Practical Performance and Credit Assignment Efficiency of Analog Multi-layer Perceptron Perturbation Based Training Algorithms

Marwan A. Jabri

Systems Engineering and Design Automation Laboratory
Sydney University Electrical Engineering
NSW 2006 Australia
marwan@sedal.su.oz.au

SEDAL Technical Report 1-7-94

Abstract

Many algorithms have been recently reported for the training of analog multi-layer perceptron. Most of these algorithms were evaluated either from a computational or simulation view point. This paper applies several of these algorithms to the training of an analog multi-layer perceptron chip. The advantages and shortcomings of these algorithms in terms of training and generalisation performance and their capabilities in a limited precision environment are discussed. Extensive experiments demonstrate that a trade-off exists between the parallelisation of perturbations and the efficiency of credit assignment. Two semi-parallelisation heuristics are presented and are shown to provide advantages in terms of efficient exploration of the solution space and fewer credit assignment confusions.

1 INTRODUCTION

Analog microelectronic implementations of the multi-layer perceptron (MLP) offer a number of attractive aspects: they provide an inherent parallelism since computational elements can be highly compact; in many real-world applications they do not require expensive and bulky analog to digital converters; when implemented using sub-threshold design techniques, the resulting system can operate at very low power; analog implementations can also provide higher operating speeds than digital implementations within their precision and noise margins.

The attractive aspects of analog computation come with a list of “obstacles”. Resolution is limited in practice to less than 8 bits which can degrade the performance of learning algorithms. Noise immu-

nity is a serious design constraint. Automated analysis and simulation tools of analog circuits are limited when compared to digital simulation and timing verification tools. Finally, memory is not accurate and bulky to implement in analog.

One of the main hurdles is the efficient training of large networks. Over the last few years several analog neural network training algorithms have been reported. This paper reviews several of these algorithms and presents experimental results on their training and generalisation performance when they are operated on an analog sub-threshold VLSI MLP.

The results of the experiments demonstrate that a trade-off exists between the parallelisation of perturbations and the efficiency of credit assignment. Two semi-parallel perturbation algorithms are presented and are demonstrated to provide advantages in terms of efficient exploration of the solution space and less credit assignment confusions.

In Section 2, we discuss the issues of neural network training in an analog framework. Section 3 reviews the training algorithms and strategies used in the experiments. In Section 4, we discuss credit assignment efficiency of parallel perturbation based algorithms. Semi-parallel perturbation heuristics are described in Section 5. Section 6 describes the experiments and in Section 7 we discuss the results and their implications.

2 LEARNING IN AN ANALOG FRAMEWORK

An analog multi-layer perceptron (AMLP) can be trained in three ways:

1. In-loop: The chip response is used in the weights optimisation process.
2. On-chip: Training is performed on chip.
3. Off-chip: Training is performed completely off the chip and the weights are downloaded to the chip (and truncated if required).

The process of in-loop training of AMLP chips is commonly used either as a direct chip training method or as supplementary training following simulations on a workstation. The chip response includes its output as well as the activity of any hidden neurons, which may or may not be required by the training algorithm[1]. The training often requires the computation (usually off-chip) of either a Pat-

tern Mean Squared Error (PMSE) or the error over all the patterns which we call here the Total Mean Squared Error (TMSE).

Although on-chip learning is a desired objective in AMLP training, there are a number of problems to be overcome. There have been several reports (see [8] for example) indicating that techniques such as backpropagation require over 12 bits of resolution for the weights and states for successful training. Another drawback in the implementation of backpropagation on a chip is the complexity of the circuitry and the additional interconnections required for the backward propagation of the errors.

The difficulties in the implementation of backpropagation and its precision requirements have led to the investigation of algorithms that are simpler to implement and are less sensitive to precision. The algorithms on which most interest has focused so far have been based on the approximation of the error gradient by a finite difference. We will describe a collection of such finite difference based techniques in Section 3.

3 LEARNING ALGORITHMS AND STRATEGIES

We make the distinction in this paper between learning algorithms and strategies. A learning algorithm refers to the weight update rule, while a learning strategy refers to the way the weight updates are applied to the weights (e.g. on-line, batch,...). In some cases, the algorithm and strategy are inherently linked.

3.1 NOTATION

The notation used in the algorithm description is as follows. The matrix $\boldsymbol{w}$ represents the network weights and its element w_{ij} represents the weight from neuron i to neuron j. The matrix $\Delta\boldsymbol{w}$ represents the update value of the weights. Unless indicated otherwise, the scalar $\varepsilon(\boldsymbol{w})$ represents the network error TMSE or PMSE. The matrix $\boldsymbol{p}^{(n)}$ represents the perturbation to be applied to the weights at iteration n of the learning process, when fully parallel perturbations are applied. In the case of semi-parallel perturbations a subset of the weights are perturbed simultaneously, and the perturbation matrix is then denoted $\boldsymbol{p}_{+}^{(n)}$ and the weight update matrix is denoted $\Delta\boldsymbol{w}_{+}$. When a single weight

1. It is assumed in this paper that the weights of an MLP are stored on the host computer and down-loaded to the chip.

w_{ij} is being perturbed at any time, only the element $p_{ij}^{(n)}$ of the perturbation matrix $\boldsymbol{p}^{(n)}$ is non zero. Finally, the learning rate is represented by η.

3.2 WEIGHT UPDATE STRATEGIES

We will distinguish six different learning strategies:

1. normal on-line weight update (NONLINE): the weight updates are computed for each weight and for a given training pattern and are then applied. (see Figure 1).

FIGURE 1. Normal On-line update strategy (NONLINE)

```
for all patterns {
   feedforward();
   compute updates;
   apply updates;
}
```

2. normal batch mode weight update(NBATCH): the weight updates are computed for each weight and accumulated over all the training patterns and are then applied. (see Figure 2).

FIGURE 2. Normal Batch update strategy (NBATCH)

```
for all patterns {
   feed forward;
   compute updates;
}
apply updates;
```

3. Single pattern conditional immediate update (CIONLINE): Assuming that a weight update is available a weight is updated if a condition is met (e.g. error decreasing). If the condition evaluation involves error computation, then only the PMSE is evaluated. (see Figure 3).

FIGURE 3. Single pattern conditional immediate update (IONLINE).

```
for all pattern {
   feed forward;
   measure PMSE;
   perturbe weight(s);
   feed forward;
   measure new PMSE;
   If (condition hold)
       keep perturbations;
}
```

4. Batch pattern conditional immediate update (CIBATCH): Similar to CIONLINE above with the exception that any error computation is done over all the patterns in the training set (that is the TMSE is computed). (see Figure 4).

FIGURE 4. Batch pattern conditional immediate update (CIBATCH).

```
measure TMSE;
forall weights {
   perturbe weight(s);
   measure new TMSE;
   If (condition hold)
       keep perturbations;
}
```

5. Perturbation on-line weight update (PONLINE): the pattern mean squared error (PMSE) is measured; the weight(s) (depending on the perturbation parallelisation scale) are perturbed and the new PMSE is measured; the weights updates are computed and applied; This is repeated to the next pattern and so on. (see Figure 5).

