FG02-92ER75700
CONF-9111215--2

10.8 A Dynamic Node Architecture Scheme for Backpropagation Neural Networks

This paper was presented at and published in the proceedings of the *Artificial Neural Networks in Engineering* conference in November 1991.

CONF-9111215--2

DE93 010311

EP

A DYNAMIC NODE ARCHITECTURE SCHEME FOR BACKPROPAGATION NEURAL NETWORKS

Eric Bartlett and Anujit Basu
Iowa State University

ABSTRACT

Typically, artificial neural network (ANN) training schemes require network architectures to be set before training. However, the learning speed and generalization characteristics of ANNs are dependent on their architectures. Thus, the viability of a specific architecture can only be evaluated after training. This work seeks to reduce the dependence of ANN capabilities on the preselection of network architectures. The present work describes an ANN dynamic node architecture (DNA) scheme which determines the appropriate number of nodes for a given network by defining an importance function which assigns an importance to each node in the network. Optimizing the network architecture becomes part of the training objective. The backpropagation learning algorithm has been implemented with this new DNA scheme.

INTRODUCTION

Although promising, the application of ANNs is restricted due to limitations such as slow learning, inefficient scaling to large problems and uncertainty as to generalization results [1,2]. One cause of these symptoms is the inability to predetermine the network architecture before training is attempted. One approach, for example, is to train many different architectures on the same problem and use one with the best post-training characteristics. This method significantly increases training time, since many ANNs must be trained. Furthermore, an optimal architecture is not necessarily obtained with this technique since it may not be one of the initial selections. This approach is, in effect, an attempt at an exhaustive search for a optimal ANN architecture.

This paper presents a systematic architecture optimization method which eliminates the need to predetermine network architectures. The DNA scheme presented requires little computational overhead and actually reduces learning time in some cases. The method is applied to the backpropagation paradigm and examples are given.

NETWORKS AND MAPPINGS

The networks used for this demonstration utilize layered continuous perceptrons and the backpropagation learning algorithm [3]. A brief review of the mapping problem and the backpropagation paradigm is given below. A mapping, M which may be continuous or discrete, such that

$$M(X_{1,n}) = X_{l+1,n} \tag{1}$$

is modeled by a network of layered nodes where

$$X_{1,n} = (x_{1,1,n}, x_{1,2,n}, \ldots, x_{1,J(1),n})^T \tag{2}$$

is the input vector and

$$X_{I+1,n} = (x_{I+1,1,n}, x_{I+1,2,n}, \ldots, x_{I+1,J(I+1),n})^T \tag{3}$$

is the output vector, which corresponds to the output of the I'th layer of active (hidden or output) nodes, and J(1) and J(I+1) are the number of nodes in the input and output layers respectively. Note that the input nodes are inactive in that their input is equal to their output. Also note that n is the training set exemplar (input-output pattern) number. Each active node has the following input-output relation,

$$x_{i,j,n} = (1/\pi)\bullet\arctan(u_{i,j,n}) + 1/2, \tag{4}$$

$$u_{i,j,n} = g\bullet \sum_{k=1}^{K} (w_{i,j,k}\bullet x_{i-1,k,n}) . \tag{5}$$

The trainable parameter set is $\{w_{i,j,k}\}$. Note that the arctangent function is used here rather than the usual exponential sigmoid function.

The cost (energy, error, merit, objective) function has the form,

$$c(W) = \{ 1/(N\bullet J(I+1))\bullet \sum_{n=1}^{N} \bullet[\sum_{j=1}^{J(I+1)} (\Omega_{I+1,j,n} - x_{I+1,j,n})^2]\}^{1/2}, \tag{6}$$

where N is the total number of training exemplars in the training set, $\{\Omega_1, \Omega_{I+1}\}$. This cost function is simply the root mean square (RMS) error of the network output over the training set. Note that $\{\Omega_1\}$ is a subset of all possible inputs $\{X_1\}$, and $\{\Omega_{I+1}\}$ is a subset of all correct or desired outputs $\{XD_{I+1}\}$ associated with $\{X_1\}$. The problem is to reconstruct or approximate the unknown desired mapping Z, such that,

