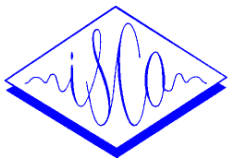

First European Conference on
Speech Communication and Technology
EUROSPEECH '89
Paris, France, September 27-29, 1989

Fast Back-Propagation Learning Methods for Large Phonemic Neural Networks

P.Haffner*, A.Waibel**, H.Sawai and K.Shikano

ATR Interpreting Telephony Research Laboratories, Seika-cho, Soraku-gun, Kyoto, 610-02, Japan
*Centre National d'Etudes des Télécommunications, 22301 Lannion, France
**Carnegie Mellon University, Computer Science Dept, Pittsburgh, PA 15206, USA

Several improvements in the Back-Propagation procedure are proposed to increase training speed, and we discuss their limitations with respect to generalization performance. The error surface is modeled to avoid local minima and flat areas. The synaptic weights are updated as often as possible. Both the step size and the momentum are dynamically scaled to the largest possible values that do not result in overshooting. Training for the speaker-dependent recognition of the phonemes /b/, /d/ and /g/ has been reduced from 2 days to 1 minute on an Alliant parallel computer, delivering the same 98.6% recognition performance. With a 55000-connection TDNN, the same algorithm needs 1 hour and 5000 training tokens to recognize the 18 Japanese consonants with 96.7% correct.

1. Introduction

Recently, the advent of new learning procedures and the availability of fast supercomputers have made it possible to use connectionist architectures to recognize "real world" patterns. For instance, very high performance has been obtained with Time Delay Neural Networks (TDNN), which are trained using Back-Propagation(1)(2). However, these high results were attained at the expense of training speed, which could be an obstacle for learning a large database. A scaling problem emerges and it is useful to evaluate learning time as a function of the dimensions of the task and the network. If we denote the number of connections w, the number of training samples m and the average number of times one has to present each training sample to have it learnt properly p, then the learning time is $t=p.w.m$(in number of connections which are processed). In our problem, m is given by the size of the database available, the order of magnitude of w is related to m (6) and p should be a function of m and w. The main criticism which has been made against the Back-Propagation procedure is the large size p may reach. If one uses the original Back-Propagation algorithm, which uses a very small gradient step size and only updates the weights after presentation of the whole training set, p may be larger than 10,000. The solution seems obvious: for instance increase the step size, but one has to be careful about oscillations and overshooting. Moreover, slow and smooth learning seems to yield better generalization performance. In the present paper, several improvements in the Back-Propagation learning procedure are proposed to increase training speed. Based on empirical results, we then discuss their limitations, particularly with respect to generalization performance. Finally, we show that with this new learning procedure, a TDNN is able to learn the Japanese consonants within 1 hour.

2. Phoneme recogntion using TDNN

We present here the network architecture and the database, which are exactly the same as in (1) and (2).

A TDNN is a Multi-Layer Perceptron (MLP) including a time dimension: two physical units are connected through several weights, each one corresponding to a different delay in time. It is possible to transform this time dimension into a spatial dimension. For instance, in figure 1, each physical unit of Hidden Layer 1 is represented at each of the 13 instants in time by one virtual unit, with its specific activation. All the virtual units that stand for the same physical unit have the same incoming weights. Therefore, a TDNN may be regarded as a MLP with some connection weights constrained to be equal.

We used a large vocabulary database of 5240 Japanese words. These words are uttered in isolation by one Japanese professional announcer. Melscale coefficients were computed from the power spectrum at an overall 10msec frame rate. The database was split into a training and a testing set, from which the actual phonetic tokens were extracted. The training tokens were randomized within each phoneme class. The training set was then built by alternating each class to be learnt.