FIGURE 5. Perturbation on-line update (PONLINE).

```
forall patterns {
   measure PMSE;
   perturbe weight(s);
   measure new PMSE;
   compute updates;
   remove perturbations;
   apply updates;
}
```

6. Perturbation batch (PBATCH): The total mean squared error (TMSE, over all patterns) is measured. The weights (depending on the parallelisation scale) are perturbed and the new TMSE is measured. The weight updates are computed and applied. (see Figure 6).

FIGURE 6. Perturbation batch update (PBATCH).

```
feed forward;
measure TMSE;
forall weight(s) {
   perturbe weight(s);
   measure new TMSE;
   compute updates;
   remove perturbations;
}
apply updates;
```

Several of the weight update strategies described above can be applicable to a learning algorithm. The strategies PONLINE and PBATCH are targetted to sequencial, semi-parallel and parallel weigt perturbation. Although update strategies may seem trivial, in an in-loop or on-chip learning context, they have far reaching implications on the design and implementation aspects of an AMLP. Further, batch strategies are simply impractical and do not scale well.

3.3 LEARNING ALGORITHMS

In this section, learning algorithms that have been reported in the literature and used in the experiments are described. All algorithms reported in this paper are part of the MUME simulation environment [9] and are available free of charge by writing to the author.

3.3.1 BACK PROPAGATION (BP)

This is the standard back-propagation algorithm. The computation of the gradient with respect to the weights is performed on the host computer, based on the readings of the chip's neuron outputs. The derivative of the neuron transfer function was based on a numerical model developed by Leong [12]. It is expected that the numerical model of the derivative would introduce errors, but such errors would also be present and might be larger if an analog derivative circuit implementation was used. The update strategies used for the BP algorithm are NONLINE and NBATCH.

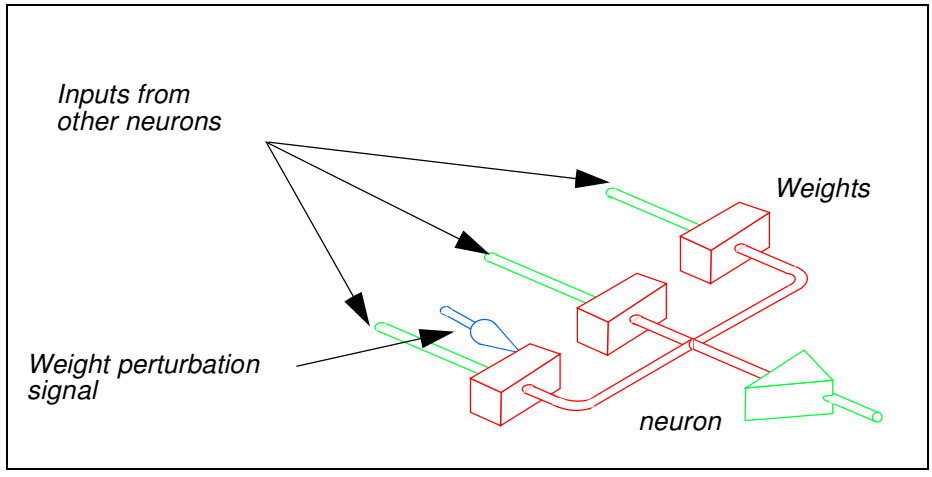
3.3.2 WEIGHT PERTURBATION (WP)

This is as reported in [2] and depicted in Figure 7. The weight updates are computed as

$$\Delta w_{ij} = -\eta \frac{\varepsilon(\boldsymbol{w} + \boldsymbol{p}^{(n)}) - \varepsilon(\boldsymbol{w})}{p_{ij}^{(n)}} \qquad \text{(EQ 1)}$$

Note here only the element p_{ij} of the matrix $\boldsymbol{p}^{(n)}$ is non-null. The error ε represents either the TMSE or the PMSE. As only one weight is perturbed at any one time, the algorithm is sequential with respect to the weights. This is the simplest of the weight perturbation techniques which aims to compute a finite difference approximation to the gradient of the error with respect to the weight. The update strategies used in the experiments are PONLINE and PBATCH.

FIGURE 7. Weight Perturbation. A perturbation is injected at the weight and the difference of network's error before and after the injection is used to approximate the error's gradient.



3.3.3 PARALLEL WEIGHT PERTURBATION SCHEMES

The WP update rule is sequential, that is only one weight is perturbed at any one time. This can be a slow process for large networks. The algorithms that follow implement parallelisations of the weight perturbation scheme.

3.3.3.1 FULLY PARALLEL PERTURBATION ALGORITHMS

Two algorithms have recently been reported to perform parallel weight perturbation. The first was developed by Gert Cauwenberghs and is called "stochastic error descent". The perturbations are depicted in Figure 8 and its learning rule is expressed as

$$\Delta \boldsymbol{w} = -\eta \left(\varepsilon \left(\boldsymbol{w} + \boldsymbol{p}^{(n)} \right) - \varepsilon \left(\boldsymbol{w} \right) \right) \boldsymbol{p}^{(n)} \qquad \text{(EQ 2)}$$

Note that all weights in the net are perturbed simultaneously and that the elements $p_{ij}^{(n)}$ of the perturbation matrix $\boldsymbol{p}^{(n)}$ have to be orthogonal (spatially and temporally decorrelated)

$$E\left(p_{ij}^{(n)} p_{kl}^{(m)} \right) = \sigma^2 \delta_{(ij,\,kl)} \delta_{(n,\,m)} \qquad \text{(EQ 3)}$$

where E denotes expectation, σ is the perturbation intensity and δ represents the Kronecker symbol

$$\delta_{(ij,\,kl)} = \begin{matrix} 1 & i = k \quad and \quad j = l \\ 0 & otherwise \end{matrix} \qquad \text{(EQ 4)}$$

and the Kronecker symbol $\delta_{(n,\,m)}$ is defined as

$$\delta_{(n,\,m)} = \begin{matrix} 1 & n = m \\ 0 & otherwise \end{matrix} \qquad \text{(EQ 5)}$$

