The Convergence of Backpropagation Trained Neural Networks for Various Weight Update Frequencies

Jim Torresen*

Abstract

One of the problems concerning the backpropagation training of feed-forward neural networks is the effect of the weight update frequency. This aspect influences the efficiency of parallel implementations of the training algorithm where the training vectors are distributed among processors. In this paper the convergence of two applications for various weight update intervals is reported. Further, several models are proposed for describing convergence and learning rate aspects in the context of a set of weight update intervals. The results show that the convergence by updating the weights after each training vector leads to about 10 times less number of training iterations compared to updating the weights only ones for the whole training set.

Keywords: Backpropagation, convergence, NETtalk, weight update interval.

1 Introduction

The weight update frequency of the Back Propagation algorithm (BP) [1] has got a major impact on the error convergence during neural network training. However, no theory exactly describes this relation. In this paper the relation between the weight update interval — i.e. the number of training patterns that is presented between weight updates, and the error convergence of the BP training algorithm will be tried modelled. Two weight layer feed-forward neural networks are used in the work.

In the forward phase the hidden layer weight matrix $\mathbf{W}_h$ is multiplied by the input vector $\mathbf{X} = (x_1, x_2, \ldots, x_{N_i})^T$, to calculate the hidden layer output

$$y_{h,j} = f(\sum_{i=1}^{N_i} w_{h,ji} x_i - \theta) \tag{1}$$

where $w_{h,ji}$ is the weight connecting input unit i to unit j in the hidden neuron layer[1]. The weights and input values are both floating point variables. The θ is an offset termed bias,

*The work has been conducted at Department of Information Science, Kyoto University, Japan and at Department of Computer and Information Science, Norwegian University of Science and Technology, Norway. J. Torresen is presently at NERA Telecommunications, Excom Division, P.O. Box 91, N-1361 Billingstad, Norway. E-mail: jim@nera.no

[1] To distinguish the different neuron layers, the indices i, j, and k are used for indexing the input, hidden, and output neuron layer, respectively.

incorporated into the training algorithm by a weight connected to +1 for each neuron [3]. This bias-weight is trained like an ordinary weight.

The function f is a nonlinear activation function. Normally the S-shaped sigmoid function

$$f(\alpha) = \frac{1}{1 + e^{-\alpha}} \tag{2}$$

is used. It compresses the output value to lie in $< 0, 1 >$. Moreover, the function is differentiable, which is a demand of the training algorithm.

The output from the hidden layer, $y_{h,j}$, is used to calculate the output of the network, $y_{o,k}$

$$y_{o,k} = f(\sum_{j=1}^{N_h} w_{o,kj} y_{h,j} - \theta) \tag{3}$$

In the backward phase the target, d, and output, y_o, are compared and the difference (error) is used to adapt the weights to reduce the error. The error to be used to update the weights can be shown [1] to be

$$\delta_{o,k} = y_{o,k}(1 - y_{o,k})(d_k - y_{o,k}) \tag{4}$$

Similar to computing the output delta error, the hidden delta error value for neuron j is

$$\delta_{h,j} = y_{h,j}(1 - y_{h,j}) \sum_{k=1}^{N_o} \delta_{o,k} w_{o,kj} \tag{5}$$

The error is not explicitely given and is computed based on the impact of the fan-in of the output delta errors. To perform steepest descent in the weight space, the weight changes become

$$\Delta w_{o,kj} = \eta \delta_{o,k} y_{h,j} \qquad \wedge \qquad \Delta w_{h,ji} = \eta \delta_{h,j} x_i \tag{6}$$

where η is the learning rate coefficient.

If *learning by pattern* is applied, the output layer weights are changed to $w'_{o,kj}$

$$w'_{o,kj} = w_{o,kj} + \eta \delta_{o,k} y_{h,j} \tag{7}$$

The hidden layer weights are updated accordingly

$$w'_{h,ji} = w_{h,ji} + \eta \delta_{h,j} x_i \tag{8}$$

The training continues for each vector in the training set until the error for the entire set becomes acceptably small.

Instead of updating the weights after each training pattern presentation, they can be updated less frequently by using *learning by block*. For updates after μ patterns have been presented Equations 7 and 8 become 9 and 10

$$w'_{o,kj} = w_{o,kj} + \eta \sum_{p=p'+1}^{p'+\mu} \delta_{p,o,k} y_{p,h,j} \tag{9}$$

$$w'_{h,ji} = w_{h,ji} + \eta \sum_{p=p'+1}^{p'+\mu} \delta_{p,h,j} x_{p,i} \tag{10}$$

The total number of training patterns is P and $\mu \leq P$. The different weight update schemes will be described in more detail below.

To obtain true gradient descent requires infinitesimal small changes of the weights. This is obtained by selecting a small value for the learning rate. However, we want to choose a learning rate as large as possible without leading to oscillation[2], since experiments show that this offer the most rapid learning [1]. To increase the learning rate and avoid oscillation, a momentum can be included. Rumelhart et al. proposed [1] to add a fraction, equal to α, of the previous weight update value to the current weight change

$$\Delta w_{o,kj}(p+1) = \eta \delta_{o,k}(p+1) y_{h,j}(p+1) + \alpha \Delta w_{o,kj}(p) \tag{11}$$

where p is the training pattern index. The weights are updated

$$w_{o,kj}(p+1)' = w_{o,kj}(p) + \Delta w_{o,kj}(p+1) \tag{12}$$

Similarly, Sejnowski et al. [12] proposed a smoothing term, α

$$\Delta w_{o,kj}(p+1) = \alpha \Delta w_{o,kj}(p) + (1-\alpha) \delta_{o,k}(p+1) y_{h,j}(p+1) \tag{13}$$

The weights are updated

$$w_{o,kj}(p+1)' = w_{o,kj}(p) + \eta \Delta w_{o,kj}(p+1) \tag{14}$$

The smoothing makes it less neccesary to scale the learning rate if the weights are to be infrequently updated. The equations for updating of the hidden weights can be similarly derived. The term α is normally set to around 0.9. The smoothing term has been applied in the experiments presented in this paper.

One training iteration contains one presentation of the whole training set. The following weight update strategies can be applied:

- *Learning by pattern (lbp)*, update the weights after *each* training pattern has been presented.
- *Learning by block (lbb)*, update the weights after a subset of the training patterns has been presented.
- *Learning by epoch (lbe)*, update the weights after all patterns have been presented (i.e. one training iteration).

[2] The error is not constantly decreasing but is oscillating between large and small error values without reaching convergence.

