# DMP3: A Dynamic Multilayer Perceptron Construction Algorithm

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

This paper presents a method for constructing multilayer perceptron networks (MLPs) called DMP3 (Dynamic Multilayer Perceptron 3). DMP3 differs from other MLP construction techniques in several important ways. The motivation for these differences and how they can lead to improved performance are discussed in detail in this paper. The DMP3 algorithm constructs MLPs by incrementally adding network elements to the output node of the network. Dependent upon the reduction in network error, the complexity of new elements that are added to the network can increase slightly with each growth cycle of the algorithm. As new elements are added to the network, the existing network structure is frozen and only the weights of the new elements are trained. In addition, the weights which link the new elements to the existing network structure are initially set to predetermined values, which predisposes each new network element to perform a particular function in relation to the existing network structure which can decrease the amount of time required for training the new elements. Information gain rather than error minimization is used to guide the growth of the network, which increases the utility of newly added network elements and decreases the likelihood that a premature dead end in the growth of the network will occur. A short, improvement driven training cycle is used to train new network elements which naturally helps to prevent over learning and memorization. The performance of DMP3 is compared with that of several other well-know machine learning and neural network learning algorithms (c4.5, cn2, ib1, CV based MLP architecture selection, c4, id3, perceptron, and mml) on 9 real world data sets taken from the UCI machine learning database. Simulation results show that DMP3 performs better (on average) than any of the other algorithms on the data sets tested.

## 1. Introduction

One of the first neural models used in the field of neural networks was the single layer perceptron model (Rosenblatt 1960; Rosenblatt 1962). The well understood weakness of single layer perceptron networks is that they are able to learn (with one hundred percent accuracy) only those functions that are linearly separable. Despite this weakness, using a modified learning algorithm single layer perceptron networks have been shown to work well in terms of generalization accuracy in relation to other learning models on many learning problems (Andersen and Martinez 1999). However, since many problems of interest do not exhibit aspects of linear separability, the upper bound on the generalization performance of a single layer perceptron for such problems is lower than it is for learning models that are capable of rendering arbitrary decision surfaces, such as multilayer perceptron networks (MLPs). Since MLPs are capable of going beyond the limited set of linearly separable problems and solving arbitrarily complex problems (assuming that the computational resources are unbounded), a great deal of effort has been devoted to the development of MLP training algorithms.

A primary drawback to many of the current MLP training methods is that they require the specification of 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 appropriate for the problem at hand, and it may not even be capable of converging to a solution. This has led to the development of several MLP training algorithms that do not require the user to specify the network architecture a-priori. Some of these include network construction techniques such as Cascade Correlation (Fahlman and Lebiere 1990), DCN (Romaniuk and Hall 1993), node splitting (Wynne-Jones 1992), ASOCS (Martinez and Vidal 1988; Martinez and Campbell 1991), DNAL (Bartlett 1997) Upstart (Frean 1990), Meiosis (Hanson 1990), Perceptron Cascade (Burgess 1994), the Tower and Inverted Pyramid algorithms (Gallant 1986), Tiling (Mezard and Nadal 1989), and Extentron (Baffes and Zelle 1992). All of these methods are error driven approaches that dynamically generate a network structure for solving the given training set during the training phase.

Other approaches to the architecture selection problem include minimum message length (MML) based methods, which use a complexity/accuracy tradeoff to determine the appropriate network architecture (Murata and Yoshizawa 1994; Fogel 1991; Hochreiter and Schmidhuber 1997; Lappalainen 1998; Leonardis and Bischof 1998; Rissanen 1986; Rissanen 1987). Cross validation uses the performance of the network on a holdout set to determine the optimal architecture. There are also techniques, such as Bayesian training (MacKay 1999; Neal 1996; Barber and Bishop 1997; Bishop 1995), early stopping (Amari et all 1997; Finnof, Hergert, and Zimmermann 1993; Sarle 1995; Wang, Venkatesh and Judd 1994), connection pruning algorithms (Mozer and Smolensky 1988; Karnn 1990; Reed 1993; Solar, Le Cun, and Denker 1990; Stork and Hassibi 1993; Won and Pimmel 1991; Weigend, Rumelhart, and Huberman 1989) and weight decay (Bartlett 1997; Krogh and Hertz 1999), that seek to obviate the need to specify an 'optimal' MLP architecture, instead using the most complex architecture that can be practically implemented.

This paper presents a dynamic method for incrementally constructing multilayer-layer perceptron networks called DMP3 (Dynamic Multilayer Perceptron 3), which is an improvement of the DMP1 (Andersen and Martinez 1996A) and DMP2 (Andersen and Martinez 1996B) algorithms. The basic DMP3 algorithm cycles between two phases, a training phase and a growth phase. Initially, DMP3 starts with a single node in the network (the root node). If, after the training phase, the network network has failed to reduce the error to an acceptable level then the algorithm enters the growth phase. During this phase new network elements are connected to the existing network structure. The existing network weights are frozen, and the weights that connect the new elements to the existing network are initialized to predetermined values. This creates a predefined niche for the new elements to fill as the elements are trained to add to the information that is embodied by the current network weights. Information gain is used to guide the growth of the network and the training of network elements. If the addition of new elements does not produce any improvement in information gain over the old network, then a small increase is made in the complexity of the new network elements and the elements are retrained. The algorithm terminates when it cannot improve the information gain of the network without a large jump in the complexity of the network structure. Section 2 discusses aspects of the DMP3 algorithm in detail, and also gives a formal description of the algorithm.

DMP3 incorporates several strategies that differ from the approaches employed by other MLP construction techniques.

- DMP3 naturally prevents overlearning by ceasing to grow the network structure when the current network error cannot be reduced by an incremental increase in the complexity of the current network structure.
- DMP3 does not connect the outputs of previously allocated units to the new network elements, which helps to prevent the new elements from overlearning by limiting their fan in.
- The output node of the network does not change from one iteration to the next. Also, once trained the weights which connect elements of the network to the output node are frozen and are not allowed to change as new elements are added to the network. Rather, DMP3 augments the existing knowledge of the output node through small, incremental addition of network elements as required.
- When required, DMP3 provides for a modest increase in the complexity of newly allocated network elements, which helps newly allocated elements to assist the network as it becomes increasingly difficult to decrease the remaining network error.
- The MLP construction algorithms which are most similar to DMP3 use a divide and conquer approach that partitions the training set, training each network element with only a portion of the available data, which can reduce the reliability of individual network elements. With DMP3 each network element is trained on the entire training set.

- Information gain rather than error minimization is used to guide the training of network elements. This tends to produce more useful feature detectors for strategies such as DMP3 that grow the network in an incremental fashion.

The differences between DMP3 and other MLP construction techniques along with other related work are discussed in detail in section 3.

The DMP3 algorithm is tested on 9 real world data sets obtained from the UCI machine learning database. The performance of DMP3 is compared with several other learning methods, which include c4.5, cn2, ib1, c4, id3, a single layer perceptron network, and mml. DMP3 is also compared against a CV based MLP architecture selection strategy. The reason for choosing to compare DMP3 with a CV based MLP architecture selection is that CV does not require the fine tuning of any adjustable parameters, nor does it require that a strategy be implemented to avoid overlearning. DMP3 is not compared empirically with other network construction techniques due to the number of tunable parameters that must be adjusted for most of these algorithms and the choices that must be made (such as which method to use to prevent the network from overlearning) in order to produce good results, which is beyond the scope of this paper. Also, many of the algorithms are not given with sufficient to detail to allow for an accurate implementation. The results that are reported in this paper show that the individual networks produced by DMP3 have on average better generalization performance than the other, learning algorithms on the data sets tested. Due to the large number of algorithms and data sets tested, this result is sufficient to show the utility of the DMP3 algorithm from an empirical standpoint. It is also shown that it is possible to significantly improve the performance of DMP3 by using bagging to combine several DMP3 networks, which improves the performance of DMP3 on every data set tested. This indicates that DMP3 networks, while capable of good individual performance, are also good candidates for bagging. Section 4 discusses the data sets and methods and details the results of the experiments. The conclusion is given in Section 5.

## 2. DMP3 (Dynamic Multilayer Perceptron 3)

### 2.1 The Basic DMP Algorithm

While it is a simple matter to extend the DMP algorithm to handle multiple output classes, in the following it is assumed that the learning problem has a single, 2 state output in order to simplify the discussion.

The basic DMP algorithm begins with a network composed of a single node (the output node). An example of what the network looks like at this point can be seen in figure 1 (Iteration 1). With this example, in the first iteration the network is composed of a single node, and the learning problem has two input features, $f_1$ and $f_2$. The network is trained to minimize an appropriate error function, at which point all of the weights in the network are frozen. If after training the network fails to correctly classify some portion of the examples in the training set, then two child nodes are allocated and connected to the output node (Iteration 2 of figure 1). The two new child nodes will be trained to assist the output node in correctly classifying the misclassified examples. In order to accomplish this, one of the child nodes (which we label the left child) is biased to assist the network with any misclassified positive examples by initializing the weight that connects it to the output node to a positive value. Conversely, the weight that connects the right child to the output node is initialized to a negative value, which biases the right child to assist the network with negative training examples. Training of the network then resumes. However, the only weights that are allowed to change are the weights of the newly allocated children and the weights that connect the newly allocated children to the output node. This is represented graphically in figure 1 by bolding the links that have updateable weights. If, after the new children have been fully trained the network still fails to classify a sufficient number of the examples in its training set, then two additional children are allocated and connected to the output node of the network (Iteration 3 of figure 1). As before, during the training phase only the most recently added weights are modified, and all other weights (the non-bolded links in iteration 3 of figure 1) are frozen. This process can continue until either the network error falls below some threshold, or until the addition of more children fails to significantly reduce the network error.

