BACK-PROPAGATION:INCREASING RATE OF CONVERGENCE BY PREDICTABLE PATTERN LOADING

George N. Bebis, George M. Papadourakis
Department of Computer Science
University of Crete
Iraklion, Crete, GREECE
Michael Georgiopoulos
Department of Electrical Engineering
University of Central Florida
P.O. Box 25000
Orlando, FLA 32816

Abstract

In this paper we present a new procedure which increases the rate of convergence of the back-propagation algorithm without increasing its complexity. The Predict back-propagation algorithm takes into account the learning "difficulty" that each exemplar pattern imposes to the network in order to determine the proper sequence that the exemplar patterns must be presented to the network. Simulated results, from representative class problems, illustrate the efficiency and superiority of this procedure over the traditional back-propagation algorithm.

1 Introduction

Artificial Neural Networks, were inspired by the structure of the human brain-its nerve cells, the large number of interconnections and interactions among them-and by envy of what the brain can do. As their original model is in biological nervous systems so their main purpose is to make an artificial system to operate in the same way as the biological organisms operate. The idea of simulating the brain formed the foundation for much of the early work in Artificial Intelligence.

Especially in our days a great resurgence of interest on Artificial Neural Networks is a consequence of recent hardware advances in the construction of massively parallel machines ([1],[2]). A number of Artificial Neural Networks have recently received widespread attention as potential new architectures for computing systems.

These models appear to have the greatest application in the area of pattern recognition ([3],[4]), in speech ([5],[6],[7]) and in image processing ([8],[9],[10]), where high computational rates are necessary and many hypotheses can be explored in parallel. Furthermore, many attempts have been made to solve complex problems ([11],[12],[14]) with promising results.

Much of the current research in the field of Artificial Neural Networks is focused on the design of efficient learning algorithms to train an Artificial Neural Network to perform a specific task ([13],[18],[19]). One of the most popular learning algorithms is the "back-propagation" rule, developed by Rumelhart et. al ([13]). This rule has been applied to a great number of cases ([13],[15],[16]) and has been found to find good solutions to the corresponding problems. Although its performance can be excellent, many researchers still criticize the convergence rate of this algorithm. Many attempts have been made in the hope of achieving faster rates of convergence. S. Kung et. al. [17] have performed experiments to find the optimal number of hidden nodes per layer and the optimal number of synaptic weights between two adjacent layers but their conjectures do not hold in every application.

In this paper we propose a new method which we call the Predict back- propagation algorithm and it relates the convergence rate of the back- propagation algorithm with the sequence in which we must present the exemplar patterns to the network. We have tested our method on a representative class of problems. Our method decreases in all cases the number of learning sweeps without increasing the overall complexity of the algorithm. Thus, the timing requirements of the learning phase are greatly reduced, without sacrificing the precision of the retrieving phase. In the second section of the paper we present a special class of Artificial Neural Networks, the multilayer feed-forward Artificial Neural Networks. In the third section we make a brief overview of the back-propagation algorithm and in the fourth section we present the proposed new method. In the fifth section we test our algorithm on a representative class of problems. Finally, we present certain comparisons of our algorithm with other methods.

2 Multi-Layered Feed-Forward Artificial Neural Networks

Artificial Neural Networks are specified by the topology of the network, the characteristics of the nodes(i.e., neurons) and the processing algorithm. The pow-

erful information processing properties of an Artificial Neural Network arise from the above network specifics.

We are interested in the special class of Artificial Neural Networks, the multilayer feed-forward Artificial Neural Networks, which is illustrated in Figure 1. The topology of a multilayer feed-forward Artificial Neural Network is a structured hierarchical layered network. It consists of several distinct layers of nodes. Two of these layers play the role of the input layer and the output layer. Between the input and the output layer we may have one or more layers of nodes which are called hidden layers. Hidden nodes, the nodes in the hidden layers, are used to represent domain knowledge useful for the problem solving. Generally, each node in one layer is interconnected with all the nodes in adjacent layers with connections, known as synapses, as illustrated in Figure 1. Each connection is associated with a weight which measures the degree of interaction between the corresponding nodes. Nodes are retatively simple processing elements and the capabilities of multilayer feed-forward Artificial Neural Networks stem from the nonlinearities used within them.

The algorithms for multilayer Neural Net processing can be divided into two phases: retrieving and learning. In the retrieving phase of the algorithm, information flows from the input layer through the hidden layers to the output layer. Basically, in this phase the nodes update their own activation values based on the system equations. In the learning phase, an adaptation of the weights corresponding to the connection nodes takes place.

