Reference: Proceedings of the IASTED International Conference on Artificial Intelligence, Expert Systems and Neural Networks, pp. 249-252, 1996.

Using Multiple Node Types to Improve the Performance of DMP (Dynamic Multilayer Perceptron)

Tim L. Andersen and Tony R. Martinez
Computer Science Department, Brigham Young University, Provo, Utah 84602
e-mail: tim@axon.cs.byu.edu, martinez@cs.byu.edu

ABSTRACT

This paper discusses a method for training multi-layer perceptron networks called DMP2 (Dynamic Multi-layer Perceptron 2). The method is based upon a divide and conquer approach which builds networks in the form of binary trees, dynamically allocating nodes and layers as needed. The focus of this paper is on the effects of using multiple node types within the DMP framework. Simulation results show that DMP2 performs favorably in comparison with other learning algorithms, and that using multiple node types can be beneficial to network performance.

1 Introduction

One of the first models used in the field of neural networking is the single layer, simple perceptron model (Rosenblatt 1958). The well understood weakness of single layer perceptron networks is that they are able to learn only those functions which are linearly separable. Since multi-layer networks are capable of going beyond the limited set of linearly separable problems and solving arbitrarily complex problems, researchers have devoted much effort to devising multi-layer network training algorithms. The unfortunate drawback to many of the current multi-layer training methods is that they require someone to specify the network architecture a-priori (make an *educated guess* as to the appropriate number of layers, number of nodes in each layer, connectivity between nodes, etc.). With a pre-specified network architecture, there is no guarantee that it will be the appropriate network for the problem at hand, and it may not even be capable of converging to a solution.

Thus, several multi-layer training algorithms have been proposed which do not require the user to specify the network architecture a-priori. Some of these include EGP (Romaniuk 1993), Cascade Correlation (Fahlman 1991), Iterative Atrophy (Smotroff 1991), DCN (Romaniuk 1993), ASOCS (Martinez 1990), and DNAL (Bartlett 1993). All of these methods seek to dynamically generate an appropriate network structure for solving the given learning problem during the training phase.

This paper discusses a dynamic method for training multi-layer perceptron networks which we call DMP 2 (Dynamic Multi-layer Perceptron 2). This method is similar in spirit to the DCN method given in (Romaniuk 1993) in that:

- The network begins with a single perceptron node and dynamically adds nodes as needed.
- As nodes are added to the network, each node is trained independently from all other nodes.
- Each node is trained on only a portion of the training set, the portion being based upon the current set of misclassified examples - a divide and conquer approach.

However, the DMP 2 method differs from DCN in several ways, some of which are:

- The network structure grows backwards from the original output node, rather than forwards.
- The output of each node is connected to at most one other node.
- Each node has as inputs the output of at most two other nodes and the training set inputs.
- A Genetic Algorithm (GA) is used to train the individual nodes of the network.

The original DMP method, DMP1 (Andersen and Martinez 95), generated networks which consisted entirely of perceptron nodes. However, since with the DMP method the individual nodes in the network are trained independently from all other nodes, it is possible to generate networks where each node implements a different type of decision function. In other words, some of the nodes in the network could be performing a radial basis function while other nodes are using hyperplane decision functions. With this type of approach, whenever it is determined that a node needs to be added to the network, several different types of decision nodes can be trained on the training set and the "best" one selected for insertion into the network.

The DMP2 method utilizes just such an approach. DMP2 uses two different types of nodes during the training of the network, with the best type being selected at each training step. The two types of nodes used are the perceptron node, which generates a hyperplane decision surface, and radial basis function (RBF) node, which has a spherical decision surface (hypersphere). This paper compares the results obtained with DMP2 (using both perceptron and RBF nodes) against those obtained with DMP1 (using only perceptron nodes). The DMP training method is also used to generate networks consisting entirely of RBF nodes, and the results obtained with the RBF only networks are compared with the DMP2 and DMP1 results.