A special case of *learning by pattern* is *delayed weight update* in which the weights are updated for pattern p *after* the forward pass for the next pattern – $p + 1$, has been computed. This method may be used for weight updating when the computation of each layer is distributed over different processors. The delta weight change values are small compared to the weights and thus the convergence should be very close to the convergence of ordinary pattern weight updates.

The number of training patterns that is presented between weight updates, as introduced above, is termed μ — indicating the weight update interval. For *lbp*, $\mu = 1$, while for *lbe* $\mu = P$, where P is the number of training patterns in the training set.

Only the *lbe* approach has been proved to converge. However, there is not known any theoretical bound on the number of presentations required for convergence. Several experiments on the other hand show that the convergence of BP improves if the weights are frequently updated. Due to the lack of proven convergence, the word *convergence* is defined as reaching a pre-determined stopping criteria.

The faster convergence for frequent weight updates is shown for a neural network trained to classify patterns into three classes by Paugam–Moisy [2]. In the experiment less frequent weight updates during training reduces the convergence rate. That is, the decrease in error per training iteration is smaller. To avoid unstable behavior during training, the learning rate had to be reduced when the weight update interval was increased. The reason for this can be explained by network paralysis [3]. Adding weight changes for many training vectors together may result in large weight change values. This may lead to large weight values, if the learning rate is not reduced. Large weights can lead to large output values to be input to the non-linear function. The derivative of the function, which is used for computing the delta error, approaches zero for large values. This results in a very small change in the weights, and the training can come to a virtual standstill.

The benefit of using infrequent weight updates is shown in [4], where a highly parallel computer is used for training neural network. For this computer, it is necessary to exploit *training set* parallelism to obtain a satisfactory performance. That implies that the network weights are updated after a *subset* of the training patterns has been computed. Thus, to be able to utilize todays highly/massively parallel computers in the best way, we ought to know the effect of the weight update frequency to the convergence. In [5], [6], a heuristic for mapping BP onto a parallel computer is given. This is based on estimating the execution time, $T_{1it}(\mu_i)$, where $1it$ denote one passage through the training set. The estimation is done for K chosen weight update intervals μ_i, $i = 1, \ldots, K$. The μ_i minimizing

$$T_{total} = T_{1it}(\mu_i)N(\mu_i)$$

is selected. $N(\mu_i)$ is the total number of iterations, needed to obtain convergence, for the weight update intervals μ_i.

At least two approaches are possible to determine $N(\mu)$. First, to use existing formulas [2],[7] for the estimation of $N(\mu)$. These are based on specific applications and computes the number of iterations needed based on the learning rate. Thus, in general they will be inaccurate.

A second approach is to run the algorithm for a few iterations (for various weight update intervals) and based on the convergence estimate $N(\mu)$ [8]. An variant of this is to train the

network for one μ and use the convergence result to estimate convergene for other values of μ. The approach benefits from the inclusion of all implemented optimizing techniques in the estimation and quite accurate values of $N(\mu)$ should be obtainable. In this paper, the second approach is described and tested.

For some neural application experiments, like training recurrent networks for speech recognition [9], *lbp* updating results in a stagnation of the error that does not occur for *lbe* updating. The quick-propagation (quickprop) algorithm, proposed by Fahlman [10], is based on *lbe*. That is, all training vectors are applied before the new weights are computed. Thus, training set parallelism can be used in parallel implementation of quickprop without leading to any reduction in the convergence rate.

The redundancy in large training sets slows down the convergence of *lbe* based algorithms according to Møller [11]. Accumulating redundant gradient weight vectors implies redundant computation. This is not a problem if the weights are updated after each training vector. In fact, redundancy has a positive effect in the beginning of the training. However, simulations show that a conjugate gradient training algorithm – which is based on *lbe*, is more efficient in the end of training even though there is redundancy in the training set. Thus, a *learning by block* approach is proposed, where the block size varies throughout training. Since the redundancy is dependent on the problem, the block size has to be selected by estimation. For each iteration the block size that lead to a confident decrease in the total error of the training set is determined. Simulation using NETtalk shows that the training starts with a very small block size and ends by updating weights only two or three times per epoch.

Two different applications — NETtalk and Sonar Return Classification, are used in this paper to get experimental results. Rules are derived for the best learning rate and the needed training iterations as a function of the weight update interval/the error at start of training. Several applicable sensitivity aspects are also considered. The convergence of the two applications are compared and a general rule for the number of training iterations is derived.

Section 2 gives a description of each of the applications studied. The convergence for various weight update intervals are described in Section 3. A comparison of the two applications are also included in the section. Finally, concluding remarks ends the paper in Section 4.

2 BP trained applications

Many different applications have arrived for the BP trained feed-forward neural network. Two of the earliest ones are NETtalk and Sonar Target Classification. Both training sets are freely available and have become benchmarks for testing new training algorithms. Below follows a short description of each application.

2.1 Speech Synthesis

NETtalk is a two layer feed-forward network that transforms text to phonemes[3], using 203 input units, 60–120 hidden units and 26 output units [12]. The input to the network is series of 7 consecutive letters from one of the training words, see Figure 1.

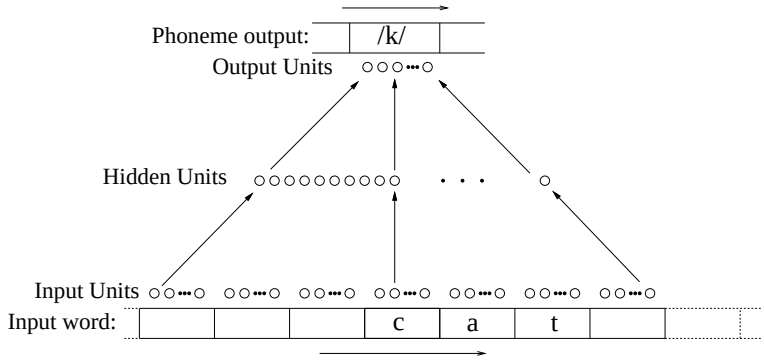

Figure 1: **Schematic drawing of the NETtalk network.**

The central letter is the one for which the phonetic output is to be produced. The three letters on each side of central character help to determine the pronunciation. The input word is scrolled through the window of 7 letters so that each letter, one after another, is placed in the center of the window. The network can be used for continuous text or a dictionary of words. For the latter approach, the words are moved through the window individually. Thus, empty letters are added in front of and after the input word as seen in the Figure 1. For a 1000 word training set, 98 % correct transformation was reported.

