DYNAMIC NODE CREATION IN BACKPROPAGATION NETWORKS

Timur Ash

February 1989

ICS Report 8901

Institute for Cognitive Science
University of California, San Diego
La Jolla, California 92093

I would like to thank Adam Harris for his untiring help and dedication to this project. Gary Cottrell's guidance and feedback from the very beginning of this work was valuable and very much appreciated. I am also grateful to Robert Hecht-Nielsen for a sense of perspective on the field and his enthusiastic support. A special note of thanks to Raul Rathmann for his help with the dirty work of writing the simulator.

I would like to extend my appreciation to the following people for reviewing the original manuscript: Rik Belew, Steve Biafore, Laurette Bradley, Jim Chen, C.K. Cheng, Jeff Elman, Karen Haines, David Hutches, John McInerney, Margaret Myers, Mark Plutowski, Pat Simpson, Victor Vianu, and Hal White. Their comments and care resulted in a much improved final product. Requests for reprints should be sent to the Institute for Cognitive Science, C-015; University of California, San Diego; La Jolla, CA 92093.

Dynamic Node Creation in Backpropagation Networks

TIMUR ASH

Large backpropagation (BP) networks are very difficult to train. This fact complicates the process of iteratively testing different sized networks (i.e., networks with different numbers of hidden layer units) to find one that provides a good mapping approximation. This paper introduces a new method called dynamic node creation (DNC) that attacks both of these issues (training large networks and testing networks with different numbers of hidden layer units). DNC sequentially adds nodes one at a time to the hidden layer(s) of the network until the desired approximation accuracy is achieved. Simulation results for parity, symmetry, binary addition, and the encoder problem are presented. The procedure was capable of finding known minimal topologies in many cases, and was always within three nodes of the minimum. Computational expense for finding the solutions was comparable to training normal BP networks with the same final topologies. Starting out with fewer nodes than needed to solve the problem actually seems to help find a solution. The method yielded a solution for every problem tried. BP applied to the same large networks with randomized initial weights was unable, after repeated attempts, to replicate some minimum solutions found by DNC.

INTRODUCTION

The ideas and results of this paper grew out of the process of training large backpropagation networks. A key design difficulty is determining the number of hidden nodes necessary to perform an accurate mapping. Formal mathematical analyses have been unable to shed much light on the question of what constitutes a minimum network size or configuration for a problem. Because of this, large network architectures are often found by trying several different topologies.

Provisions must be made for the evolution of a network's architecture during training. This is important from the practical standpoint of giving networks more flexibility in adapting to their function and finding a solution. It is also of interest from a biological point of view. Until now, fixed neural net architectures have been unsuitable for modeling the changing topology of the brain over time (Aoki, Chiye, Siekevitz, & Philip, 1988). This issue of changing architecture did not arise with perceptrons (Minsky & Papert, 1988) because the input and output layers permanently and completely specified the topology of the net. But the development of backpropagation (Rumelhart & McClelland, 1986; Werbos, 1974), with its *hidden layer(s)*, makes it possible to change not only the values of the weight and bias vectors, but their dimensions as well.

For the most part, the selection of an architecture remains difficult. Configurations are tried, and if they do not yield an acceptable solution, they are discarded. Another topology is then defined and the whole training process repeated. The possible benefits of training the previous network are lost.

In most cases, architectures are discarded for one of two reasons. If the topology chosen is too small (in an information capacity sense), the desired input to output mapping simply cannot be learned with satisfactory accuracy. Conversely, if a network is too large and the training data is presented too many times, it will learn the training set correctly, but will fail dismally when presented with novel inputs. This is

partially due to the fact that the network has *overfit* the data (i.e., the network has focused on the statistical quirks of the limited training set). Instead of learning salient regularities of the underlying relationship, the network has learned to rely on unique features of the input patterns to distinguish among them. In extreme cases, the network simply remembers all of the patterns it is exposed to and responds poorly to novel inputs.

Ideally, what is desired is a network large enough to learn the mapping, and as small as possible to generalize well (Huyser & Horowitz, 1988). There are two general approaches to finding such networks. One involves using a larger than needed topology and training it until a mapping is found. After this, elements of the network are *pruned off* if they are not actively being used (e.g. if the value of a particular weight remains very close to zero, it can be eliminated without affecting the network). By this process of pruning and fine-tuning, the network is reduced in size. Rumelhart's work in this area (Rumelhart, 1988) seeks to minimize a function of the error on the output nodes and also of the number of nodes and weights. The other approach starts with a small network and *grows* additional nodes and weights until a solution is found. An example of node addition during training in other neural network models can be seen in the restricted coulomb energy (RCE) networks used for pattern classification tasks (Reilly, Scofield, Elbuam, & Cooper, 1987).