$$XD_{I+1} = Z(X_1), \tag{7}$$

from $\{\Omega_1, \Omega_{I+1}\}$. There are, however, many solutions M_{I+1}, which satisfy the training set

$$\Omega_{I+1} = M(\Omega_1), \tag{8}$$

none of which are necessarily the desired solution

$$\Omega_{I+1} = Z(\Omega_1). \tag{9}$$

Backpropagation is by far the most widely used and understood neural network paradigm. Its popularity arises from its simple architecture and easy to understand learning process. The backpropagation scheme consists of two major steps. These are the forward activation and the backward error flows. If a DNA scheme is not used, an architecture for the specific problem must be determined. The training process begins with the assignment of random weights to the connections between the nodes of the various layers. The various input patterns are then presented to the network, and the forward activation flow produces the output patterns. These output patterns will not be the same as the desired output patterns. The errors in the outputs are calculated for the output layer nodes as the difference between the desired and actual outputs. For the hidden layers, the errors are calculated by backpropagating the errors in the output layer to the hidden layers. The errors of each of the nodes are summed over the whole set of training patterns. These errors are used to change the weights in the interconnections between the layers. The weights connecting to the output layer are changed according to the delta rule, whereas for the weights in the hidden layers the generalized delta rule is used. This process is continued untill the RMS error falls below some preaccepted value. There are many good references which describe the mathematics of the backpropagation approach in detail including [3,4].

DYNAMIC NODE ARCHITECTURE THEORY

ANNs must be able to generalize information gained through training. Without this ability, neural networks would be of little interest. Thus, networks must be evaluated on not only their speed and depth of convergence but also their generalization capabilities. Minimization of the recall set error is the ultimate goal; however the recall set is rarely known a priori. Minimization of the error over some training set does not necessarily imply minimization of the error over the recall set [5]. More than a simple optimization over an arbitrary training set must be accomplished if good generalization characteristics are to be obtained. This can be seen mathematically by comparing Equations (8) and (9) above.

Generalization is the ability to quantitatively estimate the characteristics of a phenomenon never encountered before based on its similarities with things already known. This implies the ability to distinguish between specifics and generalities. The objective is to teach the computer only important characteristics -- those which set apart the classes of interest. The mathematical solution to this is to reduce the number of ways to interpret details. In the network implementation, this means reducing the number of weights and nodes [6]. This approach seems reasonable since, when approximating functions, researchers use polynomials of reasonable degree because high power polynomials tend to oscillate about the desired interpolation values [7]. In this sense the size of a neural network can be likened to the degree of an interpolation polynomial. But since the appropriate number of nodes is unknown, a variable node architecture approach should be used. The network to be trained can be started with a small number of nodes and the network size can be increased or decreased until the network can successfully identify all classes in the training set with the minimum number of nodes and interconnections. The final network should be a better generalizer, because it has the fewest ways to distinguish the classes learned. Therefore, the network training procedure should seek to minimize both c(W) and the number of nodes. This can be achieved as follows. Start the network with only a few nodes. Since the network is most likely too small to learn the desired mapping add nodes until the network learns the training set to the desired accuracy. Once this is achieved, eliminate a node which has near-zero nodal importance, thus eliminating a nearly useless node. If the resulting error cost function is larger than desired, retrain this smaller network and repeat. The final network should give a more general implementation of the desired mapping.

The importance of a node is a function of the network outputs. If a small change in the value of a hidden nodal output changes the activation of an output node more than a similar change in another hidden nodal output, then that node is more important to the dynamic function of the network than the other nodes. Therefore, the importance of node (I,j), with respect to node (I+1,k) can be defined as,

$$I(x_{I+1,k}, x_{I,j}) = E[\ | \delta x_{I+1,k,n} / \delta x_{I,j,n} |\ ] \bullet dx_{I,j}^{max}. \tag{10}$$

Where E[•] is the expected value over the training set and $dx_{I,j}^{max}$ is the maximum change in the activation of node (I,j), also over the training set. The change in $x_{I+1,k,n}$ due to a change in $x_{I,j,n}$ is,

$$\delta x_{I+1,k,n}/\delta x_{I,j,n} = 1/\{\pi\bullet(1 + u_{I+1,k,n}^2)\}\bullet\delta u_{I+1,k,n}/\delta x_{I,j,n}$$
$$= g\bullet w_{I+1,j,k}/\{\pi\bullet(1 + u_{I+1,k,n}^2)\} \qquad (11)$$