2.2 Sonar Return Classification

Gorman et al. [13], [14] used feed-forward neural network to the classification of sonar returns from two underwater targets, a metal cylinder and a similar shaped rock. The network classification performance was shown to be better than that of trained human listeners. The network uses one hidden layer and 60 continuous-valued input units. The best performance was achieved in the case of 24 hidden units. However, this was only slightly better than that achieved with only 12 hidden units. Two output units were used, where (1,0) represented a return signal from a metal cylinder, and (0,1) represented a return signal from a rock. A set of 208 returns (111 cylinder returns and 97 rock returns) were used in the experiments.

[3]Elementary speech sounds.

3 Results

3.1 NETtalk Learning

To investigate the convergence rate for different weight update intervals, 1000 of the most common English words (a total of 5438 characters) were used. The error function for a training pattern p is given by

$$E_p(n) = \frac{1}{2}\sum_k (d_{p,k} - y_{p,o,k})^2$$

where n is the training iteration number, $d_{p,k}$ is the target output and $y_{p,o,k}$ is the output layer output. Rumelhart et al. use the values 0.1 and 0.9 as target values, since the extreme values of 0 or 1 can never be reached [1]. If the error is 0.1 per output unit the pattern error becomes

$$E_p(n) = \frac{1}{2}\sum_{k=1}^{26} (0.1)^2 = 0.13$$

and this value has been used as a threshold for determining if a pattern is trained or not. Weight change values were accumulated for a pattern p having $E_p(n) > 0.13$, while a pattern is trained when $E_p(n) \leq 0.13$. This is a slightly different error measure than reported by Sejnowski and Rosenberg, who measured the error at each individual output neuron. They used a "best guess" which compared the output values to a subset of the training set (52 target vectors). If the vector that had the smallest output error was equal to the training vector then the vector was considered correctly trained. Both methods decrease the sum of square error function

$$E(n) = \sum_{p=1}^{5438}\sum_{k=1}^{26} (d_{p,k} - y_{p,o,k})^2 \tag{15}$$

The error measure used in this research avoids over-training without a large loss in the load balance in the case of parallel computation. Since the purpose of these experiments has been to study the relation between weight update interval and convergence and not the performance of the application, a separate test set has not been used. Testing convergence on the training set requires less time than using a test set. Thus, more tests for each weight update interval could in this way be undertaken.

For each weight update interval investigated, the best learning rate η was searched for. Between 10 and 15 runs of each weight update interval were performed. These experiments in total required several hundred CPU hours. For a small μ, the best learning rate was equal to 1.5. It had to be decreased for larger values of μ. For larger μ, the convergence was more sensitive to the selection of η. The value for the smoothing term α was selected to be 0.9 as in [12]. This value was used for all the experiments. Due to this smoothing term, it was possible to keep a fairly large learning rate value even with infrequent weight updates. Bias was also used in the experiments.

Experiments with adaptive learning rate during training and other similar methods to speed up the convergence are not included. This is mainly because it would have required a *much* larger number of experiments to make a fair comparison of the convergence for different weight update intervals.

3.1.1 Results on NETtalk

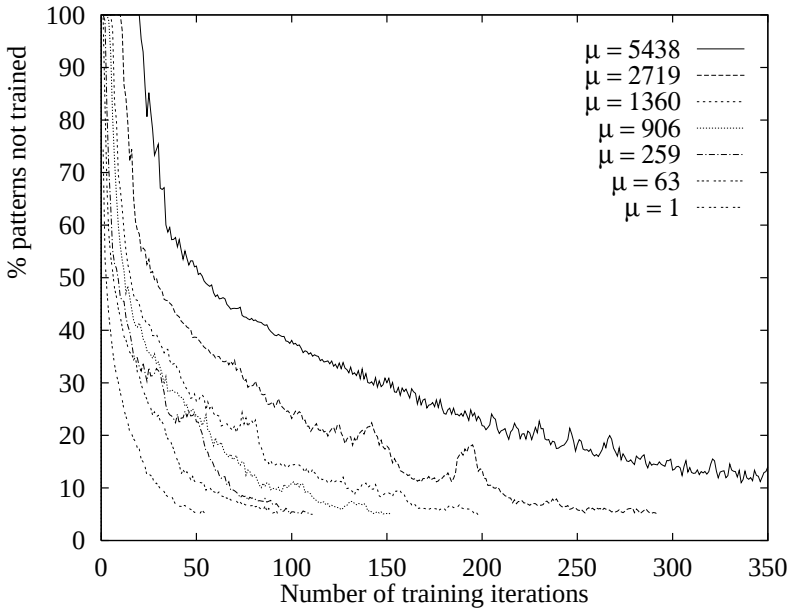

Figure 2: **Percentage of characters that are not trained for various weight update intervals.**

Figure 2 shows the results of the convergence as a percentage (called $E_{\%}$) of the characters that have not been trained. Each curve represents one weight update interval. Almost without exception, a larger μ implies a slower reduction in number of non-trained patterns. The percentage of characters not trained went down and stabilized slightly below 5%. This is due to the representation limitation of the network as mentioned in [12]. Therefore, a network is here regarded as converged, when $E_{\%} < 5\%$.

Figure 3 shows the best learning rate – the value resulting in the smallest number of iterations for convergence, for each of the μ used in the experiments. Based on these values the following rule is established

$$\eta(\mu) = 6.0\mu^{-0.5} \tag{16}$$

This is comparable to the rule derived by Paugam-Moisy in [2]: $\eta(\mu) = k_1\mu^{\alpha}$ for the application of classifying patterns into three classes – as described in the introduction.

The number of iterations required for convergence, N, is given in Figure 4. Due to the fairly linear distribution of the points, the least squares method is used to establish the experimental rule

$$N(\mu) = \frac{\mu}{11.5} + 76.4 \tag{17}$$

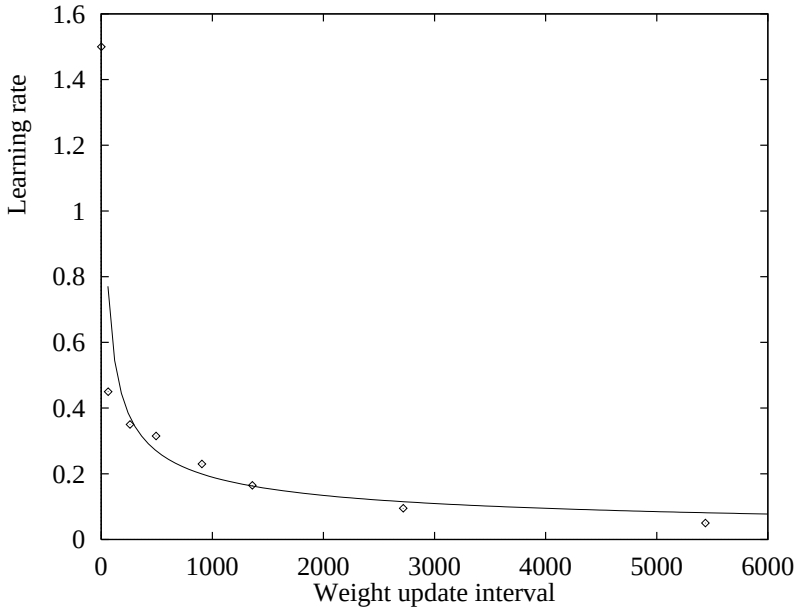

Figure 3: **The best learning rate (η) for each investigated weight update interval. The curve plots the derived rule.**