The basic DMP2 algorithm is discussed in detail in section 2. Section 3 discusses the convergence characteristics of the algorithm. Section 4 discusses the genetic algorithm method used to train individual nodes in the network, and simulation results are given in section 5. Conclusion and discussion are given in section 6.

2 The Basic Approach

Although it is relatively straightforward to extend the algorithm to problems which have multiple output classes, in the following discussion we assume that the learning problem has a single, 2 state output. Also, for the purposes of this discussion we will assume that DMP2 is using perceptron nodes. However, the basic model does not restrict the type of nodes which can be used in the network. With DMP2 it is possible to generate networks which use a completely heterogenous mixture of nodes.

DMP2 begins training with a network that contains a single output node. If the output node fails to correctly classify some of the positive examples in its training set, then a child node is allocated and trained on a subset of the parent nodes training examples. The type of decision function which the child node performs can be arbitrary and different for each node added to the network. The subset of training examples given to the new node will be the set of misclassified positive examples combined with the entire set of negative examples from the parent node.

Similarly, if the parent node fails to classify some of the negative examples in its training set, then a child node is allocated and given a training set equal to all of the parent node's positive training examples along with all of the of the parent node's misclassified negative training examples. This process continues until all of the leaf nodes in the network are able to correctly classify their respective training sets.

The intuition behind the DMP2 approach is that if a node incorrectly classifies a certain subset of positive training examples from its training set, then we create a new node (the left child) and pass the parent node's misclassified positive examples to it which, hopefully, will be able to separate the misclassified positive examples from the parent node's set of negative training examples. Conversely, if the node incorrectly classifies a set of negative training examples we create a new node (the right child) and pass those examples to the new right child. The DMP2 method thus builds a multi-layer network in the form of a binary tree. The algorithm is now discussed in greater detail. We define the following:

T = the original set of examples in the training set.
n_i = node i of the network.
$T_i \subseteq T$ = node n_i's training set.
t_{ij} = the jth example from training set T_i.
k = the number of input variables.
w_{ij} = the jth weight of n_i, $1 \leq j \leq k+2$ (the 2 extra weights are for the children of n_i).
$A(i,j)$ = the activation of node n_i when presented with training example t_{ij}.
$O(i,t)$ = the output of node n_i when presented with training example t, where

$$O(i,t) = \begin{cases} 0 \text{ if } A(i,t) \leq 0 \\ 1 \text{ if } A(i,t) > 0 \end{cases}.$$

$Z(i,t)$ = the target output of node n_i when presented with training example t.
$PARENT(i) = n_p$ (where $p = \lfloor i/2 \rfloor$) such that the output of node n_i is connected to n_p.
$LEFTCHILD(i) = n_p$ (where $p = 2i$) such that the output of node n_p is connected to n_i through weight w_{ik+1}.
$RIGHTCHILD(i) = n_p$ (where $p = 2i+1$) such that the output of node n_p is connected to n_i through weight w_{ik+2}.

Consider an arbitrarily complex, non-linearly separable, two output classification problem with training set T. Each example in T is defined to be a vector of continuous valued attributes along with an associated target output classification. The network is initially set up with a single node (the output layer). We label this node n_1, and set $T_1 = T$. The node is trained with the training set T_1 via the Delta rule training method or some other method until the total sum-squared error (TSSE) settles (the actual method DMP2 uses to train the individual nodes is discussed in section 5). During the training of node n_1, the weights w_{ik+1} and w_{ik+2} are held at zero.