The actual task we want to perform is to teach the network to associate specific output states, called target states, to each of several input exemplar patterns by calculating the appropriate connection weights. The retrieving and learning phases act one after the other until the weights take their proper values needed by the network to perform the desired task.

3 An overview of the Back-Propagation Rule

The back-propagation rule is a powerful general learning algorithm employing a gradient-or "steepest"- descent method that enables a network to adapt in a way

that improves its performance over time. Specifically, it is an iterative algorithm which in each step adjusts the connection weights in the network, minimizing the mean-square error between the target value (the desired) and the output value (the actual) of the network. This error function is a surface in the weight space and is defined as follows:

$$E = 2^{-1} \sum_j (t_j - o_j)^2 \tag{1}$$

where the summation is performed over all output nodes indexed by j. We denote by t_j the target values and by o_j the output values.

Let us assume that we have an L-layered feed-forward Artificial Neural Net with N input nodes. The number of nodes in the hidden layers is N_l for $1 \leq l \leq L$. In our notation, the input layer is not counted as a layer. So an L-layered feed-forward Artificial Neural Net has $L-1$ hidden layers and the Lth layer is the output layer. Initially, we present the input data $x_1, x_2, \ldots, x_N$ and the corresponding target data $t_1, t_2, \ldots, t_{N_l}$. The input is propagated forward through the network, which computes the activation value for each output node. This the retrieving phase, which is described by the following equations:

$$u_i(l) = \sum_{j=1}^{N_{l-1}} w_{ij}(l) a_j(l-1) + \theta_i(l) \tag{2}$$

$$a_i(l) = f(u_i(l)) = (1 + e^{-u_i(l)})^{-1} \quad 1 \leq i \leq N_l \quad 1 \leq l \leq L \tag{3}$$

where $a_i(l)$ is the activation value of the ith node in the lth layer and it is broadcasted to all the nodes of the above layer $(l+1)$. In addition, $w_{ij}(l)$ corresponds to the weight associated with the connection that originates in the jth node of layer $l-1$, and terminates in the ith node of layer l. Note that $a_j(0) = x_j$ for $1 \leq j \leq N$. Furthermore, f denotes the nonlinear sigmoid activation function and $\theta_i(l)$ is an internal offset of the ith node in layer l.

The learning phase involves a backward pass through the network during which an error signal is passed to each unit in the network and appropriate weight changes are made. For each weight the gradient of the output error with respect to that weight is computed. Then the weight is changed in the direction that reduces the error. Specifically, the adjustment of the weights is performed according to

$$w_{ij}^{(m)}(l) = w_{ij}^{(m-1)}(l) + \eta \delta_i^{(m)}(l) a_j^{(m)}(l-1) \tag{4}$$

where η is the learning rate, usually a constant, and $\delta_i^{(m)}(l)$ is the error signal generated by the exemplar pattern presented in the mth iteration. If the ith node is an output node (i.e., $l = L$) then the error signal $\delta_i^{(m)}(L)$ is described as

$$\delta_i^{(m)}(L) = a_i^{(m)}(L)(1 - a_i^{(m)}(L))(t_i^{(m)} - a_i^{(m)}(L)) \tag{5}$$

Otherwise the ith node is an internal hidden node, and the error signal is described as

$$\delta_i^{(m)}(l) = a_i^{(m)}(l)(1 - a_i^{(m)}(l)) \sum_{j=1}^{N_{l+1}} \delta_j^{(m)}(l+1) w_{ji}^{(m-1)}(l+1) \tag{6}$$

for $l = L-1, L-2, \ldots, 1$. The error signals are propagated backward updating the connection weights in all the layers.

Internal thresholds, θ_i, should be learned just like any other weights. We simply imagine that θ_i is the weight from a node that it has always constant-valued inputs. It is easy to show that the internal thresholds can be adjusted according to

$$\theta_i^{(m)}(l) = \theta_i^{(m-1)}(l) + \eta \delta_i^{(m)}(l) \tag{7}$$

4 The Predict Back-Propagation Rule

In the original Back-propagation algorithm the exemplar patterns are presented to the network continuously, in a generally random way until the network is able to recognize them. Preliminary results have shown that certain exemplar patterns can be recognized faster by the Neural network than others. We feel that instead of having the network process all the available inputs in an equal manner it should be forced to process more frequently those presentations which are more difficult to be learned. Of course, in order to implement the above idea, a measure of the relative learning difficulty each exemplar pattern causes the network must be devised.