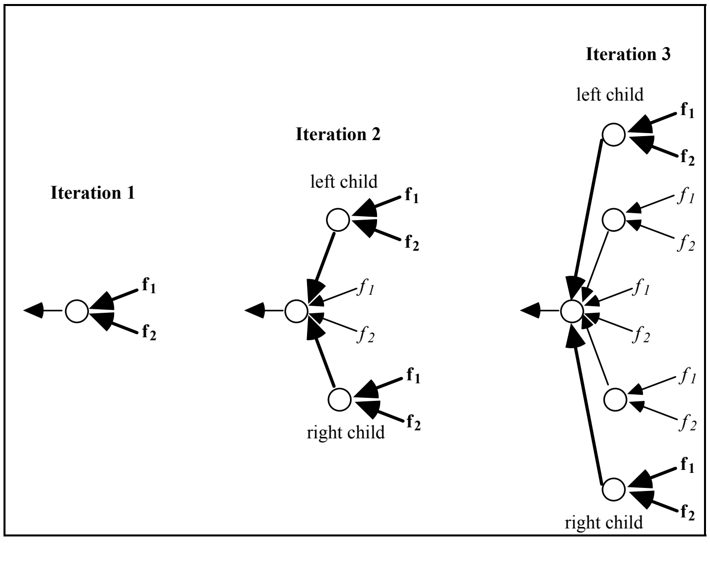

Figure 1. Progression of the DMP algorithm.

The intuition behind the DMP approach is to correct misclassified training examples by adding nodes to the network that are specifically targeted at examples from a particular class. So, if the network incorrectly classifies a subset of positive training examples from its training set, then a new node (the left child) should be allocated with the specific purpose of assisting the parent when it sees such an example. Conversely, if the network incorrectly classifies some of the negative training examples, a new node (the right child) should be allocated to assist the parent whenever a misclassified negative example is encountered. This forces a structure on the network that helps each new child to quickly find a niche to occupy in assisting the network in classifying elements of the training set.

## 2.2 Important Elements of the DMP algorithm

There are several details that must be considered in order for the DMP algorithm to produce networks that exhibit good generalization performance:

- The choice of the error function
- The types of child nodes to add to the network
- Setting the child to parent weights
- Where to grow the network
- How to train sibling nodes

These details are discussed in sections 2.2.1 through 2.2.6.

### *2.2.1 The Error Function - Guiding the Network Growth*

The choice of which function to minimize with the training algorithm is a critical element of the DMP algorithm. A standard approach is to minimize the error (the difference between the network output and the desired response), but this approach can create problems with the growth of the network. When error minimization is used as the basis for training the network it is possible for the growth of the network to reach a premature dead end, in which case the network will fail to correctly classify all (or as many as possible) of the examples in the training set. Also, if the reduction that each child node makes to the total network error is small, the size of the network can become large.

Figure 2 illustrates how the use of error minimization partitions the training set for each child. In this figure there are examples from two different output classes, X's and O's. The training set for the network is shown in the leftmost box. The best partition (in terms of number of misclassifications) that a single perceptron can do for this particular problem is represented by the line which separates the examples in this box, with the examples on the left of the line classified as X's and the examples on the right side of the line classified as O's. The O's on the left side of the line are misclassified, and a left child will be allocated to assist the network in classifying these examples. In order to assist the network in classifying the misclassified O's, the left child must be able to identify a few of the O's from the X's. In this sense, a de facto training set is defined for the left child, shown in the box located in the upper right hand corner of this figure, which is the set of misclassified O's along with the entire set of X's from the parent node's training set. Conversely, the right child must assist in correctly classifying the misclassified X's, which problem is represented by the instances contained in the box in the lower right hand corner.

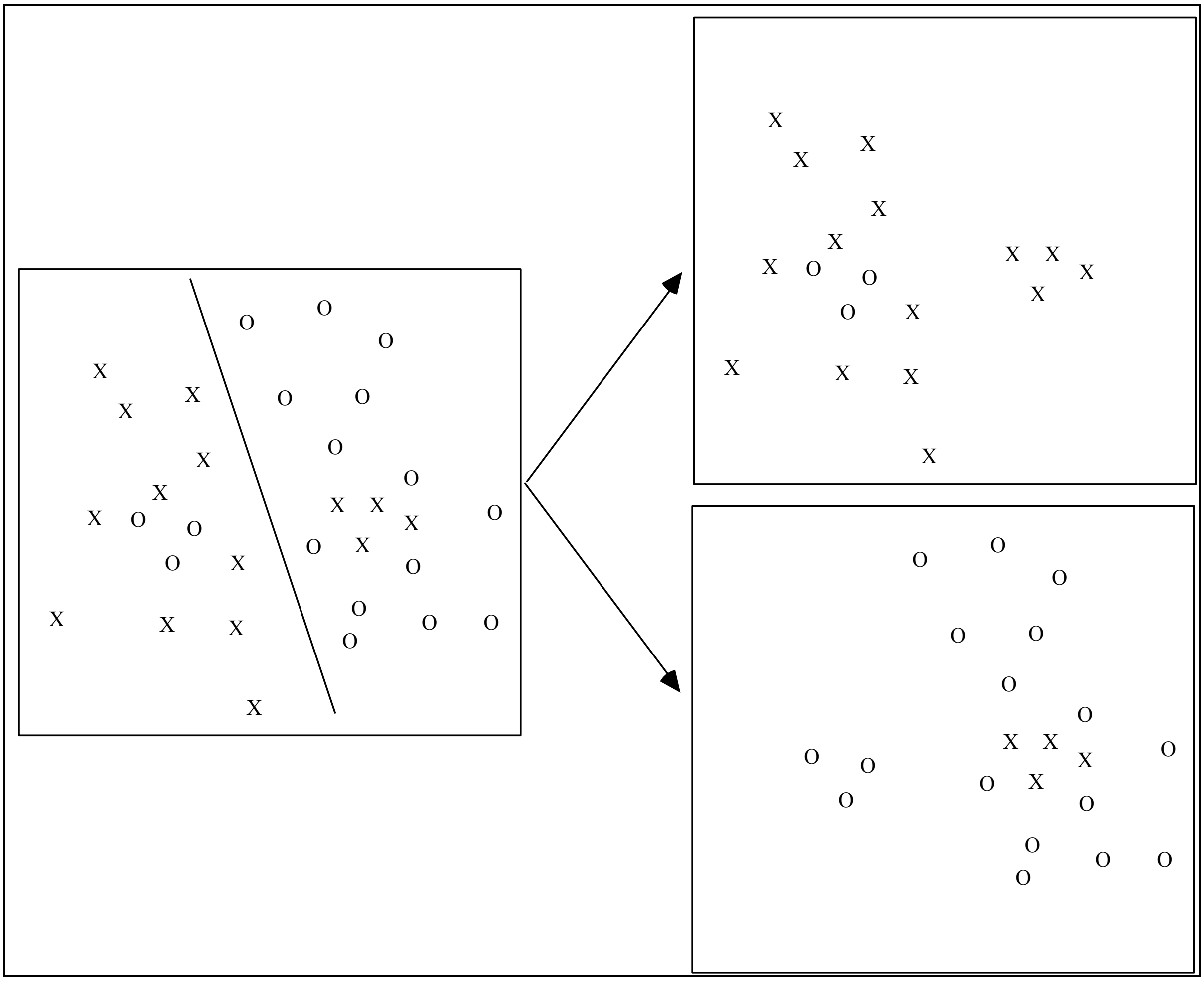

Figure 2. Example training sets for the left and right child.

Figure 2 reveals an inherent problem that is often seen when using error minimization to guide the growth of MLP networks, which is that the children are often not capable of making an appreciable reduction of the network error. Looking at the 'training sets' in figure 2, it is difficult to see how the children are going to be able to reduce the network error any further. This problem is demonstrated by trying to find a 1-d hyperplane (a line) that both maximizes classification accuracy for a child while also identifying at least one of the parent node's exceptional cases. The difficulty is that the examples of each class that the network misclassifies tend to be the hardest examples to separate from the opposite class. In other words, the network already handles all of the easy examples and passes the hardest examples off on its children.

Depending upon the training algorithm, this problem can lead to a dead end in the growth of the network. In the example shown in figure 2, if the child nodes are perceptrons like the parent then the best the left child can do, assuming the children are trained so as to minimize network error, is to classify all examples as X's. The right child is faced by a similar problem, and can do no better than classify all of the examples in its training set as O's. Neither child by itself is capable of correcting the classification of any of the incorrectly classified examples. Following the next step of the DMP algorithm new children will be allocated to handle the misclassified examples. But the set of misclassifed examples will be exactly the same as in the previous iteration, and if the new children are identical to the children allocated during the last iteration it is unlikely that they will perform any differently, and so the network has reached a dead end. Even when network growth doesn't reach a dead end, in order for the network to progress towards a solution in a timely fashion each child must correct the output of at least one training example, otherwise the size of the network could become arbitrarily large.

Due to dead ends encountered in the growth of the network structure, training each node to maximize classification accuracy does not always produce a network with the maximum possible classification accuracy on the entire training set. Therefore, if the goal is to incrementally grow a network that is both small and maximizes classification accuracy on the training set, some criteria other than minimization of training set error should be used to evaluate the performance of the individual nodes that are incrementally added to the network architecture. Obviously, in general the primary goal is not to maximize training set accuracy, but to maximize generalization performance. In some cases it may be desirable to quickly terminate the growth of the network. But for many cases training each node to minimize its training set error will make it difficult for the network to learn an appropriate decision surface for complex problems.