There are some shortcomings to the pruning approach. Since the majority of the training time is spent with a network that is larger than necessary, this method is computationally wasteful. In practice, since the solution is not known, networks much larger than required are often chosen as starting points (Sietma & Dow, 1988). In addition, many networks with different topologies are capable of implementing the same mapping. Since the pruning approach starts with a large network, it may get stuck in one of the intermediately sized solutions because of the shape of the error surface and never find the smallest solution. Also, the relative importance of nodes and weights depends on the particular mapping problem and makes it difficult to come up with a cost function that would yield small networks for arbitrary mappings.

There are two ways to train networks that grow additional nodes or weights dynamically. In both approaches, activation is spread forward through the whole network (including the newly added elements). On the backward pass, one calls for *freezing* the existing network and training only the new element(s) via backpropagation. The other approach allows complete retraining of the whole network.

The advantage of the freezing method is the relatively small retraining effort required when a new element is introduced. However, the method does not, in general, find the desired solution. When an extra degree of freedom is introduced (e.g., by adding a new weight to the network), holding the existing network values constant only allows finding the solution in an affine subset of the weight space (see Figure 1). Additional degrees of freedom (extra nodes or weights) can be introduced to allow this affine subset to pass through a global minimum point (in effect, shifting some of the previously fixed coordinates). But because of the partial duplication of effort involved (i.e., two or more weights are needed to determine the value of a single dimension), such a network cannot be minimal in size.

The remaining alternative allows retraining of the whole network after a new element is added. The dynamic node creation (DNC) method was developed to add nodes to the hidden layer(s) of the network during training. After a new node is grown with this procedure, regular BP training takes place until the desired mapping is learned, or another node needs to be added. The topologies of the networks studied along with the criteria for adding new elements are discussed below.

THE MODEL

The general class of network architectures modeled had the following features. All connections between nodes were feedforward from each layer to the one above it. No lateral or backward connections were allowed, nor connections that skip over layers. Nodes were completely interconnected to each node in the layer above. The *logistic* activation function (Rumelhart, Hinton, & Williams, 1985) was used at all layers above the input:

$$Output\ (I) = \frac{1}{1 + e^{-I}}. \tag{1}$$

An additional restriction imposed on all of the functions we tested was the use of only one layer of hidden units. This architecture does not limit the the class of mappings that can be found by the network. In fact, recent results (Hecht-Nielsen, 1988; Hornik, Stinchcombe, & White, 1988) have shown that feedforward nets with three layers can model any function of interest to an arbitrarily selected precision with a finite number of nodes in the hidden layer. However, these powerful theoretical results leave unanswered the

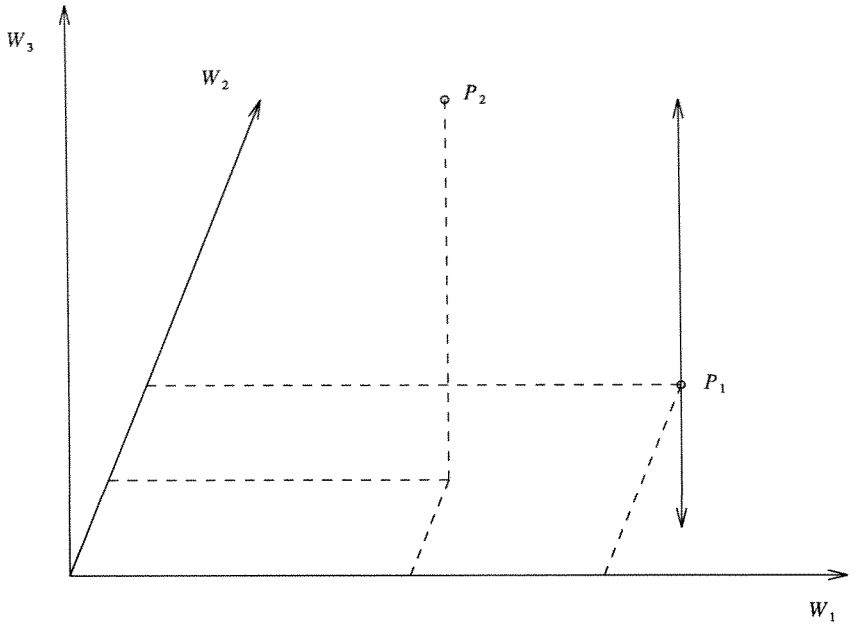

FIGURE 1. P_1 represents the point of lowest error in the plane defined by $W_1 \times W_2$. If another dimension is introduced (W_3), the global best becomes P_2. But freezing the values along W_1 and W_2 only allows solutions along the line through P_1 to be found. If we generalize this to higher dimensions; only solutions in a particular *affine subset* of the weight space can be found.

question of just how many nodes are needed for a particular problem and whether the BP procedure can learn it. Also, there is still an open question concerning the advantages of multi-hidden layer architectures. Even though such topologies are no more powerful in theory, in practice they have often resulted in much smaller (and sparser) networks that converge to a solution very quickly. This is especially true of some networks that use higher-order (sigma-pi) units (Plutowski, 1988; Rumelhart & McClelland, 1986).