This linear relation between number of training iterations and the weight update interval was also found by Paugam-Moisy. The linear distribution of points tells us that for predicting $N(\mu)$ it is sufficient to estimate the total number of iterations for a few (minimum 2) weight update intervals. These can then be used to find a linear formula for $N(\mu)$.

Several researchers have found that N is proportional to $(1-\alpha)/\eta$. Figure 5 plots this expression for the learning rates in Figure 3. The points are linearly distributed along the least squares fit line. By scaling by the number of iteration for $\mu = 906$ we get

$$N(\mu) = \frac{\mu}{8.43} + 51.2 \tag{18}$$

This gives a good estimation for $\mu = 1$ with a number of 5 iterations less than measured. However, the estimates are less precise for larger weight update intervals, e.g. for $\mu = 5438$ a number of 85 iterations more than measured is estimated.

In the following it is shown that the total number of iterations needed can be estimated based on the error after a small number of training iterations. By looking at the curves in Figure 2, we see that they are of similar form, but have different curvature. To find a curve that estimates each error curve, logarithmic regression can be used to find an expression for each μ_i of the form

$$E_{\%}(n) = k_1 n^{k_2} \tag{19}$$

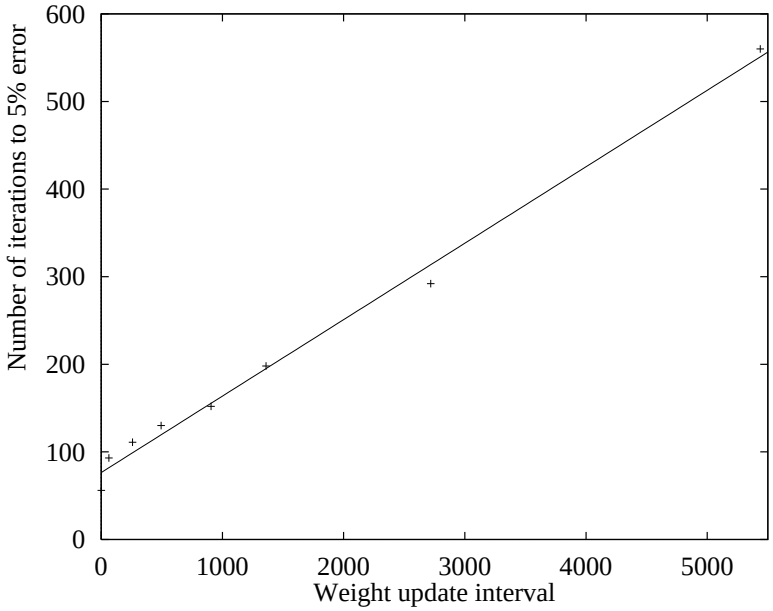

Figure 4: **Number of iterations needed to obtain convergence with $E_{\%} < 5\%$ as the stopping criteria.**

where n is the iteration index and k_1 and k_2 are parameters to be estimated. This requires that the points $(\log n_i, \log E_{\%}(n_i))\ \forall i$ are situated on a straight line. Figure 6 shows a logarithmic plot of Figure 2.

Above 20 iterations – see on the right-hand side of the vertical line in the figure, the curves are becoming more linear for increasing iterations. Thus, regression should be used on this linear part. Several different regression analyses were made to search for a curve describing the error convergence. Equation 19 was extended

$$E_{\%}(n) = k_1(n + k_3)^{k_2} \tag{20}$$

by k_3, which may be called the offset of the error curve, equal to twice[4] the number of iterations from start of training until $E_{\%} \leq 95\%$. Equation 20 was solved for each μ for the initial point $(n = 25, E_{\%}(25))$ and the convergence point (n when $E_{\%}$ becomes less than 5%, 5) with respect to k_1 and k_2. The points were given by the registered values during training – see Table 1.

The value of k_2 was found to be fairly constant for all the weight update intervals and averaged to approximately -1.25. Now k_1 can be computed based on *only* the error at $n = 25$ iterations

$$k_1 = \frac{E_{\%}(n)}{(n + k_3)^{-1.25}} \tag{21}$$

[4]Experiments showed a better approximation when using twice the number, compared to using the explicit number.

