专 题 汇 编

IJCNN' 有导师学习

北京邮电学院图书馆

②

12763

Lateral Inhibition Neural Networks for Classification of Simulated Radar Imagery

Charles M. Bachmann, Scott A. Musman, and Abraham Schultz
Code 5362, Airborne Radar Branch, Radar Division, Naval Research Laboratory
4555 Overlook Ave., SW
Washington, D.C. 20375-5000

January 8, 1992

Abstract

We investigate the use of neural networks for the classification of *simulated* inverse synthetic aperature radar (ISAR) imagery. Certain symmetries of the artificial imagery make the use of localized moments a convenient preprocessing tool for the inputs to a neural network. A database of simulated targets was obtained by warping dynamical models to representative angles and generating images with differing target motions. Ordinary backward propagation (bp) and some variants of bp which incorporate lateral inhibition obtain a generalization rate of up to $\sim$ 78% for novel data not used during training, a rate which is comparable to the level of classification accuracy that trained human observors obtained from the unprocessed simulated imagery.

1 A Simulated ISAR Database

Our database consisted of *simulated* inverse synthetic aperature radar (ISAR) images of ships. Real imagery is a Doppler vs. range profile produced by the ship's motion. Mathematically, the image shape depends on the cross-product of the line of sight vector with the instantaneous angular velocity vector of the ship. The resulting image exhibits the combined effects of the independent motions of the ship due to its roll, pitch, and yaw. Our artificial database contains target silhouettes which simulate the rudimentary shape of a ship in an image at different aspect angles and with varying degrees of roll, pitch, and yaw.

Although real imagery also has spectral characteristics, for the purposes of identifying the Perceptual Class of a ship, general shape information of a silhouette is usually sufficient. By "Perceptual Class" we mean the categories "Commercial/Auxilliary", "Combatant", "Landing Platform", "Submarine", and "Small Craft." For the purposes of this pilot study, we were interested in the determination of Perceptual Class. Therefore, we chose to simplify the problem by creating simulated images which all had binary image intensity levels.

We actually investigated a subset of the Perceptual Class problem: the ability of backward propagation and variants of bp incorporating lateral inhibition to distinguish the difference between commercial/auxilliary ships and combatants in the artificial database. For training, simulated images were presented to represent a variety of different target aspect angles and motion parameters. Each of these frames was presented once with the bow left and right. In figure 1, we show some examples of the kinds of image frames which we used.

The actual input to the network was the weighted rms image variation about the image center of mass in each range bin:

$$input_i = \sqrt{\frac{\sum_j (j - E_j(i))^2 \rho_{ij}}{\sum_i \sum_j (j - E_j(i))^2 \rho_{ij}}} \tag{1}$$

where ρ_{ij} is the intensity of the image in range-doppler cell i,j, and $E_i(j) = \sum_j j\rho_{ij}$ is the image center-of-mass in range-bin i. Figure 2 shows a sample image frame and the corresponding range-bin moments used as input. Global 2D invariant moments are of somewhat limited utility for recognition tasks such as this, which require more detailed shape information (Park and Sklansky, 1990). Therefore, we use local range-bin

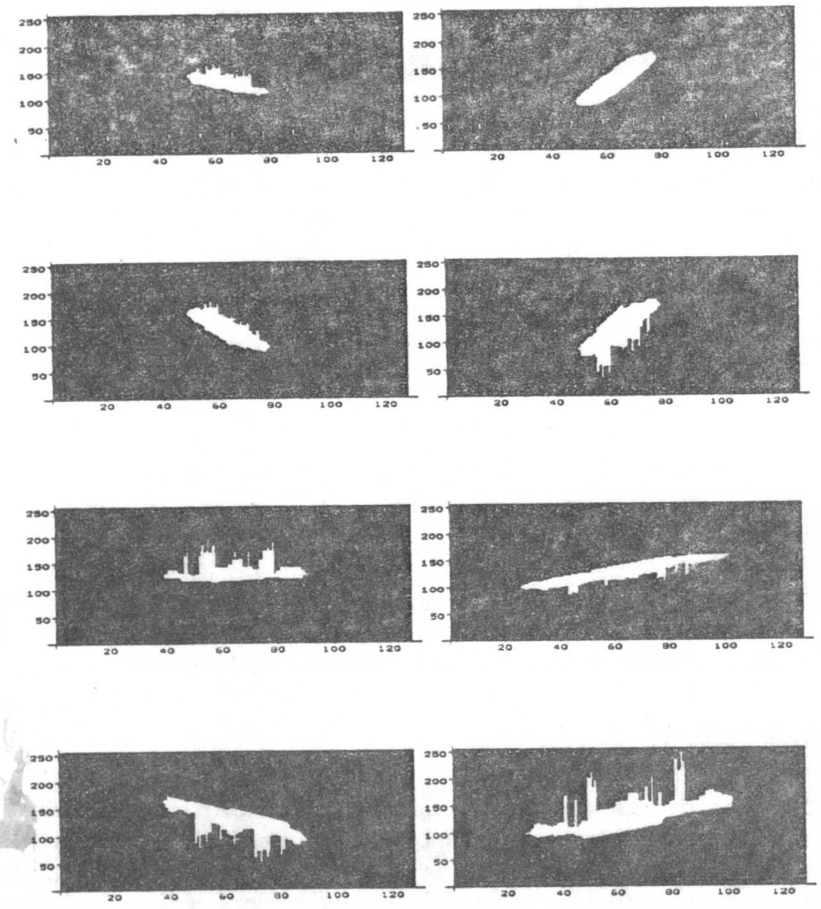


Figure 1: **Examples of simulated binary ISAR images of a commercial/auxilliary ship. The same ship is displayed at a variety of different aspect angles, and with varying degrees of profile and plan component, which model the amount of roll, pitch and yaw present in the image.**

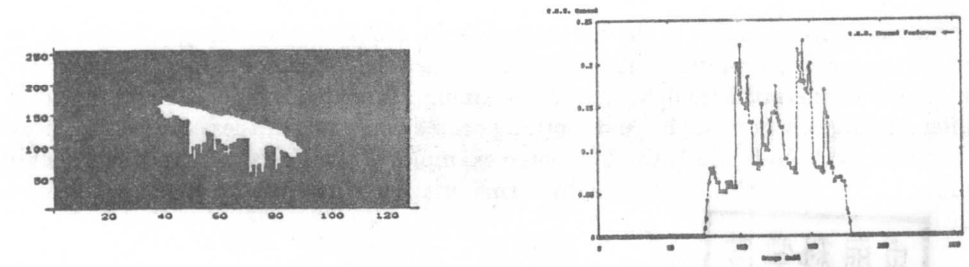


Figure 2: **(Left) An example of a simulated ISAR silhouette. The horizontal axis is range, the vertical axis Doppler. (Right) The weighted rms variation about the image center of mass in each range bin is used as the input to the neural network.**

moments to give more details about the structural shape. We also make this choice because of symmetry: while features may change their scale and sign in the Doppler domain, they will remain constant in range.

Our database consisted of 4896 simulated ISAR image frames drawn from 21 different ships. We divided the database into three subsets: one for training, a test set for monitoring generalization during the training procedure (cross-validation), and finally a second test set for evaluating generalization after training. Details of the composition of these data subsets are summarised in the following table.

