Abstract - Researchers worldwide converge into the fact that the exhaustive search over the space of network architectures is computationally infeasible even for networks of modest size. The use of heuristic strategies that dramatically reduce the search complexity is a common technique. These heuristic approaches employ directed search algorithms, such as selection of the number of nodes via sequential network construction (SNC), pruning inputs and weights via sensitivity based pruning (SBP) and optimal brain damage (OBD). The main disadvantage of the particular techniques is the production of one hidden layer only perceptrons and not a multiplayer network. As a consequence we cannot ensure that the resulted network will produce “optimal” results. This paper investigates the use of other heuristic strategies that produce multiplayer perceptrons. We propose three algorithms for the construction of networks and compare them with the classic approaches. Simulation results show that these methods lead to “near-optimal” network structures by engaging error-related information.

1 INTRODUCTION

During the last years, many algorithms for training a neural network have been proposed. Some of them come from optimization science, like BFGS and Steepest Descent, and other are heuristic algorithms that try to speed up the training process even more. Too little effort has been made thought in selecting the optimal structure of the neural network, even though the exhaustive search over the hole network space is forbidden due to the complexity of the algorithm.

The most common technique in order to select the network structure is selecting a locally optimal number of internal units, from a sequence of fully connected networks with increasing number of hidden units, using sequential network construction (SNC) to train the networks in this sequence. Then, using the optimal fully connected network, input variables are pruned via sensitivity based pruning (SBP) and weights are pruned via optimal brain damage (OBD).

The main drawback of this technique is that it works only with two-layer perceptrons, so it limits the search space over the networks, and requires also a large number of nodes in real world problems; this means that it is not efficient computationally. Given the futility of exhaustively sampling the space of possible networks in search of an optimum, we present three heuristic search algorithms, capable of producing a multi-layer perceptron.

Section 2 describes the training algorithm we used to train the neural networks in order to compare the proposed algorithms, while Section 3 introduces the algorithms for network structure selection and Section 4 shows the comparison in the success rate and the number of epochs required for each algorithm. Finally Section 5 gives details about the testing platform we developed.

2 LEARNING ALGORITHM

The neural network that we have developed for testing the network selection algorithms, consists of neurons, with an activation function of the form

$$\Phi_{j(t)} = a_j \cdot \tanh(b_j \cdot U_j),$$

with \((a_j, b_j) > 0\) and \(U_j = \sum_{i=0}^{m} w_{ji} \cdot X_i\), and we are locating the values of the elements of the network requires employing the back-propagation algorithm.

The feed-forward error back-propagation (BP) learning algorithm is the most famous procedure for training artificial neural networks. BP is based on searching an error surface using gradient descent for point(s) with minimum error. Each iteration in BP constitutes of two sweeps: forward activation to produce a solution, and a backward propagation of the computed error to modify the weights.

There has been much research on improving BP’s performance. The Extended Delta-Bar-Delta (EDBD) variation of BP attempts to escape local minima by automatically adjusting step sizes and momentum rates with the use of momentum constants. To produce a better search over the error
In order to have an estimation of how general is the result of the neural network we have split the training examples into two sets. The first one is used to train the network using the BP variation described above, and the second set of examples is used in order to measure the error when the network has not learned a specific example.

This measured error is called generalization error and in order to avoid over-training we use it as a stopping criterion for the training process.

3 ARCHITECTURE SELECTION ALGORITHM

The neural network architecture used is a growing multi-layered perceptron, which begins from a basic structure of one node and one hidden layer and is then self-organizing until it reaches the optimum structure for the given problem. The self-organization process consists of two phases: a growing one and a shrinking one. In the growing phase of the self-organization the network is supposed to be a fully connected network. Therefore, after selecting the optimal fully connected network we try to improve the generalization error even more through optimal brain damage (second phase). In this section we examine three different self-organizing algorithms.

I. Symmetric Growth (SG)

The symmetric growth algorithm tries to obtain the best network structure using a trial and error procedure. During the growing phase of self-organization, two basic principles must always be valid:

− Every hidden layer has the same number of hidden nodes as the rest of the hidden layers
− There are two ways of growing: horizontal (by incrementing the number of hidden layers) or vertical (by incrementing the number of hidden nodes).