At this point, node n_1 can not make further progress towards a solution. If the network has not converged to a solution (TSSE > ε), then two new nodes, $LEFTCHILD(1) = n_2$, $RIGHTCHILD(1) = n_3$, are created and connected to node n_1 through the weights w_{1k+1} and w_{1k+2} respectively. The left child will be responsible for classifying the positive training examples which were misclassified by the parent node, and the right child will be responsible for classifying the misclassified negative training examples. The values for the weights w_{ik+1} and w_{ik+2} are thus determined by the following, where $SGN(x)$ returns 1 if $x \geq 0$ and -1 otherwise.

$$POS\_SUM(i) = \sum_{j=1}^{k} [(1 + SGN(w_{ij})) \times w_{ij} / 2],$$

$$NEG\_SUM(i) = \sum_{j=1}^{k} [(1 - SGN(w_{ij})) \times w_{ij} / 2],$$

then set

$$w_{ik+1} = \max\{ POS\_SUM(i),\ NEG\_SUM(i) \} + 1$$

$$w_{ik+2} = -w_{ik+1}$$

Note that for any node n_i, if the output of $LEFTCHILD(i)$ is high, this forces the output of n_i to be high. On the other hand, if the output of $RIGHTCHILD(i)$ is high, this forces the output of n_i to be low. If the outputs of both $LEFTCHILD(i)$ and $RIGHTCHILD(i)$ are high then they cancel each other.

The 2 children of node n_i are then passed a training set which is a subset of the parent node's training set. The training set for each child node is determined as follows.

Let $LEFTCHILD(i) = n_p$. Then we define the training set T_p for n_p as follows:

$$T_p = \{t_{ij} \mid t_{ij} \in T_i \wedge [(Z(i,j) = 0) \vee (Z(i,j) = 1 \wedge O(i,j) = 0)]\}.$$

Then for all $t_{pm} \in T_p$ (where $1 \leq m \leq |T_p|$), we know that $t_{pm} = t_{ij}$ for some $t_{ij} \in T_i$. For each such $t_{pm} = t_{ij}$ set $Z(p,m) = Z(i,j)$.

Let $RIGHTCHILD(i) = n_q$. Then

$$T_q = \{t_{ij} \mid t_{ij} \in T_i \wedge [(Z(i,j) = 1) \vee (Z(i,j) = 0 \wedge O(i,j) = 1)]\}.$$

Then for all $t_{qm} \in T_q$ (where $1 \leq m \leq |T_q|$), we know that $t_{qm} = t_{ij}$ for some $t_{ij} \in T_i$. For each such $t_{qm} = t_{ij}$ set $Z(q,m) = 1$ if $Z(i,j) = 0$, else $Z(q,m) = 0$.

Each child node is then trained with its corresponding training set. The input weights on nodes which have children (in this case node n_1) are frozen. If either or both of nodes n_2 and n_3 do not converge to a satisfactory solution on their respective training sets, then the algorithm is repeated recursively with nodes n_2 and/or n_3 as the parent nodes. This process can be repeated until TSSE < ε or the number of positive/negative elements in a node's training set is too small to be statistically significant (i. e. noise). For this paper, the process was simply repeated until no further progress could be made at reducing the TSSE.

After the training phase has been completed the network enters the execution phase. During this phase, incoming examples are presented to every node in the network and the output of node n_1 is then taken to be the predicted output classification for the given example.

3 Convergence Properties of DMP2

The basic DMP2 algorithm has two important properties. The first property is that if all leaf nodes in the network correctly classify their respective training sets, then the output of the root node n_1 will be correct for all examples in its training set T_1. This implies that the network output will be correct for all examples in the original training set T, since $T_1 = T$. It also possible to show that the network is guaranteed to converge to a solution on any consistant training set, assuming that each node in the network correctly classifies at least one example from each output class in its training set. Both of these properties are proven as formal theorems in (Andersen and Martinez 95).

The decision function used and the way it is implemented can affect the guaranteed convergence properties of the network. The GA training method used in DMP2 utilizes a fitness function that strongly favors decision boundaries which have at least one positive and one negative example classified correctly, but the very nature of the GA approach makes it impossible to guarantee such a condition. While the GA training method does not guarantee convergence of the network, it does make convergence highly likely for consistent training sets.