An intuitive example serves to illustrate the basic idea of our approach. This example presents a naive but realistic way of learning a set of unknown words. In teaching words to humans, a common method used, is to repeatedly expose these unknown words, one at a time, randomly or sequentially, to a person until it learns them. Experiments have shown that after some time, before completion of the

learning process, a person easily recognizes a subset of these words while the rest are more "difficult" to be recognized. Instead of repeating the process involving the whole set of words, we concentrate on the subset of the "difficult" words only. Using such an elimination method all words are learned in a faster way. In the above example, although all the words are learned, some of the words were exposed to the person more frequently than others and the frequency rate was proportional to the learning "difficulty" that each word imposed.

In order to increase the rate of convergence of the back-propagation algorithm we will present the exemplar patterns to the network in a sequence which depends on the learning "difficulty" that each exemplar pattern imposes to the network. The error signals associated with each exemplar pattern yields the measure of the learning "difficulty". Thus, we can predetermine the sequence to use as exemplar patterns by choosing at each current step the exemplar for which the error signals take their maximum values.

Let $\epsilon_i^{(m)}$ denote the error, associated with the ith output node of the mth exemplar pattern, which is defined as

$$\epsilon_i^{(m)} = t_i^{(m)} - a_i^{(m)}(L) \tag{8}$$

The total error $\epsilon^{(m)}$ can be calculated by choosing any vector norm of $\epsilon_i^{(m)}$'s over all the ouput nodes. For example, we can define $\epsilon^{(m)}$ by choosing the max-norm

$$\epsilon^{(m)} = \max_i \mid t_i^{(m)} - a_i^{(m)}(L) \mid \tag{9}$$

for each $m = 1, 2, \ldots, M-1$.

The rate of convergence can be incresed by increasing the learning rate of equation (4) as indicated by Rumelhart et. al ([13]). However this could lead to oscillation. The effect of our algorithm is that we maximize adaptively another parameter of this equation, namely $\delta_i^{(m)}(l)$, which depends indirectly on equation (8). Specifically, the proper exemplar which maximizes the error signal $\delta_i^{(m)}(L)$, given in equation (5) is chosen, and as a result $\delta_i^{(m)}(l)$, given in equation (6), is also maximized.

It should be noted that the weights change every time we load a new exemplar pattern. In order for our algorithm to function properly, an update of the error

vector of $\epsilon^{(m)}$'s must be performed each time we load an exemplar pattern. However, the computational cost of such an action is very high and it negatively affects the performance of the algorithm, although it results in a decreased number of learning steps. So, in order to improve performance we do not update the error vector each time a new exemplar is loaded into the network. The update is done in relation to the termination condition we have imposed in our algorithm, as shown in a later section. Such a strategy decreases the number of retrieving steps required for the updating by a large amount, without increasing seriously the number of learning steps.

5 Implementation of the Predict Back-Propagation rule

In order to evaluate the efficiency of our proposed algorithm we have experimented with three tasks which were chosen because their error surfaces possess a wide variety of terrains. These are the same tasks that were used by Jacobs in [18] in order to evaluate the performance of the heuristics that he has suggested. These tasks are the learning of the exclusive-or, multiplexer and binary-to-local functions. In order to obtain a proper comparison we also used the same network architectures for all the experiments. Both the input to the network and the output were given in binary notation. In each task the back-propagation rule was simulated with the sequence of the exemplars determined randomly (random back-prop). sequentially (sequential back-prop) and as described by our algorithm (predict back-prop). All the experiments were implemented on a Sun 4/110 Workstation system using C.

The learning rate that we used in all the experiments was kept constant at $\eta = 0.3$. In order to define $\epsilon^{(m)}$ we used the max-norm as given by equation (9). Our simulation results showed that this norm performs better than other norm definitions. The termination condition was kept the same in all the algorithms for comparison reasons. This condition determines that a task is solved when the error $\epsilon^{(m)}$ associated associated with exemplar pattern m for all $m = 1, 2, \ldots, M-1$, was less than a given threshold. Every time that a termination was true for some exemplar m, the rest of the exemplars were presented to the network in order to calculate the error associated with each exemplar pattern and to check whether the current set of weights satisfy the termination condition. In the case of the sequential and random back-prop there is no reason to pass all the exemplar patterns through the network. For these procedures, each time that the termination condition was

true for some exemplar the rest of the exemplars were passed through the network, one after the other, until for some exemplar the termination condition was false. This is a very logical procedure since it is not necessary to compute the errors associated with all the exemplar patterns, as we did with the predict back-prop.