Table 1: Simulated ISAR Imagery Database

Subset	Total Ships	# Commercial/Aux. # Combatants	# of Frames	# Commercial/Aux. # Combatants
		5		1020
Train	9	4	2108	1088
		3		680
Monitor	6	3	1496	816
		3		612
Novel	6	3	1292	680

2 Lateral Inhibition Neural Networks

Lateral inhibition has been studied in a variety of contexts, for example in unsupervised learning networks (Cooper and Scofield, 1988; Intrator, 1990; Seabach, 1990), in reinforcement learning (Sutton, personal communication) and dynamical models (Horn and Usher, 1990). Also, recent work (Giraud, Liu, bernard, and Axelrad, 1991) has looked at networks with excitatory-inhibitory pairs of neurons.

In the context of backward propagation (Rumelhart, Hinton, and Williams, 1986; Werbos, 1974), Sandon (1987) developed an ad hoc method for incorporating a linear competition between the error signals. At a given level in the network, the error signal for a particular neuron was modified by subtracting out an amount proportional to the sum of the error signals at the other nodes:

$$\delta_i^{(n)} \rightarrow \delta_i^{(n)} - \frac{1}{N-1}\sum_{j=1}^{N}(\delta_j^{(n)} - \delta_i^{(n)}) \tag{2}$$

Heuristically, it is appealing to allow competition between the error signals; however, such an approach does not guarantee that the network will implement a gradient descent minimization procedure. In contrast, we have taken the approach of imbedding the lateral inhibition in the energy functional and carrying out gradient descent on the modified energy functional. We accomplish this by introducing fixed lateral inhibition in the forward pass of the data through the network. With this approach, we have the advantage of being certain that we are actually carrying out a minimization procedure by gradient descent.

In our network, during the forward propagation of the pattern, the linear net input is computed for each cell, then the lateral inhibition of these inputs is computed, after which the nonlinear sigmoid is applied to the result to compute the firing rate of each cell. Therefore, the feedforward equations for pattern s are:

$$\tilde{x}_i^{s,(n)} = \sum_{j=1}^{N_{n-1}} w_{ij}^{(n-1)} \tilde{o}_j^{s,(n-1)} + \phi_i^{(n)} \tag{3}$$

$$\tilde{o}_k^{s,(n)} = \sigma(\tilde{x}_k^{(n)} - \mu^{(n)} \sum_{j \neq k} \tilde{x}_j^{(n)}) \tag{4}$$

where $\tilde{x}_i^{s,(n)}$ is the net feedforward input, $\tilde{o}_k^{s,(n)}$ is the cell activity, and $\sigma(x;\lambda) = \frac{1}{1+e^{-\lambda x}}$ is the nonlinear sigmoid input-output function. Here, $\mu^{(n)} \equiv \frac{\alpha^{(n)}}{N_n - 1}$ is the fixed parametric inhibition in layer n, and N_n is the number of cells in that layer. Figure 3 compares the lateral inhibition architecture with that of ordinary backprop. With these modifications, the energy function to be minimized is defined in terms of the inhibited outputs of the final layer of the network. When we take the gradients of the modified energy functional with respect to the synaptic weights, we find that the ordinary backprop error signals, $\delta_{si}^{(n)}$, must be replaced by inhibited error signals, $\tilde{\delta}_{si}^{(n)}$, of the form:

last layer synapses :

$$\tilde{\delta}^{(n)}_{si} = \gamma_i^{(n)} - \mu^{(n)} \sum_{k \neq i} \gamma_k^{(n)} \quad (5)$$

with

$$\gamma_\alpha^{(n)} = (\tau_\alpha^s - \tilde{o}_\alpha^{s,(n)}) \lambda_\alpha^{(n)} \tilde{o}_\alpha^{s,(n)} (1 - \tilde{o}_\alpha^{s,(n)}) \quad (6)$$

interior synaptic layers :

$$\tilde{\delta}_i^{(n)} = \lambda_i^{(n)} \tilde{o}_i^{s,(n)} (1 - \tilde{o}_i^{s,(n)}) \sum_k \tilde{\delta}_k^{(n+1)} (w_{ki}^{(n)} - \mu^{(n)} \sum_{j \neq i} w_{kj}^{(n)}) \quad (7)$$

and the modification rule is written in terms of the inhibited error signal and inhibited neuronal input:

$$\Delta_s(w_{ki}^{(n)}) = -\eta \frac{\partial \xi_s(t)}{\partial w_{ki}^{(n)}} = \eta \tilde{\delta}_{sk}^{(n+1)} \tilde{o}_i^{s,(n)} \quad (8)$$

where $\xi_s = \sum_{k=1}^{N_m} \frac{1}{2}(\tilde{o}_k^{s,(m)} - \tau_k^s)^2$ is the LMS network error. Because the strength of the inhibitory connections is fixed, the inhibition appears parametrically in the expressions for the modified error signals. In fact, we can think of the inhibition equation as being equivalent to inserting a layer of non-modifiable connections $L_{ij}^{(n)} = \delta_{ij}^{(Kronecker)} - (1 - \delta_{ij}^{(Kronecker)})\mu^{(n)}$ between each layer of modifiable connections $w_{ij}^{(n)}$ ($\delta_{ij}^{(Kronecker)}$ is the Kronecker delta, or identity matrix). This fixed architectural constraint allows competition between the cells in a layer to represent particular features in the data.

Notice that the last layer error signals are equivalent to the ad hoc formula used by (Sandon, 1987); this layer is looking at the spatial difference in how close the output layer cells are to their respective targets. Our model differs from Sandon's in the interior layer: error signals for interior layers are propagated back across synaptic vectors which inhibit each other laterally. When $\mu^{(n)} \rightarrow \frac{1}{N_n}$, each cell is comparing itself to the average of all of the other cells in both the forward and backward passes. In contrast, as $\mu^{(n)} \rightarrow 0$, the algorithm reduces to the ordinary backprop network.

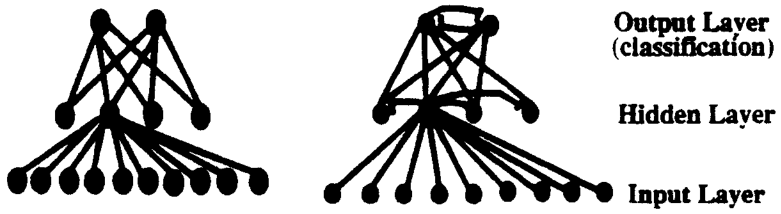


Figure 3: (Left) Ordinary backprop architecture. (Right) A backward propagation network with lateral inhibition. After the linear input in a layer is computed, the neurons laterally inhibit one another and then the nonlinear sigmoid is applied to obtain the actual cell response.

3 Results

3.1 Benchmarks for Backprop and Lateral-Inhibition Variants

We considered both three- and four-layer architectures (one and two layers of hidden units respectively). In the end, we settled on a three-layer architecture as the best configuration. Experiments with five hidden units revealed poor generalization to novel data. However, we obtained good generalization with seven or eight hidden units. More cells led to overfitting. Amplitudes of some sample first layer synaptic vectors are shown in figure 4.

