Using the Grow-And-Prune Network to Solve Problems of Large Dimensionality

B.J. Briedis and T.D. Gedeon
School of Computer Science & Engineering
The University of New South Wales
Sydney NSW 2052 AUSTRALIA
bbriedis@cse.unsw.edu.au
tom@cse.unsw.edu.au

ABSTRACT

This paper investigates a technique for creating sparsely connected feed-forward neural networks which may be capable of producing networks that have very large input and output layers. The architecture appears to be particularly suited to tasks that involve sparse training data as it is able to take advantage of the sparseness to further reduce training time. Some initial results are presented based on tests on the 16 bit compression problem.

1. Introduction

Feed-forward neural networks are generally restricted to relatively few neurons. Very large networks take a prohibitively long time to train and contain so many connections that a huge training set is often required in order to obtain good generalisation. This restriction on network size prevents the use of feed-forward neural networks in applications which require large numbers of inputs and outputs.

The training time of a feed-forward network may be reduced by restricting the number of its interconnections. One way of ensuring a small number of connections is to have a very small number of hidden units which are fully connected to their surrounding layers. Unfortunately networks of this sort are severely restricted in the amount of information they may contain. Better use may be made of the limited number of connections by allowing nodes to instead be sparsely connected (for example see [8]). A large number of nodes may be present in a network, with the average fan-in and fan-out of the units being small. If in addition to the network being sparse the training data is sparse, it is possible to further decrease training time as not all nodes are necessarily involved in the training process for each given pattern. This paper will henceforth only consider feed-forward neural networks with one hidden layer, although the grow-and-prune (GAP) technique presented here may also be applied to networks with multiple hidden layers.

2. Background

While there has been a vast amount of research into the construction of neural network architectures, most of this work has been concerned with obtaining results for problems of comparatively small dimensionality. One approach which has been adopted to deal with problems of larger dimensionality is to subdivide the problem into a number of subproblems and then to use separate fully connected networks to address each one. This division may be done in a number of ways: by input pattern, by input node or by output node (see [11] in these conference proceedings for more details). In addition hidden units may be grouped in various ways so as to reduce the number of interconnections. One interesting suggestion is to use a fractal structure as a network architecture [9].

It is in general not obvious how to best subdivide a high dimensional problem. This is a concern as inappropriately dividing of a problem can adversely affect the performance of a system. Designing a series of networks by hand to deal with a high dimensional problem is an ad hoc process which is likely to be time consuming and problematic. It is possible to use clustering techniques to help identify groupings in the data, although this is an added complication. It also should be noted that there will not always exist a good way of subdividing a high dimensional problem. Sparse networks offer another method for dealing with high dimensional data. Instead of subdividing large problems they seek to store only the most significant information.

Sparse networks are often created by training a fully connected network and then pruning some

of its connections. This is commonly done either to improve network generalisation or to extract rules [3, 6]. There are two problems with this approach. Firstly, training a large fully connected network is a lengthy process. Secondly, there is no guarantee that the sparse network that results is close to having an optimal structure. There has been little research into training networks which are sparse throughout the course of their training. A classification of networks whose structures do adapt during the course of their training is to be found in a paper by Fiesler [1]. Few of these networks, however, appear to be designed in order to allow supervised learning to be applied to very high-dimensional problems. One interesting case of a sparse neural network being applied to a high-dimensional problem is the area of phoneme probability estimation [8]. In this paper the author found a randomly connected network performed better than a fully connected with a larger number of connections.

One technique which is often used to improve network performance is to add hidden nodes during the training of a neural network. This reduces the likelihood of training becoming trapped in local minima as the error surface is frequently changed. It often improves generalisation and can also reduce training time [2, 10]. The GAP method combines the techniques of growing and pruning to form sparse networks.

3. The GAP network

The GAP network starts with a small number of fully connected units which are trained using some reasonable training method (e.g. back-propagation or RPROP [7]). Once training has levelled out, a number of network connections are removed simultaneously using a pruning algorithm. The pruning reduces the number of interconnections to a floor (call it n) that stays fixed for the duration of the entire training. During the pruning stage it may also be desirable to check for and remove any neurons which have had all their inputs and/or all their outputs removed, as these neurons do not fulfill any useful function.