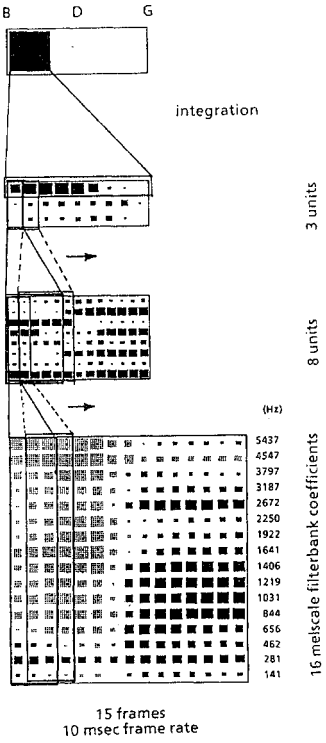

Figure 1: A 3-class TDNN for the BDG task.

3. Methodology

The *trial* (learning run) involves several *epochs* (presentations of the whole training set), and is traced with the output units *Mean Square Error(MSE)*. An *iteration* is the presentation of only one sample. After training, the network is evaluated on the testing set. A pattern is recognized when the output unit with the maximum actual activation corresponds to the unit with the desired activation equal to 1. The recognition rate is defined as the percentage of samples correctly classified. The error rate is therefore the complement of the recognition rate with respect to 100.

To make valid statistics, we generally perform several trials on the same task, with the same algorithm, but with different initial weights. We call a trial *converging* if it yields an error rate of less than 2% on training data. We rate our recognition performance with two numbers: the percentage of converging trials and the error rate on test data averaged over the converging trials.

For our simulations, we have used an Alliant computer, with 8 vector processors, which computed 2 Million Connections Per Second (MCPS).

The task we have been using as a benchmark is the recognition of the 3 consonants /b/, /d/ and /g/. The dimensions of the BDG task appear in table 2, we have been able to reduce learning time from 2 days (epoch updating, small fixed step size, 0.9 momentum) to 1 minute. We have also tried our methods on the XOR and encoder problems and were able to reduce learning time to a few dozen iterations.

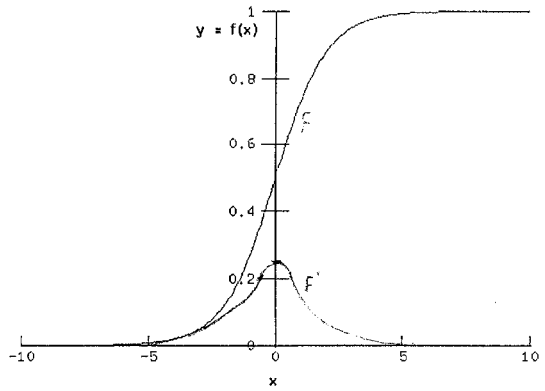

Figure 2: The Sigmoid Function f(x) = 1/(1 + e^{-x}) and its derivative f'.

Epochs	10	20	30	40	50
Standard Sigmoid	2.5 /50	1.9 /70	1.7 /100	1.7 /100	1.6 /100
Sigmoid 1	2.4 /50	2.1 /90	2.1 /90	2.1 /100	1.8 /100
Sigmoid 2 1=0.01	2.4 /90	1.9 /100	1.9 /100	1.9 /100	1.9 /100
Sigmoid 3 1=0.01	2.1 /100	2.0 /100	1.8 /100	2.0 /100	2.0 /100
New Error	2.4 /90	2.4 /100	2.1 /100	2.1 /100	2.1/100

Table 1: BDG task, Error Rate / % converging trials (10 trials, Error averaged over converging trials).

4. Modeling the error surface

We examine first the effects of modifying the sigmoid function or the output error in order to get a steeper error surface.

The presentation of a pattern modifies the weight connecting units i to j through $f(x_i)f'(x_j)\partial E/\partial y_j$. The Back-Propagation learning rate is then proportional to the values of the sigmoid function and its derivative f'. But these functions flatten out at infinity, as seen in figure 2. There are several ways to make these functions non zero at infinity.

1 - Use a symmetric sigmoid whose value is never zero at infinity, by substracting 0.5 from the sigmoid. This generally gives a slightly better learning speed, as shown in (3). However, at the beginning of the learning phase, when the weights are small and the activations close to zero, learning may be slow to initiate.

