes have the same target output. Another advantage of these constructive algorithms is that the number of neurons in the hidden layer need not be specified a priori. Of course, constructive algorithms are not well-suited for implementation in hardware, but the resulting compact threshold networks are.

3.2: Kohonen's self-organizing maps

The basic elements of Kohonen's algorithm for self-organizing maps that reproduce the input probability distribution in a compact way are, at time t:

(i) The selection of the neuron with weight vector $\mathbf{w}(t)$ closest to the input pattern $\mathbf{x}(t)$ (*winner neuron*), that is, with minimal distance $d(t)$:

$$d(t) = ||\mathbf{x}(t) - \mathbf{w}(t)||$$

(ii) The update of the weights according to

$$\mathbf{w}(t+1) = \mathbf{w}(t) + \alpha(t)\Lambda(t)(\mathbf{x}(t) - \mathbf{w}(t)),$$

where $\alpha(t)$ is the *adaptation gain* and $\Lambda(t)$ is the *neighbourhood function* which depends on the winner neuron and the neuron under consideration.

From the above description it is clear that the original algorithm is demanding in terms of computing resources like calculation of the Euclidean distance, multiplication, weight storage, and adaptation functions. Various adaptations to the algorithm have been proposed to assist its hardware implementation.

[Thiran-94] A crucial issue in hardware implementations is the influence of the quantization of weights and inputs on the behaviour of the learning algorithm. This paper studies the quantization effects on a Kohonen network and demonstrates that its consequences can be greatly reduced by having a neighbourhood function that decreases with the distance between the winner neuron and its neighbours. Their experiments show that five bits can be sufficient to guarantee convergence to a solution close to the solution obtained without quantization [40].

[Rueping-94] The key features of their digital design are: (i) the use of the easily calculable Manhattan distance $\sum_i |\mathbf{x}_i - \mathbf{w}_i|$ in stead of the Euclidean distance (see also [25]), (ii) quantization of the adaptation gain to powers of two and a rectangular neighbourhood function, (iii) binary input values and quantization of the weights to 4 bits. In this way an implementation can be obtained that uses no multipliers, has a high degree of parallelism, and at the price of slightly bigger map size (10% to 15%) shows results comparable to the original algorithm [34].

[Vassilas-95] One of the demands of a NN implementation on systolic arrays is the effective use of the processor resources. In general, batch processing is an appropriate means to obtain better parallelisation. Kohonen's original algorithm, however, has both on-line winner selection and on-line weight update. Two possible variants are (i) batch winner selection and batch weight update (ii) batch winner selection and on-line weight update. In Vassilas' paper it is shown that the convergence properties of these variants are comparable with the original on-line algorithm and that the second variant is almost identical in performance to the original algorithm [41].

3.3: Recurrent networks

The class of recurrent networks exhibits complex dynamical behaviour and needs a reliable method for training and recall. Two such widely used paradigms for training recurrent networks are the Boltzmann machine and mean field theory (MFT) learning.

Boltzmann machine The Boltzmann machine is a stochastic learning rule which uses only locally available information and is for that reason well-suited for hardware implementation. The parallelism of a potential hardware implementation is, however, severely constrained by the required asynchronous update of neurons. Therefore, in a recent analog neurocomputer a synchronous version of the Boltzmann machine is used [33]. Another peculiarity of the Boltzmann machine is its use of simulated annealing by a gradual increase of the gain of the activation function. In Bellcore's implementation of a Boltzmann machine this annealing schedule is replaced by a gradual decrease of additive noise, which can be efficiently implemented in analog hardware [1].

Mean field theory This method is based on the idea that the simulated annealing process in the stochastic Boltzmann machine is too time-consuming and can be replaced by a deterministic mean field approximation. In an optical design [29] it is demonstrated that an MFT algorithm with synchronous updating of neurons leads to good results and is suitable for hardware implementation.