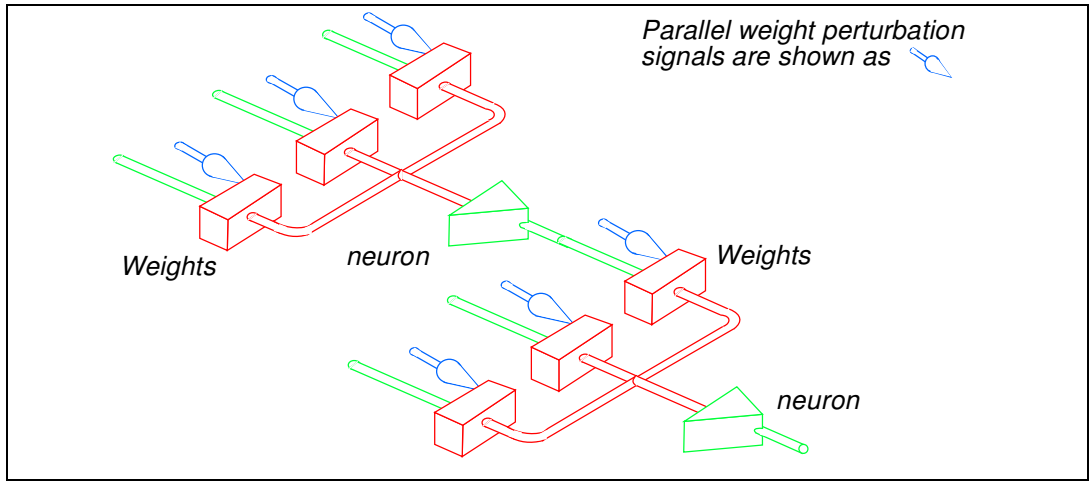
If the condition of EQ. 3 is not met, then greater credit assignment confusions will occur which will lead to a deterioration in the training efficiency. We will elaborate more on this issue in Section 4.

In an analog limited precision environment, the perturbation strength will have a minimum which is equal to the smallest change that can be applied to a weight distinguishable from noise. In practice, the algorithm can be reduced to the case where the perturbation magnitude is a small constant and the perturbation sign is randomly generated. This leads to the second fully parallel perturbation algorithm developed independently by Josh Alspector and colleagues [6] which we call here “constant perturbation random sign” (CPRS). Its learning rule can be expressed as

$$\Delta \boldsymbol{w} = -\eta \left(\varepsilon \left(\boldsymbol{w} + \boldsymbol{p}^{(n)} \right) - \varepsilon \left(\boldsymbol{w} \right) \right) \boldsymbol{p}^{(n)} \qquad \text{(EQ 6)}$$

All the perturbations $p_{ij}^{(n)}$ are equal in strength but random in sign. All weights in the net are perturbed simultaneously. The update strategies used are PONLINE and PBATCH.

FIGURE 8. Parallel weight perturbation. Perturbations of random values are injected at all the weights. The difference in the network's errors before and after the perturbations approximates the error's gradient with respect to the weights.



3.3.4 SUMMED WEIGHT NEURON PERTURBATION (SWNP)

The Summed Weight Neuron Perturbation algorithm is similar in form to CPRS and was developed independently by Flower and Jabri [4]. It is a compromise between sequential and parallel weight perturbation in that only the weights feeding into one neuron are perturbed in parallel. This means that several sequences of perturbations are required for the complete net.

The SWNP learning rule iterates over all the neurons. For neuron j the expression of the learning rule is

$$\Delta \boldsymbol{w}_{+} = -\eta \left(\varepsilon \left(\boldsymbol{w} + \boldsymbol{p}_{+}^{(n)} \right) - \varepsilon \left(\boldsymbol{w} \right) \right) \boldsymbol{p}_{+}^{(n)} \qquad \text{(EQ 7)}$$

where the + subscript indicates that the rule applies only and simulatneously to all the weights feeding into a single neuron. Note that in form the rule is identical to CPRS with the only difference being in the "scheduling" of the perturbations. The update strategies used are PONLINE and PBATCH.

3.4 COMBINED LOCAL /RANDOM SEARCH ALGORITHMS

The perturbation algorithms we have presented above (sequential or parallel) are local in their search, that is, they follow the gradient (or its approximation) of an error with respect to the parameters being

optimised. When the weights are quantised, the implications are that the error surface will present discontinuities and areas of zero gradient. These will affect the learning algorithms which may become unstable and/or stuck in local minima [8][7].

The instability results from the failure of the learning algorithm to evaluate small enough moves on the error surface because of the quantisation resolution. If the resolution cannot be improved, the only available choice is to search the solution space for another state which has non-zero gradient for the given weight resolution. This is difficult to achieve using local search techniques. The algorithms we review below address these problems[2].

3.4.1 COMBINED SEARCH ALGORITHM (CSA)

This algorithm is based on [8]. It performs a combination of modified weight perturbation and a partial random search. Modified Weight Perturbation (MWP) is a local learning rule that tests for a drop in the TMSE before adopting an immediate weight change. Partial random search is a global rule that randomly changes a weight and adopts the change if a reduction of the TMSE is observed. The update strategy used is CIBATCH.

3.4.2 SIMULATED ANNEALING (SA)

A temperature parameter T is used to "anneal" the learning process. The temperature and the TMSE are used to produce a move acceptance threshold function $MAT(T, TMSE)$. The simulated annealing weight update rule randomly changes a weight and the change is adopted either if the TMSE is reduced or if a randomly generated number R satisfies the relation

$$R \geq MAT\left(T, TMSE_{w_{new}} - TMSE_{w_{old}}\right)$$

The update strategy used is CIBATCH.

2. The CSA, SA and MWP-SA techniques do not scale particularly well, meaning that they are inefficient if the network being trained is large.

3.4.3 COMBINED LOCAL SEARCH AND SIMULATED ANNEALING (MWP-SA)

This method alternates the use of a modified weight perturbation with simulated annealing. The update strategy used is CIBATCH.

4 CREDIT ASSIGNMENT EFFICIENCY

There are two main motivations behind the parallelisation of weight perturbation. The first, and the obvious, is to explore the weight solution space faster. The second motivation is to introduce some diffusion in the learning process which could assist in the avoidance of local minima. However, the outcome must be seen in light of the efficiency of the credit assignment, that is, whether the training time (number of feedforwards) is effectively reduced and whether the quality of the training solution is maintained (consistency in training convergence and generalisation performance). This paper argues that in practice fully parallel perturbation algorithms can be much less efficient than semi-parallel algorithms. We do not prove this theoretically, but we will provide an intuitive analysis and later in the paper present extensive experimental results to support this hypothesis.

A proof that fully parallel weight perturbation algorithms can perform gradient descent on average can be found in [5]. The speed up of fully parallel perturbation methods with respect to sequential weight perturbation is a factor of $\sqrt{N}$ where N is the number of parameters[5]. A proof that SWNP achieves a $\sqrt{N}$ "speed-up" over sequential weight perturbation was presented in [4] together with the proof that, in SWNP, the descent steps in the proper directions of the negative of the gradient were larger in size (regardless of the learning rate) than the steps in the wrong direction, and hence that SWNP does implement an error descent scheme.