Under what conditions should additional hidden nodes be created? This central question can be answered by making some observations about typical learning behavior of backpropagation nets. The overall error as a function of time, as measured using the training data, usually decreases monotonically. For a given learning rate, the error tends to reach a plateau. In fact, these error curves often resemble decaying exponentials (see Figure 2; see Table 1 for a definition of the variables involved). It quickly becomes clear that a particular network is incapable of learning a mapping when the error curve begins to inch slowly downward. Decreasing the learning rate (α) at this point usually leads to diminishing returns.

What is needed is a way to detect the flattening of the average error curve at unacceptably high values. When this event occurs, a single new node should be added to the hidden layer. The ratio of the drop in squared error over the last w trials to the squared error when the last node was added is computed. When this value falls below a user defined *trigger slope* Δ_T, a new node is added.

Mathematically expressed, a new node should be added if both of the following conditions are met:

$$\frac{|\, a_t - a_{t-w} \,|}{a_{t_0}} < \Delta_T \tag{2}$$

and

$$t - w \geq t_0. \tag{3}$$

Equation 2 detects when the error curve has flattened out to an unacceptable level. Equation 3 guarantees that all of the error terms in Equation 2 deal with the same topology. Another consequence of this is that a new node can only be added after at least w trials have gone by.

When a new node is added to the hidden layer, it is completely connected to all nodes in the layer above it. All nodes in the layer below it are also connected to it. This maintains the complete forward and back interconnection required.

Another important consideration is when to turn off the node growing. This should be done when when the desired mapping is learned to a user specified precision. Often, additional training with a separate large

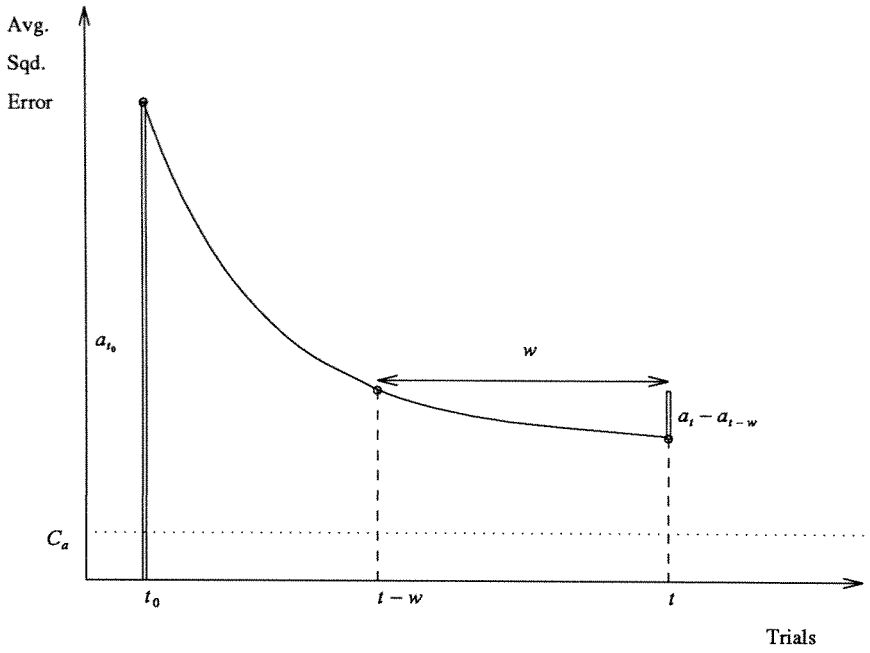

FIGURE 2. Addition of a single new hidden node is triggered when a flattening of the average squared error curve has been detected.

TABLE 1. DEFINITIONS OF VARIABLES INVOLVED IN DNC

Name	Definition
t_0	The time (in number of training trials) when the last node was added to the hidden layer.
t	The current time.
w	Width of window (in trials) over which trigger slope is determined.
a_t	Average squared error at time t (per node).
m_t	Maximum squared error at time t on any output node in any pattern.
C_a	The desired cutoff for average squared error.
C_m	The desired cutoff for maximum squared error.
Δ_T	Trigger slope - A measure of flatness of the error curve below which a new node should be added.

test set is used to gauge the acceptability of network performance. Once the desired level of performance has been reached, it is important to prevent further node growth. If this is not done, the algorithm will continue to add nodes simply to get more precision on the outputs. After node growth is disabled, normal BP fine tuning can still be carried out to improve precision. The following relations determine when to stop adding nodes (Note: both C_a and C_m are user specified.):

$$a_t \le C_a \quad \text{and} \quad m_t \le C_m. \tag{4}$$

These two constants allow separate desired precision levels to be set for the overall and worst case errors.