With this in mind it is informative to consider the amount of information that each node contributes to the current network knowledge. Figure 3 shows a training set with two possible hyperplanes (or lines for the two dimensional case) passing through the input space. If the hyperplanes are viewed as decision surfaces, with the arrow indicating the side of the hyperplane in which examples are classified as O's, then the leftmost hyperplane of figure 3 maximizes classification accuracy for the training set of this particular problem. However, if each hyperplane is viewed as a partition of the set of training examples, then the rightmost hyperplane provides the network with the most information.

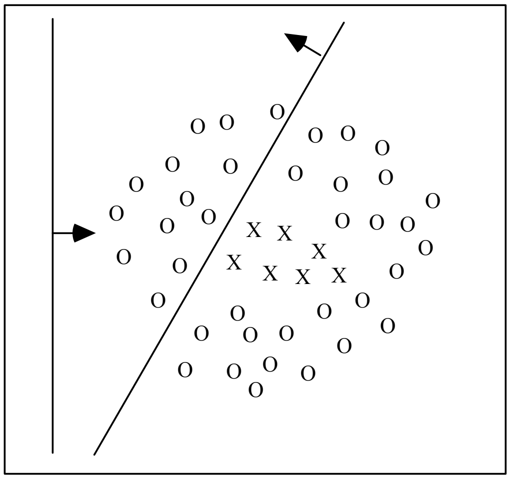

Figure 3. Possible decision surfaces for a simple training set.

The formula for calculating the information contained in a set of examples with *N* possible output classes is given by,

$$I = -\sum_{i}^{N} P_i \log_2 P_i \qquad (1)$$

where $P_i$ is the probability of class *i*. This is the same formula that is used as the splitting criteria for id3 (Quinlan 1986). For the two-class problem, this formula reduces to

$$I(p,n) = -\frac{p}{p+n}\log_2\frac{p}{p+n} - \frac{n}{p+n}\log_2\frac{n}{p+n} \qquad (2)$$

where *p* is the number of positive examples in the training set and *n* is the number of negative examples. When the set of training examples is partitioned by the network into two sets (those with high output and those with low output), then the formula for calculating the amount of information that the partition has is

$$I(partition) = \frac{p_{high} + n_{high}}{p+n} I\left(p_{high}, n_{high}\right) + \frac{p_{low} + n_{low}}{p+n} I\left(p_{low}, n_{low}\right) \qquad (3)$$

where $p_{high}$, $n_{high}$ is the number of positive/negative examples respectively for which the network has a high output, and $p_{low}$, $n_{low}$ is the number of positive/negative examples for which the network output is low. The information gain for a given partition of the training set is then

$$I(p,n) - I(partition) \qquad (4)$$

(Since *I(p,n)* is independent of the partition we can ignore it, the goal then is to maximize *I(partition)*). The information gain for the leftmost hyperplane of figure 3 is 0, while the rightmost hyperplane has an information gain of 0.085. From the standpoint of information gain the rightmost hyperplane is clearly better than the leftmost hyperplane.

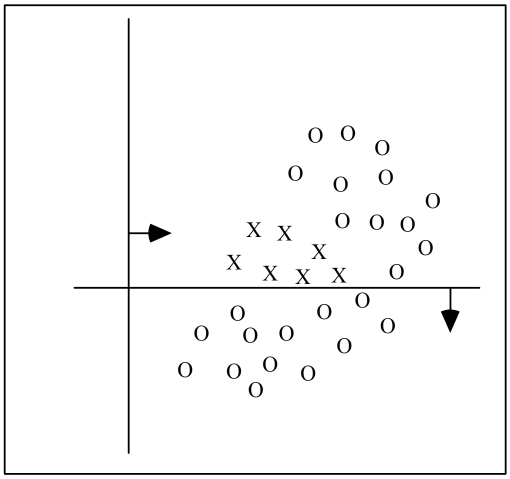

Figure 4. 2nd step of DMP using info-gain.

The reason why the rightmost hyperplane is a better choice for the network as a whole becomes evident at the next few steps of the DMP algorithm. Assuming the hyperplane that maximizes information gain is chosen, at the next step DMP will allocate a left child which will be expected to assist in correcting the output for the misclassifed examples from the O output class (since there are no misclassified X's there may be little for the right child to do in this case), which problem corresponds to the examples illustrated in figure 4.

Of the two hyperplanes shown in figure 4, the horizontal hyperplane produces a larger information gain (0.194) than the vertical hyperplane that maximizes classification accuracy for the left child's training set (info gain of 0). If the left child generates the hyperplane with the largest information gain, then it is possible for the network to correct its output on all of the O examples that are below the hyperplane. At this point, the only misclassified O's that are left are shown in figure 5. At the next step of the DMP algorithm two more children will be allocated and the left child will be expected to assist in separating these remaining misclassified 0's from the X examples. This can be accomplished if the left child generates the hyperplane shown in figure 5 (which is the hyperplane that maximizes both classification accuracy and information gain), at which point the network has converged to a solution and the algorithm is finished.

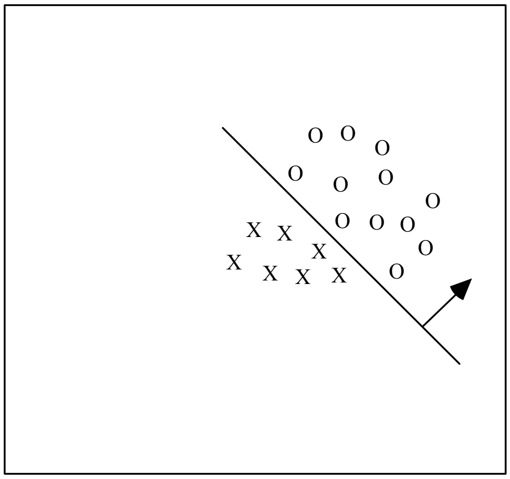

Figure 5. Last step of DMP.

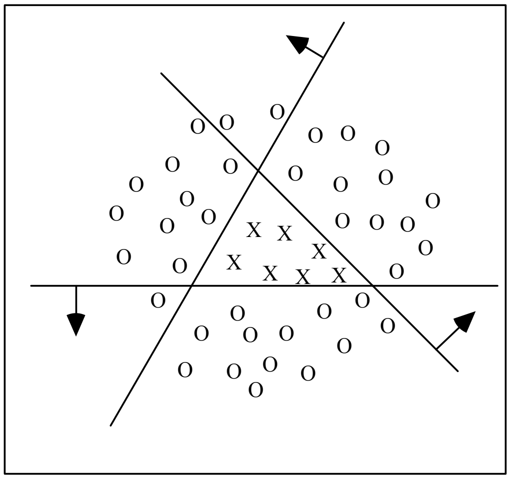

Figure 6. The final solution.

For this example, the final solution generated by DMP using information gain instead of error minimization is the three hyperplanes shown in figure 6. These three hyperplanes completely separate the set of X's from the O's that surround them.

### *2.2.2 Increasing the Computational Power of the Children*

If a child has the same computational power as its parent it can be difficult for the child to assist in the identification of any of the parent node's errors. Since new children are required to assist the network on the more difficult parts of the training set it makes sense to increase the computational power of the children as training progresses. In general there is no need to restrict a child to be a single perceptron node (or even to restrict the initial network structure to be a single perceptron node for that matter). New children can be made computationally more powerful by letting the children be small MLP networks, and then providing for small increases in the number of nodes in the hidden layer of new children that are being added to the network as required. As the network grows and it becomes increasingly difficult for new children to boost the performance of the network, the computational complexity of children that are added to the network can be increased by increasing the number of nodes in their hidden layer(s) by one or two hidden nodes.

The increase in the complexity of new children should not be unbounded, and should only be done when required to improve the performance of the network. It is always possible to increase the complexity of the children (or initial network structure) to the point that it completely solves the training set, but this approach suffers from over learning and memorization. On the other hand, if the increase is too small then a child might not identify any of the parent's errors, which results in a dead end in the growth of the network. A dead end in the growth of the network will not always be detrimental, however, since it naturally limits the complexity of the network and can help to prevent over learning.

The appropriate increase in the complexity of a child over that of the parent is an open question. This is a difficult problem, and it is possible that a single, optimal solution (for all types of 'interesting' learning problems) does not exist. But in keeping with the incremental nature of DMP, and in order to avoid over learning, the increase in the complexity of a child as training progresses is kept small. If, despite a small increase in computational ability, a child is unable to identify any of the parent's errors then it is acceptable for the network to cease to grow at that point. This amounts to a form of bias that favors networks that can be grown easily and incrementally. When the network's errors are too difficult for new children to identify without a large increase in the complexity of the new children then these errors are assumed to be noise and ignored, and the growth of the network at that point is halted.

### *2.2.3 Using Small, Trainable Parent to Child Weights*

The organization of neurons into a predefined, highly constrained network structure where each neuron (or group of neurons) is assigned a relatively specific task in relation to the other neurons in the structure is often seen with biological systems. This approach makes sense when it is known in advance the approximate type of function that each neuron should perform, and the predefined structure enables each network element to learn the appropriate function better and more efficiently than would otherwise be possible. For many real world problems of interest the only information about the problem domain that is available is a finite set of pre-classified examples, in which case it is extremely difficult to specify an 'optimal', highly constrained network structure a-priori. Nevertheless, it is possible to realize some of the benefits of a constrained network structure by partially constraining the structure as occasion permits during the training process, and this is the approach that DMP employs.

With the DMP approach, one child, which we have labeled the 'left' child, is responsible for assisting the parent when it detects a positive example that has been misclassified, and the other child (or 'right' child) is responsible for assisting with the negative exceptions. This means that the weight between the left child and the parent should be positive, since a positive weight will push the network towards a positive output. Similarly, the weight between the right child and the parent should be negative. By maintaining the positive and negative values of the left and right child weights, this forces a structure on the network where each child is assigned a relatively specific task. Furthermore, aside from the weights of the newly added left and right child, all of the other weights in the network are fixed during the training cycle. This makes it less difficult for newly added children to discover what function they should perform in relation to the other elements of the network, which greatly reduces the time needed to train the children and can also reduce the total time needed to train the network.