In semi-parallel and fully parallel weight perturbation algorithms credit assignment confusion are often introduced. A credit assignment confusion is the situation where the update to a weight (or collection of weights) leads to an increase in the error regardless of the size of the learning rate. A simple illustrative example of credit assignment confusion is the situation where the output state of a neuron in a network is zero and yet the weights being driven by that neuron are changed by the updates. This can often happen in semi- and fully parallel weight pertubation schemes. However, in algorithms that compute the gradient analytically (using the chain rule like backpropagation) these weights will not change if the state of the feeding neuron is zero. Confusions can also exist if the condition of EQ. 3 is

not met. Alhtough such confusions can take place in semi- and parallel perturbation schemes, we argue that the reasons why SWNP in general leads to a more efficient learning in the case of AMLPs are:

1. In fully parallel perturbation schemes such as those of [5] and [6], credit assignment confusions are introduced regardless of whether the perturbating signals are totally uncorrelated or not. This is a direct implication of the proof in [5]. Some perturbations will lead to bad credit assignment, and some will lead to good ones. There is no argument that the number of good assignments is on average higher and hence gradient descent take place. The question is how efficient the credit assignment process is. In SWNP all the weights being perturbed feed into a single neuron. Confusions in credit assignment as expected in fully parallel perturbations are "contained" or localised within the fan-in neuron.
2. Due to limited precision, noise and perturbation injection implementations in AMLPs, the smallest perturbation is equal to the smallest quantity that can be injected into the weights and be distinguished from correlated noise. Smallest perturbations would favour fully parallel schemes[5]. The design of circuitry that will allow high precision perturbation and computation in analog is itself a major engineering project. The experiments presented in this paper indicate that semi-parallel perturbation schemes tolerate larger perturbations than fully parallel weight perturbation schemes.

We are not claiming that partially parallel perturbation techniques such as SWNP will always out-perform fully parallel perturbation techniques, but that conditions exist for semi-parallel schemes to be more effective.

5 PARALLELISATION HEURISTICS

We describe in this section two new parallelisation heuristics: fan-out and fan-in-out. The proof that both these techniques do perform gradient descent on average is a trivial case of the proof provided for the SED algorithm in [5].

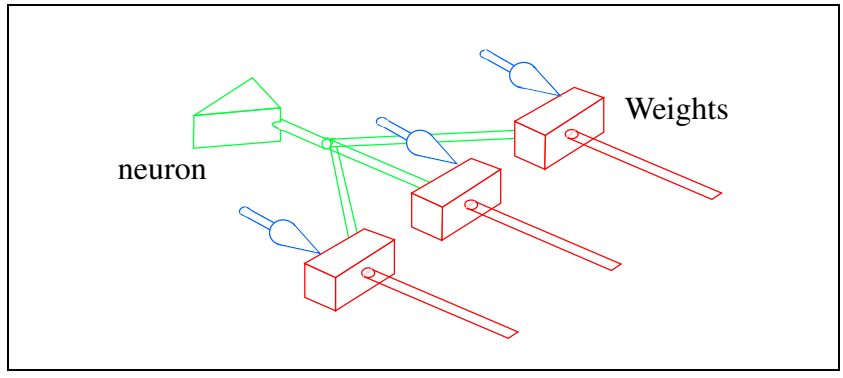
5.1 FAN-OUT SEMI-PARALLEL PERTURBATION

The technique we describe in this section can be called a "fan-out" technique (in contrast with the fan-in SWNP) and is depicted in Figure 9. In a fan-out technique all the weights leaving a neuron are perturbed simultaneously using uncorrelated random values. The learning rule is

$$\Delta \boldsymbol{w}_{+} = -\eta \left(\varepsilon \left(\boldsymbol{w} + \boldsymbol{p}_{+}^{(n)} \right) - e\left(\boldsymbol{w} \right) \right) \boldsymbol{p}_{+}^{(n)} \qquad \text{(EQ 8)}$$

where the + subscript indicates that the rule applies only and simultaneously to all the weights leaving a single neuron. The advantage of fan-out over a fan-in technique (ie. SWNP) can be described as follows. Since all the weights leaving a neuron will feed either to neurons in the following layer or to the outputs, a wider range of the solution space is explored. In a fan-in technique, the range can be constrained by the fact that only one neuron's state is effectively being affected. This at first glance may appear in conflict with the "containment" argument described in Section 4.1. It is true that the containment effect is lost by a fan-out method, but an important gain (probably more advantageous than the containment) is achieved and that is all the weights being perturbed are being fed with the same neuron state. As a result credit assignment confusions due to varying neuron state as in SWNP and fully parallel perturbations are removed. The experiments later in the paper show a significant improvement in training performance of the fan-out method over all other methods reported so-far in this paper.

FIGURE 9. Fan-out based partial perturbation.



5.2 FAN-IN-OUT SEMI-PARALLEL PERTURBATION

In this semi-parallel perturbation scheme, all the weights feeding and leaving a neuron are simultaneously perturbed using uncorrelated random values. The learning rule is

$$\Delta \boldsymbol{w}_{+} = -\eta \left(\varepsilon \left(\boldsymbol{w} + \boldsymbol{p}_{+}^{(n)} \right) - e \left(\boldsymbol{w} \right) \right) \boldsymbol{p}_{+}^{(n)} \qquad \text{(EQ 9)}$$

where the + subscript indicates that the rule applies only and simulatneously to all the weights **feeding** and **leaving** a single neuron. The FAN-IN-OUT heuristic is nothing but a combination of SWNP and FAN-OUT. Although one cannot expect that the full advantages of SWNP and FAN-OUT will be achieved in FAN-IN-OUT, we expect that learning efficiency would be as good as FAN-OUT and the speed-up of exploring the fan-in weights will still be maintained.

5.2.1 EPOCH TIME COST

The experiment results reported in the next section show the number of epochs used by the training algorithms. A hidden effect in the number of epochs for each of the algorithms is the number of feedforwards to be performed. For example, the algorithms CSA, MWP-SA, SA and WP are all sequential with respect to the weights, and for a given epoch training mode, there are as many feedforwards as there are weights. In contrast, BP or any of the fully parallel weight perturbation algorithms require one or two feedforward(s) per epoch when training in an on-line mode. In the case of the sequential perturbation algorithms and for a fully parallel analog network such as Kakadu, a feedforward is the time required to write a weight value (if it has changed on the host computer cache), apply the input pattern and read the output of the network after some chip propagation delay time (in the order of few hundred nano seconds in the case of Kakadu). Therefore, the actual propagation from input to output is *independent* of the size of the network assuming the size fits within the 10-6-4 architecture of KAKADU.