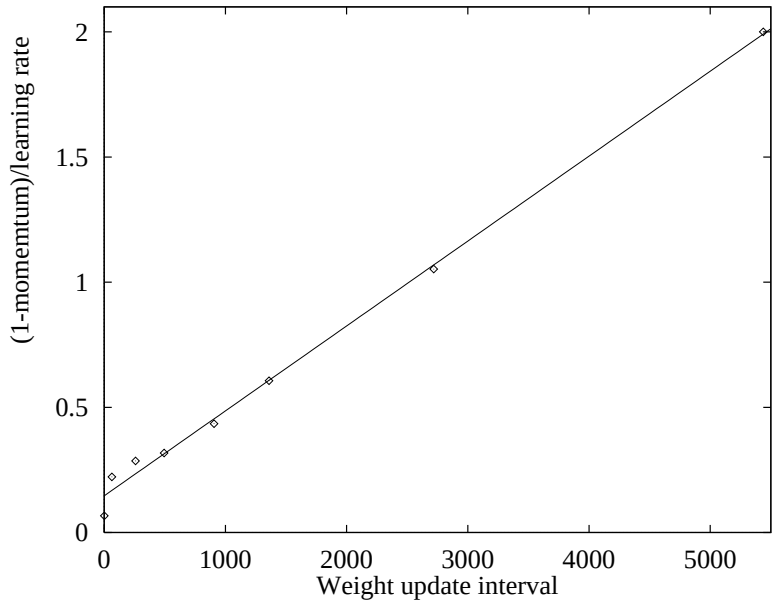

Figure 5: **Plotting of** $(1-\alpha)/\eta$**.**

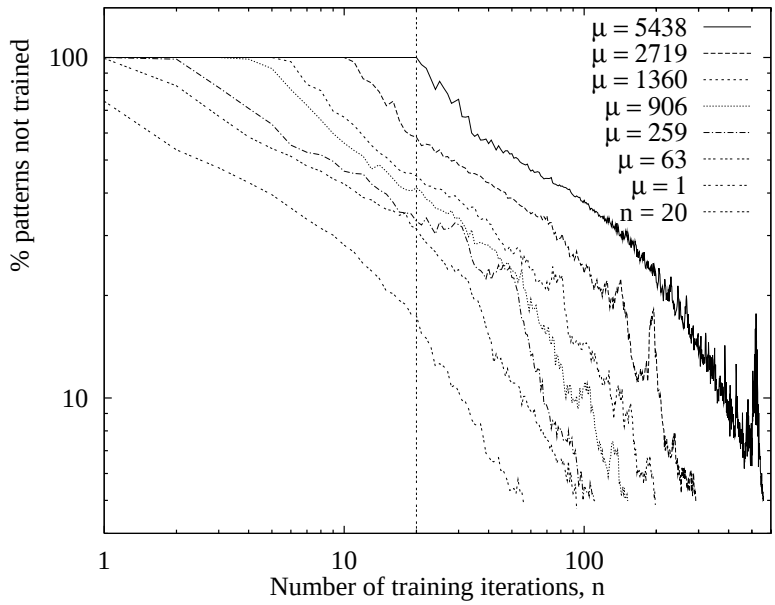

Figure 6: **Convergence of NETtalk, logarithmic plot. A vertical line is plotted for** $n = 20$**.**

Weight update interval (μ)	k_3	$E_{\%}(25)$	n when $E_{\%}(n) < 5$
1	0	12.61	56
63	0	25.65	93
259	4	32.86	111
494	6	32.38	130
906	8	35.77	152
1360	12	41.12	198
2719	22	53.51	292
5438	42	85.20	560

Table 1: **The data on NETtalk convergence used for estimation.**

Furthermore, the total number of iterations is given from Equation 20 by

$$N = \left(\frac{5}{k_1}\right)^{1/-1.25} - k_3 \tag{22}$$

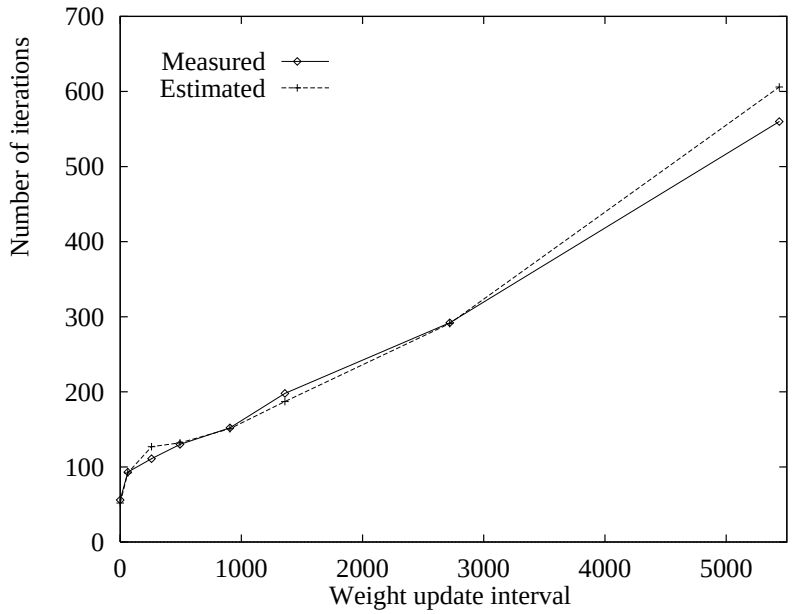

Figure 7: **Comparing measured and estimated values of $N(\mu)$.**

Figure 7, which shows both measured and estimated values of $N(\mu)$, proves the accuracy of the estimation model for this application. However in general this accuracy may not be the case. This is a field of research where very little published work using real applications exists. Thus, more studies are required to form a more general framework and to see if it is possible to estimate the convergence in general.

3.1.2 Sensitivity and Convergence Aspects

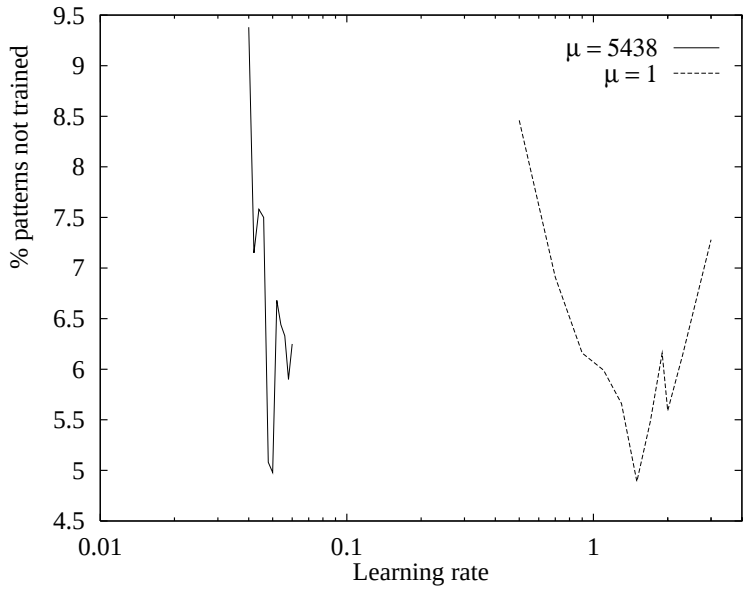

Figure 8: **The sensitivity of the learning rate (η) on the convergence for epoch versus pattern learning.**

The BP algorithm is more sensitive to the selection of the learning rate when the weights are infrequently updated. Figure 8 shows the convergence for $\mu = 1$ (*learning by pattern*) and $\mu = 5438$ (*learning by epoch*). The error $E_{\%}$ is plotted and the minimum error corresponds to the learning rate which gives the quickest convergence – i.e. $E_{\%}$ is less than 5%. The x-axis is logarithmic, and still the epoch learning has a narrower valley in the error curve.