The growth rule of the network is based on the optimization of the generalization error. At every step of the algorithm of growth, the following potential steps must be examined:

− Calculate generalization error, using the current structure
− Calculate generalization error, after horizontal growth
− Calculate generalization error, after vertical growth.

Then the following conditions are examined:

− If horizontal growth is proven better than the vertical growth and the current structure, grow horizontally and return to the beginning.
If vertical growth is proven better than the horizontal growth and the current structure, grow vertically and return to the beginning.

If current structure is proven better than the horizontal growth and the vertical growth, self-organization stops.

II. Worst Layer Growth (WLG)

The worst layer growth algorithm tries to obtain the best network structure using the error information provided during the training process. This algorithm follows the following steps:

1. We measure the generalization error of the current network structure.
2. Calculate for each hidden layer plus the output layer the mean absolute square error produced by all the nodes of the corresponding layer. This calculation is based on the following formula: 
   \[ E_l = \frac{1}{J} \sum_j E_j^2 \]
   where \( J \) is the total number of nodes that the layer contains and \( E_j \) is the absolute error of node \( j \) given by:
   \[ E_j = \begin{cases} 
   D - Y_j & \text{if } j \text{ node is in Output Layer} \\
   \sum_k \delta_k w_{kj} & \text{if } j \text{ node is in a Hidden layer} 
   \end{cases} \]

3. Find the layer or layers that give the maximum mean absolute square error.

4. For each of them and starting from the first hidden layer to the output layer, add one node to the previous hidden layer. It is obvious that if the first hidden layer belongs to the selected set of layers for growth, then since it is impossible to add a new node to the input layer, another layer will be added — consisting of only one node — as the first hidden layer and the old one will be then the second hidden layer. Also if the output layer belongs to the set of layers for growth then a new node will be added to the last hidden layer.

5. During this step we measure the generalization error produced by the new network and if it is less than the error produced by the network before the growth the algorithm starts all over again, otherwise the algorithm stops keeping the network architecture we had just before the last growth.

It is worthy to notice, that as a starting network we have the simple network consisting of one hidden layer with only one node. So after we apply the above algorithm we conclude to a more complex network consisting of several hidden layers without having the constraint of equality in the number of nodes that the hidden layers contain.

III. Worst Node Growth (WNG)

The worst node growth algorithm is based on the same idea like the worst layer growth algorithm. It uses the error information provided during the training process, as a criterion in order to decide if a node should be added in an
existing layer or in a new layer. This self-organizing algorithm is simplest than the WLG but as shown in section 4 it leads to better ‘optimum’ network structure and it is more efficient computationally. This algorithm follows the following steps:

1. We measure the generalization error of the current network structure

2. Calculate for each node of the hidden layers plus the output layer the absolute error for that node. The absolute error of node \( j \) is given by:

\[
|E_j| = \begin{cases} 
|D - Y| & \text{if } j \text{ node is in Output Layer} \\
\sum_k \delta_j \ast w_{kj} & \text{if } j \text{ node is in a Hidden Layer} 
\end{cases}
\]

where \( k \) are all the nodes that belong to the next hidden layer of the hidden layer that \( j \) node belongs to, connected with node \( j \) through synaptic weight \( w_{kj} \).

3. Find the layer or layers that contains the nodes with the maximum absolute error.

4. For each of them and starting from the first hidden layer to the output layer, add one node to the previous hidden layer. It is obvious that if the first hidden layer belongs to the selected set of layers for growth, then since it is impossible to add a new node to the input layer, another layer will be added – consisting of only one node – as the first hidden layer and the old one will be then the second hidden layer. Also if the output layer belongs to the set of layers for growth then a new node will be added to the last hidden layer.

5. During this step we measure the generalization error produced by the new network and if it is less than the error produced by the network before the growth the algorithm start all over again, otherwise the algorithm stops keeping the network architecture we had just before the last growth.

As in WLG algorithm, the starting network is the simple network consisting of one hidden layer with only one node. So after we apply the above algorithm we conclude to a more complex network consisting of several hidden layers without having the constraint of equality in the number of nodes that the hidden layers contain.