We show in Table 1 the relationship between the number of epochs and the number of feedforwards for each of the learning algorithms, assuming a batch type update strategy is used. In the case of on-line update strategies, the relationship is shown in Table 2 for the relevant algorithms. In both tables

*TABLE 1. Number of feedforwards per epoch for each of the training algorithms in **batch** update strategies. Npatterns is the number of patterns in the training set. Nweights is the number of weights in the network. Nneurons is the number of neurons in the network. Ninputs is the number of inputs (including any bias). Noutputs is the number of outputs.*

Algorithm	**Update Strategy**	**# Feedforwards per epoch**
BP	NBATCH	Npatterns
CSA	CIBATCH	Nweights*Npatterns
MWP-SA	CIBATCH	Nweights*Npatterns
SA	CIBATCH	Nweights*Npatterns
CPRS	PBATCH	2*Npatterns
SWNP	PBATCH	(Nneurons - Ninputs)*Npatterns
FAN-OUT	PBATCH	(Nneurons - Noutputs)*Npatterns
FAN-IN-OUT	PBATCH	(Nneurons - Ninputs - Noutput)*Npatterns
WP	PBATCH	Nweights*Npatterns

we have neglected the differences in the amount of computation required to modify the weights for the perturbation based algorithms, which according to the experience does not significantly affect the time cost of an epoch.

*TABLE 2. Number of feedforwards per training pattern for each of the training algorithms in **on-line** update strategies. The descriptions of the parameters are shown in the caption of Table 1.*

Algorithm	**Update Strategy**	**# Feedforwards per training pattern**
BP	NONLINE	1
CPRS	PONLINE	2
SWNP	PONLINE	Nneurons - Ninputs + 1
FAN-OUT	PONLINE	Nneurons -Noutputs + 1
FAN-IN-OUT	PONLINE	Nneurons - Ninputs - Noutputs + 1
WP	PONLINE	Nweights + 1

6 EXPERIMENTS

Experiments on two problems were conducted: intra-cardiac electro-gram classification and the 4 dimensional parity learning problem. The selection of these two problems was a matter of data convenience and the architecture (size) of the network implemented on the chip. The two spiral problem was considered, but the chip architecture (only two layers of weights) is simply not appropriate.

Each of the experiments was repeated at least 10 times, and some were repeated 20 times. Clearly the rate at which experiments can be performed is limited by the fact that only one hardware set up was available (described below) and the training bottleneck which is the bus interface between the chip and the computer (as the training is being performed in-loop). As a result, not all combinations of training strategies/algorithms/task were performed. We show the experiments that were performed in Table 3.

TABLE 3. Summary of the experiments. The weight and weight update on computer column has two cases: floating point and both weight updates and weights quantised. Note that all the weights are quantised before being loaded onto the Kakadu chip regardless of the way they have been modified on the computer.

Training Algorithm	Update Strategy	ICEG		Parity 4	
		Floating Point	W & DW quantised	Floating Point	W & DW quantised
BP	NBATCH	✓			
	NOLINE	✓			
CSA	IBATCH	✓			
MWP-SA	IBATCH	✓			
SA	PBATCH	✓			
CPRS	PONLINE	✓	✓	✓	✓
	PBATCH	✓			
SWNP	PONLINE	✓	✓	✓	✓
	PBATCH	✓			
FAN-OUT	PBATCH	✓			
	PONLINE	✓		✓	✓
FAN-IN-OUT	PONLINE	✓	✓	✓	✓
WP	PBATCH	✓			
	PONLINE	✓	✓	✓	✓

6.1 THE ANALOG MLP

The AMLP chip used in the experiments is called KAKADU and consists of a 10-6-4 architecture[3] [1]. The weights are stored on the chip as 6-bit numbers (5 plus the sign) in a matrix of static registers. The chip is interfaced to a host computer by means of an analog/digital board (called Jiggle) located in a VME cage.

3. Depending on the experiments, unused weights for a particular architecture are set to 0.

In all the experiments the weights and weight updates were maintained on the host and were downloaded to the chip to perform the feedforward.

6.2 DATA

6.2.1 INTRA-CARDIA ELECTROGRAM DATA SET

These data are classifications of a patient's intra-cardiac electro-gram (ICEG) signals. The input component of each training vector is 10 samples of the right ventricular apex (RVA) lead. The output component is an indication of whether the heart rhythm is a normal sinus rhythm (NSR) or a retrograde ventricular tachycardia (VTR). The data was recorded at an electro-physiological studies (EPS) laboratory and has been normalised using a fixed scale to allow a range between -200mV and 200mV. For each experiment 8 training patterns and 220 testing patterns were used. The 220 test patterns are divided into 150 VTR and 70 NSR. Note the small number of training patterns is not surprising as the network's computation is limited in precision. Larger training sets do not improve the generalisation performance and only lead to longer training time.

The network architectures used in the experiments are 10-5-1 for the batch strategies and 10-6-1 for the on-line strategies.

6.2.2 PARITY 4

In Parity 4, the task is to classify the binary 4-dimensional input vectors (16 of them) according to their parity (even or odd). A typical MLP that solves this problem (in floating point simulation) consists of a 5-4-1 with one of the inputs used as a bias. In the experiments we report below, an architecture of 5-5-1 or 5-6-1 was used. A binary '1' corresponds to an input to the chip of 200mV and a binary '0' corresponds to 0V. The 5-5-1 architecture was used when the weights were maintained (on the host computer) in floating point. The 5-6-1 architecture was used when the weights on the host computer were maintained in truncated form (Section 6.5.1.2 describes the truncation procedure).

6.3 TRAINING CONTROL

6.3.1 TRAINING CONVERGENCE

Each training session is terminated if either the maximum number of iterations (MAXITER) is reached or the TMSE has reached a critical error value (CEV) for which the learning process would be considered to have "converged". In many of the experiments, learning algorithms would reach TMSE values very close to CEV but fail to "converge" within the maximum number of iterations. When training reaches 100% correctness, the training is considered to have converged, regardless of whether it has reached CEV or MAXITER.

6.3.2 LEARNING RATE

In most experiments, the learning rate had to be optimised for the algorithm. The single output of the KAKADU chip can present a maximal mean square error of about 0.002. Obviously if one has truncated the weight updates (to make them integers between -31 and +31) a large learning rate is required. This however may lead to the training "oscillating". Trial and error was used to determine appropriate rates for the learning algorithms. The rate that was found to be the most successful was then selected. In some experiments (especially those with truncated weights and weight updates) up to one hundred pre-trials have been performed to determine appropriate learning rates. Overall, the experiments took over 14 months to complete!

6.3.3 PERTURBATION STRENGTH

In all the experiments reported in this paper, the strength of the perturbation was set to 1. The strength of the perturbation is an important parameter for perturbation algorithms. Like digital MLPs, weight storage in AMLPs will be subject to quantisation regardless of the storage technology. Of course the effects of quantisation could be reduced (but not indefinitely) by using more sophisticated storage circuitry at the cost of an increased chip area. The perturbations applied to the weights of an AMLP are larger or equal to the minimum quantum that the circuitry can manage. In the case of KAKADU where the weights are stored in digital and converted to analog, the minimal perturbation value should be one. This is not a limitation of KAKADU style weight storage. Other implementation methods such as those described in [10] and [11] have equivalent constraints on the minimum perturbation strength.

During in-loop training, perturbation strength could be reduced if the weights are maintained in floating point format (and truncated when down-loaded to the chip). From the experiments, however, we do not find any improvements, but on the contrary, the training sessions required a much larger number of epochs.