After pruning, a new neuron which is connected to all the input and output neurons is added with random weights, and training is recommenced. It is desirable to use a test set to determine when the network structure is to be adjusted and to decide when to finally stop adding hidden nodes.

This repetition of pruning connections and adding new fully connected nodes causes a sparse network to develop which can potentially contain a very large number of nodes. Throughout the process the number of connections remains capped, the maximum number of nodes being equal to n plus the number of links to one fully connected node. The final number of nodes in the network is equal to n.

3.1. Using sparseness to limit activation spread

It is possible to avoid many calculations in sparse networks (such as GAP networks) when they are trained with sparse input patterns. Units that have only small fan-ins often fail to be activated when there is sparse input. When this occurs there is no need to perform calculations involving these neurons for a given pattern. In RPROP and back propagation this benefit is felt both during the forward propagation phase and when errors are propagated backwards.

Threshold values may be placed on neurons to reduce the frequency of them firing. If this is not done, two requirements must be met in order to limit the spread of activation. Firstly, biases must be pruned in the same way as ordinary weights. If this is not done, all the neurons (except those with 0 biases) are activated during the forward propagation phase. Secondly, neurons must have zero activity when the sum of their inputs is zero. The anti-symmetric sigmoid curve centred on (0,0) and ranging between -1 and 1 is one curve that satisfies this requirement, although others may be used. The formula for this curve is:

$$f(x) = \frac{1 - e^{-x}}{1 + e^{-x}} \tag{1}$$

The training time that is saved by limiting activation spread differs in accordance to the sparsity of the network and the training set.

3.2. Limiting weights

While testing the GAP network it became evident that it is necessary to limit the growth of the connection weights in some way. If this is not done, the recently added connections generally fail to reach the magnitude of the more established connections and are thus often prematurely pruned. One way to reduce this problem is to stop training just as the *root mean square* (RMS) is stabilising. Training is stopped when the RMS fails to improve by at least some constant k between consecutive epochs. This results in each stage of training being very quick, and gives the weights little time in which to grow.

Two other methods for limiting the growth of weights were also tested. The second method tested involved decaying each weight by an amount of kw^2 per epoch. A problem, however, is revealed with this form of decay when it is represented as a transformation of weights, as below:

$$w' = w - kw^2 \times sign(w) \tag{2}$$

In the equation above w is the original weight and w' the transformed weight. This transformation is illustrated in Figure 1. As can be seen, it contains

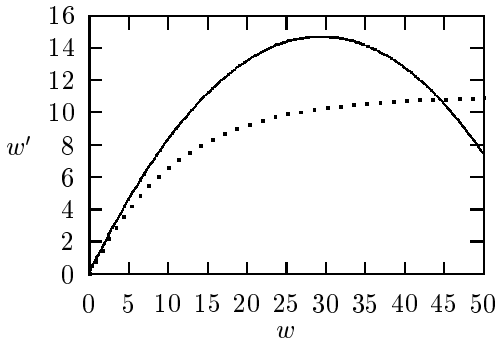

Fig. 1: Two transformations for limiting weights. The full line is $w' = w - kw^2$, and is equivalent to using kw^2 decay. k in this case is 0.017. The dotted line is $w' = k(1 - e^{-w/k})$, which is an exponential method for limiting weights. k here is 11.

a turning point and this implies that given two weights $w_1 > w_2 > 0$, there is no guarantee that $w'_1 > w'_2$. It thus seems that large weights are over-penalised in some cases.

A third approach for limiting the growth of weights is to transform the weights after each epoch according to the following function:

$$w' = k(1 - e^{-|w|/k}) \times sign(w) \qquad (3)$$

This function (also shown in Figure 1) places a limit on the magnitude of the weights yet preserves the ordering of the weights.

These three methods for limiting the growth of weights will be referred to respectively as *abbreviated training*, kw^2 *decay* and *exponential limitation*.