In the training scheme used, a pattern is said to be trained if the error is below a certain threshold. Since the weights are changed during an iteration, there is a risk of trained patterns becoming de-trained, i.e. have an error larger than the threshold after the training iteration. Thus, the number of patterns trained correctly may be less than the number counted during an iteration. Figure 9 compares the $E_{\%}$ accumulated *during* each training iteration against $E_{\%}$ computed for the whole training set *after* each training iteration for $\mu = 906$. Even though the epoch computed error is less stable, the two convergence criteria harmonize well with little difference in the final convergence. Similar results were recorded for other values of μ.

Figure 10 shows the convergence when $E_{\%} < 10\%$. 31 iterations are required before convergence using *learning by pattern*. In the NETtalk paper [12], the training continued for 30 iterations. As for the $E_{\%} < 5\%$ convergence criteria, there is a linear relation between the weight update interval and the number of iterations required before convergence.

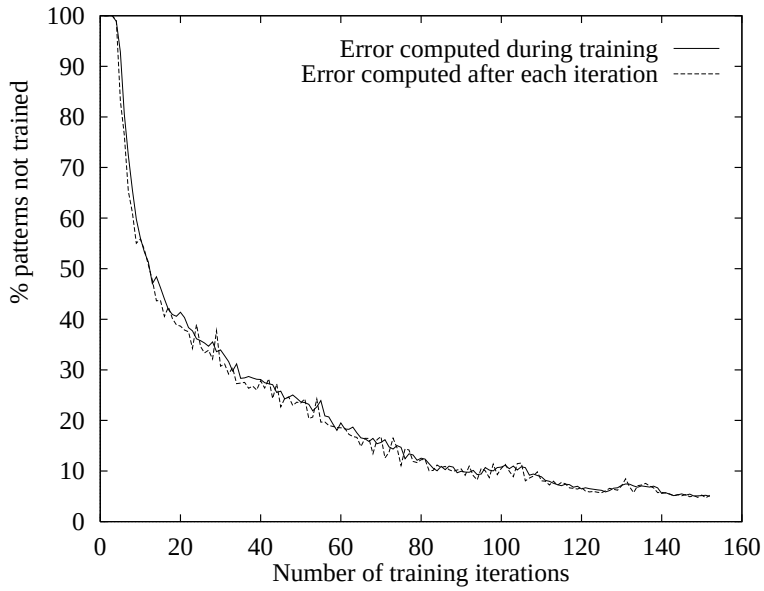

Figure 9: **Error computed during training compared to error computed after each iteration (epoch) for $\mu = 906$.**

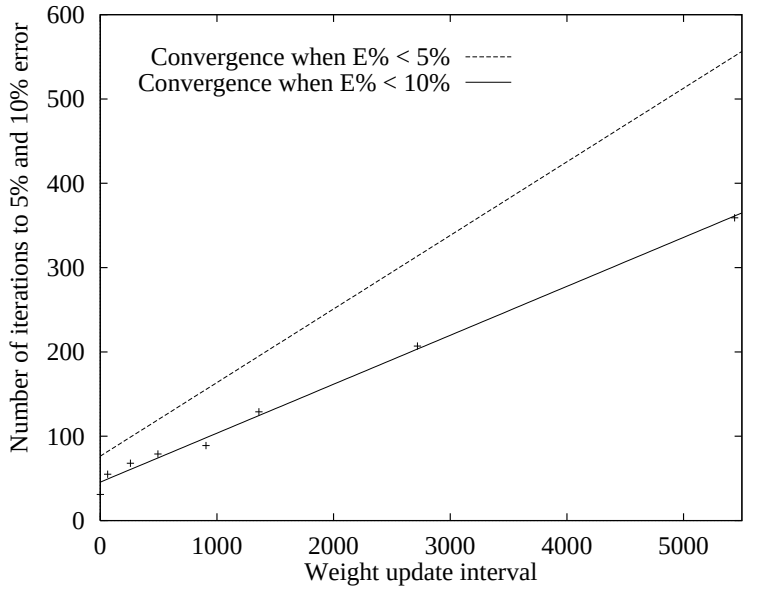

Figure 10: **Comparing $E_{\%} < 5\%$ and $E_{\%} < 10\%$ error criteria.**

An interesting aspect is the ratio

$$k_e = \frac{N(P)}{N(1)}, \tag{23}$$

where P is the number of patterns in the training set. Then, $N(P)$ represents the number of iterations for *learning by epoch*. For NETtalk training with 5% error threshold

$$k_e = \frac{560}{56} = 10 \tag{24}$$

Thus, *learning by epoch* is a factor of 10 slower than *learning by pattern*. For a 10% error threshold it becomes

$$k_e = \frac{359}{31} = 11.58 \tag{25}$$

For the least strict threshold (10%), there is a larger difference in number of iterations for *learning by pattern* compared to *learning by epoch*.

3.2 Sonar Target Classification

In this section, the results on the convergence of the sonar return application are given. In the experiments, all the available patterns (208) have been used as a training set. The weights are updated – or accumulated for *learning by block*, for a pattern where the error is larger than 0.04. That is

$$E_p(n) > \frac{1}{2}\sum_{k=1}^{2}(0.2)^2 = 0.04$$

This threshold is based on an error of 0.2 on each of the output units, which was used by Gorman et al. [13], [14]. As in their work, the smoothing term was set to 0. The number of hidden units was set to 24, which was reported as giving the best recognition rate on the training set.

3.2.1 Results on Sonar Target Classification

The convergence for different weight update intervals is given in Figure 11. The y-axis represents the percentage of training patterns with $E_p > 0.04$. The results are very similar to NETtalk – a larger weight update interval leads to slower learning. However, the convergence curves are steeper and it is not possible to use the same logarithmic regression as for NETtalk to estimate the number of training iterations.

The convergence was very sensitive to small changes in initial weights and other parameters. It was seen that the resolution of floating point variables influenced the convergence rate. Continuous values in the input units, instead of two-level values as in the case of NETtalk, probably make the algorithm more sensitive to the accuracy of the variables.