2 - Add a linear function to the sigmoid: $f_1(x)=f(x) + 1.x$. The derivative becomes $f'_1(.x)=f'(x) + 1$, and therefore cannot be zero (1 is generally between 0.01 and 0.1).

3 - Only add a small constant 1 to the sigmoid derivative, without changing the sigmoid function.Therefore, during the backward phase, we multiply $\partial E/\partial y$ by a factor that is no longer the real sigmoid derivative.

These three methods, especially those dealing with a change in the sigmoid derivative, generally lead to an increase in speed and guarantee convergence to a zero error global minimum The results we found are consistent with (4). However, this forced convergence is most often undesired, for it leads to network configurations which have overlearnt the training set, yielding slightly worse performance on test data. We see in table 1 that for the BDG task, models 2 and 3 allow for fast convergence in all 10 trials, but the final average error rate on test data is higher than the standard sigmoid.

Because of the multiplication by the sigmoid derivative, output units whose activations are close to 0 or 1 have delta values which are close to 0, regardless of the size of the actual output error for these units. To cope with this problem, instead of the standard Mean Square Error, a new error has been proposed by McClelland:

$$E = - \Sigma_{samples} \Sigma_j \ln(1 - (y_j - d_j)^2)$$

(y_j is the actual output and d_j is the desired output)

The derivative of this new error is maximum when $y_j - d_j = 1$. This new error ensures that the system will always learn and has roughly the same effect as adding a small constant to the sigmoid derivative, as we see in table 1. Despite a slightly worse generalization performance, we have found this new error to be particularly useful with large tasks involving many classes, such as learning the 18 consonants.Here, over 18 consecutive iterations, the desired activation of any output unit is equal to *1.0* during 1 iteration and *0.0* during the 17 others. A uniform *0.0* output activation for the 18 samples is an easy local minima for the network to learn. The sample which should output *1.0* is not correctly learnt, but no correction occurs, as the sigmoid derivative is equal to *0.0*. The new error prevents this problem and increases learning speed by several orders of magnitude.

5. Learning strategy

Learning strategy describes the parts of our learning algorithm that deal with the way the training samples are presented to the network. For instance, how often should we update weights and when can we skip samples? When compared to learning algorithms that update the weights at each epoch, those which update the weights after a small number of iterations are much faster (in the BDG task, by a factor of 10) and yield better generalization. However, to update the weights after each iteration does not seem useful and is costly to compute, especially at the end of the learning phase, when one has to sum over many iterations to get a significant delta value. For our 3-class BDG task, we have used the following procedure: update the weights every 3 iterations during the first epoch and then increment the updating period by 3 at each new epoch.

It is also very useful to skip samples that are correctly learnt. We set a minimum error E_0: the sample m should be skipped during a number of epochs which is proportional to $E_0 - E_m$.

6. Dynamic scaling of the learning parameters

Finding an optimal value for the step size ε is a key problem with the Back-Propagation learning rule. Most methods which have been proposed to solve this problem generally deal with line search or Newton algorithms(4)(5). For our tasks, they bring two kind of problems. First, they generally assume that the weights are always updated after presentation of the same set of samples (epoch updating). Second, they allow large moves in the weight space, they do not try to find the solution with the smallest possible weights as it happens with very slow learning. Therefore, they may yield a poorer generalization performance(6).

We have however introduced a control that allows the algorithm to use a reasonably large constant step size with no danger of overshooting or sudden changes in the weights. This control is local to each unit i, whose activation is calculated through its incoming weights:

$$x_i = f(\Sigma_j W_{ij} x_j)$$

We resize the step size for unit i:

$$\varepsilon_i = \frac{\varepsilon}{1 + \frac{\varepsilon}{\omega}\sqrt{\Sigma_j (\partial E/\partial W_{ij})^2}}$$

It ensures that:

$$\varepsilon_i \sqrt{\Sigma_j (\partial E/\partial W_{ij})^2} < \omega$$

ω=1 prevents most overshooting. With this method, we avoid brutal changes in the weights without having to use a very small step size.