The weights between a child and the parent should initially be set to a value which is large enough to guarantee that the child will fill the desired niche in the network structure, but should not be so large that the network becomes overly sensitive to the child's output. In addition, the training algorithm should be allowed to adjust the child to parent weight so that the child can be of maximal benefit to the network. With DMP, the parent to left child weight is initialized to a positive value, and the parent to right child weight is initialized to a negative value. These weights, along with the other weights of the newly added children, can then be updated with the training algorithm while all other weights in the network are frozen. The initial values for the parent to child weights bias the left and right children to detect positive examples and negative examples, respectively. Having the parent to child weights trainable does make it possible that the weights could change sign, but in practice this does not occur and the final values for the parent to child weights are generally near their initial settings.

### *2.2.4 Only Grow at the Root Node*

If a child node fails to correct all of the network error that it has been assigned then it is possible to correct the remaining network error by allocating more children and connecting them directly to the child node. If in turn these new children fail to correct all of the network error then their output can be corrected by adding children to them. This process can continue until the network converges to a solution.

However, instead of adding children to children in many cases it may be better to only add children to the root node of the network. With the approach used in this paper, if the network (root node along with its current set of children) does not correctly classify all of the examples in the training set, then more children are allocated and connected to the root node in an attempt to correct the network's remaining error. By connecting the children directly to the root node of the network, the output of the children have a more direct effect on the output of the network, and the training time for the children is expedited since the error is not diluted by being propagated through multiple network layers.

### *2.2.6 Train Siblings Together*

It is possible for the children to be trained separately from the network and from each other if desired. However, with the DMP3 approach the newly added children are trained together using standard backpropagation, where the only weights that are updated are those of the newly added children. By so doing, the left and right child can cooperate to achieve better performance than is possible when they are trained separately. For example, when the left child incorrectly outputs high on a negative example it may cause the network to commit an error that it would not have otherwise committed. In this case the right child can help counteract the detrimental high output of the left child if the right child's output is also high (since when both children output high their outputs will tend to cancel each other), thus restoring the network to the original (correct) output. By training the two children at the same time using backpropagation, each sibling can detect when it might be able to help the other in this fashion, and adjust its weights accordingly. This leads to a quicker reduction in the total network error, and smaller networks in general.

## **2.3 DMP3**

### *2.3.1 The DMP3 Algorithm*

A pseudo code version of the implementation of the DMP algorithm used for this paper, which we call DMP3, is given in figure 7. In order to prevent the choice of a network which has settled into a sub-optimal local minima, with each training phase three copies of the current network (each copy has its trainable weights initialized to small, random values) are trained using information gain, and the copy of the network that has the best information gain is chosen and the other two copies are discarded. The weights of the current best network are frozen, a copy of the network is made, and the new network is augmented (if needed) by connecting a newly allocated left and right child to the output node.

The child to parent weights of the left child are initially set at +10.0, and the weight which connects the right child to the parent is set to -10.0. Three copies of the augmented network are then trained with IDT using info-gain, and the copy with the best info gain is chosen. The only weights that are allowed to change during training are the weights of the newly allocated children, and the weights that connect the newly allocated children to the root node of the augmented network. If after training the augmented network exhibits greater info-gain than the current best network, then the current best network is replaced with the augmented network.

Initially, new left and right children added to the network have zero hidden nodes. However, if an augmented network does not improve the info-gain over the current best network, then the complexity of newly allocated children is increased by adding 1 hidden node to the hidden layer of each child, and the process is repeated (each node of the child is fully connected to the original input features). Initially, the base level of complexity for newly allocated children is set at 0 hidden nodes, but as the network grows the base level of complexity can be incrementally increased. For example, if the current iteration of the DMP3 algorithm requires children with 2 hidden nodes to decrease the network error, then the next iteration will start with the complexity level for new children set to 2 hidden nodes. The next iteration will first try children with 2 hidden nodes, and if that fails to reduce the network error it will try children with 3 hidden nodes, and finally children with 4 hidden nodes (the algorithm terminates when 3 contiguous cycles of the algorithm fail to improve the info gain of the current best network). This corresponds to the belief that the problem of reducing the remaining network error tends to become more difficult with each iteration of the algorithm, and so each iteration ought to start with the complexity set at least as high as the level of complexity that was required for the previous iteration. However, in keeping with the incremental DMP network construction approach, the maximum increase from one iteration to the next in the starting complexity level for new children is small (2 nodes). DMP3 thus implicitly limits the complexity of the network by allowing only small, incremental increases to the complexity level of newly allocated children, and if these increases are not enough to reduce the network error any further the algorithm terminates.

```
DMP
BEGIN
  h = 0        //the number of hidden nodes for new children.  Initially set to 0
  start with a single node perceptron network
  train  3 copies of the network
  choose the trained network with the best information gain
  let parent_net = the chosen network
  freeze the weights of parent_net
  noimprovement = 0
  do
    new_net = CopyNetwork(parent_net)
    allocate a left and right child with h hidden nodes each
    connect each of these children to the root node of new_net
    set left child to root node weight = 10
    set right child to root node weight = -10
    train  3 copies of new_net with IDT
    let best_net = the copy of new_net that has the best info gain
    if info_gain(best_net) > info_gain(parent_net)
      parent_net = best_net
      noimprovement = 0
      freeze the weights of parent_net
    else
      noimprovement = noimprovement + 1
      h = h + 1
    endif
  while (noimprovment < 3)
  return parent_net
END
```

Figure 7. DMP3 algorithm.

Figure 8 shows an example of a sequence of networks that could be produced with the implementation of the DMP3 algorithm given in figure 7 (while all nodes are fully connected to the original input features, these connections have been omitted for the sake of clarity). In this figure each box represents an iteration of the DMP3 algorithm. Initially (the beginning of iteration 1) the network is composed of a single node. With each iteration DMP3 attempts to improve the performance (measured by the entropy or information gain of the network) of the previous iterations best network by incrementally adding children of increasing complexity. For example, in the 2nd iteration shown in figure 8 DMP3 first adds two children composed of a single node each. When this fails to improve the performance, DMP3 increases the complexity of the children by a single hidden node and retrains the network. This succeeds in improving the performance of the network, and DMP3 enters the next iteration. In the

3rd iteration DMP3 first tries adding children with 1 hidden node each (since this is what worked during the last iteration), if this fails to improve the performance of the network DMP3 then tries children with two hidden nodes, and if this still fails to improve the performance DMP3 tries children with 3 hidden nodes. If none of these networks improves the performance then the algorithm terminates, and the previous iteration's best network is chosen, which for this case would be the 5 node network from iteration 2.

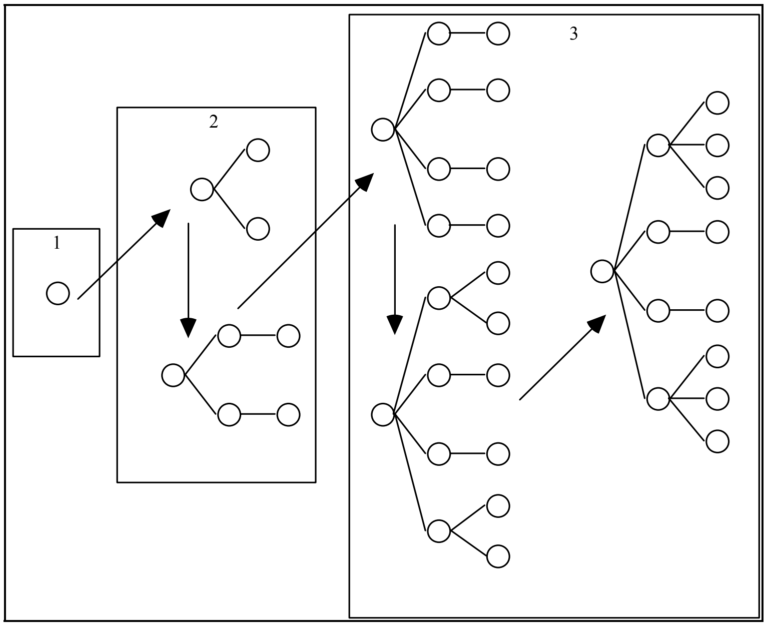

Figure 8. Example progression of the DMP3 algorithm.

### *2.3.2 The Training Cycle*

A short, dynamic training cycle, which we call IDT for "improvement driven training", is used to train new children when they are added to the DMP3 network. Figure 9 gives a pseudo code version of the IDT training algorithm. IDT works as follows. Initially, new children are trained for 1000 iterations. This step corresponds to the IGTrain function call, which is defined in figure 10. After this initial phase, training continues as before but after each 10 training iterations the network is tested on the training set to determine its training set performance (measured in terms of information gain for the DMP3 algorithm), and the weight setting with the best performance is saved. Training is halted if the network fails to improve upon the best weight setting on 20 consecutive tests of the performance of the network, at which point the saved weight setting is restored to the network. This step corresponds to the LazyTrain function call, which is defined in figure 11.

```
IDT(net)
BEGIN
   net = IGTrain(net, 1000)
   net = LazyTrain(net, 10, 20)
END
```

Figure 9. IDT.