EXPECTATIONS

There were several issues of concern when the DNC technique was developed. Primary among them, of course, was whether this procedure could find good networks at all. If they could be found, were they small? The hope was that minimal solutions (in terms of the number of nodes) could be found. For problems where a lower bound solution was not known, it was hoped that DNC would at least produce a reasonable network topology. Another issue concerned the assumption of a monotonically decreasing average error curve. To deal with this, the absolute value bars were introduced in Equation 2. But it was not clear what effect a nonmonotonic average error curve would have on the triggering of new nodes. Along the same lines, it was anticipated that the introduction of a new node (and its associated weights) would upset the balance of the current network and actually cause a substantial rise in the average error curve before proceeding down to some lower value. This *hump effect*'s magnitude and duration may have made the retraining expense of adding a new node prohibitive. In other words, if the network had to relearn a completely new representation with the introduction of each new node, such a growing procedure would not be wise. The more general issue of computational expense was also a concern. DNC was expected to take longer than training an equivalent BP network with the same final topology (assuming that the correct best size is known in advance). It was not clear whether this cost difference would be large or small. The total computational burden of finding and training an appropriate network by use of DNC was expected to be much lower than the usual trial and error approach. This also needed to be verified.

IMPLEMENTATION AND TEST PROBLEMS

The simulator for this investigation was written in standard C and ran on a multiuser Sun 4 under the UNIX operating system. All weights were initialized to small random values ranging from -0.1666 to +0.1666. Example problems used were culled from Rumelhart and McClelland (1986; Ch. 8). These included the encoder problem, symmetry, parity, and binary addition with carry. For each run, the network was started with one node at the hidden layer. A constant learning rate of .5 and a momentum value of .9 were used. The trigger slope value used throughout was .05. The values of C_a and C_m were .001 and .01 for all runs. Table 2 summarizes the problems tested along with known upper bounds on the number of nodes needed in the hidden layer.

Representatives of the above classes of problems were chosen for investigation. In each case, the number following the name refers to the N mentioned in the input column of Table 2. The problems were ENC16, SYM4, SYM6, PAR2 (XOR), PAR4, PAR5, PAR6, ADD2, and ADD3. An exhaustive set of possible input data was used to train each network. One complete presentation of the data was considered one time trial.

RESULTS

The basic validity of dynamic node creation was demonstrated with all of the test problems. In *every* run, a solution to the problem at hand was discovered. Reasonably small solutions were found in all cases. In fact, minimal solutions were found for all problems with known upper bounds (see Table 2). In the case of ENC16, the problem was solved with three hidden nodes instead of the anticipated four. Not every run resulted in a minimal topology. One run (out of eight) for PAR6 required nine nodes instead of six to learn the mapping, and another required seven. Similarly, PAR2 required three nodes instead of the minimal two during two out of five attempts, despite using a value of .01 (instead of .05) for the trigger slope. This extra growing is partly due to an inappropriate selection of Δ_T and w. If regions containing local and global minima are separated in weight space, more hidden units may be "permuted" (White, 1987) to yield multiple global minima and facilitate the finding of a solution. This may help to explain why networks slightly larger than the minimum size often have an easier time finding solutions. ADD2 was consistently solved with four nodes, and ADD3 was solved with seven. These results should be examined to see if they represent minimal solutions for one hidden layer binary addition.

Computational expense was competitive with learning using BP networks that had the "correct" number of hidden nodes to start. Since expense grows geometrically with increasing network size, DNC benefits by initially working with smaller networks. A comparison of training expense with equivalent BP networks is presented in Figure 3 below. As can be seen, DNC networks were competitive with BP and yielded a solution to the problem of finding a good topology in the bargain.

TABLE 2. TEST PROBLEMS ALONG WITH EMPIRICAL UPPER BOUNDS ON THE NUMBER OF HIDDEN LAYER UNITS

Name	Input	Output	Known Solution (# of hidden units)
Encoder Problem (ENC)	N bit binary vector with 1 bit on	Same as input	$\log_2 N$
Symmetry (SYM)	N bit binary vector	1 if symmetric, 0 if asymmetric	2
Parity (PAR)	N bit binary vector	1 if # of 1's is odd, 0 otherwise	N
Binary Addition (ADD)	Two N bit binary vectors	N bit result and 1 carry bit	None known for one hidden layer

It was anticipated that DNC would be more expensive than BP. But even in the worst case, the difference was only forty percent. In some cases, the growing procedure was actually *cheaper* than BP. These results were obtained despite naively starting the DNC networks with one hidden node each time. In practice, networks for more complicated tasks can be initialized with a larger number of hidden nodes and allowed to grow from there. It is also important to note that BP was unable to find smallest topology solutions for several of the problems despite repeated attempts and extended learning times. BP networks were tried with learning rates of .1, .25, and .5 and allowed to run for 50,000 trials.

