Weight Perturbation: An Optimal Architecture and Learning Technique for Analog VLSI Feedforward and Recurrent Multi-Layer Networks

Marwan Jabri & Barry Flower

Systems Engineering and Design Automation Laboratory

School of Electrical Engineering

University of Sydney

Abstract

Previous work on analog VLSI implementation of multi-layer perceptrons with on-chip learning has mainly targeted the implementation of algorithms like back-propagation. Although back-propagation is efficient, its implementation in analog VLSI requires excessive computational hardware. In this paper we show that using gradient descent with direct approximation of the gradient instead of back-propagation is cheapest for parallel analog implementations. We also show that this technique (we call "weight perturbation") is suitable for multi-layer recurrent networks as well. A discrete level analog implementation showing the training of an XOR network as an example is also presented.

I. Introduction

Many researchers have recently proposed architectures for VLSI implementation of multi-layer perceptron. Most of the reported work has addressed digital implementation [2]. Furman and associates [1] have reported an analog implementation of back-propagation. In both digital and analog reported work, back-propagation was selected because of its efficiency and popularity. For analog, the implementation of on-chip learning algorithms requires bi-directional synapses and the generation of the derivative of neuron transfer functions with respect to their input. As the area and power of a synapse has an important effect on the total chip area and the resulting size of network that can be implemented, its minimisation is important. Furthermore, in analog implementation, the generation of the derivative is a rather of a difficult task. These complications are even more amplified when recurrent networks are being implemented as the number of synapses may grow as the square of the number of neurons.

Recently, the Madaline Rule III (MR III) was suggested as an alternative to back-propagation for analog implementation [4]. This rule can be considered implementing gradient evaluation using "node perturbation" according to:

$$\Delta w_{ij} = -\eta \frac{\Delta E}{\Delta net_i} x_j \tag{1}$$

where $net_i = \sum_j w_{ij} x_j$ and $x_j = f(net_j)$ with f being the non-linear squashing function.

Therefore, in addition to the actual hardware needed for the normal operation of the network, the implementation of the MR III learning rule for an N neuron network in analog VLSI requires:

- An addressing module and wires routed to select and deliver the perturbation to each neuron.
- either one or N multiplication hardware to compute the term $\frac{\Delta E}{\Delta net_i} x_j$ in addition to the multiplication by the learning rate. If one multiplier is used then additional multiplexing hardware is required.
- An addressing module and wires routed to select and read the x_j terms.

Note that if greater training flexibility is required in the sense of off-chip access to the gradient values, then the states of the neurons (x_j) would need to be made available off-chip as well, which will require a multiplexing scheme and N chip pads.

An alternative approach to node perturbation is "weight perturbation" where the gradient is approximated to a finite difference, and we show in this paper that gradient evaluation using "weight perturbation" is a cheaper solution, hardware and complexity wise, and can be equally used to train recurrent networks.

II. Gradient Evaluation using Weight Perturbation

The gradient with respect to the weight can simply be evaluated by the approximation

$$\frac{\partial E}{\partial w_{ij}} = \frac{\Delta E}{\Delta_{pert} w_{ij}} + O(\Delta_{pert} w_{ij}) = \frac{E(w_{ij} + pert_{ij}) - E(w_{ij})}{pert_{ij}} + O(\Delta_{pert} w_{ij}) \qquad (2)$$

if the perturbation $\Delta_p ert w_{ij}$ is small enough. This is the forward difference method and the weight update rule becomes:

$$\Delta w_{ij} = \frac{E(w_{ij} + pert_{ij}) - E(w_{ij})}{pert_{ij}} \qquad (3)$$

where $E()$ is the total mean square error produced at the output of the network for a given pair of input and training patterns and a given value of the weights.

