Reference: WCNN, pp 177-181, 1996.

The Effect of Decision Surface Fitness on Dynamic Multi-layer Perceptron Networks (DMP1)

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

The DMP1 (Dynamic Multi-layer Perceptron 1) network training 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. This paper introduces the DMP1 method, and compares the preformance of DMP1 when using the standard delta rule training method for training individual nodes against the performance of DMP1 when using a genetic algorithm for training. While the basic model does not require the use of a genetic algorithm for training individual nodes, the results show that the convergence properties of DMP1 are enhanced by the use of a genetic algorithm with an appropriate fitness function.

1 Introduction

One of the first models used in the field of neural networking is the single layer, simple perceptron (Rosenblatt 1958). The well understood weakness of single layer 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.).

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 building multi-layer perceptron networks called DMP1 (Dynamic Multi-layer Perceptron 1). 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.

However, the DMP1 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 focus of this paper is on the effect that different types of decision boundary fitness have on the performance of DMP1. The usual approach to training simple perceptron nodes is to utilize the well known delta rule method or some variation thereof. In general, this method searches for boundaries which minimize misclassification error on the training set. However, the results in this paper show that such decision surfaces do not always lead to a network with good performance when using the DMP1 algorithm.

The basic DMP1 algorithm is discussed in section 2. Section 3 gives the fitness function and method used to train individual nodes in the network, and simulation results are given in section 4. Conclusion and discussion are given in section 5.

2 The DMP1 Algorithm

In the following discussion we assume that the learning problem has a single, 2 state output. DMP1 begins training with a single output node. If, after training, 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 node's training examples. 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 DMP1 approach is that if a node misclassifies a certain subset of positive training examples from its training set, then a new node is allocated and given the misclassified positive examples, which it will hopefully be able to distinquish from the parent node's set of negative training examples. Conversely, if a node misclassifies some of its negative training examples a node is allocated (the right child) and passed those examples. The DMP1 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.
$T_i \subseteq T$ = the set of examples which are used to train node n_i.
t_{ij} = the jth example from training set T_i.
k = the number of input variables.
w_{ij} = the jth weight of n_i, $1 \le j \le k+2$ (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

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

$Z(i,t)$ = 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 DMP1 uses to train the individual nodes is discussed in section 5). During the training of node n_1, the weights w_{1k+1} and w_{1k+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_{1k+1} and w_{1k+2} are thus determined by the following, where $SGN(x)$ returns 1 if $x \ge 0$ 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
$w_{ik+1} = \max\{POS\_SUM(i), POS\_SUM(i)\} + 1$
$w_{ik+2} = -w_{ik+1} = -max\{POS\_SUM(i), POS\_SUM(i)\} - 1$

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 $1 \le m \le |T_p|$), 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 $1 \le m \le |T_q|$), 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).

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 Training Method

The standard method used to train simple perceptron nodes is the well known delta rule training method, which searches for a separating hyperplane which minimizes classification error on the training set. While such decision surfaces are appropriate when the network is restricted to a single layer, they may be less desirable when the goal is to build a multi-layer network

This problem is exhibited in figure 1. In this example, a small set of positive examples are surrounded by a large set of negative examples.

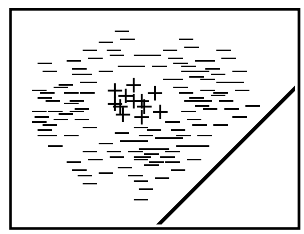

Figure 1. Example pattern distribution.

With this example, the delta rule training algorithm is likely to generate a decision surface which has all of the patterns on one side, since this would minimize the number of misclassification errors for a single hyperplane. However, if the goal is to separate the positive examples from the negative examples then such a decision surface would be useless to the DMP1 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 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.

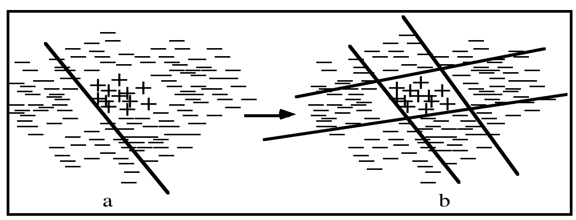

Figure 2. Decision surfaces.

By using a GA and appropriate fitness function it is possible to select hyperplane decision boundaries that can deal with the situation shown in figure 1. For example, the decision surface in figure 2a is more desirable since it could eventually lead to the set of decision surfaces in figure 2b which completely separates the set of positive examples from the set of negative examples.

DMP1 uses the following fitness function.

Given a vector of real values $\boldsymbol{i}$ 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 $\boldsymbol{i}$.

This function ranges between 0 and 2, and tends to favor decision boundaries which correctly classify at least one example from both output classes. For example, the fitness for a boundary with all of the positive and negative examples on one side is 1, while the fitness for the boundary in figure 2a is approximately 1.5.

4 Results

DMP1 was tested on several different data sets which were obtained from the UCI Machine Learning Database. These are Pima Indian (diabetes), Bupa (liver disease), Monks 1-3, Sonar, and Tic-tac-toe. In addition to reporting the generalization results for DMP1 using the delta rule (DMP1-DR) and GA (DMP1-GA) training method, table 1 also reports results obtained using three well known learning methods for comparison purposes. 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. The results obtained with these methods show that DMP1 is capable of performing comparably with other learning methods.