The assumption of monotonically decreasing average error held true for most of the test problems. When it did not, the spikes in the error curve did not cause a problem in triggering the growth of new nodes (see node 4 in Figure 4). The only problem with allowing the absolute value bars in Equation 2 can arise when the average error curve consistently rises. Under such conditions, new node growth would not be

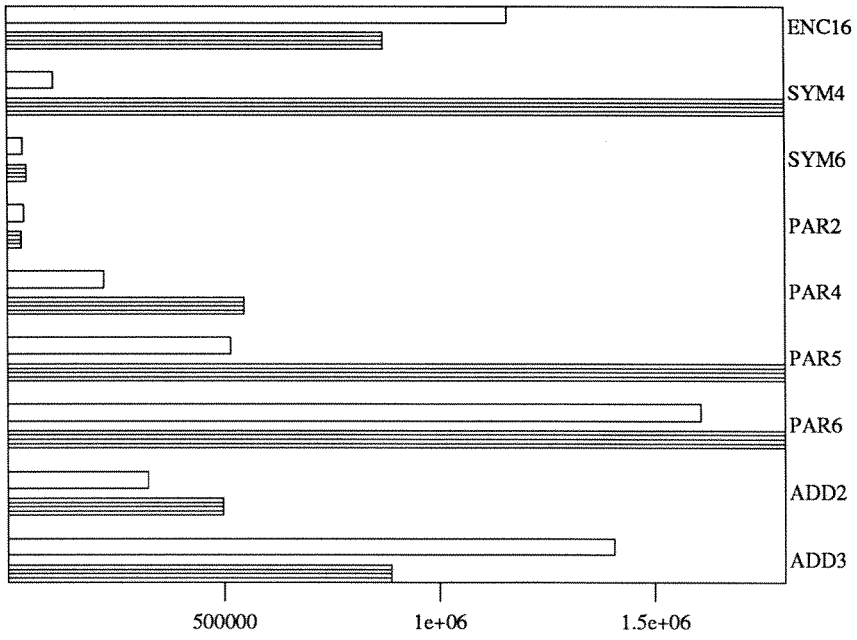

FIGURE 3. Computational effort (in number of floating point multiplications) required to learn the mapping using DNC and normal BP with the same final topology. Results shown are for smallest topologies found by DNC. Light bars are for DNC, dark ones for BP. Note: BP solutions for SYM4, PAR5, and PAR6 were not found after 50,000 trials.

triggered even though performance is getting worse. In practice, it is unlikely that this condition can persist for very long (certainly not on the same time scale as w), and should not cause problems.

The predicted hump effect did not materialize. When a new node is added, the weights connecting it to the rest of the network are small. Active weights in the existing network are usually at least an order of magnitude larger. The new node neither helps nor hinders the activity of the existing network initially. The change occurs smoothly and usually causes the average error curve to continue downward (sometimes after a very small glitch).

DISCUSSION

It is important to understand the effect of training the smaller networks on the way to finding the larger solution. This can be evaluated by considering how many trials (complete presentations of the data set) the DNC network spent with its largest architecture before the mapping was learned. If lower dimensional training led it away from the actual solution, this number could be expected to be larger than the corresponding time for training a regular BP network with the same topology. If the training had no effect (resulted in an essentially random distribution of the weights), the number should be about the same as the BP training time. In fact, in every case tested, DNC networks spent less time training at maximum size than their BP counterparts (see Figure 5). This indicates that the initial training in lower dimensional space actually helped to find a solution. This is also supported by the fact that minimal solutions for SYM4, PAR5, and PAR6 were found by growing the networks, but could not be found after repeated attempts with normal BP learning. This is partly due to the fact that optimization is easier in lower dimensional spaces.

The *infinity norm* m_t is an important criterion for determining when a mapping has been learned. Concentrating on the values of the average error curve a_t can often give misleading information about the performance of the network. In the test problems selected it was very common for the network to get one output node in at least one pattern completely wrong in order to drive down the overall average error. This "sacrificing one for the good of the many" behavior can be uncovered by watching m_t (see Figure 4). Low levels of a_t and m_t should be the indicators of good network performance. This goal may not be possible for all problems. It will only work in an essentially noise-free, deterministic environment. When noise is present, attainable values for C_a and C_m may be impossible to determine at the start of the