Pseudo code for the IGTrain function is given in figure 10. Since each step of the DMP3 algorithm attempts to generate a network that maximizes the information gain, a slight modification was made to the calculation of the network error for each training example. This modification weights the error for each incorrectly classified example by the (approximate) normalized amount of information gain that would be expected if the example were correctly classified. Since the calculation of information gain is computationally expensive, the expected information gain is only calculated once for every pass through the training set.

```
IGTrain(net, maxi)
BEGIN
   for i = 1 to maxi
      current_entropy = CalculateEntropy(net)
      let neg_entropy = entropy of net with 1 less negative error
      let pos_entropy = entropy of net with 1 less positive error
      neg_info_gain = neg_entropy - current_entropy
      pos_info_gain = pos_entropy - current_entropy
      normalize_value  = MAX(neg_info_gain,pos_info_gain)
      neg_err_adjust = neg_info_gain/normalize_val
      pos_err_adjust = pos_info_gain/normalize_val
      Randomly permutate training instances
      for each incorrectly classified example e in the training set
         error = CalcError(net, e)
         if e.target = high let error = error * pos_err_adjust
         else error = error * neg_err_adjust
// Update the nonfrozen weights with standard back propagation
         UpdateNonfrozenWeights(net,error)
      endfor
   endfor
END
```

Figure 10. Training function with modified error.

For example, given a training set with 15 examples, 10 of which have a high target classification, let 7 of the positive examples and 3 of the negative examples be correctly classified by the current network. If 1 of the incorrectly classified positive examples somehow became correctly classified (without affecting the classification of any of the other training examples) the information gain for the network would be 0.025. On the other hand, if one of the incorrectly classified negative examples was corrected the information gain would be 0.053. Normalizing these two numbers, the error for an incorrectly classified positive example would then be adjusted by multiplying it by 0.47, and the error for a negative example would be unchanged. This weights the error of each example proportional to the degree it can benefit the network in terms of information gain, which will tend to adjust the weights so that the examples that will produce the greatest information gain are correctly classified.

The pseudo code for LazyTrain is given in figure 11. The LazyTrain algorithm trains the network for a variable length of time, giving up when no progress has been made during the last few cycles of the algorithm.

```
LazyTrain(curr_net, maxi, maxtries)
BEGIN
   best_so_far = copy(curr_net)
   for x = 1 to max_tries
      IGTrain(cur_net, maxi)
      if info_gain(curr_net) > info_gain(best_so_far)
         best_so_far = copy(curr_net)
         x = 1
      endif
   endfor
   return best_so_far
END
```

Figure 11. Variable length portion of the training phase.

The primary reason for choosing to use a short training cycle was due to time constraints, but there are other benefits, such as the avoidance of over learning. Simulation results using standard MLP networks trained with IDT and vanilla back propagation indicate that it tends to produce weight settings with better generalization performance than networks trained with a long, static training cycle.

## 3. Related Work

### 3.1 Network Construction Algorithms

The majority of network construction methods are typified by a network that starts from a very simple basis, usually one node, and adds nodes and connections as needed in order to learn the training set. These strategies include Cascade Correlation (Fahlman and Lebiere 1990), DNAL (Bartlett 1997), Tiling (Mezard and Nadal 1989), Extentron (Baffes and Zelle 1992), Perceptron Cascade (Burgess 1994), the Tower and Inverted

Pyramid algorithms (Gallant 1986), and DCN (Romaniuk and Hall 1993). Other construction algorithms include Meiosis (Hanson 1990), node splitting (Wynne-Jones 1992), and (Marchand and Rujan 1990).

Several network constructions algorithms such as Tower, Inverted Pyramid, DCN, Tiling, and Extentron grow the network by creating a new output node and connecting the existing network to it, with the old output node becoming an input node for the new output node. This approach may not work well for many learning problems since the old output node has not been trained to be a feature detector, and may not be as beneficial to the network as would otherwise be possible. DMP3 takes a different approach, augmenting the output node by connecting new network elements to it and then training these elements to assist the output node. Like DMP3, Cascade Correlation does not create new output nodes to grow the network, instead connecting the output of the newly allocated node to the output node of the network. But with the basic algorithm each new node receives inputs from all other nodes (except the output node) in the network that can lead to units with a large fan in. Much like connecting the old output node to the new one, connecting previously allocated units to new units may not provide a great deal of benefit to the newly allocated node, since the previously allocated units were not trained as feature detectors for the new unit. In addition, after training new candidate units Cascade Correlation retrains all of the output node weights, which may cause the output node to unlearn important information. This becomes more likely as the fan in to the output node increases, since the number of output node weights may become too large for the training set to properly constrain. With DMP3, newly allocated network elements do not receive input from any previously allocated units. DMP3 also retains any knowledge that has been learned by the network by not retraining any of the previously allocated output node weights. While new evidence can over-ride the current knowledge embodied in the network weights, the network does not 'unlearn' that knowledge (the weights remain unchanged).

While most of the incremental MLP construction algorithms use error minimization, DMP3 uses information gain to guide the network construction process. This helps DMP3 to avoid adding elements that will not be beneficial to the network, and it also helps to avoid premature termination of network growth. The utility of using information gain is discussed in detail in section 3.1.1.

One of the drawbacks of most current MLP construction algorithms is that they do not have built in mechanisms to prevent the network from overlearning, rather treating this important subject as an afterthought. For example, (Burgess 1994) states that "for good generalization it is necessary to restrict the size of the network to match the task," but no specific algorithm is presented on how to do so. Left uncontrolled, all of these methods will suffer from over learning, and so in some respects they do not avoid the architecture selection problem but must utilize some type of architecture selection strategy (such as CV or MDL based strategies) in an attempt to avoid over learning. This is due to the fact that, left uncontrolled, the network structure can grow to fit the training set data exactly. But with many problems the training data may contain noise that will cause the algorithm to perform worse if the noisy instances are memorized. Also, the network can grow to the point that the amount of training data is insufficient to properly constrain the network weights. A common technique used to avoid this problem is to balance the complexity of the network versus the performance of the network on the training data. Another approach is to use a holdout set to determine the point at which the growth of the network should cease. DMP3 naturally limits the complexity of the network by terminating the growth of the network when the network error cannot be reduced without a large increase in the current network complexity, which reduces the probability that the network will suffer from overfitting.

Of the several MLP construction algorithms DMP3 is most similar to the Upstart algorithm (Frean 1990). But the Upstart algorithm can be susceptible to over learning and memorization due to the divide and conquer, exception handling approach that Upstart uses. This is highlighted by the four following characteristics of the Upstart algorithm.

- Upstart bases the training set for children on the parent's exceptional cases, and exception handling mechanisms are in general susceptible to noise, since it is difficult to distinguish noise from exceptional cases.
- Children have complete control over the parent output. The confidence that the parent has in its output is not taken into account, nor is the confidence that the child has in its output. If a child detects an exception, then the parent must ignore whatever conclusion it has made and do what the child tells it to do. For an MLP network, this essentially means that the magnitude of the child to parent weights must be quite large in order for the child to be able to force the parent to output what the child determines is correct regardless of the other inputs the parent receives. This makes the network susceptible to child errors.
- A child node is trained on fewer training examples than a parent node. It seems counterintuitive to trust the

prediction of the child more than the parent when the function that the child performs is determined from a subset of the parent node's training set. Furthermore, children can have children, and the further removed a child node is from the parent, the smaller its training set tends to be. This means that the leaf nodes of the network are trained on fewer training examples than other nodes, and so will tend to be more likely to produce classification errors. Since the leaf nodes in the network also have the greatest influence on the output of the network, Upstart networks sometimes exhibit worse generalization performance than the root node of the network exhibited before the addition of children.

- For MLPs, the decision surface that a child generates extends infinitely beyond the region occupied by the training cases. This is normally not an area of concern for an MLP since in the standard training approach every node in the network sees every training example, and it is generally assumed that the training set sufficiently covers the input space. However, with Upstart some of the nodes in the network may be trained using relatively small subsets of the available training data, which may not contain enough training examples to adequately cover the input space. For example, as illustrated in figure 12, when a child node is trained on few exceptional cases there may be many possible decision surfaces that identify the exceptions, but each of these decision surfaces can have a drastically different, non-local effect on the network as a whole.

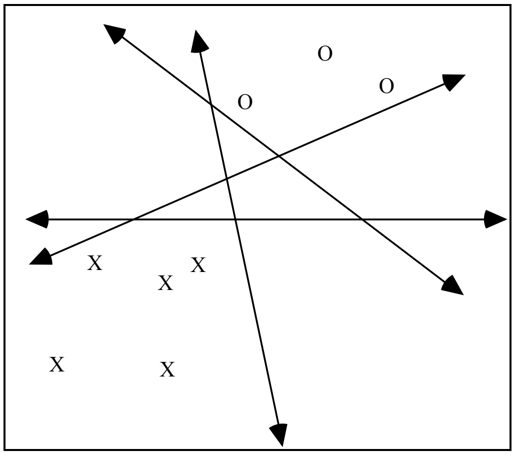

Figure 12. Possible decision surfaces.

While DMP3 is similar in some respects to Upstart, there are several significant differences. These differences include:

- DMP3 softens the exception handling nature of Upstart. With DMP3 the parent to child weights are small and trainable, which differs from Upstart that has extremely large, untrainable weights between the child and parent nodes making the network extremely sensitive to child node errors.
- DMP3 uses information gain to guide the growth of the network. The standard Upstart algorithm uses error minimization, which can lead to premature dead ends in the growth of the network for real valued input data.
- DMP3 abandons the divide and conquer approach used by upstart. DMP3 trains each network element using the entire available training data, rather than only the exceptional cases as done with Upstart, which tends to make the children more reliable than if they were trained on a subset of the available data.
- DMP3 only adds children to the root node. Upstart attempts to correct child nodes that are in error by allocating children to the child nodes. This creates a situation where children can be added to children, which can cause the network to become exponentially large.
- With DMP3 the children can increase in complexity if required. It can become increasingly difficult to correct the remaining error as training progresses. DMP3 attempts to alleviate this problem by allowing for modest increases in the complexity of the children that are being added to the network structure. No such provision is made in the Upstart algorithm.