6.4 ICEG BATCH MODE EXPERIMENTS

The batch mode set of experiments were performed on the ICEG classification problem only. A 10-5-1 AMLP architecture was used. The weights and weight updates were maintained in floating point representations. The weights are quantised to 6 bits before being loaded into the chip to produce its response to an input pattern. For each algorithm 20 training trials were performed. Each trial starts with different initial conditions. An upper limit of 2000 training iterations was set. This upper limit was determined by:

- excessive experimental time required by some algorithms,
- average performance of most algorithms

We show the training performance of the eight algorithms in Table 4. The first column shows the algorithm; the second column shows the update mode; the third column has two sub-columns showing the average training error and its standard deviation respectively (on the runs that have successfully trained); the fourth column shows the average and standard deviation of the number of epochs[4]

TABLE 4. Training performance summary of the learning algorithms on the ICEG classification problem for 20 trials. See text for explanation.

Algorithm	**Update mode**	**Training TMSE**		**Epochs**		**Feedforwards**		**Converged**
Max Iter: 2000 CEV: 0.0004		*Average*	*Standard Deviation*	*Average*	*Standard Deviation*	*Average*	*Standard Deviation*	
BP	NBATCH	3.48e-04	5.96e-05	960	329	7680	2632	14
CPRS	PBATCH	3.46e-04	5.50e-05	1391	216	22256	3456	12
CSA	CIBATCH	3.36e-04	4.80e-05	35	18	15400	7920	20
MWP-SA	CIBATCH	3.50e-04	5.30e-05	88	81	38720	35640	20
SA	CIBATCH	3.16e-04	8.09e-05	165	85	72600	37400	20
SWNP	PBATCH	3.66e-04	3.65e-05	1270	277	60960	13296	18
FAN-OUT	PBATCH	3.82e-04	2.06e-05	1165	323	139800	38760	20
WP	PBATCH	3.57e-04	2.61e-05	466	162	205040	71280	20

for the runs that have successfully trained. The fifth column also has two sub-columns showing the average and standard deviation of the number of feedforwards performed during training. Note these numbers are related to the number of epochs using the information of Table 1. The last column shows the number of times an algorithm has successfully trained all the patterns (i.e. converged) out of the 20 runs.

Table 4 shows that all algorithms have a probablity of more than 50% to converge. Epoch wise, CSA is the fastest. Based on the number of feedforwards, BP is the fastest followed by CSA. While BP has fewer feedforwards, it is more computationally expensive than the other algorithms when computing the weight updates. Note that all algorithms except BP, CPRS and SWNP, managed to converge in all 20 trials.

The generalisation (or testing) results are shown in Table 5. The first column in the table shows the

TABLE 5. Generalisation performance summary of the learning algorithms on the ICEG classification problem.

Algorithm	**Testing Error (TMSE)**		**Classification (% correct)**	
	Average	*Standard Deviation*	*Average*	*Standard Deviation*
BP	4.31e-04	2.56e-04	95.62	4.56
CPRS	2.01e-04	8.8e-05	96.97	2.29
CSA	3.27e-04	3.92e-04	95.57	5.55
MWP-SA	1.08e-03	1.06e-03	86.89	12.98
SA	7.70e-04	5.81e-04	90.72	7.45
SWNP	2.22e-04	1.20e-04	96.21	3.27
FAN-OUT	2.06e-04	1.3e-04	96.86	2.93
WP	3.80e-04	1.69e-04	94.48	2.99

algorithm; the second column has two sub-columns showing the average and the standard deviation of the TMSE over the runs that have successfully trained. The smaller the standard deviation the more consistent the performance of the network was across the successful runs. The third column shows the classification performance (for the successful runs), again with a column showing the average and

4. An epoch is an iteration through the training set.

another column showing the standard deviation. Most algorithms generalise pretty well, with a cluster of them performing in the high nineties. The best average performance was achieved by CPRS followed very closely by FAN-OUT and SWNP[5]

6.5 ON-LINE MODE

In the case of on-line training, two sets of experiments were performed for both the ICEG and Parity 4 problems:

1. weights and weight updates in floating point
2. weights and weight updates truncated to 6 bits.

We describe first the ICEG on-line mode experiments for both floating point weights and updates.

6.5.1 ICEG ONLINE MODE EXPERIMENTS

In the ICEG on-line mode experiments a 10-6-1 AMLP architecture was used. The 10-5-1 architecture used in the batch mode experiments was evaluated and found to be inferior to the 10-6-1 architecture (within the constraint of the maximum number of iterations). The algorithms CSA, MWP-SA and SA were not experimented with as earlier work has shown very poor performance when on-line update modes are used.

6.5.1.1 WEIGHTS AND WEIGHT UPDATES IN FLOATING POINT

In this set of experiments, 20 trials were conducted for each algorithm. The weights were maintained on the host in floating point representation (double precision) and were truncated to integers in the range [-31, 31] when used on the chip. A summary of the training performance of the algorithms is shown in Table 6. Note the poor performance (in terms of number of successful runs) of BP. All perturbation based algorithms have a respectable convergence rate. The algorithm with the highest probability of successful training is FAN-IN-OUT, only failing 2 out of the 20 runs. Interestingly, FAN-IN-OUT also has a competitive feedforward average and standard deviation when compared to BP and CPRS.

5. The number of successful runs of SWNP and CPRS is however smaller.

TABLE 6. Training performance on the ICEG classification problem in on-line update mode. Weights and weight updates are maintained in floating point representation.

Algorithm	**Update Mode**	**Training TMSE**		**Epochs**		**Feedforwards**		**Converged**
Max Iter: 1000 CEV: 0.00008		*Average*	*Standard Deviation*	*Average*	*Standard Deviation*	*Average*	*Standard Deviation*	
BP	NONLINE	6.65e-05	1.56e-05	919	63	919	63	4
CPRS	PONLINE	6.62e-05	1.50e-05	810	227	1620	454	11
SWNP	PONLINE	7.04e-05	1.23e-05	597	242	4776	1936	11
FAN-OUT	PONLINE	7.57e-05	3.65e-06	662	100	11254	1700	16
FAN-IN-OUT	PONLINE	6.98e-05	5.87e-06	701	209	4907	1463	18
WP	PONLINE	7.08e-05	6.23e-06	633	224	42411	15008	12

The generalisation performance of the algorithms is shown in Table 7. The best average generalisation performance is achieved by FAN-IN-OUT with an average correct classification of 98.61% and a very small standard deviation. FAN-OUT achieves a very close average performance and a very small standard deviation as well. All the algorithms in fact achieve an average performance in the high 90s.

TABLE 7. Generalisation performance of the algorithms on the ICEG classification problem.