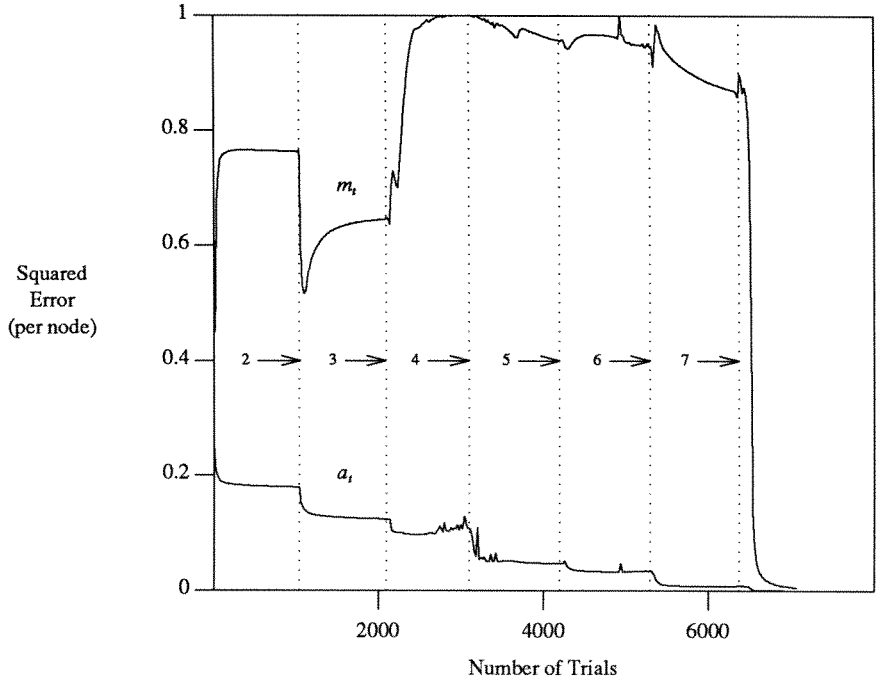

FIGURE 4. Squared error graph for ADD3. Dashed vertical lines indicate the creation of new nodes. Node growing was unaffected by the presence of spikes in the a_t curve (see node 4). The hump effect was negligible.

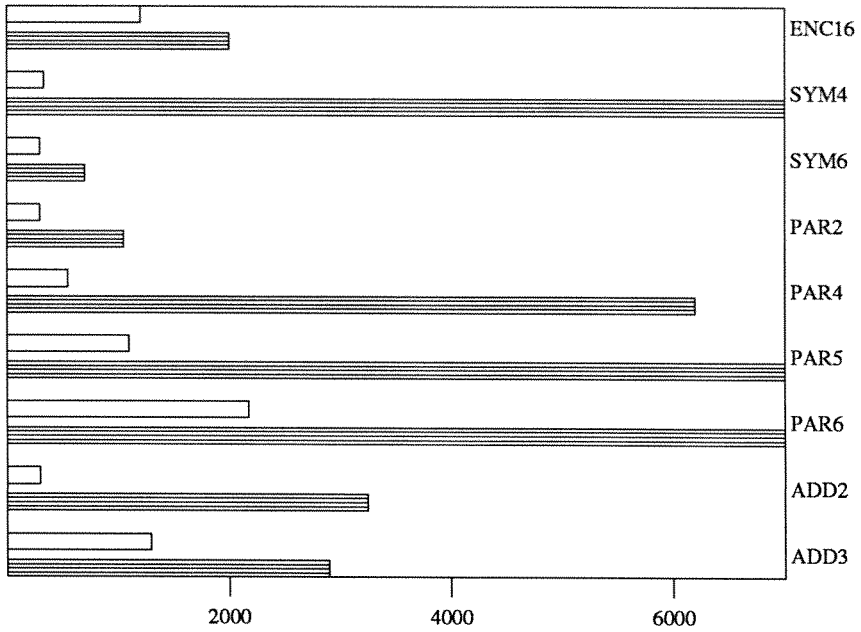

FIGURE 5. Number of trials spent with the final architecture before mapping is learned. Light bars are for DNC, dark ones for BP. In all cases lower dimensional DNC training helped to find a solution. Note: BP solutions for SYM4, PAR5, PAR6 were not found after 50,000 trials.

simulation. Noisy data may also result in some kind of inverse relationship between a_t and m_t (i.e., forcing the average error down may cause the maximum error to rise, and visa versa).

It is also important to note that the average error curve should not be calculated using the training examples. Another *test set* (drawn from the same population as the training set) should be used to calculate the error values. This issue was not a problem for the boolean functions tested because a complete training set was used in each case. Since the training examples represented the whole universe of possible data points, the network was learning an exact relationship. In general, the network's training should be stopped at the point when the average squared error curve (based on the test set) begins to rise. Past this point, overtraining will occur, and the network's ability to generalize will degrade.