15 runs for each data set were conducted. For each run, the data is randomly split into two equal parts, one part for training the second part for testing. Table 1 reports the average accuracy on the test set, and also lists generalization results obtained using the three other learning methods. The first column lists the data sets tested, the second column gives the accuracy for DMP1-DR, the second column gives the accuracy for the DMP1-GA method, and the last three columns list the results of the other methods. The numbers in parenthesis give the standard deviations for the reported results. The results for the other methods are taken from (Zarndt 1995). The last row of the table gives the average accuracy across all of the data sets for each of the methods.

Data Set	DMP1-DR	DMP1-GA	c4.5	Back Prop	Perceptron
bupa	69.3 (3.8)	61.4 (3.2)	59.1	69.8	66.4
monk1	94.0 (3.5)	94.6 (4.3)	100.0	100.0	72.3
monk2	73.0 (9.6)	94.2 (4.2)	65.4	100.0	61.6
monk3	96.6 (2.6)	98.1 (1.4)	98.9	98.7	98.2
pima	68.9 (2.9)	62.2 (3.2)	67.0	76.2	74.6
sonar	75.0 (3.5)	75.0 (5.4)	70.1	76.4	73.2
tictactoe	97.1 (0.8)	80.4 (2.4)	65.4	98.6	97.0
average	82.0	80.8	75.1	88.5	77.6

Table 1. Average test set accuracy.

Table 2 reports the average training set accuracy with DMP1 using both the delta rule training method and the GA training method. The results show that DMP1 is much more likely to converge to a solution that correctly classifies all training examples when using the GA training method. While the convergence characteristics of DMP1-GA are interesting, the concern is that it could lead to memorization of noise and other idiosyncrasies in the training set data which would cause degradation of generalization performance. Indeed, it can be seen from table 1 that training set accuracy doesn't always translate to good results on the test set, in particular with the bupa, pima, and tic-tac-toe data sets.

Data Set	DMP1-DR	DMP1-GA
bupa	77.0	100.0
monk1	99.5	100.0
monk2	84.6	100.0
monk3	100.0	100.0
pima	75.0	100.0
sonar	100.0	100.0
tictactoe	98.9	100.0
average	90.7	100.0

Table 2. Average training set accuracy.

In spite of the fact that over learning can be a problem with the DMP1 training methods, it does appear from table 1 that the method is an improvement over a single layer perceptron network. However, neither of the DMP1 methods did as well as the multilayer backpropogation network, which bested DMP1-DR by 6.5% on average.

Table 3 shows the average number of nodes the network contained after convergence for each of the data sets for the DMP1 method. From this table it can be seen that large networks tend to be strongly correlated with poor generalization performance. For example, the DMP1-GA method required on average 145 nodes to converge to a solution on the pima data set. 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 other learning methods.

Data Set	DMP1-DR	DMP1-GA
bupa	9.4	60.5
monk1	7.5	6.8
monk2	9.1	6.0
monk3	1.6	2.5
pima	10.9	144.8
sonar	4.3	12.6
tictactoe	3.3	38.4

Table 3. Average number of nodes in network.

For every data set except one, whichever method produced the smaller average network size also had the best average test set accuracy. The only exception to this rule occurred on the sonar data set, where DMP1-GA had slightly better test set results than the smaller networks produced by DMP1-DR. This suggests a training strategy which utilizes both the DMP1-GA and DMP1-DR methods on the training set, and then simply selects the training method which produces the smallest average size network. Table 4 lists the results of this approach.

Note that the average accuracy, at 85.1 percent, is significantly higher than the best result reported in table 1 for the DMP1 methods individually, and is much more comparable with the results obtained with the multilayer backpropogation network.

Data Set	DMP1-Small
bupa	69.3
monk1	94.6
monk2	94.2
monk3	96.6
pima	68.9
sonar	75.0
tictactoe	97.1
average	85.1

Table 4. Hybrid approach.

5 Conclusion and Future Work

The overall empirical performance of DMP1-DR and DMP1-GA as measured by average test set accuracy was better than c4.5 and a single layer perceptron, although somewhat worse than the performance of the multi-layer backpropogation network. Choosing decision boundaries which maximized classification accuracy on the individual nodes' test sets generally led to poor network convergence, a result that might at first seem counter-intuitive. However, the DMP1-GA approach, while it always led to convergence on the test set, often generated overly large networks that had correspondingly poor generalization performance.

These results suggest several areas where improvements could be made that could increase the generalization performance of DMP1. One of the main areas for potential modification is the GA fitness function, which could be modified to favor smaller networks that generalize better while at the same time preserving most of the convergence characteristics of the algorithm. Other modifications could include ways to prune child nodes, such as using a pruning set to delete nodes which are not helping the network to correctly classify examples.

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 DMP1 (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 if it is deemed beneficial.

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

- Training of individual nodes - fitness function refinement, GA refinement, and other training methods.
- Pruning algorithms - using a pruning set, statistical tests.
- Modifications to the basic algorithm which allow nodes to have more than two children.

6 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.