3.4: Other types of neural networks

RAM-based networks This is a special class of neural networks based on random access memories that are fit for hardware implementation. This network model can be easily implemented in standard available components, but has the disadvantage of a limited learning capacity. Therefore, various generalizations of the original model have been developed with extended capabilities, but also a more complex realization in hardware. A recent overview of RAM-based networks and related implementation aspects can be found in [3].

Cellular neural networks Cellular neural networks are of particular interest for VLSI implementation because of their sparse connectivity. Every unit of the network is a simple analog processor that interacts directly only with its neighbouring units within an often small range. Since the range of the network dynamics and the connection complexity are independent of the number of units, its implementation scales up well to bigger networks. An extensive overview by Chua and Roska of the cellular neural network paradigm can be found in [9].

4: Inaccuracy and robustness

A key issue in hardware implementations of all neural network models is the required precision of its parameters, since any hardware implementation is liable to imperfections such as limited numerical precision, component imprecision, noise, and stuck-at faults in weights and neurons. All of these inaccuracies have been subject of investigation and most neural network models show in fact a remarkable degree of robustness when these inaccuracies are incorporated during the training of the network [14] [27] [28].

4.1: Limited precision

Two different types of limited precision can be discerned. Firstly, the limitation by the representation of values by a small number of bits; this plays a major role in digital neural network implementation. Secondly, limited precision can be caused by component imprecision, for example non-linear responses of multipliers and variations between components. This problem is paramount in both electronic and optical implementations. A large range of theoretical and experimental studies has been performed to investigate and confine the effects of limited precision computation.

Limited numerical precision The accuracy that is needed for representing the weights of a neural network is area consuming and is incompatible with the system noise in analog implementations. Hence, the number of different weight values of the network should be as small as possible in order to obtain an efficient and accurate implementation. For different network architectures and learning algorithms the effect of such a limited weight precision has been investigated. The common tenor of these investigations is that below a certain level these limitations have a large influence on the behaviour of the network. For example, the accuracy needed in the standard backpropagation training algorithm in order not to deviate too much from the ideal learning trajectory is generally found to be 14 to 16 bits [19]. Note that, the accuracy needed in the forward pass lies around 8 bits [19] [30].

In order to reduce the chip area needed for weight storage and to overcome system noise, a further reduction of the number of allowed weight values is desirable. Hence, weight discretization algorithms based on the backpropagation learning rule have been designed for training multilayer networks with a very limited number of weight values ($2-4$ bits) [12] [38]. The rationale behind these weight discretization algorithms is to keep and update the weights with a high resolution off-line and use the discrete weights in the forward pass; these methods are therefore best-suited for off-chip training.

Component imprecision The state-of-the-art in analog (optical and electronic) hardware has progressed considerably over the last decade However, compared to digital technology it is not yet a mature discipline and the design of reliable and identical components gives rise to problems. In analog electronic implementations it is, for example, complex to efficiently construct a linear multiplier with a sufficient operating range, and simple non-linear multipliers are therefore often preferable or even inevitable. Examples of the use of non-linear multipliers can be found in both analog electronic [24] and in optical implementations [28]. It is also shown how the backpropagation learning rule can compensate for the non-linearity of multiplications by incorporating this non-linear multiplier in its derivation [24].

Another problem of analog hardware is the construction of an activation function that is close to the widely-used standard sigmoid. However, the incorporation of an accurate model of the hardware activation function in the training algorithm can compensate for this inaccuracy [24]. Additional difficulties arise in an analog optical implementation of the sigmoidal function based on intensity encoding, namely the limitation to non-negative values which means that the non-linearities are shifted into the non-negative domain and have a gain (steepness) that differs greatly from one [36]. While in analog electronic implementations one can easily deal with a non-standard gain by including a gain stage [18], this is impossible in an intensity-based optical implementation. An adapted backpropagation learning rule, based on a precise relation between the gain and the other initial parameters provides compensation for this non-standard gain [26] [39].