We have also developed similar methods to reduce the momentum from its maximum value of 0.99 when the weights or the weight variations grow too large.

7. Experiments on consonant recognition

Our first experiments with all-consonant recognition used an architecture derived from the 3-class architecture described in figure 1, but with 18 classes, as shown in Fig.3. We have selected from above the improvements in the learning procedure which were robust, fast and kept good generalization performance on test data. The standard sigmoid and McClelland's new error are used. The weights are updated with a period which is incremented with time, from 9 iterations to 72, with an increment of 3. The step size has a maximum value of 0.001 and the momentum is dynamically scaled between 0.5 and 0.99. The dimensions of the task are given in table 2. We have tried 6 different networks, corresponding to different numbers of units in the first hidden layer. Our recognition results after 30 epochs are shown in table 3. The networks with 36 to 72 hidden units are able to learn the training set nearly perfectly. Generalization performance does not seem to decrease as the size and the complexity of the network increase. Fig.4 shows that the average size of the weights is smaller in large networks.

We have found, a posteriori, our results to be consistent with the relation proposed by Haussler(6). It gives a theoretical upper bound on the size of the training set for MLPs with a continuous sigmoid function: to be practically certain to get a given generalization performance, one has to feed the network a number of training samples which is proportional to $W(Log(w_{max})+b)$, where W is the number of weights and w_{max} the maximum weight. We have indeed found that the quantity $N(Log(\sqrt{<w^2>})+b)$ is constant for networks that yield approximately the same generalization performance. N is the number of units in hidden layer 1, which ranges here from 27 to 72 (the number of weights W= 1500N). $\sqrt{<w^2>}$ is the root mean square of the weights, and not the maximum value. This relation was however the best one we could get from the data given by Fig.4. The fact that the bound proposed by Haussler seems to generalize for TDNNs is very encouraging, and it sould be possible to control numerically the generalization capacity of a network.

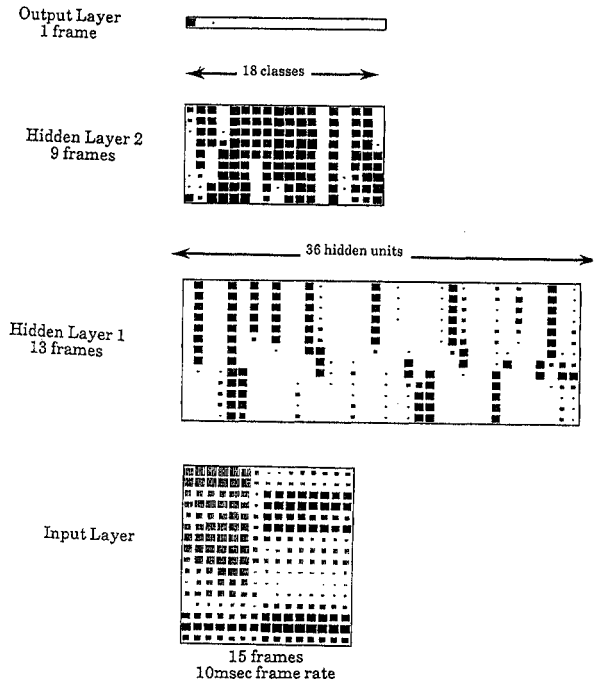

Figure 3: A non modular consonant TDNN with 36 units in Hidden Layer 1

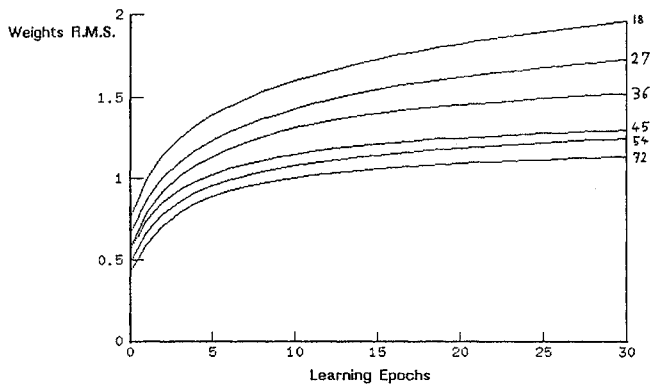

Figure 4:Root Mean Square Weight Vs. Learning Epochs for Consonant TDNNs with 18 to 72 Hidden Units.