4 Training Method

The method used to train each node in the network is based upon genetic algorithms. Each individual in the population consists of a vector of real values, where each individual is considered to be a potential weight vector or prototype point for the node being currently trained.

The population of RBF nodes is initialized by randomly selecting elements from the training set as RBF centers. A good initial sphere of influence for each center is then selected by searching outwards in discrete steps. The search proceeds until all of the examples from the center's output class are contained within the sphere of influence, at which point the sphere which generated the highest fitness score is selected. The genetic algorithm then takes over and randomly moves the centers and adjusts the sizes of the spheres of influence through crossover and mutation operators in an attempt to improve the fitness scores for the population of

RBF nodes. The population of perceptron nodes is initialized by randomly generating a set of weight vectors with weights taking on real values between -10 and 10. The genetic algorithm then adjusts these vectors through the same genetic operators used on the RBF nodes to find individuals with higher fitness scores. The single node with the highest fitness score is selected from the population of RBF and perceptron nodes for insertion into the network.

The reason for selecting the genetic algorithm method for training each node was so the network would be able to deal with the situation shown in figure 1. In this example, a small set of positive examples are surrounded by a large set of negative examples.

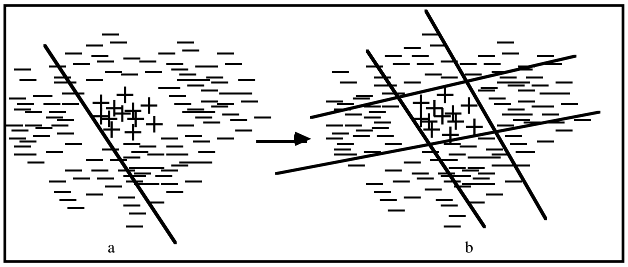

Figure 1. Decision surfaces.

With this example, the well known delta rule training algorithm is likely to generate a decision surface which has all of the patterns on one side, since that would minimize the number of misclassification errors for a single separating hyperplane. However, assuming that we wanted to separate out the positive examples in figure 1, a decision surface that had all examples on one side would be useless to the DMP2 training method, since it would fail to distinguish at least one positive example from at least one negative example. In this case (if the network were allowed to continue) a left child would be allocated and given as its training set the set of misclassified positive examples along with the entire set of negative examples (there would be no need for a right child since all of the negative examples are correctly classified). Thus, the left child's training set would be exactly the same as its parent's training set. Since in this case there is no reason to expect the left child to find a more useful separating hyperplane than its parent, the addition of the left child will not provide any benefit to the network.

Continuing this line of reasoning, with the DMP2 method it is generally better to correctly classify a large number of examples from both classes, rather than correctly classifying all of one class and just a few examples of the other. If only a few examples from one class are correctly classified, then the child node assigned the misclassified examples will have essentially the same training set as its parent, albeit with a few less examples from one class.

Thus, each node is trained with a GA so the network would be able to deal with this situation. Using the same example given in figure 1, the decision surface in figure 1a may be more desirable for the purposes of DMP2, since it could eventually lead to the set of decision surfaces in figure 1b, which completely separates the set of positive examples from the set of negative examples. In order to generate these types of decision surfaces with a GA, it is necessary to use a fitness function which favors them. For DMP2, the following simple fitness function was selected:

Given a vector of real vaues **i** of length k, and a node n_j of the network, let

$$fitness(i) = \sin\left(\frac{\left\{e \middle| e \in T_j \wedge Z(j,e) = 1 \wedge O(j,e) = 1\right\}}{\left\{e \middle| e \in T_j \wedge Z(j,e) = 1\right\}} \times \pi/2\right) + \sin\left(\frac{\left\{e \middle| e \in T_j \wedge Z(j,e) = 0 \wedge O(j,e) = 0\right\}}{\left\{e \middle| e \in T_j \wedge Z(j,e) = 0\right\}} \times \pi/2\right)$$

where the weights of node n_j have been set to **i**.