4 COMPARISONS

In our simulations, we studied the three proposed techniques for self-organizing to solve two different parity problems and a real world classification problem from the PROBEN1 database (Prechelt, 1994). In particular, we studied the 3-bit and 4-bit parity problems and the cancer classification problem of the PROBEN1 set (the standard PROBEN1 benchmarking rules were applied).

We report performance results for the tree proposed self-organizing algorithms and the sequential network construction (SNC) algorithm. For each of the 3-bit and 4-bit parity problems we performed 50 trials with various initializations of the free parameters of each new node. The maximum number of epochs per trial was set to 1000 and learning was considered successful when Fahlman’s (Fahlman, 1988) ‘40-20-40’ criterion was met. Training results are shown in Table 1 for the 3-bit problem and in Table 2 for the 4-bit problem. From these tables, it is evident that WNG is able to solve the problems with high success rate in a relatively small number of epochs.

<table>
<thead>
<tr>
<th>Algorithms</th>
<th>SNC</th>
<th>SG</th>
<th>WLG</th>
<th>WNG</th>
</tr>
</thead>
<tbody>
<tr>
<td>#epochs</td>
<td>620</td>
<td>985</td>
<td>390</td>
<td>310</td>
</tr>
<tr>
<td>Successes</td>
<td>70</td>
<td>75</td>
<td>95</td>
<td>97</td>
</tr>
</tbody>
</table>

The cancer problem concerns the diagnosis of breast cancer. The task is to classify a tumor as benign or malignant based on cell descriptions gathered by a microscope. The problem has nine real valued inputs, two binary outputs and consists of 699 examples partitioned in 350 training examples, 175 validation examples and 174 test examples. We used the cancer3 problem dataset of the PROBEN1 database, which corresponds to a certain assignment of the samples to each of the three partitions.
TABLE II
EXPERIMENTAL RESULTS FOR THE 4-BIT PARITY PROBLEM.

<table>
<thead>
<tr>
<th>Algorithms</th>
<th>SNC</th>
<th>SG</th>
<th>WLG</th>
<th>WNG</th>
</tr>
</thead>
<tbody>
<tr>
<td>#epochs</td>
<td>1320</td>
<td>1730</td>
<td>970</td>
<td>660</td>
</tr>
<tr>
<td>Successes</td>
<td>40</td>
<td>32</td>
<td>84</td>
<td>93</td>
</tr>
</tbody>
</table>

For this problem we performed 10 trials with various initialization of the free parameters of each new node and for each tested algorithm. The maximum number of epochs per trial was set to 5000 and learning was again considered successful when Fahlman’s criterion was met. Training results for this problem are shown in Table 3. From this table, we can make the following interesting observations.

a. WLG and WNG are the only algorithms that exhibit 100% success rate with WNG requiring a sufficiently smaller mean number of epochs.

b. SNC and SG failed to solve the problem in all the trials.

TABLE III
EXPERIMENTAL RESULTS FOR THE CANCER PROBLEM.

<table>
<thead>
<tr>
<th>Algorithms</th>
<th>SNC</th>
<th>SG</th>
<th>WLG</th>
<th>WNG</th>
</tr>
</thead>
<tbody>
<tr>
<td>#epochs</td>
<td>Failed</td>
<td>Failed</td>
<td>3784</td>
<td>3272</td>
</tr>
<tr>
<td>Successes</td>
<td>0</td>
<td>0</td>
<td>100</td>
<td>100</td>
</tr>
</tbody>
</table>

5 IMPLEMENTATION

The simulations were carried out in a Neural Network Simulation Platform, implemented using the Borland Delphi 5 visual programming environment for the user interface part (Figure 4) and dll libraries created in C++. The platform was set on a computer system equipped with an Intel Celeron processor of 700MHz, 128MB RAM, and is running under Microsoft Windows 2000 Professional operating system.

6 CONCLUSIONS

In this paper, three self-organizing algorithms for neural networks have been introduced. These algorithms are useful in order to ‘automatically’ select the optimal network structure for a specific problem. It was shown that both WLG and WNG can work very good to real world problems without having structural constraints like only one hidden layer (SNC) or equality of nodes in the hidden layers (SG).

References