The order of the error of the finite difference approximation can be improved by using the central difference method, so that;

$$\frac{\partial E}{\partial w_{ij}} = \frac{\Delta E}{\Delta_{pert} w_{ij}} + O(\Delta_{pert} w_{ij}^2) = \frac{E(w_{ij} + \frac{pert_{ij}}{2}) - E(w_{ij} - \frac{pert_{ij}}{2})}{pert_{ij}} + O(\Delta_{pert} w_{ij}^2) \quad (4)$$

if the perturbation $\Delta_p ert w_{ij}$ is again small enough, and the weight update rule becomes:

$$\Delta w_{ij} = \frac{E(w_{ij} + \frac{pert_{ij}}{2}) - E(w_{ij} - \frac{pert_{ij}}{2})}{pert_{ij}} \tag{5}$$

however, the number of forward relaxations of the network required is of the order N^3 rather than N^2 for the forward difference method. Thus either method can be selected on the basis of a speed/accuracy trade-off.

Note, that as η and $pert_{ij}$ are both constants, the analog implementation version can simply be written as:

$$\Delta w_{ij} = G(pert_{ij}) \Delta E(w_{ij}, pert_{ij}) \tag{6}$$

with

$$G(pert_{ij}) = -\frac{\eta}{pert_{ij}}$$

and

$$\Delta E(w_{ij}, pert_{ij}) = E(w_{ij} + pert_{ij}) - E(w_{ij})$$

The weight update hardware involves the evaluation of the error with perturbed and unperturbed weight and then the multiplication by a constant.

This technique is ideal for analog VLSI implementation for the following reasons:

1. As the gradient $\frac{\delta E}{\delta w_{ij}}$ is approximated to $\frac{E_{pert} - E}{\Delta_{pert} w_{ij}}$ (where $\Delta_{pert} w_{ij}$ is the perturbation applied at weight w_{ij}), no back-propagation pass is needed and only forward path is required. This means, in terms of analog VLSI implementations, no bidirectional circuits and hardware for the back-propagation are needed. The hardware used for the operation of the network is used for the training. Only single simple circuits to implement the weight update are required. This simplifies considerably the implementation.

2. Compared to node perturbation our technique does not require the two neuron addressing modules, routing and extra multiplication listed above.

Our technique does not require any overheads in routing and addressing connections to every neuron to deliver the perturbations as the same wires used to access the weights are used to deliver weight perturbations. Furthermore, node perturbation requires extra routing to access the output state of each neuron and extra multiplication hardware is needed for the $\frac{\Delta E}{\Delta net_i} x_j$ terms which is not the case with weight perturbation. Finally, with weight perturbation, the approximated gradient values can be made available if needed at a rather low cost [1]

III. Simulations

The "weight perturbation" technique was used on two test cases: XOR (feedforward and recurrent) and Intra-Cardiac Electro-Grams (ICEG). The learning procedure is implemented as shown in Figure 1.

A. XOR

i. Feedforward XOR

The XOR network used has 1 hidden layer with two hidden units, two input units acting as pins and one output unit. The training was done in the on-line mode. The network parameters are shown in Table 1. The total mean squared error is shown in Figure 2 for both training with back-propagation and weight perturbation.

[1]If the mean square error is required off-chip then only one single extra pad is required. Otherwise, if approximated gradient values are to be calculated off-chip, then no extra chip area or pads are required as the output of the network would be accessible anyway.

```
for each pattern $p$ {
     E = ForwardPass()
     ClearDeltaWeights()
     for each weight $w_{ij}$ do {
          Epert = ApplyPerturbate($w_{ij}$)
          DeltaError = Epert - E
          DeltaW[i][j] = - $\eta$ * DeltaError/Perturbation
          RemovePerturbation($w_{ij}$)
     }
}
```

Figure 1: Weight perturbation algorithm in its simplest form. This procedure can be used either for on-line or batch training.