The previous architecture does not include any knowledge about phonetics. It is possible to sort the 18 consonants into 6 classes: the modular TDNN(2) is built with 7 different groups of hidden units, 6 to discriminate within each class, and the last one for inter-class discrimination. As shown in table 2, our algorithm only needs 1 hour and 5063 samples to achieve a 96.7% recognition rate on test data. Modular architecture seems to perform slightly better than the non-modular architecture whose best performance is 96.3%. To check the influence of the training set size, the network has also been trained on a smaller set of 3114 samples. Recognition is only 95.3%.

Task	BDG	Consonants	Modular Consonants	
Phoneme classes	3:b d g	18 consonants	18: b d g / p t k / m n N s sh h z / ch ts / r w y	
Units in 1st hidden Layer	8	N	68	
Number of Connections	6233	1500 N	55757	
Number of Free Parameters	521	120 N	4730	
Learning Time	1 minute	1 hour	1 hour	1 hour
Training Set Size	780	5063	3114	5063
Recognition on Training Set	99.4 %	See below	99.4 %	99.2 %
Testing Set Size	760	3061	3061	3061
Recognition on Test Set	98.6 %	See below	95.3 %	96.7 %

Table 2: Dimensions of the different tasks.

Number of units	18	27	36	45	54	72
Training set	98.7	99.5	99.7	99.8	99.7	99.9
Testing set	94.4	94.9	96.3	95.4	96.2	95.6

Table 3: % Recognition on the training and testing sets as a function of the number of units in Hidden Layer 1.

Conclusion

We have shown in this paper that learning speed for the Back-Propagation procedure could be reduced by several orders of magnitude, thanks to improvements in the modeling of the error surface, the learning strategy and the scaling of the learning parameters. In our experiments, training patterns only require 10 to 20 presentations. Learning reasonably large phonemic databases is in the reach of computers currently available. Further improvements in learning speed can be achieved by modular learning: training separately different parts of the network separately.

The networks learns using examples which are randomly drawn from some unknown distribution, it is therefore important to guarantee good generalization performance on open data. This can be achieved by controlling the size of the weights. Learning appears therefore as: do use small initial weights and increase progressively the weights until the solution with the smallest possible weights is found; don't jump as quickly as possible at the first good solution.

(1) A.Waibel, T.Hanazawa, G.Hinton, K.Shikano and K.Lang: "Phoneme Recognition Using Time Delay Neural Networks". *IEEE Trans. On Acoustics, Speech and Signal Processing, Vol 37 n° 3, March 1989.*

(2) A.Waibel, H.Sawai and K.Shikano: "Modularity and Scaling in Large Phonemic Neural Networks". *Proceedings of the IEEE ICASSP, may 1989.*

(3) W.S.Stometta and B.A.Huberman: "An improved Three Layer Back-Propagation Algorithm". In *Proceedings of the IEEE International Conference on Neural Networks,* pages 619-627. San Diego, CA, 1987.

(4) S.E.Fahlman: "An Empirical Study of Learning Speed in Back-Propagation Networks". *Technical report CMU-CS-88-162, Carnegie Mellon University, June 1988.*

(5) R.L.Watrous: "Learning Algorithms for Connectionist Networks: Applied Gradient Methods for Non-Linear Optimization". In *Proceedings of the IEEE International Conference on Neural Networks,* pages 619-627. San Diego, CA, 1987.

(6) D.Haussler: "Generalizing the PAC Model: Sample Size Bounds From Metric Dimension-based Uniform Convergence Results". *COLT '89, April 1989.*