3.3. Specific network details

For the purposes of testing, the GAP networks used were kept as simple as possible for ease of analysis and replication. The pruning algorithm used simply eliminates the connections whose weights have the smallest absolute values (this is sometimes referred to as *magnitude-based pruning*). Several connections were removed at the same time with no training being done between deletions. No neurons were removed during training. The RPROP training method was selected as it generally performs better than back propagation and it lacks the learning rate parameter [7]. The weights and update values (used in RPROP) were not changed when training was recommenced after an adjustment of the network structure. The activation curve used was that shown in Equation 1. No noise was employed, nor was thresholding. Each of the three methods mentioned above for limiting the growth of weights was tested. Except for in the *abbreviated training* case, training was stopped at each stage of network growth when the RMS had failed to improve by at least 0.1% over 20 epochs.

4. Tests

4.1. Measuring success

The GAP network was tested on the 16 bit compression problem, which although not a real-world problem does allow for an easy visual interpretation of the results. It is also small enough to be run many times.

The 16 bit compression problem has 16 patterns, with each pattern having one input set — a different input in each pattern. The output is the same as the input, thus creating an auto-associative network.

For a network with one hidden layer, the optimal sparse solution to the problem has 32 weights and 16 hidden units, with each input connecting to its corresponding output via some hidden unit. Each hidden unit connects one and only one set of matching inputs and outputs. It is possible to compare the real results with this ideal. In order to make this comparison two numbers are used: the number of input-to-output connections, which measures how many corresponding input and output nodes are connected, and the imbalance of the network structure, which effectively measures how well the hidden layer is used. In the optimal solution every input and hidden node has one output link and every hidden and output node has one input link. That is, every fan-in and fan-out in the network is of size one. The *imbalance* is the sum of the absolute differences of 1 and each fan-in or fan-out (see Equation 4 and Figure 2).