Similarly, the chain rule can be used to find $I(x_{I+1,k}, x_{i,j})$. An estimate of the importance of any hidden node can be obtained by summing the effects of a change in the output of that node on the activations of the nodes in the output layer. Thus, the total importance of node (i,j) is the sum of the importance over all nodes in the output layer,

$$I(x_{i,j}) = \sum_{k=1}^{J(I+1)} I(x_{I+1,k}, x_{i,j}). \qquad (12)$$

If a node has little affect on the output of the network then it is of little dynamic value to the network and has little importance. The importance of a network layer can be similarly defined as the sum of the importance of each node in the layer of interest.

COMPUTER SIMULATION RESULTS

This section illustrates how backpropagation networks learn with DNA. Two examples are given; more examples and theory are given in [8].

The exclusive-nor problem

A straightforward example of network learning is the exclusive-nor problem. The training data is shown in Table I. A DNA network was started with a 2 X 1 X 1 architecture. Thus it has two input nodes, one node in one hidden layer and one output node. After training, the resultant network has a minimal architecture of 2 X 3 X 1. Table II shows the DNA history. This table shows the RMS training error (cost) obtained by the network in the architecture listed. The target cost is 0.05, therefore if the network learns the training set to this value the network is assumed to have learned its task. Note the oscillation in the number of hidden layer nodes. Apparently it requires more nodes to learn the training set than are required for its correct recall.

Table I. Exclusive-nor training data set.

Pattern Number	Inputs		Desired Output
1	0.0	0.0	1.0
2	0.0	1.0	0.0
3	1.0	0.0	0.0
4	1.0	1.0	1.0

Table II. Exclusive-nor dynamic node architecture history.

	RMS	Architecture				RMS	Architecture				RMS	Architecture		
	Error	In	Hdn	Out		Error	In	Hdn	Out		Error	In	Hdn	Out
1)	.4994	2	1	1	15)	.0942	2	4	1	28)	.0498	2	4	1
2)	.3311	2	2	1	16)	.0930	2	5	1	29)	.0623	2	3	1
3)	.1152	2	2	1	17)	.0921	2	5	1	30)	.0498	2	3	1
4)	.1140	2	2	1	18)	.0911	2	6	1	31)	.0979	2	2	1
5)	.1116	2	3	1	19)	.0902	2	6	1	32)	.0876	2	2	1
6)	.1029	2	3	1	20)	.0892	2	7	1	33)	.0861	2	3	1
7)	.1009	2	4	1	21)	.0884	2	7	1	34)	.0852	2	4	1
8)	.0495	2	4	1	22)	.0873	2	8	1	35)	.0844	2	5	1
9)	.0544	2	3	1	23)	.0492	2	8	1	36)	.0836	2	6	1
10)	.0494	2	3	1	24)	.0485	2	7	1	37)	.0829	2	6	1
11)	.1061	2	2	1	25)	.0483	2	6	1	38)	.0823	2	7	1
12)	.0982	2	2	1	26)	.0492	2	5	1	39)	.0817	2	7	1
13)	.0964	2	3	1	27)	.0528	2	4	1	40)	.0810	2	8	1
14)	.0951	2	4	1										

Gaussian distribution separation

ANNs exhibit well known pattern classification capabilities. An example of this is the ability to distinguish between two groups of numbers randomly drawn from two different Gaussian (normal) distributions. In this example an ANN is trained to reveal which of two populations a sample of five numbers belongs. The two populations have means of 0.40 and 0.60 and standard deviations of 0.10 and 0.10 respectively. The training data is shown in Table III. The training was started with a 1 X 1 X 1 network. The resultant network has a 5 X 31 X 1 architecture. Table IV shows the DNA history for this example. This table shows the RMS training error obtained by the network along with the associated architecture. Note that the target cost is 0.05.

Table III. Gaussian distribution separation data set.