Another important aspect of analog implementations is the non-uniformity of the elements such as multipliers and activation functions. It has been exemplified that accurate modeling of these non-idealities or on-chip learning can compensate for these component-to-component variations [7] [14].

4.2: Robustness

Until a few years ago robustness of neural networks was mainly a folk theorem, but it has been investigated quite thoroughly these last years. In the above, several examples have already been given of the robustness in neural networks to inaccuracies. Here, the influence of faulty weights or neurons and noise will be discussed in some more detail.

Faulty weights and neurons The removal of the interconnection weights in a network and the occurrence of stuck-at faults in neurons are two types of faults that can serve as a test bed for the robustness of neural networks. The robustness of a backpropagation trained multilayer network to removing weights to/from the hidden layer and the influence of redundancy in the form of excess hidden neurons has been investigated [11]. While graceful degradation of the network performance under weight removal was observed, the addition of more hidden neurons did only deteriorate the results. Hence, it can be concluded that standard backpropagation training is not *inherently* fault-tolerant. An augmentation technique that tries to introduce linear dependencies in the already trained network by adding hidden neurons leads to better robustness.

The effect of "stuck-at-0" and "stuck-at-1" neurons on the solutions found in recurrent optimization networks is investigated in [31]. This type of network exhibits a high degree of fault-tolerance and an ability to find sub-optimal solutions to optimization problems in the presence of stuck-at faults in neurons.

It is widely believed that one should not verify the robustness of a neural network model *a posteriori*, but incorporate a robustness criterion in the training phase. This can be done by changing the objective function that has to be optimized during training or by injecting the expected faults during training. An illustration of this fact is an adaptation of the backpropagation learning rule that uses only a random subset of hidden neurons during each iteration. The resulting network is far more robust to the destruction of hidden neurons with only a small loss of accuracy in the noiseless case [22].

Noise and perturbation The surprising effects of the injection of random noise on the weights of a multilayer neural network when training by the backpropagation algorithm have been elaborately discussed by Murray and Edwards [27]. Both analytically and experimentally it is demonstrated that synaptic noise improves the network's fault tolerance to weight damage, generalization on unseen patterns, and the training trajectory. Similar results have been obtained when injecting additive noise into the weights of recurrent neural networks [21].

A theoretical study of the effect of perturbations of the parameters in a general class of feedback neural networks is studied in [42]. In this type of network it is important to know whether the stable states of the perturbed networks are close to the original stable states. It is shown that under reasonable conditions a linear relationship holds between the perturbations of the network parameters and the resulting error in the stable states.

5: Summary and conclusions

In this paper an overview has been given of a variety of methods that have been developed to facilitate the hardware implementation of neural network models. Each of the well-known neural network models brings along its specific problems for hardware implementation. While, for example, for the standard backpropagation algorithm the use of an accurate derivative of the activation function complicates the implementation, the realization in hardware of a Boltzmann machine is hindered by the sequential update of neurons. Most of the hardware-friendly algorithms that have been described here are geared towards the implementation of on-chip learning. The advantages of on-chip learning are manifold and include besides the gain in speed, an inherent compensation for component inaccuracies and the adaptation to new training patterns. However, most of the on-chip learning rules described in this paper have not been realized in hardware and their efficacy is difficult to judge. Some notable exceptions are Bellcore's implementation of a Boltzmann machine and the mean field theory algorithm [1], and Battiti's TOTEM-chip based on the reactive tabu search [4].

Attention has also been given to learning algorithms that are not suited for hardware implementation themselves, but the resulting network of which can be efficiently implemented. An important example of this class are the threshold networks, the training of which is often based on constructive methods that evolve the network's topology during training.

A key problem for all realizations for neural networks in hardware are the inaccuracy and imperfections of the hardware components. This ranges from quantization of the weight values and component-to-component variations to stuck-at faults of weights and neurons. Most of these aspects have been discussed and it has been exemplified that neural network models are remarkably robust to this limited precision when the inaccuracies are incorporated during the training of the network.