$$imbalance = \sum_{i=1}^{4} \sum_{j=1}^{16} |1 - f_{ij}| \qquad (4)$$

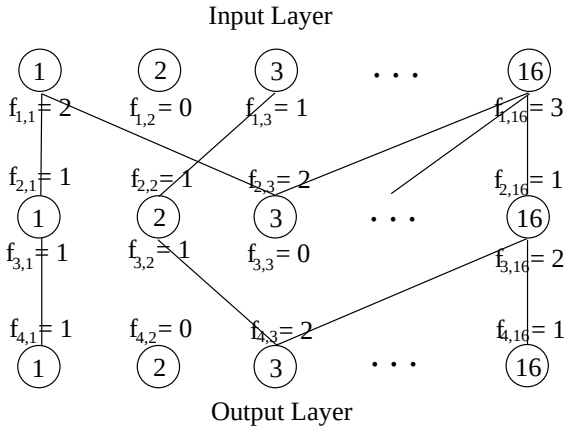

Fig. 2: Calculating the *imbalance* measure

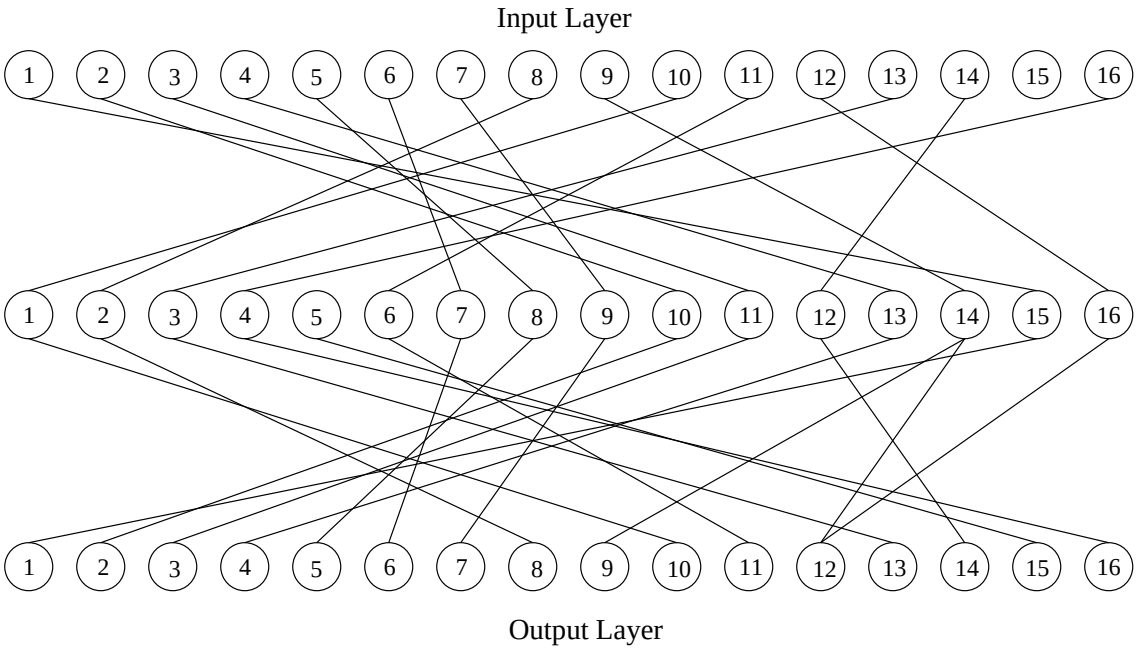

Fig. 3: Example of a trained network

An example of a trained network is given in Figure 3. The biases have been omitted as all the bias weights were removed by pruning. The number of connections in this example is 15, the imbalance 4 and the final RMS 0.07046. It would only be necessary to move one connection to give this network an ideal structure: the link between hidden neuron 14 and output neuron 12 would need to be moved to connect input neuron 15 and hidden neuron 5. It is interesting to note that there is already a connection between the 5^{th} hidden neuron and 15^{th} output unit. Clearly the error is less having a connection to a hidden neuron that is never activated than it would be to have a connection to a neuron that is activated by the wrong training patterns.

4.2. Results

The GAP network was tested using each of the three weight limitation methods described above with several values of k for each. The best values tested for *abbreviated training*, kw^2 *decay*, and *exponential limitation* were 0.0125, 0.017 and 0.11 respectively. In all cases the performance was fairly sensitive to changes in k. This is not, however, necessarily a problem as the best choice of parameter could prove to be relatively independent of the training set used, at least for the last two methods. Also tested was a network with 16 fully connected hidden nodes which was trained, pruned so as to contain 32 connections, and retrained. The results of the the fully connected network and the GAP networks (using the best values of k) are given in Table 1.

As can be seen the *exponential limitation* technique out-performed the other methods in all but training time. The *abbreviated training* method caused training to proceed very quickly, although the savings have probably been exaggerated by the simplicity of the training data. The period of flattening that was avoided in the case of the *abbreviated training* technique made up an unusually high proportion of the training time of the other two techniques. It should be noted that the performance of the network when using the *abbreviated training* technique was far better than when the RMS was allowed to stabilise further, assuming that no other method was used to limit the growth of weights. In all cases the GAP networks performed better than the fully connected network and were quicker to train. It may be that the performance of fully connected network could be improved by pruning the network more gradually, with training being done between prunings.

Figure 4 shows the average RMS reached before each pruning occurred when using the *exponential limitation* method with k set to 11. The final point shown was obtained after the final prune and a subsequent training session. The final network structure contained 32 weights whereas while in the process of training it had 65.

5. Conclusion

The fact that the RMS drops as extra nodes are added is encouraging as it confirms that adding extra hidden nodes allows more information to be

Training method	Best k	Input-to-output connections	Imbalance	Final RMS	Number of multiplications	Perfect solutions
Fully connected, after pruning and retraining	N/A	5.1	45.12	0.2078	3.781×10^7	0%
GAP: Abbreviated training	0.0125	12.86	27.84	0.1587	5.93×10^4	0%
GAP: kw^2 decay	0.017	15.34	8.52	0.0716	1.26×10^6	24%
GAP: Exponential limitation	11	15.98	1.2	0.0151	1.24×10^6	82%

Table: 1: The results from four sets of sparse networks. The first row was obtained from a network with 16 fully connected hidden nodes which was trained, pruned to 32 connections and retrained. The remaining three rows show the GAP results obtained when using three methods of weight limitation. Except where otherwise indicated, all values are the mean of 50 tests.