We chose a small step-constant $\eta = 0.1$ with moderate momentum $\kappa = 0.6$ for the experiments. We trained the networks to distinguish between artificial images of combatants and those of commercial/auxiliaries as described above. Synapses were saved in a buffer whenever the generalization on the monitoring data set improved. The synapses in this buffer, representing the best level of generalization

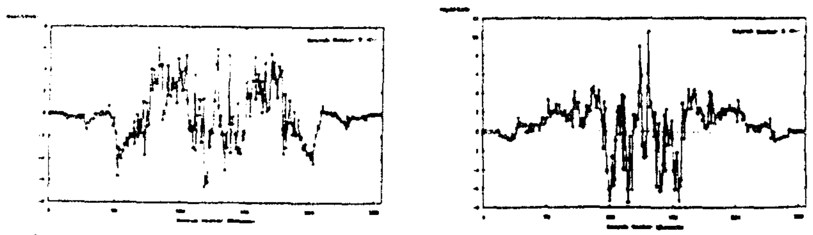


Figure 4: **Examples of Amplitude plots for the elements of two first-layer synaptic vectors after training. Samples are from a 256-7-2 network configuration.**

attained on the monitoring data set were eventually tested at the conclusion of each trial on the the third subset of the database as described above. The generalization results in the table are for this third independent database. This method of training was designed to compensate for the overtraining problem that is frequently encountered with backprop. By buffering the synapses only when generalization improves we lessen the chance that the network will overfit the data which often happens in the later phases of training. Such a training approach has been used by (Intrator, 1990) with BCM networks (Bienenstock, Cooper, and Munro, 1982) and in ordinary backprop (Bachmann, 1990).

Table 2: Neural Network Generalization
Classification of Range-Bin Moments of Simulated ISAR Imagery
Two-Class Problem: Combatants vs. Commercial/Auxilliary Ships

$\alpha^{(n)}$	trials	Mean		Mean	Best
(Inhibition)		Combat.	Commerc.	Combined	Combined
0.0	20	84.0 ± 5.8 %	54.5 ± 5.8 %	69.2 ± 5.2%	77.6%
0.15	20	79.6 ± 5.9 %	59.3 ± 5.6 %	69.4 ± 4.0%	76.7%
0.25	20	82.5 ± 7.4 %	55.0 ± 8.0 %	68.8 ± 3.4%	76.4%
1.0	20	78.8 ± 6.8 %	56.5 ± 6.8 %	67.6 ± 5.3%	77.9%

3.2 Understanding the Results

There is not a significant difference in generalization over the range of inhibition levels which we explored with the neural networks. We feel that ordinary backprop is probably doing as well as can be expected given the limited number of ships in the database. Also, some of the viewing angles were extreme and therefore difficult to classify. Also, a number of frames had very little profile component in the image and therefore were not easily categorized. Furthermore, the initial representation using range-bin moments loses some of the shape information and could be improved. Obtaining generalization of up to ~ 78% ("Best Combined" column in Table 2), therefore, is a good result.

To better understand our results, we tested human observors trained in ISAR ship classification, asking them to classify the simulated binary images. A strict comparison is not really possible since our networks were trained with only a couple of thousand frames, whereas some of the people participating in the human tests had many years of experience in ISAR classification. Furthermore, the human observors looked at the raw images, not the range-bin moments. Nevertheless, their performance can give an indication of the difficulty of the problem. One hundred patterns were selected at random from our simulated database. The human observors were asked to classify the ships as either "commercial/auxilliary", "combatant", or "other (submarine, landing platform, or small craft)". Even though there were no submarines, landing platforms, or small craft present in the simulated images, this third perceptual category was provided as a means of estimating how often combatants and commercial / auxilliary ships would be confused with other perceptual classes.

Table 3: Performance of Human Observors
Classification of Simulated ISAR Images (silhouettes)

Observor	% Correct		% Misclassified as "other"		% Undecided	
	Combat.	Commerc.	Combat.	Commerc.	Combat.	Commerc.
AL	70.9 %	42.2%	18.2 %	37.8 %	0 %	0 %
WL	83.6 %	31.1%	12.7 %	20.0 %	1.8 %	2.2 %
JM	89.1 %	48.9%	5.5 %	13.3 %	5.5 %	15.6 %
LB	92.7 %	60.0%	5.5 %	15.6 %	0 %	0 %
DD	89.1 %	66.7%	7.3 %	6.7 %	0 %	0 %
MB	98.2 %	80.0%	1.8 %	2.2 %	0 %	0 %
Means:	87.3 ± 9.3 %	54.8 ± 17.6 %	8.5 ± 5.9 %	15.9 ± 12.4 %	1.2 ± 2.2 %	3.0 ± 15.6 %
Both:	71.0 %		12.2 %		2.1 %	
Best:	89.1 %		2.0 %		0 %	

Some of the variation can be accounted for by degree of experience. It is interesting, however, that the most experienced human observor, DD in the table, did not obtain the best score. Nevertheless, this experiment provides a baseline for the degree to which the perceptual class of the simulated image silhouettes can be identified.

The results which we have obtained for the neural networks for a range of randomly chosen initial starting conditions appear to have a similar mean classification accuracy, although we can not make a strict comparison because we have not allowed for an "other category" in the neural net experiments. An expanded database of "other" simulated targets will be useful in future work. We also plan to develop an on-line version of the simulation which will present randomly chosen views of the ships in the database to the neural network. Work on hybrid networks combining unsupervised and supervised learning algorithms is also planned.

4 Acknowledgement

We wish to acknowledge Dave Drake for assisting us in the creation of our artificial radar database and to thank the other members of the Airborne Radar Branch at NRL who participated in the human classification test. The theoretical results on lateral inhibition were completed at the Center for Neural Science of Brown University during the fall of 1990 by one of us (Bachmann) under the support of the National Science Foundation (grant # EET 8719102), the Army Research Office (grant # DAAL03-88-K-0116), and the Office of Naval Research (grant # N00014-88-K-1316). The remainder of the work, including all of the simulations, was completed at the Airborne Radar Branch, Naval Research Laboratory, under the support of the Office of Naval Research (53-1501-0) and (53-1501-1).

References

[1] Bachmann, C. M., *Learning and Generalization in Neural Networks*, Ph.D. Dissertation, Brown University, Department of Physics, May 1990.

[2] Bienenstock, E. L., Cooper, L. N., Munro, P. W. *Theory for the Development of Neuron Selectivity: Orientation Specificity and Binocular Interaction in Visual Cortex*, The Journal of Neuroscience, Vol. 2, No. 1, pp. 32-48, January, 1982.

[3] Cooper, L. N., Scofield, C. L. *Mean-Field Theory of a Neural Network*, Proceedings of the National Academy of Sciences USA, Vol. 85, pp. 1973-1977, March 1988, Neurobiology.

[4] Horn, D. and Usher, M., *Excitatory-Inhibitory Networks with Dynamical Thresholds*, International Journal of Neural Syustems, Vol. 1, No. 3 (1990), pp. 249-257.

[5] Intrator, N. I. *Feature Extraction Using an Unsupervised Neural Network*, in Proceedings of the 1990 Connectionist Models Summer School, Touretsky, D. S., Ellman, J. L, Sejnowski, T. J. (eds.), San Mateo, CA: Morgan Kaufmann.

[6] Giraud, B., Liu, L. C., Bernard, C., Axelrad, H., *Optimal Approximation of Square Integrable Functions by a Flexible One-Hidden-Layer Neural Network of Excitatory and Inhibitory Neuron Pairs*, Neural Networks, Vol. 4, pp. 803-815, 1991.

[7] Park, Y., Sklansky, J., *Automated Design of Linear Tree Classifiers*, Pattern Recognition, Vol. 23, No. 12, pp. 1393-1412, 1990.

[8] Rumelhart, D. E., Hinton, G. E., Williams, R. J. *Learning Internal Representations by Error Propagation* in **Parallel Distributed Processing, Explorations in the Microstructure of Cognition**, Vol. 1, Rumelhart, D. E., McClelland, J. L. (eds.), pp. 318-362, MIT Press: Cambridge, Mass., 1986.