References

[1] J. Alspector, A. Jayakumar, and S. Luma. Experimental Evaluation of Learning in a Neural Microsystem. *Advances in Neural Information Processing Systems (NIPS92)*, vol. 5, pp. 871–878, Morgan Kaufmann, San Mateo, 1993.

[2] J. Alspector, R. Meir, B. Yuhas, and A. Jayakumar. A Parallel Gradient Descent Method for Learning in Analog VLSI Neural Networks. *Advances in Neural Information Processing Systems (NIPS92)*, vol. 5, pp. 836–844, Morgan Kaufmann, San Mateo CA, 1993.

[3] J. Austin. A Review of RAM Based Neural Networks. *Proceedings of the Fourth International Conference on Microelectronics for Neural Networks and Fuzzy Systems*, pp. 58–66, Turin, Italy, September 26–28, 1994, ISBN 0-8186-6710-9.

[4] R. Battiti and G. Tecchiolli. TOTEM: A Digital Processor for Neural Networks and Reactive Tabu Search. *Proceedings of the Fourth International Conference on Microelectronics for Neural Networks and Fuzzy Systems*, pp. 17–25, Turin, Italy, September 26–28, 1994, ISBN 0-8186-6710-9.

[5] R. Battiti and G. Tecchiolli. Training Neural Nets with the Reactive Tabu Search. *IEEE Transactions on Neural Networks*, vol. 6, no. 5, pp. 1185–1200, September 1995.

[6] R. D. Brandt and F. Lin. Supervised Learning in Neural Networks without Explicit Error Back-Propagation. *Proceedings of the Thirty-Second Allerton Conference on Communication, Control, and Computing*, pp. 294–303, Monticello, Illinois, September 28–30, 1994.

[7] H. C. Card and C. R. Schneider. Analog CMOS Neural Circuits - In Situ Learning. *International Journal of Neural Systems*, vol. 3, no. 2, pp. 103–124, 1992

[8] G. Cauwenberghs. A Fast Stochastic Error-Descent Algorithm for Supervised Learning and Optimization. *Advances in Neural Information Processing Systems (NIPS92)*, vol. 5, Morgan Kaufman, San Mateo CA, 1993.

[9] L. O. Chua and T. Roska. The CNN Paradigm. *IEEE Transactions on Circuits and Systems-I: Fundamental Theory and Applications*, volume 40, no. 3, pp. 147–156, March 1993.

[10] E. Corwin, A. Logar, and W. Oldham. An Iterative Method for Training Multilayer Networks with Threshold Functions. *IEEE Transactions on Neural Networks*, vol. 5, no. 3, pp. 507–508, May 1994.

[11] M. D. Emmerson and R. I. Damper. Determining and Improving the Fault Tolerance of Multilayer Perceptrons in a Pattern-Recognition Application. *IEEE Transactions on Neural Networks*, vol. 4, no. 5, pp. 788–793, September 1993.

[12] E. Fiesler, A. Choudry, and H. J. Caulfield. A Universal Weight Discretization Method for Multi-Layer Neural Networks. *IEEE Transactions on Systems, Man, and Cybernetics (IEEE-SMC)*, (Accepted for publication). See also: E. Fiesler, A. Choudry, and H. J. Caulfield. A Weight Discretization paradigm for Optical Neural Networks. *Proceedings of the International Congress on Optical Science and Engineering*, vol. SPIE 1281, pp. 164–173, SPIE, Bellingham, Washington, 1990. ISBN: 0-8194-0328-8.

[13] B. Flower and M. Jabri. Summed Weight Neuron Perturbation: An $O(N)$ Improvement over Weight Perturbation. *Advances in Neural Information Processing Systems (NIPS92)*, vol. 5, 212–219, Morgan Kaufmann, San Mateo CA, 1993.