In at least one case (PAR4), similar error curve behavior was observed for DNC and BP networks. The shape of the curves in Figure 6A is basically a squash of those in Figure 6B. The first 2000 trials for both networks indicate a_t and m_t values around .25, corresponding to error of .5 on the output node. This indicates that only the average value of the parity function has been learned (the fact that it is on half of the time and off the other half). This is understandable for the DNC network, because it may not be able to model more complex functions with its simple initial architectures. But BP makes the same crude approximation first. It tries the simplest (linear) solution before going on to the correct one, even though all four hidden nodes and associated weights are available from the very start.

It is still not clear whether DNC finds solutions similar to those found by regular BP. Although the above graphs seem to indicate that this is the case, we have not investigated it yet. A similar question concerns generalization. If the types of networks found by DNC are different from those found by BP, do they generalize better? There was no opportunity to test this with the current problems because a complete training set was used for each one.

OPEN QUESTIONS

Several issues are raised by this investigation and merit further attention. One question involves the width of the history window being used. For the boolean problems tested, a width of 1000 trials worked very well. However, on some runs, this was insufficient in preventing the growth of extra nodes beyond the known minimum. This will continue to be a problem because BP error curves can often seem

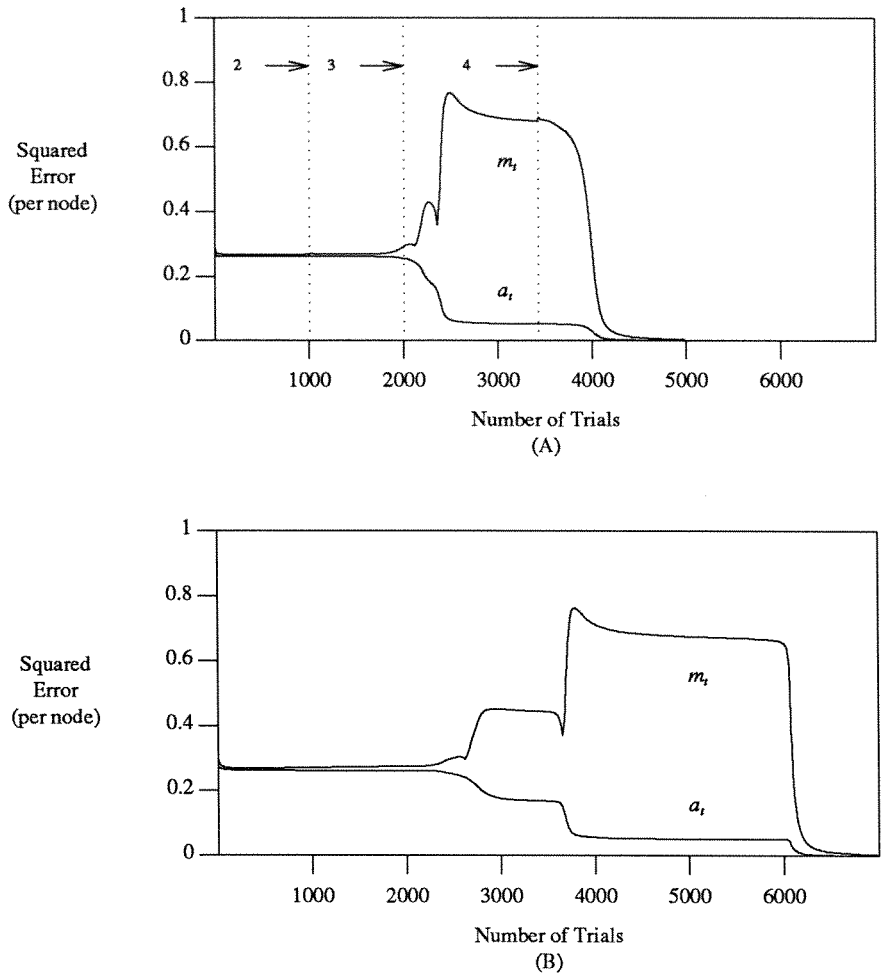

FIGURE 6. PAR4 error curves: (A) DNC network, (B) BP network. Flat shape at the beginning indicates that only the bias has been learned. This is surprising for the BP network because it has the capacity to model the correct function from the start.

nearly flat for a long time before rapidly falling. It is impossible to predict when flatness is a precursor to a rapid drop and when it represents a shallow area of the error surface or an actual local minimum (McInerney, Haines, Biafore, & Hecht-Nielsen, 1988). This is similar to the *horizon effect* in game playing look-ahead problems. What needs to be determined is a range of effective w values that works well in practice. This problem may be avoided altogether by other flatness heuristics that do not need a back history of error values. It may be possible to use some sort of fading window integrator equation. It has been pointed out (G. Cottrell, personal communication, 1988) however, that Equation 2 has the desirable quality of adding nodes more rapidly near the beginning of the training, when the remaining absolute error is highest.