This function ranges between 0 and 2, and tends to favor nodes which correctly classify several examples from both output classes. For example, the fitness for a boundary with all of the positive and negative examples on one side will be 1, while the fitness for the boundary in figure 3a would be approximately 1.5.

5 Results

The DMP2 training algorithm was tested on several different data sets which were obtained from the UCI Machine Learning Database. The real-world data sets used are Pima Indian (diabetes), Bupa (liver disease), Monks 1-3, and Sonar. The Monks data sets are translated into a binary format where each input had two possible states. For example, a 6 state single input in the original file would be translated into 6 two state inputs, only one of which is set to 1 depending on the original input value. The other data sets which have real-valued attributes are left in their original formats.

15 runs for each data set were conducted with each DMP method. For each run, the data is randomly split into two equal parts, one part being used for training the network and the second part for testing generalization accuracy. Table 1 reports the average accuracy on the test set, and also lists generalization results obtained using three other learning methods for comparison purposes. The first column lists the data sets tested, the next three columns give the accuracy on the test set for the DMP training methods, and the last three columns are the results of the other methods. The numbers in parenthesis are the standard deviations for the reported results. The three other learning methods reported on are c4.5 (Quinlan 1986), a mutli-layer network trained with back propogation, and a single layer perceptron network. In addition to reporting the results for DMP2, which uses both perceptron and RBF nodes, table 1 reports the results for DMP restricted to only perceptron nodes (DMP1) and for DMP restricted to only RBF nodes (DMP-RBF). The last row of the table gives the average accuracy across all of the data sets for each of the methods.

From table 1 it can be seen that the overall accuracy of DMP2 is better than that of either DMP-RBF or DMP1. Also, DMP2 is a definite improvement on a single layer perceptron network, since the single layer perceptron network had an average generalization accuracy that was approximately 9% lower than that of DMP2. However, it does appear that the multilayer back propogation network performs slightly better than DMP2, at least on the data sets tested. It should be mentioned here that the results reported for the back-propogation network were obtained using a much larger training set (a 90% training set, 10% test set split), which could possibly give the back propogation network an advantage.

Data Set	DMP2	DMP-RBF	DMP1	c4.5 rule	Back Prop	Perceptron
bupa	64.5 (5.9)	60.5 (6.3)	61.3 (5.3)	59.1	69.8	66.4
monk1	97.2 (2.5)	70.8 (3.1)	96.8 (2.9)	100.0	100.0	72.3
monk2	98.0 (2.2)	76.9 (8.4)	98.2 (3.0)	65.4	100.0	61.6
monk3	98.5 (1.8)	83.3 (3.9)	98.4 (1.0)	98.9	98.7	98.2
pima	68.6 (3.1)	67.1 (3.2)	63.2 (2.3)	67.0	76.2	74.6
sonar	74.5 (4.4)	64.7 (5.4)	72.6 (7.4)	70.1	76.4	73.2
Average	83.6	70.6	81.8	76.8	86.9	74.4

Table 1. Average test set accuracy.

One surprising result is the extremely poor performance of DMP-RBF. This is probably due in part to the fitness function which the genetic algorithm was using, which seemed to work well with perceptron nodes, but not so well with the RBF nodes. The results for DMP-RBF could probably be improved by fine tuning the way the genetic algorithm searches for good RBF nodes.

Table 2 shows that all of the DMP training methods are generally able to come up with a solution which correctly classifies each example in the training set. The only exception to this occurred with the DMP-RBF method, which still achieved at least 99% accuracy on each of the data sets. While the convergence characteristics of DMP2 are interesting, it could lead to memorization of noise and other idiosyncrasies in the training set data which would cause degradation of generalization performance.

Data Set	DMP2	DMP-RBF	DMP1
bupa	100.0	99.9	100.0
monk1	100.0	100.0	100.0
monk2	100.0	100.0	100.0
monk3	100.0	100.0	100.0
pima	100.0	99.9	100.0
sonar	100.0	99.0	100.0

Table 2. Average training set accuracy for DMP.