[9] Sandon, P., Lecture on Extensions to Backward Propagation, G.T.E. Connectfest Conference, Waltham, Mass., Sept. 25, 1987.

[10] Seebach, B. S., *Evidence for the Development of Phonetic Property Detectors in a Modified BCM Neural Network without Innate Knowledge of Linguistic Structure*, Ph. D. Dissertation, Brown University, Program in Neural Science, 1990.

[11] Werbos, P. J., *Beyond Regression: New Tools for Prediction and Analysis in Behavioral Sciences.* Ph.D. Thesis, Harvard University, 1974.

Dynamic Learning Using Exponential Energy Functions

Maqbool Ahmad and Fathi M. A. Salam

System and Circuits & Artificial Neural Nets Laboratories
Department of Electrical Engineering
Michigan State University
East Lansing, MI 48824
Phone: (517)-355-7695 Fax: (517)-353-1980
E-mail: salam@egr.msu.edu

0 ABSTRACT

We employ a continuous-time gradient descent weight update law for supervised learning of feedforward artificial neural networks due to specific advantages over its discrete-time counterpart. We also employ an Exponential energy function in the update law. We analytically show that this energy function would speed up the learning dynamics and ensures faster convergence to a useful minimum. The dynamics would "skip" minima which are at higher energy levels and converge to one at a lower energy level. We also describe software implementation of the learning dynamics based on the Exponential energy function. Various supporting simulations on the XOR and character recognition problems are also included.

1 INTRODUCTION

Artificial neural networks are mathematical models, originally designed to mimic some of the functions of the nervous systems of higher animals. Carefully formulated and designed, they can be implemented via software using conventional digital computers or hardware using digital/analog VLSI. We resort to the first approach with a view to finally implement these artificial neural models in digital/analog VLSI.

We focus on the feedforward artificial neural networks (FFANN) using back-propagation architecture. Without question, back-propagation is currently the most widely applied neural network architecture for FFANNs. This popularity primarily revolves around the ability of back-propagation networks to learn complicated multidimensional mappings.

The automatic learning rules for FFANN give them most of their unique capabilities. The Generalized Delta Rule (GDR) proposed by Rumelhart *et al* [1], which is an error back-propagation (EBP) learning rule, has been popularized since the publication of their work in 1986. Following the presentation of an input/output pattern $p = 1,2,....P$, the rule for updating the weight w_{kj}, connecting the jth node in a layer to the kth node in the subsequent layer, is given by the difference equation [1]

$$\Delta w_{kj} = -\gamma \nabla_{w_{kj}} E , \qquad (1.1)$$

where γ is a constant known as the learning rate and $\nabla_{w_{kj}}$ is the partial derivative with respect to w_{kj} . E is the energy function and is usually given by the sum-of-the-squared error function

$$E = \frac{1}{2} \sum_p \sum_n \varepsilon_{pn}{}^2, \qquad (1.2)$$

where $\varepsilon_{pn} := t_{pn} - y_{pn}$, while t_{pn} and y_{pn} are, respectively, the nth components of the desired and the actual output vectors for input/output pattern p. The update (1.1), however, has been known to be (a) extremely slow if it converges, and (b) sometimes it does not converge and may even generate oscillations.

Describing the update law as a system of differential equations [2-5] in the form

$$\dot{w}_{kj} = -\gamma \nabla_{w_{kj}} E , \qquad (1.3)$$

has its advantages over the system of difference equations. In particular, being a continuous-time gradient system, the learning rule (1.3): (a) ensures convergence to only minima, i.e., stable equilibria, (b) prevents the existence of oscillations and complicated behaviors, and (c) lends itself to natural implementation via digital/analog LSI/VLSI circuits [2-5].

The choice of the energy function specifies the back-propagated error update law. It also determines the learning performance in the sense of the speed of convergence and the size of domains of attraction of the stable equilibrium points in the weight space. For the faithful implementation of the update law (1.3), the learning rate γ and the time step Δt of the integration routine should be small enough to ensure that the essence of the continuous-time weight update is not lost. However, as a consequence of the small value of $\gamma.\Delta t$ (a) the learning is slow and (b) the algorithm converges to the nearest local minimum -- which in most cases may not be useful.

0-7803-0559-0/92 $3.00 © 1992 IEEE

One approach in speeding up the learning is to choose a suitable energy function E. One can either choose the Cauchy energy function [6] given as

$$E^c = \frac{1}{2} \prod_p \prod_n (1+\varepsilon_{pn}{}^2), \tag{1.4}$$

the Polynomial energy function [7] of order r, which is

$$E^p = a \sum_p \sum_n (|\varepsilon_{pn}| + b)^r, \tag{1.5}$$

with a and b any real constants or any other energy function that helps speeding up the learning process. While the Cauchy [6] or the Polynomial [7] energy functions speed up the convergence to their local minima, the values of these minima may not be satisfactory from a practical point of view.

One may consider increasing the value of the learning rate γ, to be an immediate solution to the above problems. This may not work in all the cases. With a large value of γ, while we increase the tendency of "skipping" over local minima and increasing the speed of convergence, we also make it prone to skip desirable minima and delay the convergence.

There is a need that the learning algorithm should skip the undesirable local minima and converge to a minimum which ensures reasonably smaller error value. An aspect of this need is captured by the simulating annealing approach. However, the simulating anealing lacks precision in the sense that the approach may not cease at or near global (or acceptable) minima. In this work we tackle the problem of skipping the undesirable local minima and increasing the learning speed to achieve an acceptable set of weights of the network. We propose the Exponential energy function of the form

$$F = e^{\kappa E}, \tag{1.6}$$

for the weight update law (1.3). Here κ is any positive real number whose determination is discussed later.

The paper is organized as follows. In section 2 we include the learning dynamics of the update law (1.3) using the Exponential energy function F. In section 3, we compare the learning dynamics of the weight update law (1.3) using the Exponential energy function F and a simple energy function E. E can be the Gaussian, the Cauchy, the Polynomial or any other suitable energy function. Software implementations of these derivations discussed in section 4 are then used to train a FFANN. As illustrative examples, we specialize the software to solve the XOR and the pattern recognition problems. Simulation results are presented in section 5 and finally we summarize our conclusions in section 6.

2 THE LEARNING DYNAMICS

We derive the learning dynamics for the Exponential energy function (1.6). Consider a multilayer FFANN having an input layer, m hidden layers and an output layer. We index the input through the output layers by $IN, H_a, .., H_j, H_k, .., H_m, O$ consecutively. The input layer is assumed to have I nodes with indices $1,..,i,..,I$ and the output layer has N neurons with indices $1,..,n,..,N$. The hidden layers are assumed to have sufficient number of neurons to perform the assigned task satisfactorily [8-10]. The last hidden layer H_m is assumed to have M nodes which are indexed $1,..,m,..,M$. In general, any hidden layer H_j and its subsequent hidden layer H_k is assumed to have J and K nodes, which are indexed $1,..,j,..,J$ and $1,..,k,..,K$ respectively. The neurons in any particular layer, as depicted in Figure 1, are assumed to have no lateral or feedback connections.