[14] R. C. Frye, E. A. Rietman, and C. C. Wong. Back-Propagation Learning and Nonidealities in Analog Neural Network Hardware. *IEEE Transactions on Neural Networks*, vol. 2, no. 1, pp. 110–117, January 1991.

[15] G. M. Georgiou and C. Koutsougeras. Complex Domain Backpropagation. *IEEE Transactions on Circuits and Systems 2: Analog and Digital Signal Processing*, vol. 39, no. 5, pp. 330–334, May 1992.

[16] A. Hamilton, A. F. Murray, D. J. Baxter, S. Churcher, H. M. Reekie, and L. Tarassenko. Integrated Pulse Stream Neural Networks: Results, Issues, and Pointers. *IEEE Transactions on Neural Networks*, vol. 3, no. 3, pp. 385-392, May 1992.

[17] J. Hertz, A. Krogh, B. Lautrup, T. Lehmann. Non-Linear Back-Propagation: Doing Back-Propagation without Derivatives of the Activation Function. Preprint available from Neuroprose , March 1994, file://archive.cis.ohio-state.edu/pub/neuroprose

[18] P. W. Hollis and J. J. Paulos. A Neural Network Learning Algorithm Tailored for VLSI Implementation. *IEEE Transactions on Neural Networks*, vol. 5, no. 5, pp. 784–791, September 1994.

[19] 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.

[20] M. Jabri and B. Flower. Weight Perturbation: An Optimal Architecture and Learning Technique for Analog VLSI Feedforward and Recurrent Multilayer Networks. *IEEE Transactions on Neural Networks*, vol. 3, no. 1, pp. 154–157, January 1992.

[21] K. Jim, C. L. Giles, and B. G. Horne. Synaptic Noise in Dynamically-Driven Recurrent Neural Networks: Convergence and Generalization. Technical report UMIACS-TR-94-89 / CS-TR-3322, Institute for Advanced Computer Studies, University of Maryland, College Park, MD 20742, May 1994

[22] P. Kerlirzin and P. Réfrégier. Theoretical Investigation of the Robustness of Multilayer Perceptrons: Analysis of the Linear Case and Extension to Nonlinear Networks. *IEEE Transactions on Neural Networks*, vol. 6, no. 3, pp. 560–571, May 1995.

[23] J. H. Kim and S.-K. Park. The Geometrical Learning of Binary Neural Networks. *IEEE Transactions on Neural Networks*, vol.6, no. 1, pp. 237–247, January 1995.

[24] J. Lont and W. Guggenbühl. Analog CMOS Implementation of a Multilayer Perceptron with Nonlinear Synapses. *IEEE Transactions on Neural Networks*, vol. 3, no. 3, pp. 385-392, May 1992.

[25] D. Macq, M. Verleysen, P. Jespers, and J.-D. Legat. Analog Implementation of a Kohonen Map with On-Chip Learning. *IEEE Transactions on Neural Networks*, vol. 4, no. 3, pp. 456–461, May 1993.

[26] P. Moerland, E. Fiesler, and I. Saxena. The Effects of Optical Thresholding in Backpropagation Neural Networks. *Proceedings of the International Conference on Artificial Neural Networks (ICANN95)*, Paris, France, October 11–13, 1995.

[27] A. F. Murray and P. J. Edwards. Enhanced MLP Performance and Fault Tolerance Resulting from Synaptic Weight Noise During Training. *IEEE Transactions on Neural Networks*, vol. 5, no. 5, pp. 792–802, September 1994.

[28] L. Neiberg and D. Casasent. High-Capacity Neural Networks on Nonideal Hardware. *Applied Optics*, vol. 33, no. 32, pp. 7665–7675, 1994.

[29] C. Peterson, S. Redfield, J. D. Keeler, and E. Hartman. An Optoelectronic Architecture for Multilayer Learning in a Single Photorefractive Crystal. *Neural Computation*, vol. 2, pp. 25–34, 1990.

[30] S. W. Piché. The Selection of Weight Accuracies for Madalines. *IEEE Transactions on Neural Networks*, vol. 6, no. 2, p. 432–445, March 1995.