The best learning rate for each weight update interval is plotted in Figure 12. A similar experimental rule to Equation 16 is established

$$\eta(\mu) = 3.0\mu^{-0.5} \tag{26}$$

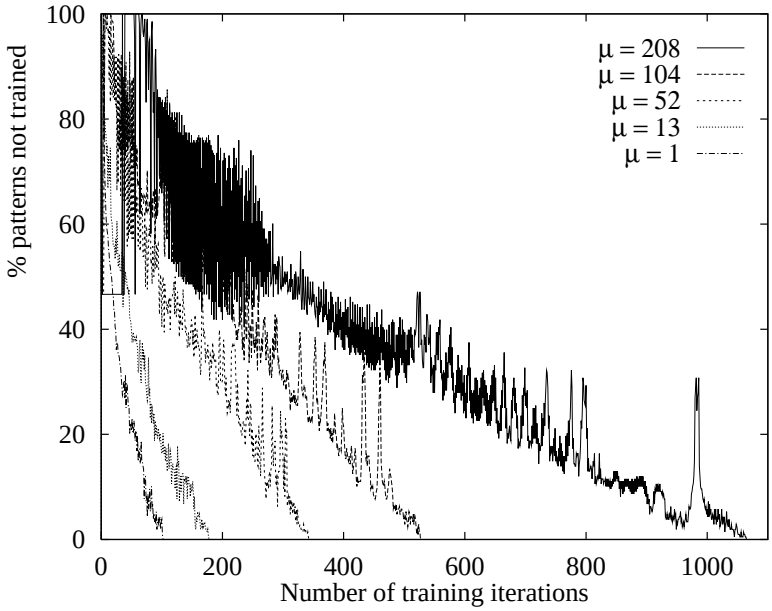

Figure 11: **The percentage of patterns that have $E_p > 0.1$, for learning sonar return classification.**

The total number of iterations until *all* training patterns have reached $E_p \leq 0.04$ during a learning cycle is plotted in Figure 13 together with the derived least squares fit of the points

$$N(\mu) = 106.66 + 4.51\mu \tag{27}$$

The number of iterations required for an error threshold equal to 5% not trained patterns is plotted in Figure 14. This is equivalent to the error threshold used for NETtalk. The least squares error equation is given by

$$N(\mu) = 97.04 + 3.88\mu \tag{28}$$

3.3 Comparing Sonar Results to NETtalk Results

In this section, common properties of the two applications are highlighted. The idea is to search for a general way of estimating the number of iterations needed for convergence, N, as a function of the weight update interval.

For NETtalk, the learning ratio was found to be $k_e = 10$ for a 5% error threshold and $k_e = 11.58$ for a 10% error threshold. Correspondingly for the sonar target classification (all of the patterns trained) we have

$$k_e = \frac{1065}{102} = 10.44 \tag{29}$$

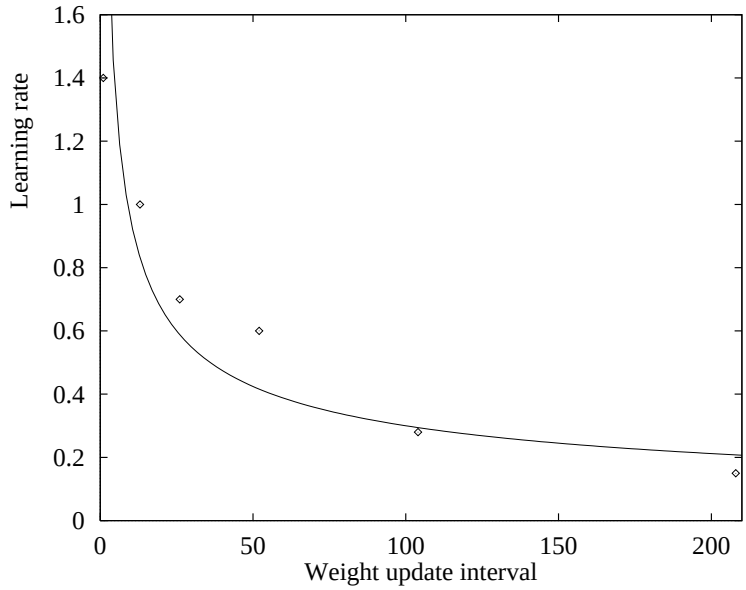

Figure 12: **The best learning rate η for each of the investigated weight update intervals. The curve plots the derived rule.**

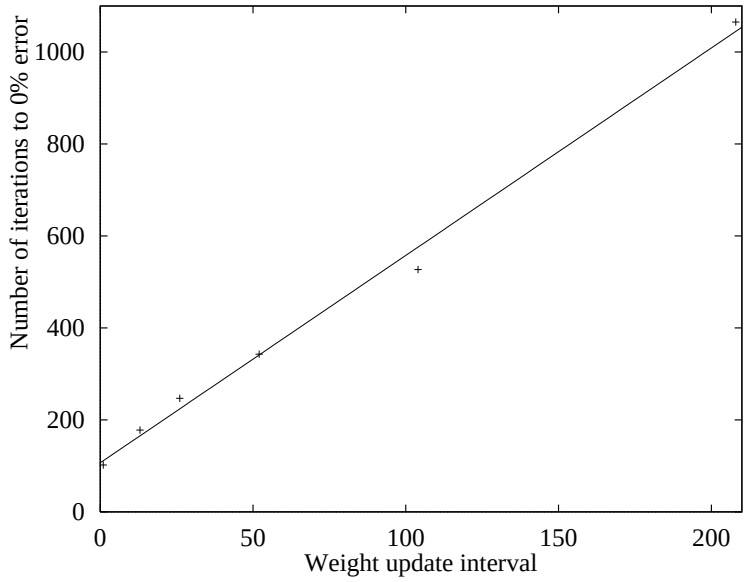

Figure 13: **Number of iterations required to obtain convergence, i.e. $E_p \leq 0.04$ for all training patterns.**

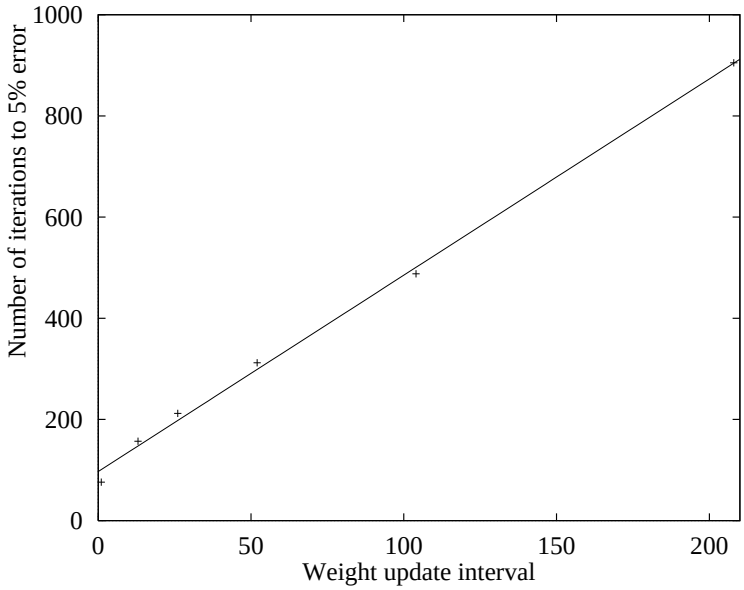

Figure 14: **Number of iterations needed to reach a stopping criteria of 5% error.**

For the less strict stopping criteria of 5%, $k_e = 11.9$. Thus, k_e is in the 10-12 range for both applications. Thus, *learning by pattern* trains 10-12 times faster than *learning by epoch*. Moreover, a less strict error threshold increases k_e. The resemblance between these two applications was not expected, since they are different in both network size and number of training patterns.