Algorithm	**Testing Error**		**Classification (% correct)**	
	Average	*Standard Deviation*	*Average*	*Standard Deviation*
BP	3.45e-04	1.13e-04	97.39	0.78
CPRS	2.17e-04	1.01e-04	97.48	1.96
SWNP	2.84e-04	1.80e-04	96.49	3.46
FAN-OUT	1.98e-04	6.71e-05	98.18	1.17
FAN-IN-OUT	1.67e-04	7.26e-05	98.61	1.46
WP	3.18e-04	1.03e-04	97.76	1.28

The generalisation performance of BP has to be treated carefully as it is based on the 4 successful trials only.

6.5.1.2 TRUNCATED WEIGHTS AND WEIGHT UPDATES

In this set of experiments, both the weights and their updates are maintained in a 6-bit integer format [-31 to +31]. The truncation process is performed as in Figure 10. Note the weight updates are first multiplied by the learning rate; they are then quantised to six bits; they are added to the weights; and

then the weights are quantised to six bits. This particular way of quantising the weights and their updates is “implementation” motivated. In an AMLP, the weight updates are most likely to be implemented as charge or discharge of capacitors (see [6], [10] and [11] for examples) and hence will be applied as “quantum”. The weights themselves have to be either maintained in a digital random access memory (RAM), electrically erasable programmable read only memory (EEPROM) or simply refreshed from their analogue values. In any of these cases, weights will be truncated to a limited precision.

FIGURE 10. Truncation of weight updates and weights.

```
foreach weight[i,j]{
   w[i,j] = w[i,j] + Quantise(Eta * dw[i,j]);
   w[i,j] = Quantise(w[i,j]);
}
```

Unlike the previous experiments, only 10 trials were conducted[6]. Due to the poor performance of BP in the previous experiment, it was excluded from this experiment. The training results are shown in Table 10. Note that only FAN-OUT, FAN-IN-OUT, and WP managed to converge 10 times out of the 10 trials. This is followed by SWNP which manages to converge 7 times and CPRS which converges 5 times.

TABLE 8. Summary of the 10 training trials on the ICEG data with truncated weights and weight updates. The update strategy PONLINE was used.

Algorithm	**Training Error**		**Epochs**		**Feedforwards**		**Converged**
Max Iter: 2000 CEV: 0.0001	*Average*	*Standard Deviation*	*Average*	*Standard Deviation*	*Average*	*Standard Deviation*	
CPRS	8.16e-05	1.83e-05	1517	357	3034	714	5
SWNP	8.53e-05	8.8e-06	872	517	6976	4136	7
FAN-OUT	9.48e-05	6.29e-06	667	622	11339	10574	10
FAN-IN-OUT	9.33e-05	5.86e-06	409	434	2863	3038	10
WP[a]	8.70e-05	1.31e-05	920	616	61640	41272	10

a. As in the previous experiment, WP’s max number of iterations was set to 1000.

Interestingly, FAN-IN-OUT is not only the most consistent, but is also the fastest epoch and feedforward wise.

6. The maximum number of iterations has been increased to 2000 and the convergence criteria was relaxed to 0.0001.

The generalisation performance of the algorithms is shown in Table 9. Note if one neglects that the sample size (number of successful runs) of CPRS is half that of FAN-OUT, FAN-IN-OUT and WP, CPRS and SWNP achieve the best average generalisation performance with SWNP being the most consistent (over 7 runs of course). In the group of algorithms that converged 10/10, FAN-OUT achieves the best and most consistent performance, followed very closely by FAN-IN-OUT.

TABLE 9. Generalisation performance on the ICEG classification problem with truncated weights and weight updates.

Algorithm	**Testing Error**		**Classification (% correct)**	
	Average	*Standard Deviation*	*Average*	*Standard Deviation*
CPRS	1.48e-04	6.09e-05	98.64	1.79
SWNP	1.94e-04	1.36e-04	98.31	1.28
FAN-OUT	3.4e-04	2.13e-04	95.77	4.27
FAN-IN-OUT	3.43e-04	2.99e-04	95.41	5.28
WP	6.59e-04	7.53e-04	92.09	8.29

6.5.2 PARITY 4 ONLINE EXPERIMENTS

6.5.2.1 WEIGHTS AND WEIGHT UPDATES IN FLOATING POINT

In this experiment a 5-5-1 architecture was used[7]. Table 10 shows the training performance summary for 20 training trials. The best performance is achieved by FAN-OUT followed by SWNP. FAN-OUT

TABLE 10. Summary of the 20 training trials on the Parity 4 problem. The update strategy PONLINE was used.

Algorithm	**Training Error**		**Epochs**		**Feedforwards**		**Converged**
Max Iter: 2000 CEV: 0.00008	*Average*	*Standard Deviation*	*Average*	*Standard Deviation*	*Average*	*Standard Deviation*	
CPRS	4.37e-04	4e-04	1995	11	3990	22	5
SWNP	7.50e-05	4.33e-06	970	406	7760	3248	12
FAN-OUT	7.26e-05	6.29e-06	1069	273	12828	3276	14
FAN-IN-OUT	1.02e-04	9.16e-05	1453	414	10171	2898	10
WP	7.6e-05	2.83e-06	1083	482	40071	17834	6

is also the most consistent. Note that FAN-IN-OUT has achieved a respectable performance when compared to WP and CPRS.

7. A 5-4-1 architecture was tried but could not lead to 100% correct training.

As the Parity 4 problem is more of a training challenge (rather than a generalisation challenge as it is the case with the ICEG classification problem), we show in Table 9 the average performance of the training algorithms over all the trials (not only the successful ones as we did in the ICEG experiments). The experiments also show that FAN-OUT performs best on average with the highest correct classification rate and smallest standard deviation over all 20 trials. The results also show a consist-

TABLE 11. Overall training performance on all trials for the Parity 4 problem (including trials that did not converge).

Algorithm	**Testing Error**		**Classification (% correct)**	
	Average	*Standard Deviation*	*Average*	*Standard Deviation*
CPRS	6.61e-04	4.24e-04	91.88	6.76
SWNP	3.14e-04	4.01e-04	96.56	5.16
FAN-OUT	1.89e-04	3.06e-04	97.19	5.16
FAN-IN-OUT	3.49e-04	3.82e-04	95.31	5.69
WP	5.43e-04	3.9e-04	94.06	5.16

ency between the performance (average) of the successful trials and the performance over all the trials.

6.5.2.2 TRUNCATED WEIGHTS AND WEIGHT UPDATES

In this experiment a 5-6-1 architecture was used. The weights and their updates were truncated as described in Section 6.5.1.2. Ten trials were performed and Table 10 shows the training performance. Here we see that training of the Parity 4 problem becomes much harder with CPRS and SWNP train-

TABLE 12. Summary of the 10 training trials on the Parity 4 problem with truncated weights and weight updates. The update strategy PONLINE was used.