We have run each task eight times. Each time the initial weight values were set to small random values and in order to have proper comparisons the same set of initial weights was used in each of the three algorithms. The results of our simulations are listed in Tables 2,4, and 6. These results contain both the number of learning sweeps (denoted sweeps in the Tables) and the number of retrieving sweeps. We define a learning sweep to be a pass of all the exemplar patterns through the network including the learning phase. Every retrieving sweep is defined as the pass of all the exemplar patterns through the network, excluding the learning phase. The reason that prompted us to discriminate between learning sweeps and retrieving sweeps is that during training the network may go through a retrieving phase without having to go through the learning phase (in the case where the termination condition for the input pattern is satisfied).

We expected that the number of retrieving sweeps in the case of the predict back-prop would be large due to the fact that we have to go through a retrieving sweep every time that we learn an input pattern. On the contrary, the simulation results show that both the number of learning sweeps and the number of retrieving sweeps are significantly reduced in the case of the predict back-prop algorithm. In the following section we present the analytical details of the three tasks involved in the simulated studies.

5.1 Exclusive-or Task

The exclusive-or problem is a classical problem which requires two hidden units to be solved. Many difficult problems involve an exclusive-or problem as a subproblem. The inputs and the desired outputs of the exclusive-or problem are defined in Table 1.

The network arcitecture for this task is illustrated in Figure 2. Table 2 presents the results of the exclusive-or task. The threshold value for the termination condi-

tion was set to 0.1. The simulation results show that the sequential back-prop and the random back-prop perform about the same. The predict back-prop outperforms both of them (by a factor of between 5 and 27 to 1 for the number of retrieving sweeps and by a factor of between 1.2 and 3 to 1 for the number of learning sweeps).

5.2 Multiplexer Task

The multiplexer task consists of six input nodes and one output node. Two of the input nodes act as a switch. Depending upon the number of values loaded to the other four nodes the switch dertermines the output value of the network. The inputs and the desired outputs in the multiplexer task are shown in Table 3, where the terms **x** and **a** denote a binary value.

The network architecture for this task is illustrated in Figure 3, and Table 4 presents the results for the multiplexer task. In this task we have also set the threshold value to 0.1. We observe again that the sequential back-prop and the random back-prop perform about the same and as expected the predict back-prop shows excellent results.

5.3 Binary to Local Task

This the most challenging task and it consists of the proper conversion from a distributed representation, over three nodes, into a local representation over eight nodes. The inputs and the desired outputs for the binary-to-local task are defined in Table 5.

The network architecture for this task is illustrated in Figure 4. This network arcitecture produces an error surface with sharp ravines. Thus, it is an interesting problem to test learning algorithms. Because of the learning difficulty imposed by this architecture the number of learning sweeps increase sharply in all cases. In Table 6, we present the results for the binary-to-local task. Here we have set a threshold value of 0.3. The symbol "*" means that the associated algorithm did not converge after 200000 learning sweeps. Although the Predict back-propagation algorithm converges in all cases, our experimental results have shown that for smaller threshold values, less than 0.3, the sequential and random back-prop algorithms did

not converge at all. As Table 6 indicates, the predict back-prop exhibits once more the best performance.

6 Comparison with other techniques

In this section we will discuss the advantages of our algorithm compared to other attempts to increase the convergence rate of the back-propagation algorithm. S. Kung et. al. ([17]), in order to reduce the number of learning sweeps, attempted to define the optimal number of hidden units per layer experimentally. Based on results from random simulations, they claimed that the optimal number of hidden units for efficient learning is equal to $M - 1$, where M is the number of exemplar patterns used. Results from our simulations, showed that S. Kung's conjecture does not hold in general.

R. Jacobs ([18]) attempted to increase the rate of convergence of the back-propagation rule by proposing the following heuristics:

i) Every parameter of the performance measure to be minimized should have its own individual learning rate. In our case the parameters of the performance measure are the weights, so we should associate an individual learning rate for each weight.

ii) Every learning rate should be allowed to vary over time. So weights should be adapted by their learning rates varying over time.

iii) When the derivative of a parameter possesses the same sign for several consecutive time steps, the learning rate for that parameter should be increased.

iv) When the sign of the derivative of a parameter alternates for several consecutive time steps, the learning rate for that parameter should be decreased.

The above heuristics can often be of value and their basic ideas have been used in many problems in numerical analysis. We believe that the associated algorithms using these heuristics can yield fewer learning sweeps than the classical back-propagation rule. Altough fewer learning sweeps are required, the overhead associated with the computation of a different learning rate for each weight is time

consuming. In addition, the adjustment of each learning rate, which is based on both the partial derivatives of the error with respect to the weights and on an estimate of the curvatures of the error surface, imposes further computational complexity to these algorithms. We believe that the additional computational compexity of these algorithms should be taken into consideration before a fair comparison with the back-propagation or the predict back-prop algorithms is conducted.