This result, of a near constant k_e ratio, will now be used in estimating $N(\mu)$ based on a single result for convergence, $N(1)$. The relation between μ and the k-ratio is illustrated in Figure 15. The straight line is assumed according to the earlier shown linear relation between the weight update interval, μ and $N(\mu)$. The equation for the line passing through the points (1,1) – *learning by pattern* and (P, k_e) – *learning by epoch* is given by

$$k_\mu - 1 = \frac{k_e - 1}{P - 1}(\mu - P) \Leftrightarrow k_\mu = \frac{k_e - 1}{P - 1}(\mu - P) + 1 \tag{30}$$

where k_μ also can be understood as $N(\mu)/N(1)$. An estimate of the $N(\mu)$ is then given by

$$N(\mu) = k_\mu N(1) \tag{31}$$

For neural training, the application is usually trained several times with different parameters. Therefore, Equation 31 can be used for the approximation of $N(\mu)$, after some initial convergence tests. To confirm the estimation method, other applications should be tested.

To set the appropriate learning rate, the previous experimental rules (Equation 16 and 28) can be formulated in a general approximation rule

$$\eta(\mu) = k\mu^{-0.5} \tag{32}$$

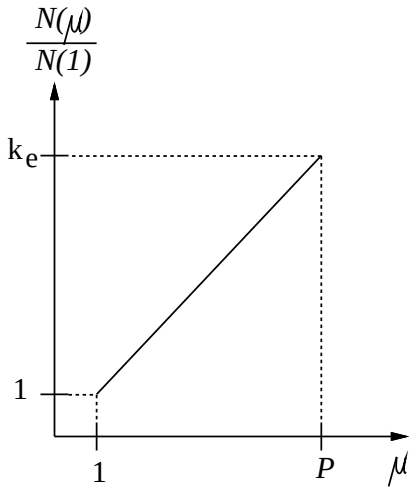

Figure 15: **The relation between μ and k_e, where P is the total number of patterns in the training set.**

where k is a positive constant less than 10. As the training set gets smaller, the appropriate k value decreases. This is in accordance to the two experiments reported in this paper.

4 Conclusions

The convergence tests of two applications in this chapter indicate a linear relation between the weight update interval and the number of iterations to reach convergence. *Learning by epoch* is a factor of 10-12 times slower than *learning by pattern*. Several sensitivity aspects on convergence have been tested. The learning rate value highly influences the convergence, especially for infrequent weight updates. For the two applications studied there exists several similarities regarding convergence. It was possible to find common rules describing the convergence behavior. However, more applications should be tested for evaluating them.

References

[1] D.E. Rumelhart, G.E. Hinton, and R.J. Williams, "Learning internal representation by error propagation," in *Parallel Distributed Processing*, vol. 1, pp. 318–362. The MIT Press, 1986.

[2] Helene Paugam-Moisy, "Parallel neural computing based on neural network duplicating," in *Parallel algorithms for digital image processing, computer vision and neural networks*, Ioannis Pitas, Ed., chapter 10, pp. 305–340. John Wiley & Sons, 1993.

[3] Philip D. Wasserman, *Neural Computing – Theory and Practice*, Van Nostrand Reinhold, 1989.

[4] Jim Torresen, Shin-ichiro Mori, Hiroshi Nakashima, Shinji Tomita, and Olav Landsverk, "Parallel back propagation training algorithm for MIMD computer with 2D-torus network," in *Proceedings of International Conference On Neural Information Processing (ICONIP'94), Seoul, Korea*, October 1994, vol. 1, pp. 140–145.

[5] Jim Torresen, Hiroshi Nakashima, Shinji Tomita, and Olav Landsverk, "General mapping of feed-forward neural networks onto an MIMD computer," in *Proc. of IEEE Int. Conference on Neural Networks (ICNN'95)*, Perth, Western Australia, 27 November – 1 December 1995, IEEE.

[6] Jim Tørresen, *Parallelization of Backpropagation Training for Feed-Forward Neural Networks*, Ph.D. thesis, Norwegian University of Science and Technology, 1996, ISBN 82-7119-906-4.

[7] Akira Sato, "An analytical study of the momentum term in a backpropagation algorithm. In Proc. of ICANN–91," in *Artificial Neural Networks*, T. Kohonen et al., Eds., vol. 1, pp. 617–622. Elsevier Science Publishers B.V. (North-Holland), 1991.

[8] Jim Torresen, Shinji Tomita, and Olav Landsverk, "The relation of weight update frequency to convergence of BP," in *Proc. of World Congress on Neural Networks (WCNN'95)*, Washington, D.C., July 1995, vol. 1, pp. 679–682, INNS Press.

[9] Michael Witbrock and Marco Zagha, "An implementation of backpropagation learning on GF11, a large SIMD parallel computer," *Parallel computing*, vol. 14, pp. 329–346, 1990.

[10] S.E. Fahlman, "Faster-learning variations on back-propagation," in *Proc. of the 1988 Connectionist Models Summer School*. 1988, Carnegie-Mellom University.

[11] Martin Møller, "Supervised learning on large redundant training sets," *Int. Journal of Neural Systems*, vol. 4, no. 1, pp. 15–25, 1993, World Scientific Publishing Company.

[12] Terrence J. Sejnowski and Charles R. Rosenberg, "Parallel networks that learn to pronounce English text," *Complex Systems*, vol. 1, pp. 145–168, 1987.

[13] R. Paul Gorman and Terrence J. Sejnowski, "Analysis if hidden units in a layered network trained to classify sonar targets," *Neural networks*, vol. 1, pp. 75–89, 1988.

[14] R. Paul Gorman and Terrence J. Sejnowski, "Learned classification of sonar targets using a massively parallel network," *IEEE Trans. on acoustics, speech and signal processing*, vol. 36, no. 7, pp. 1135–1140, 1988.