Let $p = 1,2,......,P$, be the index for the input-output patterns used in training the network. Consider the learning dynamics of the weights using the energy function (1.2), (1.4) or (1.5), specifically

$$\dot{w} = -\gamma \nabla_w E. \tag{2.1}$$

Now consider the learning dynamics of the weights using the Exponential energy function (1.6), specifically

$$\dot{w}^e = -\gamma \nabla_w F, \tag{2.2i}$$

$$= -\gamma \kappa e^{\kappa E} \nabla_w E, \tag{2.2ii}$$

$$= -\gamma \chi \nabla_w E, \tag{2.2iii}$$

$$= \chi \dot{w}, \tag{2.2iv}$$

where χ is the *skipping factor* and is defined as

$$\chi := \kappa e^{\kappa E}. \tag{2.3}$$

Observe that χ is constant at any particular step of the integration and varies exponentially with the output error (energy) from one step to another. As the learning proceeds, the output error (energy) E decreases, the skipping factor χ also decreases exponentially. Therefore the likelihood of skipping the local minima which are at higher energy levels are greater than skipping the local minima at lower energy levels. Here $\dot{w}$ gives the learning dynamics for the update law (2.1). These learning dynamics have already been explored in [2]-[7].

3 ANALYTICAL COMPARISON OF THE LEARNING DYNAMICS

We compare the Exponential learning (2.2), with the simple learning (2.1). In order to do so, we first outline a criterion for the learning performance in subsection 3.1 and then compare the learning dynamics of weight update law (1.3) using a simple energy function (1.2), (1.4) or (1.5) and the Exponential energy function (1.6) in subsection 3.2.

3.1 A criterion for the learning performance

To compare the learning dynamics given by (2.1) and (2.2), we define the performance criterion delineated in the following procedure. First we consider $V := E$ as a candidate Liapunov function [11]. Then, we calculate the derivative of V along the trajectories of the system (2.1) -- denote this derivative by $\dot{V}$. Similarly, we calculate the derivative of V along the trajectories of the system (2.2) -- denote this derivative by $\dot{V}^e$. Now if $\dot{V}^e - \dot{V} \leq 0$, then we say that *the learning performance* of the second system is "faster" than or at least is the same as the first system. This is motivated by the fact that $\dot{V}^e - \dot{V}$ gives the projection of the rate of change of the difference in the weight vectors, say $(W^e - W)$, along the divergence vector $\nabla_W V$. This we take as a measure of speed of updating the weight-vectors difference $(W^e - W)$, which in turn is equivalent to the comparative speed in updating the weight vectors W^e and W.

Formally, we observe that

$$\dot{V}^e - \dot{V} = \nabla_W V \, . \, (\dot{W}^e - \dot{W}), \qquad (3.1)$$

where W^e and W denote vectors containing all the corresponding weights. Equations (2.2) give the components of W^e and equations (2.1) give the components of W.

3.2 Comparison of the learning dynamics

We analyze the *learning performance* of weight update law (1.3) using the Exponential energy function by comparing the learning dynamics of systems (2.1) and (2.2) with the criterion outlined in subsection 3.1. In order to formally compare the *learning performance* due to a simple and its Exponential energy functions, we state

Theorem 1: Consider a candidate Liapunov function $V := E$ given by equation (1.2), (1.4) or (1.5) corresponding to the Gaussian, the Cauchy or the Polynomial energy function. Let $\dot{V}$ be the derivative of V along the the trajectories of (2.1) and $\dot{V}^e$ be the derivative of V along the trajectories of (2.2). Let the skipping factor $\chi \geq 1$. Then the *learning performance* of system (2.2) is "faster" than or at least is the same as system (2.1).

Proof : In order to analyze the learning performance, let's calculate $\dot{V}^e - \dot{V}$.

$$\dot{V}^e - \dot{V} = \nabla_W V \, . \, (\dot{W}^e - \dot{W}),$$

$$= [..\nabla_w E..] \, . - \gamma([..\chi\nabla_w E..]^T - [..\nabla_w E..]^T), \qquad (3.2i)$$

$$= -\gamma \sum_w \left[\chi - 1\right] \left[\nabla_w E\right]^2. \qquad (3.2ii)$$

Given that $\chi \geq 1$, the update law (1.3) using the Exponential energy function (1.6) improves (or at least does not deteriorate) the *learning performance.* The skipping factor $\chi \geq 1$ imposes the following restriction on the energy function as depicted in Figure 2.

$$E \geq - \frac{\ln\kappa}{\kappa}. \qquad (3.3)$$

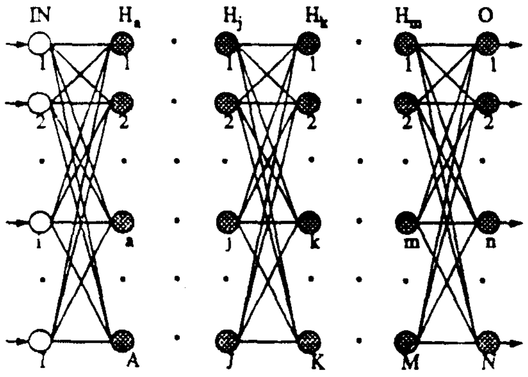


Figure 1: A model of feedforward Artificial Neural Network.

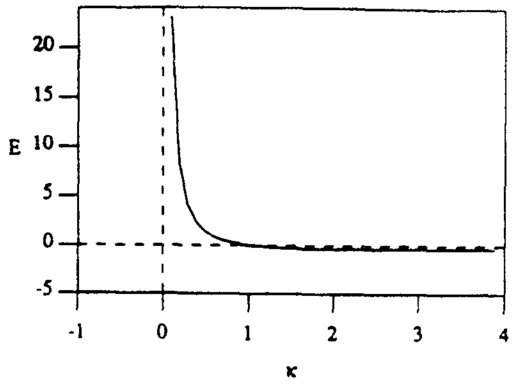


Figure 2: Graph representing the relationship (3.4).

We observe that for non-positive κ the system does not behave as a gradient descent system. For positive κ the *learning performance* only improves if for a certain value of κ, E has a value that satisfies the relationship (3.3). We also observe that if $\kappa \geq 1$, the learning with the Exponential energy function always improves.

4 SOFTWARE IMPLEMENTATION OF THE LEARNING DYNAMICS

The weight update law (2.2) is implemented into computer software using the C-language on Sun SPARC station. Fourth order Runge-Kutta integration routine is used for the integration of these update laws. A stopping criterion ω, was set to determine if the system has converged to a solution. The parameter ω was defined as

$$\omega := \sum_w |\Delta w|, \tag{4.1}$$

where w are all the weights in the network and

$$\Delta w := w(t+\Delta t) - w(t). \tag{4.2}$$

The term Δt is the time step chosen for the fourth order Runge-Kutta integration of the update laws. An error criterion ξ was used to characterize the training: the lower the value of the error, the better is the training. The (output) error parameter is defined as

$$\xi := \sum_p \sum_n \varepsilon_{pn}{}^2. \tag{4.3}$$

Note that this error parameter is exactly the Gaussian energy function. The weights of the network can be initialized by giving their values or selecting randomly by a random number generator. The training simulator, using the Runge-Kutta integration routine reads the input data from an input file, which provides the values of the time step Δt, the learning rate γ, the error criterion ξ, the stopping criterion ω, the energy function parameter κ, the network structure, the training patterns and the corresponding target values. The training is terminated when either the stopping criterion ω or the error criterion ξ is achieved. The simulator writes the intermediate or/and the final values of the network weights or/and other parameters into an output file.

5 SIMULATIONS

The *training capability* of the software simulator, using the Exponential and the simple energy functions, is investigated and a comparison is drawn. In the following we simulated two problems of different computational complexity: the XOR and the arabic numerals (0-9) recognition problem. Simulation results for both of these problems are reported in section 5.1 and 5.2.