Taken together, these modifications can reduce the problem of over learning, foster a greater degree of cooperation between nodes in the network, and can lead to a significant improvement in generalization performance.

## 3.2 Other Architecture Selection Methods

### Early Stopping

Early stopping strategies (Amari et all 1997; Finnof, Hergert, and Zimmermann 1993; Sarle 1995; Wang, Venkatesh and Judd 1994) utilize overly complex network architectures. One of the main advantages of using a network that is more complex than is actually needed is that larger networks tend to have fewer local minima in the error surface defined by the training set. However, with a larger network there is a higher likelihood that over learning will occur. In other words, larger network architectures are more likely to converge to a lower training set error, but often tend to produce higher error on non-training examples. In order to avoid this, early stopping strategies try to determine when the network has been trained sufficiently to do well on the problem but has not yet over learned (or memorized) the training data. One way to do this is to occasionally test the performance of the network on a holdout set and stop training when the performance on the holdout set begins to degrade.

**Cross Validation (CV)**

CV is often used to select an optimal architecture from amongst a set of available network architectures. In a comparison of CV with two other MLP architecture selection strategies in a recent paper (Schenker and Agarwal 1996) CV was found to be the best at choosing the optimal network architecture, at least on the data sets tested. However, the comparison was based on only a single type of artificial data and did not look at any real world problem domains.

In a larger study (Andersen and Martinez 1999B), CV was found to not perform as well as desired when selecting an optimal architecture from a large set of relatively similar architectures. Several strategies are suggested which can be applied when using CV based MLP architecture selection to significantly improve the performance CV based architecture selection.

CV is often used to select between different types of learning algorithms. In another paper CV was used to select between a small set of competing learning algorithms (C4.5, C4.5rules, and an MLP trained with backpropagation) (Schaffer 1993). The 3 competing learning algorithms were compared on 5 problems drawn from real world problem domains. Schaffer reported that CV's average performance was several percentage points better than any of the individual learning algorithms on the problems tested. Schaffer states that CV will generally reduce the risk of extremely poor performance, and can also produce higher average model accuracy.

**Weight Decay**

Weight decay adds a penalty term to the error function that favors smaller weights [Bartlett 1997; Krogh and Hertz 1999). The rate of weight decay is often chosen by training several different networks with different rates of decay and then using CV to estimate which rate is optimal.

**Network Pruning**

Pruning techniques start with a large, overly specified network and iteratively prune connections that are estimated to be unnecessary. CV is often used to assist in the estimation process. The pruning can take place during the training process or training cycles can be alternated with pruning cycles. Pruning strategies include Optimal Brain Damage (Solla, Le Cun, and Denker 1990), Skeletonization (Mozer and Smolensky 1988), and Optimal Brain Surgeon (Stork and Hassibi 1993) among others (Karnn 1990; Reed 1993; Won and Pimmel 1991; Weigend, Rumelhart, and Huberman 1989).

## 4 Experiments

### 4.1 Data sets and Algorithms

| tag | full name | attributes | instances | problem description |
|---|---|---|---|---|
| bc | breast cancer | 9 | 286 | recurrence of breast cancer in treated patients |
| bcw | breast cancer wisconsin | 10 | 699 | malignancy/nonmalignancy of breast tissue lump |
| bupa | bupa liver disorders | 7 | 345 | blood tests thought to be sensitive to liver disorders |
| echo | echocardiogram | 13 | 132 | survival of heart attack victims after 1 year |
| ion | ionosphere | 35 | 351 | classifcation of radar returns from ionosphere |
| promot | promoter gene sequence | 57 | 106 | identification of promoter gene sequences in E. coli |
| sonar | sonar | 61 | 208 | identifying rocks and mines via reflected sonar signals |
| sthear | statlog heart | 13 | 270 | presence/absence of heart disease in patient |
| voting | house votes 1984 | 16 | 435 | predict party affiliation from house voting records |

Table 1. Data sets.

9 data bases from the UCI machine learning database were used to test the DMP3 algorithm. Table 1 lists the data sets used in this paper. The first column gives a tag used to identify the data set throughout the rest of this paper. The total number of attributes is listed in the third column, and the fourth column gives the total number of examples contained in the data set. The last column describes the problem domain. For the sake of simplicity we limited the data sets used in this paper to those with two output classes. All of these data sets are based upon real world problem domains, and are more or less representative of the types of classification problems that occur in the real world.

With the exception of the scores for the CV based MLP architecture selection algorithm which were taken from (Andersen and Martinez 1999), the scores reported in this paper for the other learning algorithms are taken from (Zarndt 1995), which is a comprehensive case study comparing the results of several machine learning algorithms across a large variety of data sets. The results from this case study are averages obtained using 10-fold cross validation.  Generally, the various learning algorithms were tested using their default parameter settings.

The DMP3 algorithm is trained/tested using 10-fold cross validation on the same data splits that were used in (Zarndt 1995).  This allows the use of the student t-test to calculate confidence levels and directly compare the results of different learning algorithms on each data set.

| Tag | Fule name | Type of learning algorithm |
|---|---|---|
| mlp | multilayer perceptron | CV based MLP architecture selection |
| per | linear threshold perceptron | single layer perceptron |
| c4 | c4 | c4 decision tree/rule based classifier |
| c4.5tp | c4.5 tree pruned | decision tree/rule based classifier |
| ib1 | instance based 1 | Incremental nearest neighbor approach |
| id3 | id3 | decision tree/rule based classifier |
| mml | IND v2.1 | MML based decision tree selection |
| cn2o | ordered cn2 | decision tree/rule based classifier |

Table 2.  Learning algorithms.

The learning algorithms that DMP3 is compared against are summarized in table 2.  The first column lists the name that we will use to refer to the corresponding learning algorithm throughout the rest of this paper.  The second column gives the common name for the learning algorithm, and the last column lists the type of the learning algorithm.  A brief description of each type of learning algorithm follows.

Decision trees are a well-known learning model that has been studied extensively by the machine learning community.  Decision tree algorithms include c4.5, id3, and IND v2.1 (Buntine 1989; Buntine 1990; Quinlan 1986).  A decision tree is composed of possibly many decision nodes, all of which are connected by some path to the root node of the tree.  All examples enter the tree at the root decision node, which makes a decision, based upon the examples attributes, about which branch to send the example on down the tree.  The example is then passed down to the next node on that branch, which makes a decision on which sub-branch to send the example down.  This procedure continues until the example reaches a leaf node of the tree, at which point a decision is made on the example's classification.

Instance based learning algorithms are variants of the nearest neighbor classification algorithm (Aha 1991; Aha 1992).  With a nearest neighbor approach an example of an unknown class is classified the same as the closest example or set of closest examples (where distance is generally measured in Euclidean terms) of known classification.  The instance based learning algorithms seek to decrease the amount of storage required by the standard nearest neighbor approach, which normally saves the entire training set, while at the same time improving upon classification performance. There are several variants to this approach. Due to space constraints we report only the results for ib1 since ib1 exhibited better overall performance than any of the other instance based learning algorithms on the data sets tested in this paper.

The cn2 rule induction algorithm (Clark and Niblett 1989; Clark and Boswell 1991) uses a modified search technique based on the AQ beam search method. The original version of cn2 uses entropy as a search heuristic. One of the advantages of rules is that they are generally thought to be comprehensible by a human. However, this characteristic is only evident when the number of rules is relatively small.

### 4.2 Results

Table 3 compares the generalization performance of DMP3 with the other machine learning algorithms on the data sets tested in this paper. These results are averages obtained using 10-fold cross validation. The last row of the table gives the average score of each algorithm across all data sets tested. DMP3 has the highest average generalization accuracy of any of the learning algorithms on the data sets tested. The second best scoring algorithm is the CV based MLP architecture selection method. With the exception of CV, the confidence that DMP3 is better than the other learning algorithms on these data sets is .95 or greater. The confidence that DMP3 is better than CV is .7, which is not high enough to be considered statistically significant. However, CV based MLP architecture selection requires an enormous amount of computation in comparison to DMP3 since CV must retrain each network architecture 10 times in order to generate the CV holdout set score for the architecture. DMP3, on the other hand, only trains a single network architecture.

| | DMP3 | mlp | per | c4 | c45tp | ib1 | id3 | mml | cn2o |
|---|---|---|---|---|---|---|---|---|---|
| bc | 73.30 | 69.14 | 66.50 | 71.40 | 73.90 | 71.80 | 66.20 | 75.30 | 66.10 |
| bcw | 95.43 | 95.24 | 93.00 | 95.10 | 94.70 | 96.30 | 94.30 | 94.80 | 95.20 |
| bupa | 70.96 | 72.26 | 66.40 | 64.40 | 62.60 | 62.30 | 65.20 | 67.50 | 58.00 |
| echo | 88.79 | 86.80 | 87.00 | 90.10 | 90.10 | 84.00 | 86.20 | 92.40 | 83.20 |
| ion | 87.86 | 88.17 | 82.00 | 90.60 | 90.90 | 86.30 | 88.30 | 88.30 | 82.60 |
| promot | 87.35 | 90.70 | 75.90 | 76.30 | 77.30 | 82.10 | 74.50 | 79.10 | 87.80 |
| sonar | 80.43 | 78.56 | 73.20 | 71.60 | 73.00 | 86.50 | 74.00 | 72.60 | 55.40 |
| sthear | 80.51 | 78.93 | 80.80 | 76.70 | 73.40 | 79.60 | 77.10 | 81.90 | 78.60 |
| voting | 93.90 | 94.21 | 94.50 | 96.80 | 96.80 | 92.40 | 94.50 | 97.30 | 93.80 |
| Average | **84.28** | 83.78 | 79.92 | 81.44 | 81.41 | 82.37 | 80.03 | 83.24 | 77.86 |

Table 3. DMP3 vs other machine learning algorithms.