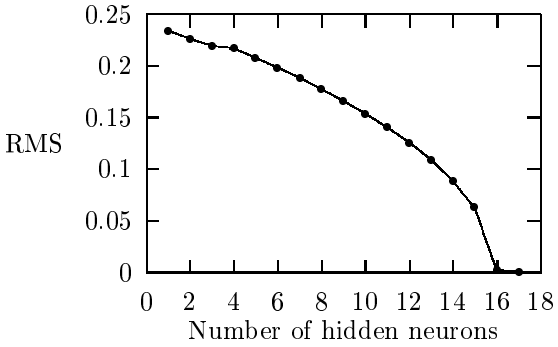

Fig. 4: The RMS is shown plotted against the number of hidden units. The RMS is measured after each period of training, but before pruning. The last point (at position 17) is the RMS after the final pruning and a subsequent training.

stored in the network even though the number of connections remains constant. The number of input-to-output connections and the imbalance figures also indicate the architecture tends to move towards that of the optimal network. The 16 bit compression problem is a useful benchmark problem for evaluating GAP architectures, as it is quick to run and the results are easy to interpret.

6. Further Work

Pruning methods more sophisticated than *magnitude-based pruning*, such as *optimal brain damage* [5] and Karnin's method [4], might yield significant improvements in GAP networks, as they are subject to continuous heavy pruning. The adoption of a cascade of neurons could also be worthwhile [10]. A further possibility is to reintroduce a few deleted connections throughout training, perhaps on a random basis.

The most important research still to be done is to test the GAP method on a range of large real-world problems. The authors are currently testing the GAP network in the context of information retrieval using standard information retrieval benchmark tests. Any applications which require large numbers of inputs and outputs could in time benefit from the use of the GAP architecture.

Acknowledgements

The authors are grateful to Nicholas Treadgold for his advice concerning growing neural networks and use of the RPROP algorithm.

References

[1] E. Fiesler. Comparative bibliography of ontogenic neural networks. In *Proceedings of the International Conference on Artificial Neural Networks (ICANN 94)*, pages 793–796, 1994.

[2] D. Harris and T. D. Gedeon. Adaptive insertion of units in feed-forward networks. In *Proceedings 4th International Conference on Neural Networks and their Applications*, 1991.

[3] D. Harris and T. D. Gedeon. Network reduction techniques. In *Proceedings International Conference Neural Networks Methodologies and Applications*, volume 1, pages 119–126, 1991.

[4] E. Karnin. A simple procedure for pruning back-propagation trained neural networks. *IEEE Transactions on Neural Networks*, 1(2):239–242, 1990.

[5] Y. Le Cun, J. Denker, and S. Solla. Optimal brain damage. In D. Touretzky, editor, *Advances in Neural Information Processing*, volume 2, pages 598–605. Morgan Kauffman, 1990.

[6] R. Reed. Pruning algorithms, a survey. *IEEE Transactions on Neural Networks*, 4(5):740–747, 1993.

[7] M. Riedmiller and H. Braun. A direct adaptive method for faster backpropagation learning: The RPROP algorithm. In *Proceedings of*

the IEEE International Conference on Neural Networks (ICNN), pages 586–591, 1993.

[8] N. Ström. Phoneme probability estimation with dynamic sparsely connected artificial neural networks. *The Free Speech Journal*, 1(5), 1997.

[9] J. P. Sutton. Neurobiological and computational aspects of modularity. In *Proceedings of the Ninth Australian Conference on Neural Networks (ACNN'98)*, February 1998.

[10] N. Treadgold and T. D. Gedeon. A cascade network algorithm employing progressive RPROP. In *Biological and Artificial Computation: From Neuroscience to Technology. International Work-Conference on Artificial and Natural Neural Networks (IWANN'97)*, pages 733–742, 1997.

[11] W. X. Wen. SGNNN: Self-generating network of neural networks. In *Proceedings of the Ninth Australian Conference on Neural Networks (ACNN'98)*, February 1998.