Table 1: Configuration parameters for training XOR

Parameter	Value
Perturbation strength	0.00001
Learning rate	0.3
Convergence criteria	0.001
Initial weight range	0.3
Sensitivity criteria	0.3

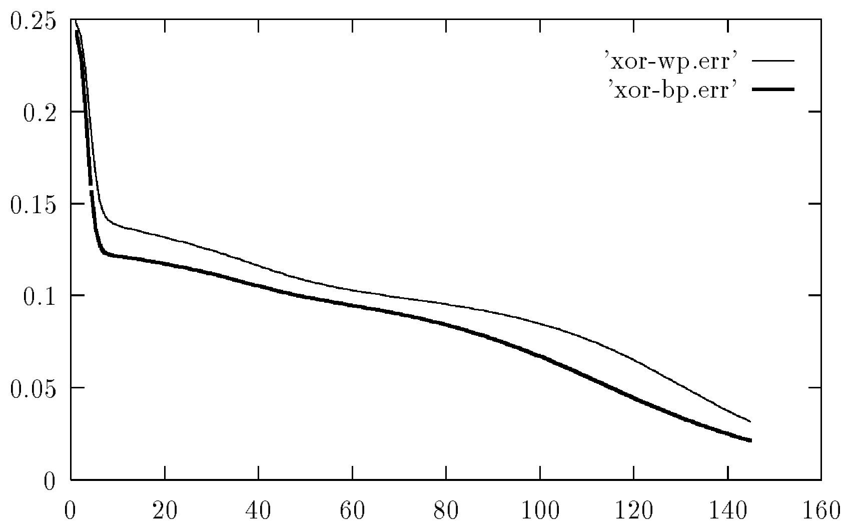

Figure 2: Mean square error for XOR training using back-propagation (xor-bp.err) and weight perturbation (xor-wp.err).

As Figure 4 shows and as one may expect, the overall shape of the error as function of training iteration are very similar. All 4 XOR patterns are training in 145 iterations with both techniques (the final average mean square error are not however equal). A study of the weight produced by both techniques show that they are extremely similar (the differences were not visible from weight density plots).

ii. Recurrent XOR

A multi-layer recurrent was trained using weight perturbation. The architecture of the network is shown in Figure 3 and the training parameters are shown in Table 2.

The same architecture was trained using Recurrent Back-Propagation based on the algorithm of Pineda [3]. The training error curves are shown in Figure 4. Although the two training techniques started from identical initial conditions, the convergence speed was different and the final weight solution was different for both techniques. This

Table 2: Parameters of the XOR recurrent network.

Parameter	RBP	RWP
Perturbation strength	NA	0.001
Neuron relaxation constant	0.01	0.01
Weight relaxation constant	0.1	NA
Network stability constant	0.000001	0.000001
Learning rate	0.3	0.3
Convergence criteria	0.1	0.1
Initial weight range	0.7	0.7
Sensitivity criteria	0.3	0.3

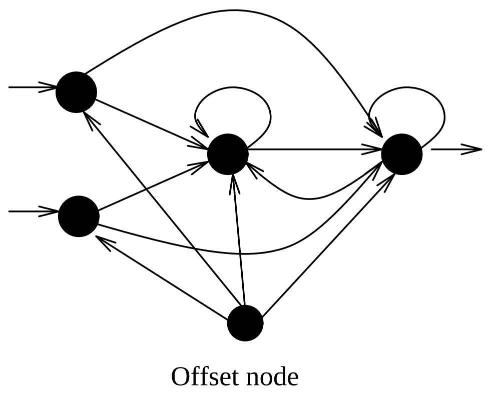

Figure 3: Architecture of the XOR recurrent network.

may be attributed to different learning steps (perturbation strength) in the case of the weight perturbation technique.