Algorithm	**Training Error**		**Epochs**		**Feedforwards**		**Converged**
Max Iter: 4000 CEV: 0.00008	*Average*	*Standard Deviation*	*Average*	*Standard Deviation*	*Average*	*Standard Deviation*	
CPRS	4.42e-04	0	4000	0	8000	0	1
SWNP	6.4e-04	0	4000	0	32000	0	1
FAN-OUT	6.71e-04	3.6e-04	4000	0	48000	0	4
FAN-IN-OUT	2.91e-04	3.17e-04	3204	1278	22428	8946	9
WP	5.4e-04	4.1e-04	3432	983	126984	36371	3

ing to 100% only once. Of all five algorithms, only FAN-IN-OUT and WP managed to bring the

TMSE below 0.00008. All others, managed to train to 100% for the number of trials shown in the "Converged" column of Table 9. We show in Table 9 the training performance for all trials (regardless of whether they have converged or not). This table shows that FAN-IN-OUT was the most successful on average and the most consistent, followed by FAN-OUT.

TABLE 13. Training performance with truncated weights and weight updates on all trials for the Parity 4 problem (including trials that did not converge).

Algorithm	**Testing Error**		**Classification (% correct)**	
	Average	*Standard Deviation*	*Average*	*Standard Deviation*
CPRS	2.12e-03	1.5e-03	70.62	20.42
SWNP	1.23e-03	1.06e-03	81.25	18.87
FAN-OUT	3.81e-04	3.66e-04	94.38	5.47
FAN-IN-OUT	2.54e-05	8.03e-05	99.38	1.98
WP	1.98e-03	2.25e-03	83.12	16.15

7 DISCUSSION

The experiments above indicate that:

- all algorithms used here can train AMLPs in an in-loop mode.
- semi-parallel weight perturbation algorithms can be much more effective than fully parallel perturbation schemes.
- when weights and weight updates are truncated, the performance of the backpropagation algorithm rapidly degrades as it has been reported in [8] and [13].
- FAN-IN-OUT has consistently produced the best performance when weights and weight updates are truncated.

Of most interest to us from these experiments is the indication for on-chip training requirements. From the experiments above, the on-line training strategy is the most likely candidate for on-chip learning. Batch strategies (PBATCH or CIBATCH) are not practical for perturbation based algorithms as they require the TMSE which is evaluated over all the training set, requires additional storage, and leads to a high computational cost per learning iteration. The second and more important requirement is the fact that in on-chip learning, weights and weight updates have to be in fixed point quantised format. Depending on how the weights are stored (digital or analog storage) precision may vary. The bottom line is that for a highly compact implementation, it is not attractive to have high precision weight and weight update storage. In that regard, the experiments demonstrate that with a precision of

the weights reduced and quantised down to 6-bits, excellent training and generalisation performance can still be achieved with semi-parallel perturbation techniques.

Due to the excessive length of time required for each experiment, it was not possible to either increase the number of trials, the maximum number of iterations or the number of classification problems. Each experiment set required one to ten days[8] and in many cases an experiment set had to be restarted because of a computer crash. Further engineering work on the interface may facilitate faster experimentations in the future.

Additionally, theses experiments were limited to algorithms that can operate on the chip and within the interface framework. There are other training algorithms, like the MADALINE Rule III or neuron perturbation [3] which could not tested because it requires write-access to the neuron states which is not possible in the chip interface framework.

8 CONCLUSIONS

The practical capabilities of perturbation and backpropagation based algorithms on the training of an analog MLP chip were evaluated. The experiments show that all algorithms can be used successfully for in-loop training, however backpropagation algorithms have difficulties when weights and weight updates are maintained in a fixed point format (truncated). Perturbation based algorithms that rely on an averaged gradient descent perform well in practice. As expected, sequential weight perturbation is consistent in achieving good training and generalisation performance but is succeptible to being trapped in local minima. The experiments demonstrate that parallel weight perturbation techniques that perform stochastic error descent (CPRS, SWNP) all achieve good generalisation performance and are less succeptible to be trapped in local minima. Amongst the parallel perturbation schemes, the FAN-OUT semi-parallel perturbation scheme seems to be the most effective. The experiments demonstrate, that at least for the type of learning problems that have been addressed, full parallel perturbation schemes are not as effective as semi-parallel schemes such as SWNP, FAN-OUT and FAN-IN-OUT.

8. Note the VME cage access cycle is about 3micro-seconds about an order of magnitude larger than the MLP chip feedforward propagation delay.

9 AKNOWLEDGEMENT

I would like to thank B. Flower, R. Coggins, E. Tinker, J. Alspector and G. Cauwenberghs for comments and suggestions. Part of this work was funded by the Australian Research Council.

10 References

[1] Philip Leong and Marwan Jabri, "A Low Power Trainable Analogue Neural Network Classifier Chip", in the IEEE Custom Integrated Circuits Conference, San Diego, pp 4.5.1-4.5.4, May 1993.

[2] Marwan Jabri and Barry Flower. Weight perturbation: An optimal architecture and learning technique for analog VLSI feedforward and recurrent multilayer networks. *IEEE Transactions on Neural Networks*, 3(1):154–157, January 1992.

[3] 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.

[4] Barry Flower and Marwan Jabri. Summed Weight Neuron Perturbation: A O(N) Improvement Over Weight Perturbation. NIPS 5, Morgan Kaufmann, pp. 212-219, 1993.

[5] Gert Cauwenberghs. A Fast Stochastic Error-Descent Algorithm for Supervised Learning and Optimisation. NIPS 5, Morgan Kaufmann, pp. 244-251, 1993.

[6] Josh Alspector, Ronny Meir, Ben Yuhas and Antonny Jayakumar. A Parallel Gradient Descent Method for Learning in Analog VLSI Neural Networks. NIPS 5, Morgan Kaufmann, pp. 836-844, 1993.

[7] Marwan Jabri, Steve Pickard, Philip Leong and Yun Xie. Algorithms and Implementation Issues in Analog Low Power Learning Neural Network Chips. International Journal on VLSI Signal Processing, Vol. 6, No. 2, pp. 67-76, March 1993, USA.

[8] Yun Xie and Marwan Jabri. On the Training of Limited Precision Multi-layer Perceptron. Proceedings of the International Joint Conference on Neural Networks, pp III-942-947, July 1992, Baltimore, USA.

[9] Marwan Jabri, Edward Tinker and Laurens Leerink. MUME - A multi-module multi-algorithm neural simulation environment. To appear in Neural Simulation Environments, Kluwer Publishing, 1993.

[10] Gert Cauwenberghs, "Analog VLSI autonomous Systems for Learning and Optimization", PhD thesis, Caltech, 1994.

[11] Barry Flower, Marwan Jabri and Steve Pickard, "An Analogue On-Chip Supervised Learning Implementation of an Artificial Neural Network", submitted to IEEE Transactions on Neural Networks, April 1994.

[12] P. Leong, "Arrhythmia classification using low power VLSI", PhD thesis, the University of Sydney, 1992.

[13] Yun Xie and Marwan Jabri. Analysis of the effects of quantization in multilayer neural networks using a statistical model. *IEEE Transactions on Neural Networks*, 3(2):334–338, March 1992.