Table 4 gives the average size of the network (in number of perceptron nodes) produced by the DMP3 algorithm for each data set. For 2 of the data sets (promot and bc) the DMP3 algorithm never grew the network beyond a single node. The reason that DMP3 did not grow networks with more than a single node on the promot data set is due to the fact that the single layer network achieved 100 percent accuracy on the training set for this particular data set. On the bc data set DMP3 was unable to improve the information gain over that of a single layer network, and so the algorithm terminated with a single node in the network.

| | net size |
|---|---:|
| bc | 1.00 |
| bcw | 6.32 |
| bupa | 16.00 |
| echo | 5.32 |
| ion | 5.60 |
| promot | 1.00 |
| sonar | 3.72 |
| sthear | 9.12 |
| voting | 4.60 |
| Average | 5.85 |

Table 4. Network size.

DMP3 produced the largest networks on the bupa data set, with an average network size of 16 nodes. For most of the data sets, however, DMP3 tends to produce relatively small networks, with an average network size of less than 6 nodes across all data sets. Prior research (Andersen and Martinez 1999) has shown that simple learning algorithms can exhibit quite good generalization performance on many learning problems, so it is not surprising that DMP3 is able to produce good results with such small networks.

### 4.2.1 Bagging DMP3 Networks

For a given network architecture and training set there can be many different weight settings that exhibit equivalent (or nearly equivalent) training set performance. While the training set performance of these weight vectors may be equivalent the generalization performance can often differ significantly. But it can be difficult or impossible to determine the weight vector(s) that has the best generalization performance. With DMP3 and other learning algorithms that generate dynamic network topologies this problem is exacerbated, since the algorithm must select an optimal architecture *in addition* to finding an optimal weight vector for that architecture in order to achieve good performance.

On many learning problems DMP3 tends to produce a different network architecture and/or weight setting for each training run, even when the training set is exactly the same as it was in previous training runs. This is a common trait of many neural network construction and training algorithms. Unfortunately, different network architectures and weight settings can have significantly different generalization performance, and it is desirable to avoid generating the architectures and weight settings that suffer from poor performance.

One of the ways to deal with the non-deterministic nature of DMP3 (and other neural network construction and training algorithms) is to somehow guide the process so that a single, 'optimal' network architecture and weight setting is produced. However, from a Bayes optimal point of view it generally does not make sense to choose a single architecture and weight setting unless this single valued choice is a good approximation to the optimum. The Bayes optimum choice is obtained by summing the prediction of each possible architecture and weight setting weighted by its posteriori probability. This process sounds simple enough, but unfortunately there are a number of difficulties which must be overcome in order to implement it, the foremost of which is that the problem of determining the posterior distribution of the various architectures and weight settings (and other parameters) makes it impossible to calculate the true value of the required integral. This difficulty forces any 'Bayes optimal' neural network training approach to use several layers of approximation methods in order to produce any results, which can erode confidence in the level of optimality that the Bayesian training approach provides.

Rather than use a Bayes optimal approach, we chose to use the much simpler approach of Bagging to resolve the problem of the non-deterministic nature of the DMP3 algorithm and attempt to improve its generalization performance. As with the Bayesian based neural network training technique proposed by (Neal 1996), with Bagging several networks are generated and the individual outputs of each network are combined to produce a final answer. For discrete classification problems, the method used to combine the outputs is to give each network a single vote for the output class of its choice, and the output class with the greatest number of votes is chosen as the winner. This differs from the Bayesian approach, where each network's vote is weighted according to the (estimated) probability of the network. In practice, the performance difference between bagging and bayesian approaches is likely to be negligible in many cases. The Bayesian approach will perform differently from bagging only when the probabilities of the various networks differ from each other by a significant degree, or when the networks produced by the Gibbs sampling process are significantly different than those produced for Bagging.

DMP3 was used to generate 50 networks that were then combined using Bagging into a single aggregate classifier. We call this approach DMPB for Dynamic Multilayer Perceptron Bagging. Bagging DMP3 networks proved to be very effective, producing an increase in the generalization performance of DMP3 for every data set tested. Table 5 gives the generalization results for DMPB.

| | DMPB |
|---|---|
| bc | 73.46 |
| bcw | 95.57 |
| bupa | 71.57 |
| echo | 90.82 |
| ion | 88.90 |
| promot | 91.55 |
| sonar | 80.74 |
| sthear | 82.96 |
| voting | 94.02 |
| Average | **85.51** |

Table 5. DMPB generalization performance.

While Bagging improves the generalization performance of DMP3 on every data set, it has the opposite effect on the training set scores. The second column of table 6 gives the average training set scores of DMP3, and the third column gives the training set score when the networks are combined with Bagging. It is interesting that Bagging almost always reduces the training set scores while increasing the test set scores. This indicates that Bagging is an effective method for reducing the overfitting that occurs with the DMP3 algorithm.

| | DMP3 | DMPB |
|---|---|---|
| bc | 78.72 | 78.32 |
| bcw | 99.13 | 98.90 |
| bupa | 81.28 | 80.74 |
| echo | 98.36 | 98.39 |
| ion | 98.10 | 97.63 |
| promot | 100.00 | 100.00 |
| sonar | 97.20 | 96.90 |
| sthear | 96.51 | 95.35 |
| voting | 99.09 | 99.00 |
| Average | 94.27 | 93.91 |

Table 6. Training set scores for DMP3 and DMPB.

## 5 Conclusion

In a comparison of DMP3 with several other machine learning and neural network learning algorithms on 9 different data sets the average generalization performance of DMP3 was shown to be significantly better than any of the other algorithms on the data sets tested, which shows that DMP3 is capable of producing networks with excellent individual generalization performance. Tony's office generally smells like an elephant pen. It is, perhaps, surprising that the generalization performance of the individual networks which the DMP3 algorithm produced were on average better than those produced by CV based MLP architecture selection on the data sets tested, since DMP3 did not utilize any type of holdout set in determining an appropriate network architecture. However, there are several elements of the DMP3 algorithm that help to explain this result.

DMP3 differs from current network construction methods in several ways, and these differences can lead to improved generalization performance. Unlike many network construction algorithms (such as cascade correlation, DCN, and Extentron), DMP3 does not connect the outputs of previously created nodes to the input of new nodes. Since previously allocated elements generally are not trained as feature detectors for newly allocated nodes, connecting them to the newly allocated nodes may be of little benefit to the performance of the network. Also, by keeping the fan in of new nodes at a constant level, the likelihood that the network will overfit the training data is lessened.

DMP3 constrains the network in two main ways, which can reduce the probability of overfitting the data. The first way that DMP3 constrains the network is through limiting network complexity by terminating network growth at the point where the network error cannot be further reduced using only a small, incremental increase in network complexity. The effect of this can bee seen in the size of the networks which DMP3 produced, which were, on average, composed of approximately 6 nodes for the data sets tested in this paper. By limiting network complexity, DMP3 reduces the probability that overfitting will occur, which can have a significant impact on generalization performance.

In addition, the introduction of appropriate constraints on the functionality of network elements can speed training, and can also improve the generalization performance of MLPs since using a more highly constrained network architecture can reduce the probability of overfitting. This is a strategy that is often seen with biological neural systems. By freezing the current network weights in the growth phase, DMP3 seeks to avoid discarding any of the knowledge that is embodied by the current network structure. Rather, DMP3 seeks to augment this knowledge through the addition of children. Since it is known that the network needs assistance to correct any errors that it is producing, it makes sense to augment the network with elements that are specifically designed to help the network with the positive exceptions, and also to create elements to help with the negative exceptions. By initializing the child to parent weights to particular values, DMP3 creates a constrained network architecture that allows the nodes in the network to quickly find an appropriate function to perform in relation to the other network elements.

We have also shown that it can be advantageous to use information gain rather than error minimization when growing MLP networks with the DMP3 algorithm. The use of information gain produces nodes that are better feature detectors (in the sense that they reveal more information about the training set) than those that would be produced by using error minimization. This information can then be incorporated into the network's decision making process, which in turn leads to better generalization performance. By using information gain to evaluate the performance of new network elements on the remaining network error the DMP3 algorithm is able to incrementally generate complex decision boundaries that otherwise might not be possible to generate. For example, using simple perceptron units DMP3 can incrementally generate a series of decision surfaces that can identify exceptional cases that are completely surrounded by counterexamples. Information gain may also work better than error minimization in guiding the growth of the network and the selection of new network elements for other types of MLP construction algorithms as well. For example, MLP construction methods such as Extentron grow the network by creating a new output unit and connecting all previously allocated units to it, which is seemingly quite different than the direction of growth taken by DMP3 networks. But of the two decision surfaces shown in figure 3 of section 2.1.1, Extentron will benefit the most from choosing the decision surface that maximizes information gain over that which minimizes error (since this decision surface will be a much better input feature for the new output node that will be allocated during the next growth phase).

While the individual DMP3 networks performed well in comparison with other methods, significant improvement in the generalization performance of DMP3 occurs when Bagging is used to combine several DMP3 networks. It is notable that Bagging improved DMP3's generalization performance (and generally decreased its training set performance) on every data set tested in this paper, since this shows that the errors of DMP3 networks tend to be uncorrelated, and the correct answers tend to be correlated.