Let us now examine the additional cost imposed by our algorithm to the timing and space requirements of the back-propagation rule. The only additional space demand on memory required by the algorithm is an M-dimensional vector which is needed for the storage of the errors $\epsilon^{(m)}$. Periodically this vector must be updated according to our policy, as discussed in section 5. This timing overhead has been calculated and presented in the simulation results (the number of retrieving sweeps). In addition, minimal time is required for the periodical computation of the maximum error.

7 Conclusions

We presented a new algorithm which we called the predict back-propagation algorithm. This algorithm increases the rate of convergence of the back-propagation algorithm without increasing its computational complexity. We simulated the proposed algorithm with two traditional implementations of the back-propagation algorithm, the sequential and the random back-prop, over a representative class of learning problems. The predict back-propagation algorithm showed great efficiency and robustness in all cases.

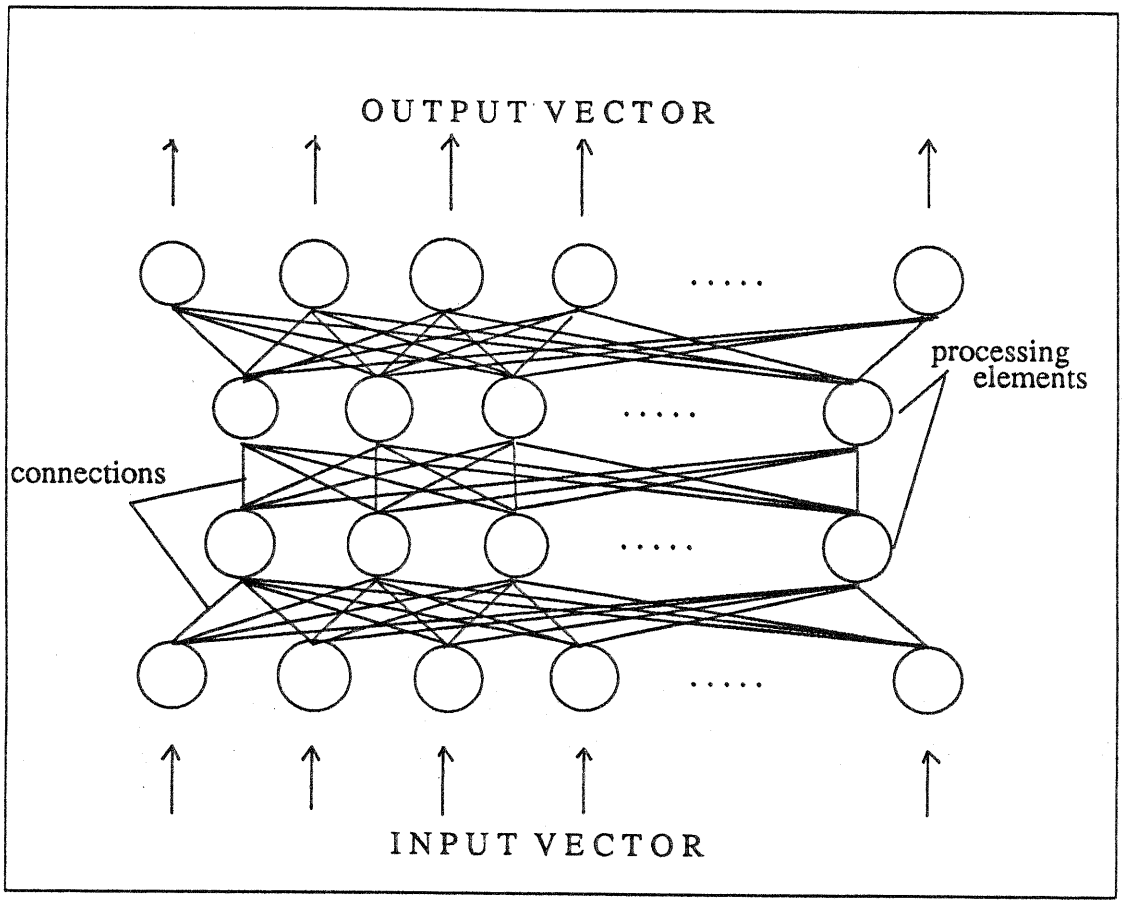

FIGURE 1. A multilayer feed-forward Artificial Neural Network

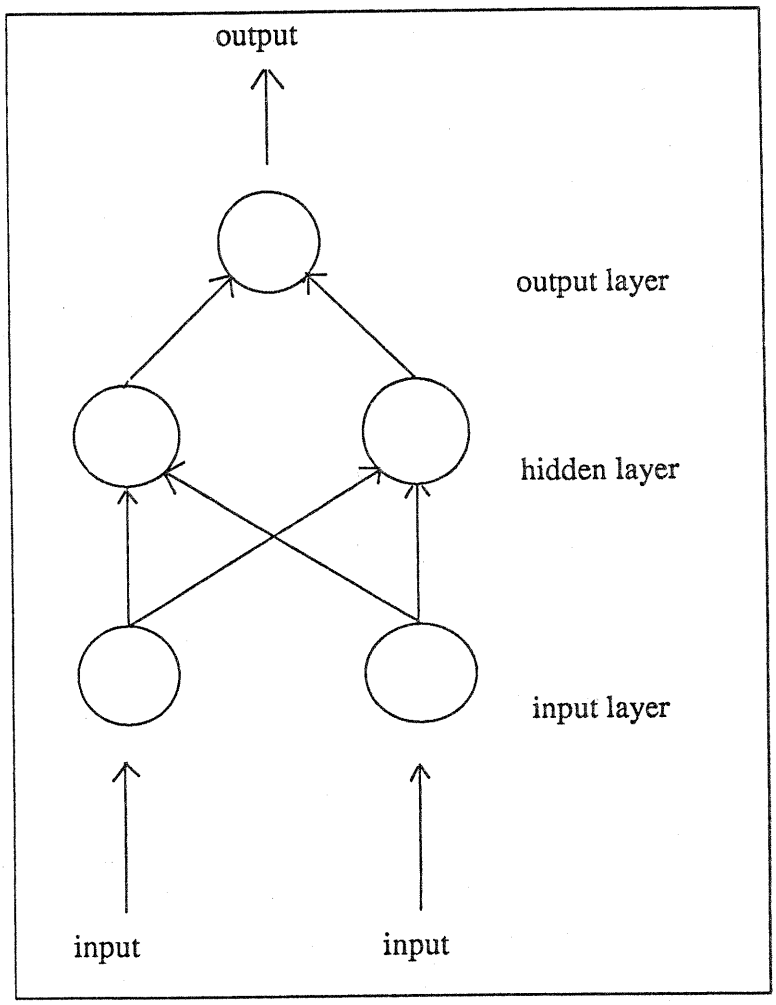

FIGURE 2. Network architecture for the exclusive-or task

TABLE 1. Input Output data for the exclusive-or task

Inputs		Output
1	1	0
1	0	1
0	1	1
0	0	0

TABLE 2. Results from the simulation of the exclusive-or task

Simulation Results									
sequent.	sweeps	7390	8680	2784	5773	7457	4279	7437	7341
	retriev.	562	840	154	499	566	495	560	561
random.	sweeps	7392	8721	2799	5773	7458	4278	7437	7341
	retriev.	562	857	154	499	566	495	560	561
predict.	sweeps	3427	2932	2286	3031	3554	3277	4335	3876
	retriev.	26	39	36	50	21	48	90	26