5.1 The XOR problem

The prototype FFANN used to solve the XOR problem had two passive input nodes (a passive node simply transmits the input to its output), two hidden nodes and one output node. The network was fully connected and weights were initialized randomly by a random number generator. The value of any initial weight w is such that $-0.9 \leq w \leq 0.9$.

We assume that the network has converged when the stoppage criteria $\omega \leq 0.0001$ is achieved. In all the simulation cases, the network converged to a local minimum. A comparison of 2 different sets of initial conditions are illustrated in Figures 3, 4 and 5. In these figures, we compared the learning dynamics of the update law (2.1) using the Gaussian , the Cauchy and the Polynomial energy function respectively, with the update law (2.2) using the Exponential energy function with κ equal to 1 and 1.8. In all of these simulations γ and Δt are chosen to be 1 and 0.1 seconds, respectively.

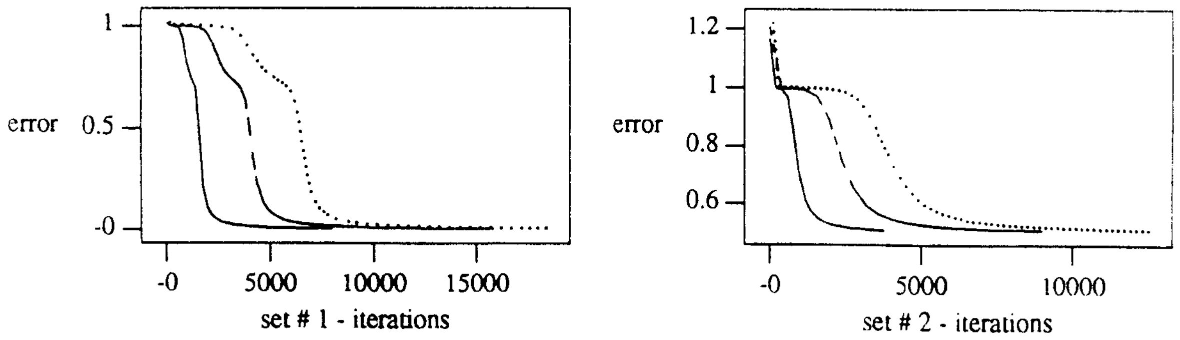
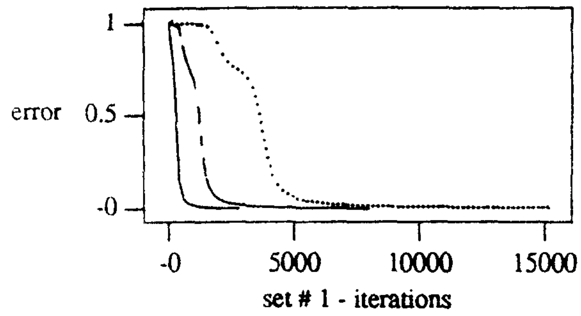


Figure 3: Comparison of the Exponential (κ=1, 1.8) and the Gaussian learning. The Gaussian is represented by (...) whereas the corresponding Exponential (κ=1, 1.8) is represented by (---) and (___) respectively.



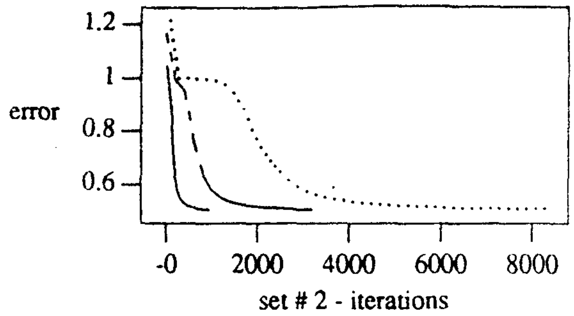
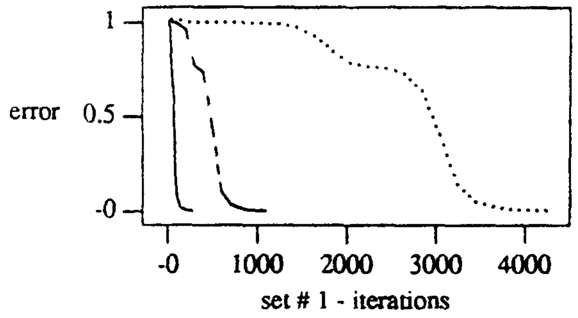


Figure 4: Comparison of the Exponential (κ=1, 1.8) and the Cauchy learning. The Cauchy is represented by (...) whereas the corresponding Exponential (κ=1, 1.8) is represented by (---) and (___) respectively.



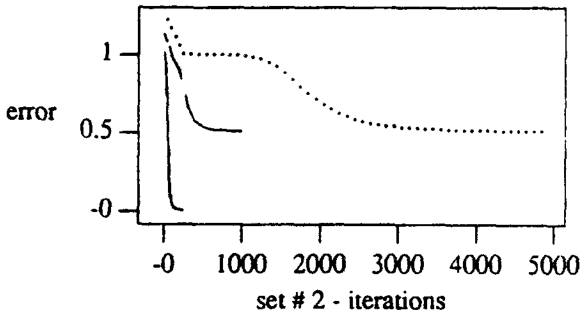


Figure 5: Comparison of the Exponential (κ=1, 1.8) and the Polynomial learning. The Polynomial is represented by (...) whereas the corresponding Exponential (κ=1, 1.8) is represented by (---) and (___) respectively.

In the case of set # 1, the training achieved excellent results. In case of set # 2 the network was stuck in an undesirable local minimum with an error value equal to 0.5. We observe that for set # 2 when $\kappa = 1$, the skipping factor was not large enough to skip the local minimum. But when κ was chosen to be equal to 1.8, we observe that the weight update law (2.2) skips the local minima and converges to a desirable minimum (Figure 5). This behavior persisted for numerous case studies.

5.2 The Character (Arabic Numerals) Recognition problem

We decompose the character recognition problem into two parts: (1) feature extraction and (2) classification. We tackle the problem of feature extraction using a two layer feedforward artificial neural network (FFANN) and that of classification using an additional layer network. The two networks are concatenated, the feature extraction network followed by the classification network. The network structure is described in detail in [12].

We have collected 100 different fonts of Arabic numerals (0-9) from printed material [13]. Each of these printed fonts were scanned using a SC-7500 Toshiba scanner and was stored as "Tiff" file. These files have subsequently been converted to "Hips" format which could easily handle binary data. The pixel image of various digits in different fonts were then obtained from the formatted Hips files.

The images of these fonts were stretched appropriately to fit within a pixel window of 17×15 resolution while leaving a small margin on all sides. With this preprocessing, we also achieve scaling invariance of the characters to some extent. The grey level of these pixels were set to be 0 or 1. The collected fonts included roman, bold and italic versions of different classes.

The feature extraction and the classification networks can be trained together or separately. We trained the two networks separately with a view to achieve the following advantages:

(1) The feature extraction and the classification networks can be viewed as two separate modules,

(2) The feature extraction being fully trained on a large data-base and the classification network can be quickly trained on the data-base when new fonts are added. By doing so we enhance the character recognition capability of the network as demonstrated in [12].

Half of the data-base was used as set 1 and the other half as set 2. We trained the feature extraction network with set 1 and update law (1.3) using the Gaussian (1.2) and the Exponential (1.6) energy functions. In all the training simulations the value of Δt and γ were chosen to be 0.0005 and 1. κ parameter of the Exponential energy function (2.2) was chosen to be 0.004 (the initial error value being in the range of 4000 to 5000).