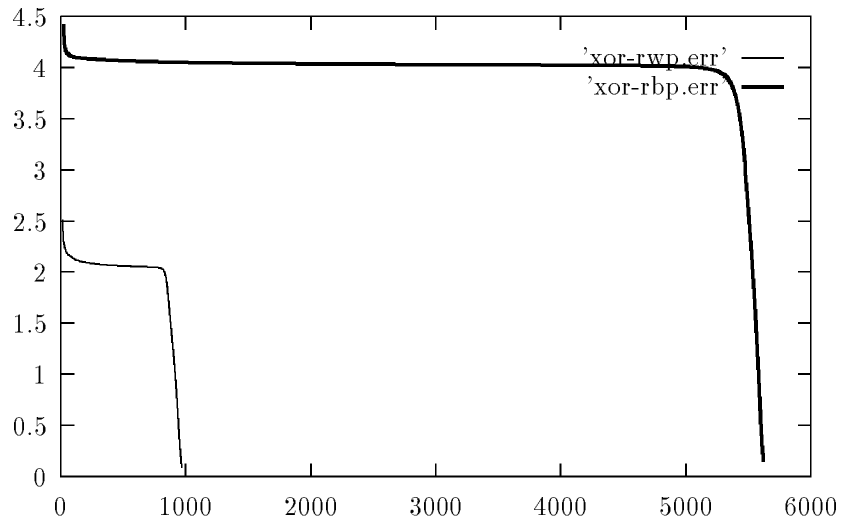

Figure 4: Mean square error for the weight perturbation and recurrent back-propagation training.

B. ICEG Classification

Another of our tests is on the training of a three layer perceptron to classify Intra-Cardiac Electro-Grams (ICEG). The size of the training set is 120 patterns, and the network has 21 input units, 10 hidden units and 5 output units. Figure 5 shows the mean square error for weight perturbation and back-propagation training.

Following the training we have tested the trained networks on a set of 2600 patterns. The training with back-propagation and weight perturbation has led to an identical performance 91% of correct classification.

IV. Implementation

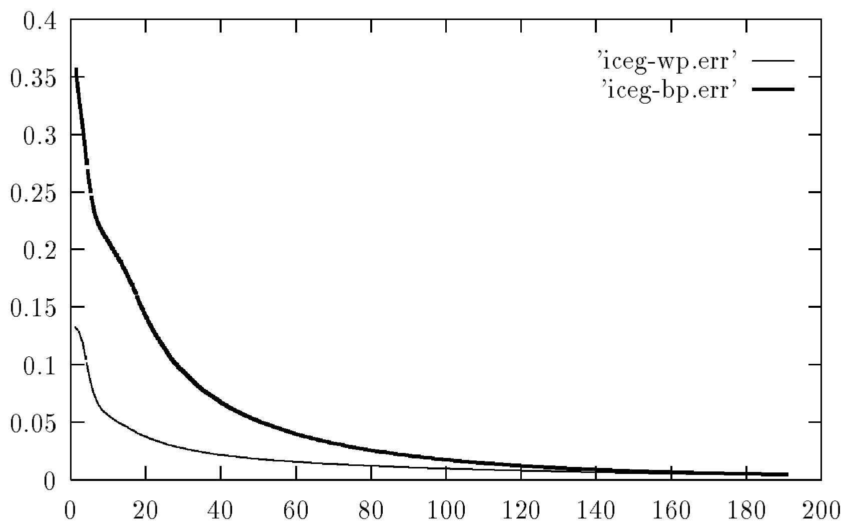

Figure 5: Mean square error for Intra-Cardiac Electro-Gram training using back-propagation (iceg-bp.err) and weight perturbation (iceg-wp.err).

To show the feasibility of learning with analog implementation of weight perturbation, we have constructed a discrete component implementation of an XOR network. Figure 7 shows a block diagram of the network used and Figure 6 shows a picture of the hardware implementation (synapse and neuron boards). In addition a PC was provided as a controller to orchestrate the presentation of training vectors and weight updates. The weights were stored as a voltage on capacitors which are periodically updated.

The voltage range for a signal is $\pm 10.0V$ and the weight values also have a range of $\pm 10.0V$. This means that a mean square error of $10.0V^2$ indicates that the network output signal and the training signal vary by 32%.