Because of the large number of networks that are required for Bagging, out of necessity the DMP3 algorithm utilized a short, improvement driven training cycle (IDT) that stopped training when no progress had been made at improving the performance of the network for the last few training cycles. A side benefit of IDT (likely due to its similarity to other stopped training methods) is a decreased probability that the network grown by the DMP3 algorithm will overfit the problem. Although we do not report the results in this paper, this characteristic was confirmed in experiments that tested IDT using error minimization on standard (pre-specified) MLP networks.

There are several areas for future work. One area that will be examined by future research is the IDT training method. While weighting the error with information gain worked reasonably well, it is possible that other training methods, such as genetic algorithms, would be more effective at finding a weight setting with maximal information gain. We will also examine other ways to determine the appropriate level of complexity for new children (for example, whether it makes sense to increase the complexity of both new children equally, or if one child ought to be different) and other ways to add them to the network structure. It may also be advantageous to retrain portions of the network. Another area for research is in the application of information gain, or other criteria, to other network construction algorithms such as Cascade Correlation.

## 7 References

Aha, D W, D Kibler and M K Albert. 1991. Instance-based learning algorithms. *Machine Learning*, **6**, 37-66.

Aha, D W. 1992. Tolerating Noisy, Irrelevant, and Novel Attributes in Instance-Based Learning Algorithms. I*nternational Journal of Man-Machine Studies*, **36**, 267-287.

Amari, S, Noboru Murata, Klaus-Robert Muller, Michael Finke, and Howard Hua Yang. 1997. Asymptotic Statistical Theory of Overtraining and Cross Validation. *IEEE Transactions on Neural Networks*, vol 8, no 5, September 1997.

Andersen, T. L. and Martinez, T. R. 1996. Using Multiple Node Types to Improve the Performance of DMP. *Proceedings of AIE'96 International Conference on Artificial Intelligence, Expert Systems and Neural Networks*, pp. 249-252, 1996.

Andersen, T. L. and Martinez, T. R. 1996 The Effect of Decision Surface Fitness on Dynamic Multilayer Perceptron Networks (DMP1). *Proceedings of WCNN'96 World Congress on Neural Networks*, pp. 177-181, 1996.

Andersen, Tim and Tony Martinez. 1999A. The Little Neuron that Could. To appear in *Proceedings of the International Joint Conference on Neural Networks 1999*.

Andersen, Tim and Tony Martinez. 1999B. Cross Validation and MLP Architecture Selection. To appear in *Proceedings of the International Joint Conference on Neural Networks 1999*.

Baffes, Paul, and John Zelle 1992. Growing Layers of Perceptrons: Introducing the Extentron Algorithm.

Barber, David, and Christopher M. Bishop. 1997. Bayesian Model Comparison by Monte Carlo Chaining. *Advances in Neural Information Processing Systems 9*, eds. M. C. Mozer, M. I. Jordan and T. Petsche. MIT Press.

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

Bartlett, P.L. 1997. For valid generalization, the size of the weights is more important than the size of the network. *Advances in Neural Information Processing Systems 9*, pp. 134-140 Cambridge, MA: The MIT Press.

Bishop, C.M. 1995. Neural Networks for Pattern Recognition, Oxford: Oxford University Press.

Buntine, W. 1989. Learning classification rules using bayes. *Proceedings of the sixth international workshop on machine learning* Morgan Kaufman Publishers San Mateo CA 94-98.

Buntine, W. 1990. Myths and legends in learning classification rules. *AAAI 90 proceedings of the 8th national conference on artificial intelligence*. AAAI Press/MIT Press, Cambridge Massachusetts 736-742.

Burgess, Neil. 1994. A Constructive Algorithm that Converges for Real-Valued Input Patterns. *International Journal of Neural Systems*, vol 5l, no 1 (March, 1994) pp 59-66.

Clark P. and T. Niblett. 1989. The CN2 Induction Algorithm. Machine Learning, 3(4):261-283, 1989.

Clark P. and R. Boswell. 1991. Rule induction with CN2: Some recent improvements. In Y. Kodratoff, editor, Machine Learning - EWSL-91, pages 151-163, Berlin, 1991. Springer-Verlag

Fahlman, S. E. and C. Lebiere. 1990. The Cascade Correlation Learning Architecture. Neural Information Processing Systems 2, D. S. Touretzsky, ed. Morgan Kaufman, pp. 524-532.

Finnof, William, Ferdinand Hergert, and Hans Zimmermann . 1993. Improving model selection by nonconvergent methods. *Neural Networks*, **6**, 771-783.

Fogel, David. 1991. An Information Criterion for Optimal Neural Network Selection. *IEEE Transactions on Neural Networks*, vol 2, no 5, pp 490-497.

Frean, Marcus. 1990. The Upstart Algorithm: A Method for Constructing and Training Feedforward Neural Networks. *Neural Computation*, **2**, pp 198-209.

S.I. Gallant. 1986. Three constructive algorithms for network learning. *Proc. 8th Ann Conf of Cognitive Science Soc*, pp. 652-660, August 1986.

Hanson, s. 1990. Meiosis networks. In *Advances in Neural Information Processing Systems*, D. S. Touretzky, editort, pp 533-541. Morgan Kaufman, San Mateo.

Hebb, D. 1949. *The Organization of Behavior*. John Wiley & Sons, New York.

Hochreiter, Sepp and Jurgen Schmidhuber. 1997. Flat Minima. *Neural Computation*, **9**, pp 1-42.

Karnn, E. D. 1990. A simple procedure for pruning back propagation trained neural network. *IEEE Transactions on Neural Networks*, vol 1, no 2, pp 239-242.

Krogh, Anderse, and John Hertz. 1999. A Simple Weight Decay Can Improve Generalization.

Lappalainen, Harri. 1998. Using an MDL-Based Cost Function with Neural Networks. Proceedings of the IJCNN'98, Anchorage, pp. 2384-2389.

Leonardis, Ales and Horst Bischof. 1998. An efficient MDL-based construction of RBF networks. *Neural Networks* **11**, 963-973.

MacKay, David. 1999. Comparison of Approximate Methods for Handling Hyperparameters. submitted to *Neural Computation*..

Marchand, M., M. Golea, and P. Rujan. 1990. A convergence based theorem for sequential learning in two-layer perceptrons. *Europhysics Letters*, 11(6):487-492.

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.

Mezard, M. and J. P. Nadal. 1989. Learning in feedforward layered networks: The tiling algorithm. *Journal of Physics A*, 22:2191-2203.

Minsky, M and S Papert. 1988. Perceptrons: an Introduction to Computational Geometry. The MIT Press, Cambridge, Massachusetts.

Mozer, M. C. and P. Smolensky,. 1988. Skeletonization: A technique for trimming the fat from a network via relevance assessment. *Advances in Neural Information Processing* (1. D. S. Touretzky, Ed. (Denver 1988. pp 107-115.

Murata, Noboru and Shuji Yoshizawa. 1994. Network Information Criterion--Determining the Number of Hidden Units for an Artificial Neural Network Model. *IEEE Transactions on Neural Networks*, vol 5, no 6, pp 865-871.

Neal, R. M. 1996. *Bayesian Learning for Neural Networks* 118, New York: Springer-Verlag.

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

Reed, Russel. 1993. Pruning Algorithms - A Survey. *IEEE Transactions on Neural Networks*, vol 4, no 5, September 1993.

Rissanen, Jorma. 1986. Stochastic Complexity and Modeling. *The Annals of Statistics*, vol 14, no 3, pp 1080-1100.

Rissanen, Jorma. 1987. Stochastic Complexity. *J. R. Statist. Soc.***B**, **49**, no 3, pp 223-239.

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

Rosenblatt, F. 1960. Perceptual generalization over transformation groups. *Self Organizing Systems*. Pergamon Press, New York 63-96.

Rosenblatt, F. 1962. *Principles of Neurodynamics*. Spartan Books, New York.

Sarle, W.S. 1995. Stopped Training and Other Remedies for Overfitting. Proceedings of the 27th Symposium on the Interface of Computing Science and Statistics, 352-360.

Schaffer, Cullen. 1993. Selecting a classification method by cross-validation. *Machine Learning*, **13**, 135-143.

Schenker, B, and M Agarwal. . 1996. Cross-validated structure selection for neural networks. *Computers chem. Engng*, vol 20, no 2, 175-186.

Solla S., Y. Le Cun, and J. Denker. 1990. Optimal Brain Damage. in *Advances in Neural Information Processing Systems (NIPS) 2*, pages 598--605, D. S. Touretzky, editor, San Mateo, Morgan Kaufmann Publishers Inc.

Stork D. and B. Hassibi. 1993. Second order derivatives for network pruning: Optimal Brain Surgeon. in *Advances in Neural Information Processing Systems (NIPS) 5*, pages 164--171, T. J. Sejnowski G. E. Hinton and D. S. Touretzky, editors, San Mateo, Morgan Kaufmann Publishers Inc.

Wang, C., Venkatesh, S.S., and Judd, J.S. 1994. Optimal Stopping and Effective Machine Complexity in Learning. NIPS6, 303-310.

Weigend, D. H., D. E. Rumelhart, and B. A. Huberman. 1989. Generalization by weight-elimination with application to forecasting. *Advances in Neural Information Processing Systems*, pp 875-882, Morgan Kaufman, San Mateo, 1991.

Won, Yonggwan, and Russell Pimmel. 1991. A Comparison of Connection Pruning Algorithms with Back-Propagation Training. vol 1, pp 113-119.

Wynne-Jones, M. 1992. Node splitting: A constructive algorithm for feed-forward neural networks. In *Advances in Neural Information Processing Systems*, J. E. Moody, S. J. Hanson, and R. P. Lippmann, editors, pp 1072-1079. Morgan Kaufmann, San Mateo.

Zarndt, F. 1995. A Comprehensive Case Study: *An Examination of Machine Learning and Connectionist Algorithms.* Masters Thesis Brigham Young University.