We assume that the network has converged when the stopping criterion $\omega \leq 0.15$ or the error criterion $\xi \leq 400$ is satisfied. For the Gaussian energy, learning was terminated when the stopping criterion ω was achieved where as for the exponential energy, learning was stopped when the error criterion was achieved. The network achieved 100% recognition results for (the training) set 1. After the network was trained with set 1, it was tested with set 2. The error(s), number of steps to converge and the character recognition performance is reported in Table I. We observe that the Exponential Gaussian outperforms simple Gaussian learning in terms of error criterion and number of computer steps. The generalization capability can further be enhanced by appropriately using the classification network [12].

Table I

energy function	ξ	steps	correct	incorrect	reject
Gaussian	526.25	16205	92.6%	3%	4.4%
Exponential	399.96	10598	92.4%	2.6%	5%

Table I shows error(s), the number of steps at convergence and the character recognition results when the Gaussian and its Exponential energy functions are used for training.

6 CONCLUSION

We have considered a class of software implemented FFANNs and addressed supervised learning using continuous-time weight update law. A solution to overcome the problem of convergence to immediate local minima is presented. We proposed the Exponential energy function to be used in the continuous-time weight update law. We have analytically shown that this will speed up the convergence to a useful minimum at lower energy level in comparison with the usual learning scheme. We also present the learning dynamics and the software implementation of this update law. Extensive simulation on the prototype XOR problem confirm the theoretical results. we have also used the Exponential learning for a character recognition problem and have shown that it considerably improves the learning speed.

Acknowledgment: This work is supported in part by ONR Grant N00014-89-J-1833, the Michigan Research Excellence Fund (REF), and NSF Grant ECS-8814027.

References

[1]. D. E. Rummelhart, G. E. Hinton and R. J. Williams, *Learning internal representations by error propagation*, in Parallel Distributed Processing: Explorations in the microstructures of cognition, Cambridge, MA: MIT Press, vol. 1, pp. 318-362, 1986.

[2]. F. M. A. Salam, *Neural Nets and Engineering Implementations*, key address at the 31st Midwest Symposium on Circuits and Systems, St. Louis, Missouri, August 10-12, 1988.

[3]. F. M. A. Salam, Artificial Neural Nets: Basic Theory and Engineering Implementations, Department of Electrical Engineering, Michigan State University, East Lansing, MI 48824, October 1989.

[4]. F. M. A. Salam and M. R. Choi, *An All-MOS Analog Feedforward Neural Circuit with Learning*, 1990 IEEE International Symposium on Circuits and Systems (ISCAS), New Orleans, Louisiana, May 1-3, 1990.

[5]. F. M. A. Salam, *Learning Algorithms For Artificial Neural Nets For Analog Circuit Implementation*, Proceedings of the 22nd Symposium of the Interface, pp. 169-178, May 1990.

[6]. M. Ahmad and F. M. A. Salam, *Supervised learning using the Cauchy energy function*, submitted to 2nd International Conference on Fuzzy Logic and Neural Networks.

[7]. M. Ahmad and F. M. A. Salam, *Error back-propagation learning using the Polynomial energy function*, submitted to 4th IEEE International Conference on System Engineering.

[8]. M. Arai, *Mapping abilities of three layer neural networks*, International Joint Conference on Neural Networks, vol. 1, pp. 419-423, 1989.

[9]. G. Mirchandni and W. Cao, *On hidden nodes of Neural Nets*, IEEE Transactions on Circuits and Systems, vol. 36, No. 5, pp. 661-664, May 1989.

[10]. S. C. Huang and Y. H. Huang, *Bounds on the number of hidden neurons in multilayer perceptrons*, IEEE Transactions on Neural Networks, vol. 2, No. 1, Jan. 1991.

[11]. H. K. Khalil, Nonlinear Systems, Macmillan, 1991.

[12]. M. Ahmad and F. M. A. Salam, *Feedforward artificial neural network structure for character recognition*, to appear in the 29th Annual Allerton Conference on Communication, Control and Computing, October 1991.

[13]. F. Lamberty, Letter forms, Hasting House Publishing, New York, 1964.

A Convergent Neural Network Learning Algorithm

Zaiyong Tang and Gary J. Koehler
Dept. of Decision and Info Sciences
University of Florida Gainesville, FL. 32611

Abstract

A globally guided backpropagation (GGBP) training algorithm is presented. This algorithm is a modification of the standard backpropagation algorithm. Instead of changing a weight w_{ij} according to the partial derivative of error, E, with respect to w_{ij}, we try to minimize E in the output space. The change in weights W is computed based on the desired changes in the output O. The new algorithm is an analog to backpropagation with a dynamically adjusted learning rate, η. This learning rate changing scheme avoids the problems associated with heuristic learning rate adjusting method. Two main advantages of GGBP are (1) fast learning speed, and (2) convergence to a global optimal solution.

1 Introduction

Backpropagation (BP) is one of the most widely used learning algorithm for multi-layered feed-forward neural networks (le Cun, 1988). The popularity of BP arises from its simplicity and successful applications to many real world problems. It is commonly recognized, however, that BP has some inherent shortcomings. Two of the often cited BP shortcomings are (1) slow or no convergence, and (2) the possibility of getting stuck in local minimum solutions.

There has been a great research effort devoted to overcome the first problem. A number of local acceleration heuristics are discussed in Jacobs (1988). Other approaches to improve the speed of convergence include the use of second order information of the error surface such as Newton's method, conjugate gradient methods (Moller, 1990; Becker and le Cun, 1988). Those improvements on backpropagation often increase the learning speed significantly in terms of training epochs at the cost of an increased computation effort.

Few researchers have considered the second problem of BP. Empirical results have shown that with ample hidden units embedded in the network, BP can usually escape a local minimum (Rumelhart et al., 1986) probably due to large degrees of freedom. However, increasing hidden units in the network may not be an appealing idea, since an unnecessarily large number of hidden units is likely to decrease the generalization capability of the network (Kruschke, 1989), and may cause overfitting problems.

In this paper we propose a modification to the standard backpropagation algorithm. The modification, while retaining the simplicity of the standard BP, introduces two nice properties: (1) there is a training time speed up, and (2) convergence to global optimal solution is guaranteed. We start with a briefly review of the standard backpropagation.

2 Backpropagation (BP)

The backpropagation leaning algorithm was introduced for feed-forward neural network training by Rumelhart et al. (1986) (although the basic idea can be traced back to 1969 (le Cun, 1988)). In a feed-forward network, the neurons (processing units) are arranged in layers. The input units simply pass on the input vector X. The units in the hidden layer and output layer are called computation units. Each computation unit has a transfer function, f, which is often chosen to be the sigmoid function.

A feedforward neural network works by training it with known examples ((X,T) pairs) where X is an input and T the desired ouput results from X. A random sample X_p is drawn from the training set $\{X_p | p = 1, 2, \ldots, P\}$, and fed into the network through the input layer. The network computes an output vector O_p and compare it to the training target T_p. An error function is defined based on the difference between O_p and T_p. A commonly used error function is the sum of squared errors (SSE) function. The error computed from the output layer is back-propagated through the network, and weights are modified according to their contribution to the error function (for BP details, see Rumelhart and McClelland, 1986).

Note that although epoch training (Updating weights after presentation of all training patterns) ensures convergence of a BP solution, the solution may be a strict local minimum. If the local minimum error is larger than the stopping criterion, then BP fails to obtain a solution. The need for global optimal solution to the feed-forward neural network motivates the globally guided backpropagation algorithm.