[31] P. W. Protzel, D. L. Palumbo, and M. K. Arras. Performance and Fault-Tolerance of Neural Networks for Optimization. *IEEE Transactions on Neural Networks*, vol. 4, no. 4, pp. 600–614, July 1993.

[32] D. Psaltis and Y. Qiao. Adaptive Multilayer Optical Networks, In *Progress in Optics*, editor: E. Wolf, vol. 31, chapter 4, pp. 227–261, 1993, Elsevier Science Publishers, Amsterdam, The Netherlands, ISBN 0–444–89836–0.

[33] H. Pujol, O. Klein, E. Belhaire, and P. Garda. RA: An Analog Neurocomputer for the Synchronous Boltzmann Machine. *Proceedings of the Fourth International Conference on Microelectronics for Neural Networks and Fuzzy Systems*, pp. 449–455, Turin, Italy, September 26–28, 1994, ISBN 0-8186-6710-9.

[34] S. Rueping, K. Goser, and U. Rueckert. A Chip for Selforganizing Feature Maps. *Proceedings of the Fourth International Conference on Microelectronics for Neural Networks and Fuzzy Systems*, pp. 26–33, Turin, Italy, September 26–28, 1994, ISBN 0-8186-6710-9.

[35] D. Rumelhart, G. Hinton, and R. Williams. Learning Internal Representations by Error Propagation. *Parallel Distributed Processing: Explorations in the Microstructure of Cognition*, vol. 1: Foundations, pp. 318–362, MIT Press, Cambridge, Massachusetts, 1986. ISBN: 0–262–18210–7.

[36] I. Saxena and E. Fiesler. Adaptive Optical Neural Multilayer Network with Optical Thresholding. *Optical Engineering* (ISSN 0091–3286), special on optics in Switzerland (P. Rastogi, editor), vol. 34 , no. 8, pp. 2435–2440.

[37] F. J. Śmieja. Neural Network Constructive Algorithms: Trading Generalization for Learning Efficiency? *Circuits, Systems, and Signal Processing*, vol. 12, no. 2, pp. 331–374, 1993.

[38] C. Z. Tang and H. K. Kwan. Multilayer Feedforward Neural Networks with Single Power-of-Two Weights. *IEEE Transactions on Signal Processing*, vol. 41, no. 8, pp. 2724–2727, August 1993.

[39] G. Thimm, P. Moerland, and E. Fiesler. The Interchangeability of Learning Rate and Gain in Backpropagation Neural Networks, accepted for publication in vol. 8 of *Neural Computation*. See also: P. Moerland, G. Thimm, and E. Fiesler. Results on the Steepness in Backpropagation Neural Networks. *Proceedings of the '94 SIPAR–Workshop on Parallel and Distributed Computing*, ed. M. Aguilar, pages 91–94 , Institute of Informatics, University of Fribourg, Fribourg, Switzerland, October 1994.

[40] P. Thiran, V. Peiris, P. Heim, and B. Hochet. Quantization Effects in Digitally Behaving Circuit Implementations of Kohonen Networks. *IEEE Transactions on Neural Networks*, vol. 5, no. 3, pp. 450–458, May 1994.

[41] N. Vassilas, P. Thiran, and P. Ienne. How to Modify Kohonen's Self-Organizing Feature Maps for An Efficient Digital Parallel Implementation. *Proceeding of the International Conference on Artificial Neural Networks*, Cambridge, June 26–28, 1995.

[42] K. Wang and A. N. Michel. Robustness and Perturbation Analysis of a Class of Artificial Neural Networks. *Neural Networks*, vol. 7, no. 2, pp. 251–259, 1994.

[43] B. Widrow and M. A. Lehr. 30 Years of Adaptive Neural Networks: Perceptron, Madaline, and Backpropagation. *Proceedings of the IEEE*, vol. 78, no. 9, 1415–1442, September 1990.