Figure 8 shows a training session of the xor network reaching a mean square error of $8.0V^2$ using the weight perturbation algorithm. The noise apparent is due to A/D sampling errors and noise on the network due to weight and training vector refresh. We note that convergence occurs inspite of this noise level, demonstrating the robustness

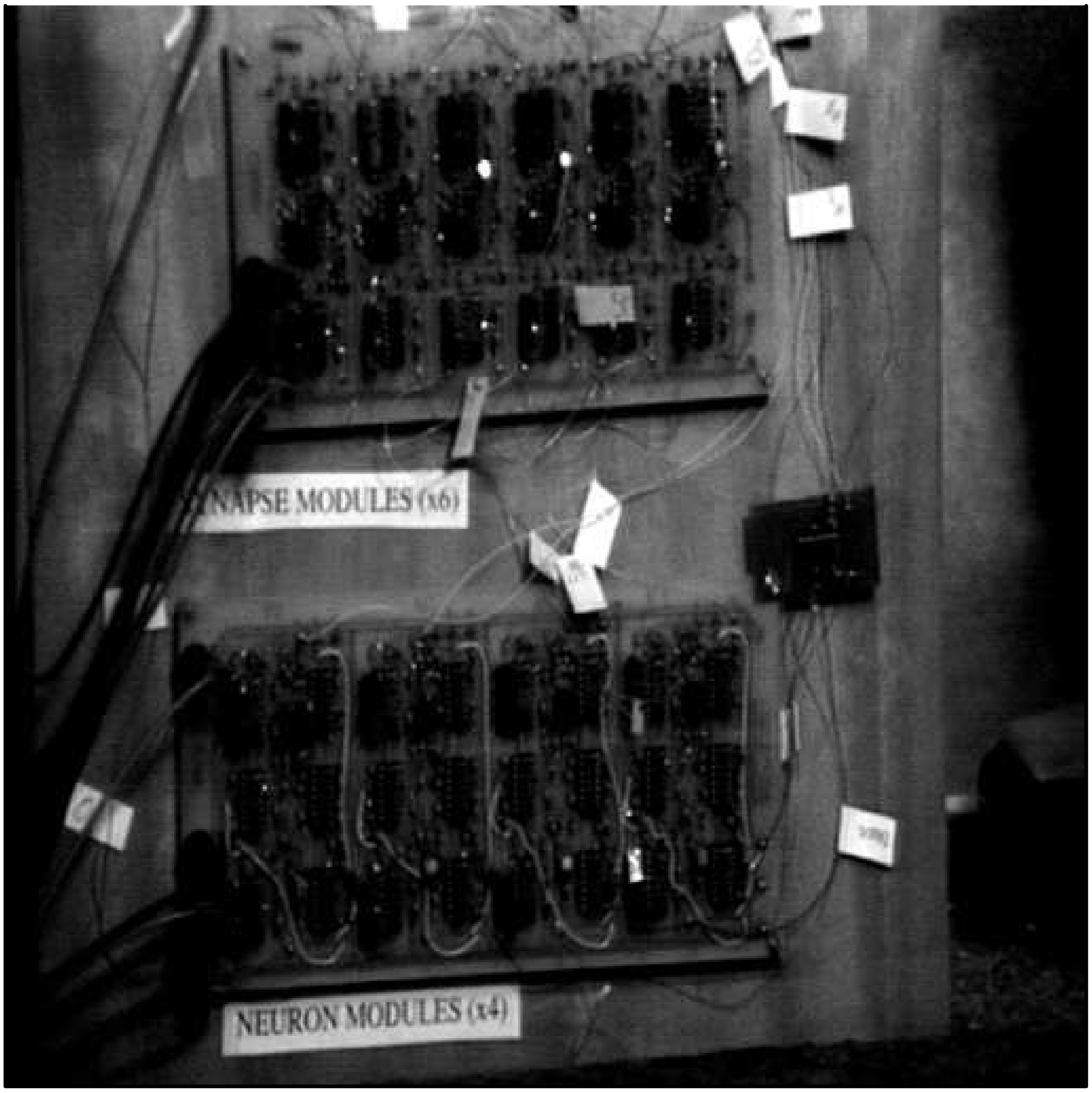

Figure 6: Picture of the Xor Hardware (synapse and neuron boards).