3 Globally Guided Backpropagation (GGBP)

3.1 The Idea

One of the main shortcomings of BP is the possibility of getting stuck in a local minimum solution. The error surface of BP networks in weight space is generally very complicated. On the other hand, the error surface of the BP network in the output space is quite simple. If we use a sum of squared error function, E, the error surface is convex quadratic in the output space. Minimization of the quadratic function is easy. The unique local minimum of E is also a global minimum solution. The optimal outputs are the target values. Unfortunately, solving for weights W through the inverse function of output O is extremely difficult, if not impossible. However, if we change the output by a small amount, we will be able to find the changes in weights W via a Taylor series expansion of O.

$$\begin{aligned}\Delta O &= O(W + \Delta W, X) - O(W, X) \\ &= \nabla_W O(W, X)\Delta W + \frac{1}{2}\Delta W^T \nabla_W^2 O(W + \xi\Delta W, X)\Delta W \qquad (1)\end{aligned}$$

where $\xi \in (0, 1)$.

If we update the weights of the network based on the changes in O, instead of $-\eta\nabla_W E$, as in standard backpropagation, then we have reason to hope that this weight updating scheme would (1) lead to faster convergence, since the search in the weight space is guided directly by the search in the output space, and (2) lead to a global optimal solution. The idea is to ignore the shape of the error surface in the weight space by moving W in a direction such that in the output space error E is always decreasing.

3.2 Learning Rule Derivation

The learning rule of GGBP is derived based on the changes in output space. Let us consider a given training pattern. The error function is

$$E = \sum_{k=1}^{K} E_k = \frac{1}{2}\sum_{k=1}^{K}(T_k - O_k)^2 \qquad (2)$$

where k is the index for the output units.

Changing output $O = (O_1, O_2, ..., O_k)^T$ based on gradient descent in the output space gives

$$\Delta O(n) = O(n+1) - O(n) = -\eta \nabla_O E(n) \tag{3}$$

where n is the iteration index. Using equation (2) results in

$$\Delta O(n) = \eta(T - O(n)). \tag{4}$$

Assuming the changing in W is small, we may use first order approximation in Equation (1) (For clarity we omit writing the variables weights W and input X in function $O(W, X)$). Hence

$$\Delta O(n) = \nabla_W O(n) \Delta W. \tag{5}$$

Note that here $\Delta O(n)$ is a K dimensional vector, ΔW is an S dimensional vector, and $\nabla_W O$ is a $K \times S$ matrix. Finding ΔW requires the pseudo-inverse of the matrix $\nabla_W O$. This is computationally undesirable. Considering the special structure of the feed-forward neural network, we notice that the weights of the output layer associated with output unit i are independent of the output units $O_k, k = 1, 2, ..., K, k \neq i$. We can rewrite ΔO as

$$\Delta O = [\nabla_{W_H} O, \nabla_{W_{O_1}} O, ..., \nabla_{W_{O_K}} O] \begin{bmatrix} W_H \\ W_{O_1} \\ \vdots \\ W_{O_K} \end{bmatrix} \tag{6}$$

where W_H denotes weights in the hidden layer(s) and W_{O_i} the output layer weights associated with output node i. Each component of ΔO becomes

$$\begin{aligned} \Delta O_k &= \nabla_{W_H} O_k \Delta W_H + \nabla_{W_{O_k}} O_k \Delta W_{O_k} \\ &= \nabla_{W^k} O_k \Delta W^k \qquad k = 1, 2, ..., K \end{aligned} \tag{7}$$

where W^k denotes all weights contributing to output O_k. ΔO_k becomes the inner product of two vectors, $\nabla_{W^k} O_k$ and ΔW^k. This inner product is maximized if we choose ΔW^k in the direction of $\nabla_{W^k} O_k$. Thus we have

$$\Delta O_k = ||\nabla_{W^k} O_k|| ||\Delta W^k|| \qquad k = 1, 2, ..., K \tag{8}$$

or

$$||\Delta W^k|| = \frac{\Delta O_k}{||\nabla_{W^k} O_k||} \qquad k = 1, 2, ..., K. \tag{9}$$

The normalized component of ΔW^k is

$$\Delta w_s = ||\Delta W^k|| \frac{\frac{\partial O_k}{\partial w_s}}{||\nabla_{W^k} O_k||}. \tag{10}$$

Substituting $||\Delta W^k||$ with Equation (9) gives

$$\Delta w_s = \frac{\Delta O_k \frac{\partial O_k}{\partial w_s}}{||\nabla_{W^k} O_k||^2}. \tag{11}$$

Replacing ΔO_k using Equation (4) results in

$$\Delta w_s = \frac{\eta(T_k - O_k)\frac{\partial O_k}{\partial w_s}}{\sum_{i \in W^k}(\frac{\partial O_k}{\partial w_i})^2}. \tag{12}$$

If w_s is a weight belong to the output layer, Equation (12) is used as weight updating rule. If w_s is a weight from a hidden layer, we need consider the effect of all the outputs on it. The changes due to each output $O_k, k = 1, 2, ..., K$ are summed up. Hence we have

$$\Delta w_s = \sum_{k=1}^{K} \frac{\eta(T_k - O_k)\frac{\partial O_k}{\partial w_s}}{\sum_{i \in W^k}(\frac{\partial O_k}{\partial w_i})^2} \tag{13}$$

for all $s \in W_H$.

Recall in standard backpropagation, the weights are updated with the following formula

$$\Delta w_s = -\lambda\frac{\partial E_p}{\partial w_s} = \lambda(T_k - O_k)\frac{\partial O_k}{\partial w_s} \tag{14}$$

for output layer and

$$\Delta w_s = -\lambda\frac{\partial E_p}{\partial w_s} = \lambda\sum_{k=1}^{K}(T_k - O_k)\frac{\partial O_k}{\partial w_s} \tag{15}$$

for hidden layer(s).

Note the similarity of the weight updating scheme of GGBP with that of the standard BP. The new methods is analogous to the standard BP with a dynamically adjusted learning rate

$$\lambda \approx \frac{\eta}{F(\frac{\partial O_k}{\partial w_s})} \tag{16}$$

where F is a function of the partials of the output with respect to the weights. The concepts of the two approaches are, however, quite different. With GGBP, η is a fixed learning parameter in the output space, while λ is a fix learning rate in the weight space.

4 Experiments

Two test problems are used to illustrate and evaluate the performance of GGBP. Both problems are standard test problems. All tests were run on a 80386-micro computer. The reported results are an average of 20 runs starting with the same random initial weights for both GGBP and standard BP. All numbers are rounded to their nearest integers

4.1 The XOR Problem

The Exclusive Or (XOR) problem has been used extensively as a benchmark for neural network algorithm evaluation due to historical reasons. A $2 \times 2 \times 1$ feed-forward network is used in our test. Results in Table 1 show that GGBP is 3 to 13 times faster than standard BP (BP used the parameters η and α recommended by Rumelhart et al. (1986)). For the sake of comparison, standard BP without the momentum term is tested, which resulted in a convergence speed about 35 times slower than that of GGBP. As the stopping criterion becomes more stringent, the difference between GGBP and BP becomes more significant. This is no surprise as the GGBP uses an approximation scheme that is best in the neighborhood of the global minimum, while standard BP slows down when the error signal becomes small. Typical learning curves of both GGBP and BP are shown in Figure 1.