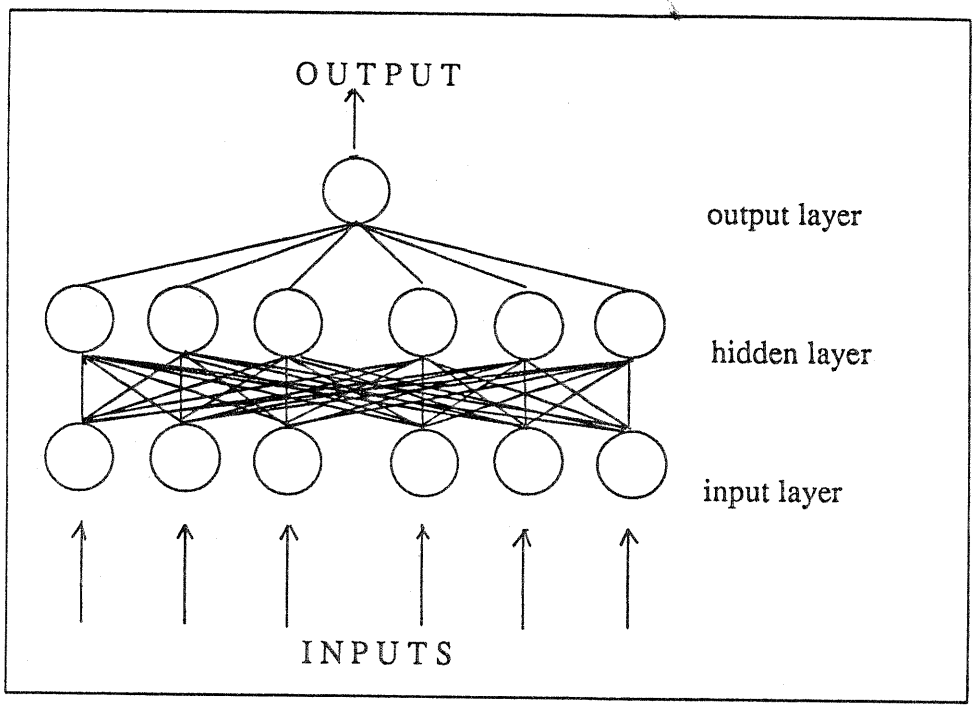

FIGURE 3. Network architecture for the multiplexer task

TABLE 3. Input Output data for the multiplexer task

Inputs						Output
x	x	x	a	0	0	a
x	x	a	x	0	1	a
x	a	x	x	1	0	a
a	x	x	x	1	1	a

TABLE 4. Results from the simulation of the multiplexer task

Simulation Results									
sequent.	sweeps	682	454	492	456	465	411	465	475
	retriev.	461	235	295	257	250	243	268	283
random.	sweeps	682	454	492	457	465	413	465	475
	retriev.	461	235	295	258	250	244	268	283
predict.	sweeps	290	272	251	243	253	243	253	252
	retriev.	3	6	4	4	4	9	3	4

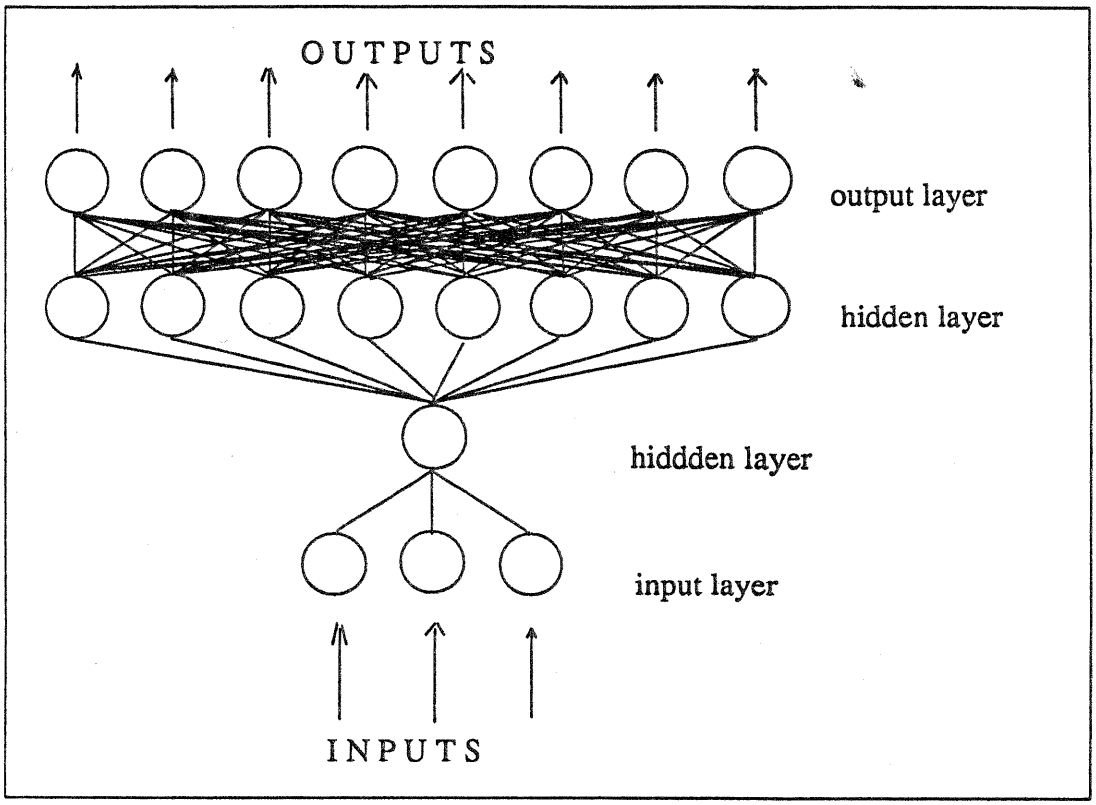

FIGURE 4. Network architecture for the binary to local task

TABLE 5. Input Output data for the binary to local task

Inputs			Output							
0	0	1	1	0	0	0	0	0	0	0
0	1	0	0	1	0	0	0	0	0	0
0	1	1	0	0	1	0	0	0	0	0
1	0	0	0	0	0	1	0	0	0	0
1	0	1	0	0	0	0	1	0	0	0
1	1	0	0	0	0	0	0	1	0	0
1	1	1	0	0	0	0	0	0	1	0
0	0	0	0	0	0	0	0	0	0	1