In fact, it does appear that this is the case. Table 3 shows the average number of nodes the network contained after convergence for each of the data sets for the DMP methods. From this table it can be seen that data sets which require large networks tend to be strongly correlated with poor generalization performance. For example, the pima data set required on average over 100 nodes to converge to a solution. This was by far the largest network required by any of the data sets, and as expected the generalization results reported for this data set also faired the worst in comparison with the back propogation network.

Data Set	DMP2	DMP-RBF	DMP1
bupa	47.5	81.0	51.6
monk1	4.9	139.3	5.1
monk2	3.1	116.4	3.6
monk3	1.6	82.2	2.3
pima	104.3	146.4	119.9
sonar	9.9	80.3	9.3
Average	28.6	107.6	32.0

Table 3. Average number of nodes in network.

6 Conclusion and Future Work

The overall empirical performance of DMP2 as measured by average generalization accuracy was better than or comparable to the other learning methods. However, it is expected that the generalization capabilities of DMP2 can be improved by reducing its sensitivity to noisy inputs. For example, statistical significance tests could be used to prune child nodes, or part of the training examples could be withheld and used as a node pruning set, or the fitness function could be further refined to help prevent memorization of noise and further limit the size of the network. Another problem could be the large weight which the algorithm tends to assign to the exceptional training cases.

Another area for improvement is in the network structure and how nodes are added to the network. It is highly likely that the basic structure of the networks generated by DMP2 (binary trees) will not be optimal for solving many problems of interest. For example, in some cases a better solution might be generated if the output node had three children instead of two. The algorithm could be modified so that it would be capable of dynamically adding more than two children to a node deemed beneficial.

With this in mind, future work is concentrating on the following areas:

- Individual nodes - GA refinement, other training methods, different node types.
- Smaller networks - using a pruning set, statistical tests, determining when to stop further network growth.
- Modifications to the basic algorithm which allow nodes to have more than two children.
- Lessen the impact of exceptional training cases by using a smarter weight assignment mechanism.

7 Bibliography

Andersen, Tim and Tony Martinez. A Provably Convergent Dynamic Training Method for Multilayer Perceptron Networks. *Proceedings of the 2nd International Symposium on Neuroinformatics and Neurocomputers,* 1995.

Azimi-Sadjadi, Mahmood (1993). Recursive Dynamic Node Creation in Multilayer Neural Networks. *IEEE Transactions on Neural Networks*, Vol 4, No 2, pp 242-256.

Bartlett, Eric (1994). Dynamic Node Architecture Learning: An Information Theoretic Approach. *Neural Networks*, Vol 7, No 1, pp 129-140.

Fahlman, S. and C. Lebiere (1991). The Cascade Correlation Learning Architecture.

Martinez, Tony and Jacques Vidal (1988). Adaptive Parallel Logic Networks. Journal of Parallel and Distributed Computing. Vol 5, pp 26-58.

Martinez, T. R. and Campbell, D. M. (1991). "A Self-Adjusting Dynamic Logic Module", Journal of Parallel and Distributed Computing, v11, no. 4, pp. 303-13.

Quinlan, J. R. (1986). Induction of Decision Trees. *Machine Learning*, 1:81-106.

Romaniuk, Steve and Lawrence Hall (1993). Divide and Conquer Neural Networks. *Neural Networks*, Vol 6, pp 1105-1116.

Rosenblatt, F. (1958). The Perceptron: A probabilistic Model for Information Storage and Organization in the Brain. *Psychological Review*, Vol. 65, No. 6.

Smotroff, Ira, David Friedman and Dennis Connolly (1991). Self Organizing Modular Neural Networks. *International Joint Conference on Neural Networks*, II, 187-192.

Zarndt, Frederick (1995). Masters Thesis at Brigham Young University.