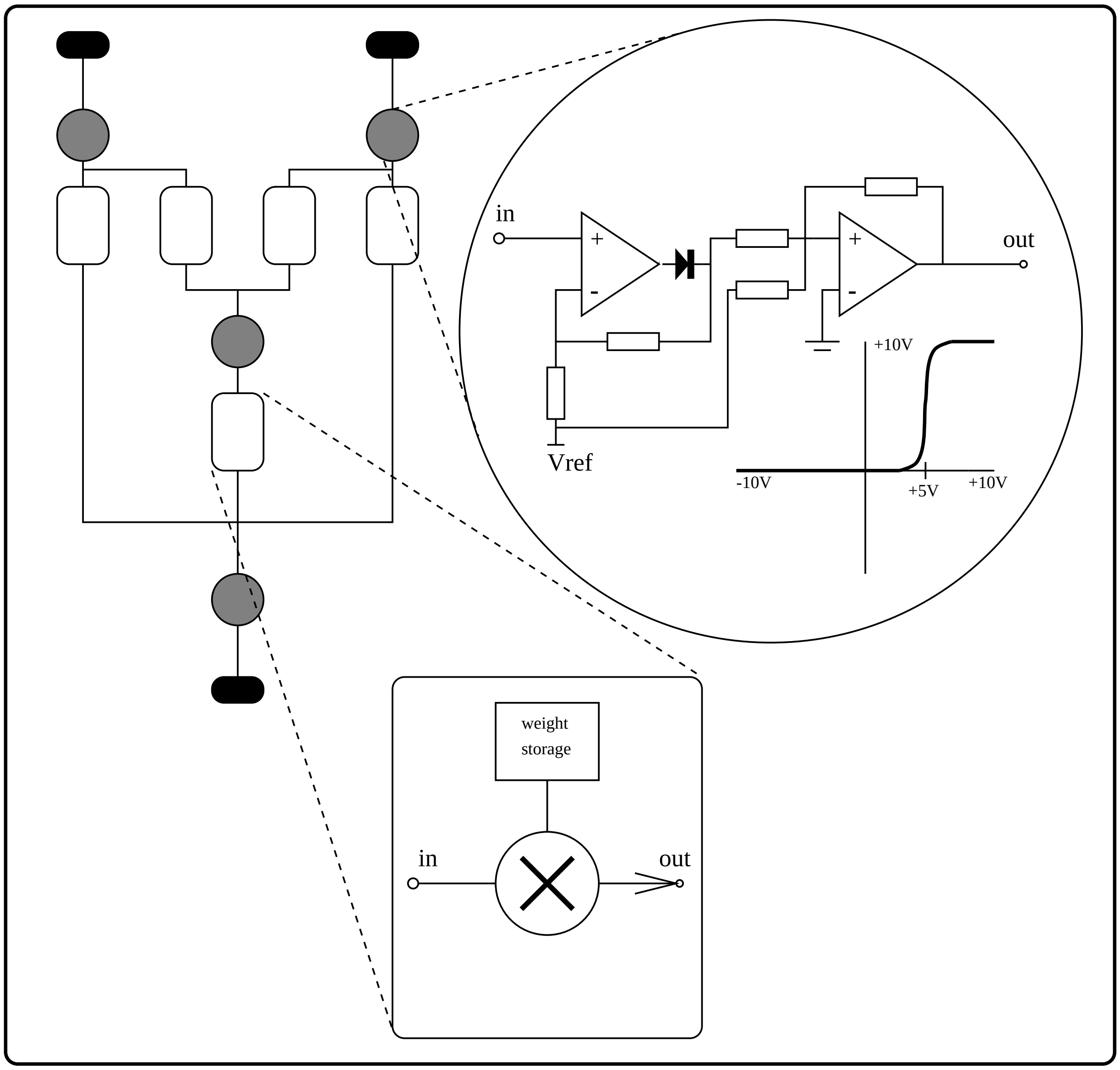

Figure 7: Xor Hardware Implementation Circuit Block Diagram

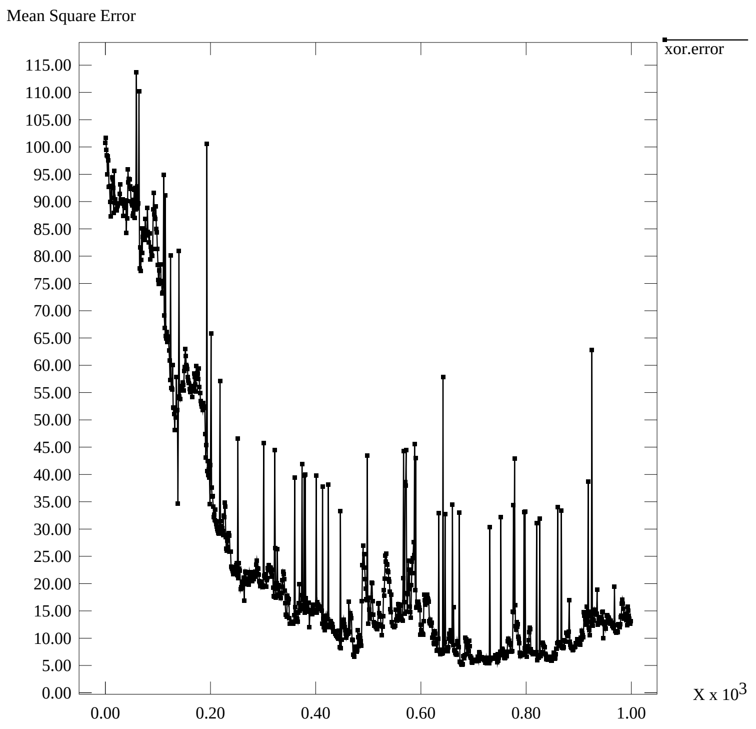

Figure 8: Mean square error of training hardware XOR network

of the weight perturbation optimising technique to noise.

V. Conclusions

In this paper we show that weight perturbation is a very cheap and flexible learning technique for analog implementations of neural networks. We also show that it is more flexible than back-propagation and node perturbation (MR III). We demonstrate using simulations that weight perturbation produces the same performance as back-propagation and recurrent back-propagation. A discrete analog implementation was used to demonstrate the feasibility of multi-layer feedforward training using weight perturbation. The same technique can be used to train simple recurrent networks (like Elman networks) and continuously running recurrent networks for temporal sequences recognition (like Williams and Zipser networks). For all these networks it is easy to see that as far as training is concerned, the hardware implementation using a weight perturbation architecture is very similar to that required for the normal operation of the networks.

VI. Acknowledgments

This research is supported by the Australian Research Council and a Sydney University Special Project grant.

References

[1] B. Furman, J. White, and A Abidi. CMOS analog implementation of back propagation algorithm. In *Abstracts of the First Annual INNS Meeting*, page 381, Boston, USA, 1988.

[2] J.N. Hwang and S.Y. Kung. Parallel Algorithms/Architectures for Neural Networks. *Journal of VLSI Signal Processing*, 221–251, 1989.

[3] Fernando J. Pineda. Recurrent Backpropagation and the Dynamical Approach to Adaptive Neural Computation. *Neural Computation*, 1(2):161–172, 1989.

[4] Bernard Widrow and Michael A. Lehr. 30 years of adaptive neural networks: perceptron, madaline, and backpropagation. *Proceedings of the IEEE*, 78(9):1415–1442, 1990.

List of Figures

List of Tables