TABLE 6. Results from the simulation of the binary to local task

Simulation Results									
sequent.	sweeps	16336	*	*	17203	*	*	*	15276
	retriev.	18756	*	*	27661	*	*	*	11554
random.	sweeps	*	*	22361	*	31803	*	*	16433
	retriev.	*	*	55334	*	84524	*	*	18940
predict.	sweeps	9275	11149	9781	10996	9742	8922	9025	11018
	retriev.	2523	5444	2304	2893	2767	2110	2021	2702

References

[1] T. Williams,"Object recognition opens the eyes of machine-vision systems," *Computer Design*, May 1988.

[2] H. P. Graf, L. D. Jackel and W. E. Hubbard, "VLSI implementation of a neural network model," *Computer*, March 1988.

[3] R. P. Lippman, "An introduction to computing with neural networks," *IEEE ASSP Magazine*, April 1987.

[4] B. Widrow, R. G. Winter and R. A. Baxter, "Layered neural nets for pattern recognition," *IEEE Transactions on Acoustics Speech and Signal Processing*, Vol.36, No.7, July 1988.

[5] R. P. Lippman, "Neural network classifiers for speech recognition," *The Lincoln Laboratory Journal*, Volume 1, 1988.

[6] T. J. Sejnowski and R. C. Rosenberg, "NETtalk:a parallel network that learns to read aloud," *NEUROCOMPUTING foundations and research*, editors J. A. Anderson and E. Rosenfeld, MIT Press, Cambridge, Massachussetts, 1988.

[7] T. Kohonen, "The neural phonetic typewriter," *Computer*, March 1988.

[8] T. Kohonen. Self-organization and associative memory. Springer-Verlag 1988.

[9] J. Hutchinson, G. Koch, J. Luo and C. Mead, "Computing Motion using analog and binary resistive networks," *Computer*, March 1988.

[10] Y. Thou, R. Chellappa, A. Vaid and B. K. Jenkins, "Image restoration using a neural network," *IEEE Transactions on Acoustics Speech and Signal Processing*, Vol. 36, No.7, July 1988.

[11] J. J. Hopfield and D. W. Tank, "Neural computation of decisions in optimization problems," *Biological Cybernetics*, 52, pp. 141-152, 1985.

[12] M. Takeda and W. Goodman, "Neural networks for computation: number representation and programming complexity," *Applied Optics*, Vol. 25, No. 18, September 1986.

[13] D. E. Rumelhart, J. L. McClelland, and the PDP Research Group. Parallel Distributed Processing (PDP), Explorations in the Microstructure of Cognition, Volume 1:Foundations. MIT Press, Cambridge, Massachusetts, 1986.

[14] D. E. Rumelhart, J. L. McClelland and the PDP Research Group. Parallel Distributed Processing (PDP), Explorations in the Microstructure of Cognition, Volume 2: Psychological and Biological Models. MIT Press, Cambridge, Massachusetts, 1986.

[15] D. Burr, "Experiments on neural net recognition of spoken and written text," *IEEE Transactions on Acoustics Speech and Signal Processing*, Vol. 36, No.7, July 1988.

[16] S. Y. Kung and J. H. Hwang, "Neural net architectures for robotic applications," *IEEE Journal of Robotics and Automation*, special issue on Computational Algorithms ans Architectures in Robotics and Automation, 1989.

[17] S. Y. Kung, J. H. Hwang and S. W. Sun, "Efficient modeling for multilayer feedforward neural nets," *Proceedings of ICASSP 1988*, Volume 4, pp. 2160-2163, New York, NY 1988.

[18] R. A. Jacobs, "Increased Rates of Convergence through learning rate adaptation," *Neural Networks*, Vol. 1, pp. 295-307, 1988.

[19] J. F. Shepanski, "Fast learning in artificial neural systems: Multilayer perceptron training using optimal estimation," *Proceedings of the ICNN*, San Diego, CA, pp. I-465–I-472, September 1988.