The unit of change in this simulation was a completely forward and back connected node. This was chosen partly because a node is the bottleneck through which connections pass, and is in some sense more important than a single weight. It also allowed the writing of very efficient code for the simulator. In general, there is no compelling reason why single weights and/or sparsely connected nodes should not be used when adding to the network. Some of the solutions found by the growing method were minimal in the number of nodes, but not in the number of weights. Adding single weights would result in a finer grained growth (adding one degree of freedom at a time instead of several) and may yield less complex networks. It is also possible to use various pruning procedures once a solution has been found and get rid of unused weights.

Adding single weights would also allow the modeling of arbitrary topologies and interconnection schemes. Once nonuniform networks are considered,

it becomes important to develop algorithms for deciding *where*, as well as when to add new elements. We are currently conducting research to determine general node allocation strategies in networks with multiple hidden layers. Work using brain-like topological constraints and an update rule similar to BP has already proved promising for some perceptual classification tasks (Honavar & Uhr, 1988a, 1988b). This method adds new elements to carefully designed hierarchical networks based on performance in recognizing each class of objects.

The test problems used in this paper were all of a discrete (boolean) nature. DNC has successfully been applied to the compression of real-valued cloud image data (McInenrney, 1989). However, further work needs to be conducted to determine DNC's effectiveness for modeling continuous functions.

CONCLUSIONS

DNC is a valuable tool for finding BP architectures for arbitrary mapping problems. This fully automatic process found solutions for *all* problems tried. The architectures yielded were small, and often minimal. Computational expense was comparable to training normal BP networks with the same final topologies, despite naively starting with one hidden unit each time. Lower dimensional training seems to help find a solution quickly once the proper dimensionality is reached. Further work is warranted into the shape of BP error surfaces, in order to explain this phenomenon. Additional benchmarking should be conducted to determine the computational expense of training large networks with this method compared to normal BP.

REFERENCES

Aoki, Chiye, Siekevitz, & Philip. (1988, December). Plasticity in brain development. *Scientific American*, pp. 56-64.

Hecht-Nielsen, R. (1988). *Theory of the backpropagation neural network.* Invited paper presented at 1988 INNS Annual Meeting.

Honavar, V., & Uhr, L. (1988a). *Experimental results indicate that generation, local receptive fields and global convergence improve perceptual learning in connectionist networks* (Tech. Rep. 805). University of Wisconsin, Madison, Computer Sciences.

Honavar, V., & Uhr, L. (1988b). *A network of neuron-like units that learns to perceive by generation as well as reweighting of its links* (Tech. Rep. 793). University of Wisconsin, Madison, Computer Sciences.

Hornik, K., Stinchcombe, M., & White, H. (1988). *Multilayer feedforward networks are universal approximators.* Unpublished manuscript, University of California, San Diego, Department of Economics.

Huyser, K. A., & Horowitz, M. A. (1988). Generalization in digital functions. *Neural Networks, 1*(Suppl. 1), 101.

McInerney, J. M., Haines, K. G., Biafore, S., & Hecht-Nielsen, R. (1988). *Can error surfaces traversed by backpropagation networks have local minima?* Unpublished manuscript, University of California, San Diego, Department of Computer Science and Engineering.

McInerney, J. M. (1989). *Dynamic node creation for image compression.* Preliminary manuscript, University of California, San Diego, Department of Computer Science and Engineering.

Minsky, M., & Papert, S. (1988). *Perceptrons* (Expanded Edition). Cambridge: MIT Press.

Plutowski, M. E. P. (1988, July). *Backpropagation with higher-order units.* Poster Session presented at *1988 IEEE International Conference on Neural Networks*, San Diego, California.

Reilly, D. L., Scofield, C., Elbaum, C., & Cooper, L. N. (1987). Learning system architectures composed of multiple learning modules. *Proceedings of the First International Conference on Neural Networks*, San Diego, California.

Rumelhart, D. E. (1988, July). *Parallel distributed processing.* Plenary lecture presented at 1988 IEEE International Conference on Neural Networks, San Diego, California.

Rumelhart, D. E., & McClelland, J. L. (1986). *Parallel distributed processing: Explorations in the microstructure of cognition. Vol. 1. Foundations.* Cambridge: MIT Press/Bradford Books.

Rumelhart, D. E., Hinton, G. E., & Williams, R. J. (1985). *Learning internal representations by error propagation* (Tech. Rep. 8506). University of California, San Diego, Institute for Cognitive Science.

Sietma, J., & Dow, R. (1988, July). *Neural net pruning—Why and how.* Poster session presented at 1988 IEEE International Conference on Neural Networks, San Diego, California.

Werbos, P. J. (1974). *Beyond regression: New tools for prediction and analysis in the behavioral sciences.* Doctoral dissertation, Harvard University.

White, H. (1987). Some asymptotic results for back propagation. *Proceedings of the IEEE First International Conference on Neural Networks* (Vol. III), 261-266.