Pattern Number	Inputs					Desired Output
1	0.2992	0.3949	0.4739	0.3733	0.2804	1.00
2	0.8346	0.6081	0.6707	0.4650	0.5755	0.00
3	0.3796	0.3880	0.4172	0.2840	0.3443	1.00
4	0.4519	0.6373	0.6055	0.5346	0.4797	0.00
5	0.3406	0.3318	0.4616	0.6410	0.3962	1.00
6	0.5784	0.6526	0.5508	0.5237	0.6557	0.00
7	0.3254	0.3917	0.4103	0.2119	0.3201	1.00
8	0.6796	0.5932	0.8022	0.6165	0.5716	0.00
9	0.3347	0.3474	0.4502	0.4883	0.3693	1.00
10	0.6741	0.6495	0.5675	0.6239	0.4688	0.00

Table IV. Gaussian distribution dynamic architecture history.

	RMS	Architecture				RMS	Architecture				RMS	Architecture		
	Error	In	Hdn	Out		Error	In	Hdn	Out		Error	In	Hdn	Out
1)	.4989	1	1	i	13)	.0607	5	9	1	25)	.0530	5	21	1
2)	.4990	2	1	1	14)	.0596	5	10	1	26)	.0527	5	22	1
3)	.1415	2	2	1	15)	.0586	5	11	1	27)	.0523	5	23	1
4)	.1275	2	3	1	16)	.0578	5	12	1	28)	.0520	5	24	1
5)	.1175	3	3	1	17)	.0571	5	13	1	29)	.0517	5	25	1
6)	.1130	4	3	1	18)	.0564	5	14	1	30)	.0514	5	26	1
7)	.0916	5	3	1	19)	.0557	5	15	1	31)	.0511	5	27	1
8)	.0898	5	4	1	20)	.0551	5	16	1	32)	.0508	5	28	1
9)	.0877	5	5	1	21)	.0547	5	17	1	33)	.0505	5	29	1
10)	.0860	5	6	1	22)	.0541	5	18	1	34)	.0502	5	30	1
11)	.0839	5	7	1	23)	.0537	5	19	1	35)	.0499	5	31	1
12)	.0622	5	8	1	24)	.0534	5	20	1					

CONCLUSIONS

A DNA scheme is presented which varies the number of hidden nodes in a feedforward backpropagation neural network during training. This new method minimizes the number of nodes and interconnections in the network consistent with the learning objective. Results show that the method can obtain the appropriate network architecture for a given training task in a systematic way. The method therefore eliminates the need to preselect network architectures.

REFERENCES

1. S. Judd, "Learning in Networks is Hard", IEEE First International Conference on Neural Networks, 2, 685-692 (June 1987).
2. J. M. McInerney, K. G. Haines , S. Biafore, R. Hecht-Nielsen, "Back Propagation Error Surfaces Can Have a Local Minima", IJCNN International Conference on Neural Networks, 2, 627 (June 1989)
3. R. Hecht-Nielsen, "Theory of the Backpropagation Neural Network", IJCNN International Conference on Neural Networks, 1, 593-606 (June 1989)
4. D. E. Rumelhart, J. L. McClelland and the PDP Research Group, Institute for Cognitive Science, University of California, San Diego, Parallel Distributed Processing: Explorations in the Microstructure of Cognition, (MIT Press, Cambridge, Mass. 1986)
5. M. T. Musavi, A. Rajavelu, A., Sahai, S., and Zhao, J., "Analysis and Generalization of Back Propagation in Neural Networks", Neural Networks, 1, Supp. 1, 118 (1988)
6. T. Ash, "Dynamic Node Creation in Backpropagation Networks", IJCNN International Conference on Neural Networks, 2, 623 (June 1989)
7. W. H. Press, B. P. Flannery, S. A. Teukolsky, and W. T. Vetterling, Numerical Recipes: The Art of Scientific Computing, (Cambridge University Press, New York 1986)
8. E. B. Bartlett, "Nuclear Power Plant Status Diagnostics Using Simulated Condensation: An Auto-Adaptive Computer Learning Technique", Ph.D. Dissertation, The University of Tennessee at Knoxville (1990)

END

DATE
FILMED
5/28/93