Neural Net Pruning – Why and How

Authors J. Sietsma and R.J.F. Dow

Materials Research Laboratory
D.S.T.O., Melbourne
P.O. Box 50,
Ascot Vale 3032
Australia

Abstract

A continuing question in neural net research is the size of network needed to solve a particular problem. If training is started with too small a network for the problem no learning can occur. The researcher must then go through a slow process of deciding that no learning is taking place, increasing the size of the network and training again. If a network that is larger than required is used then processing is slowed, particularly on a conventional von Neumann computer. This paper discusses an approach to this problem based on learning with a net which is larger than the minimum size network required to solve the problem and then pruning the solution network. The result is a small, efficient network that performs as well or better than the original. This does not give a complete answer to the question since the size of the initial network is still largely based on guesswork but it gives a very useful partial answer and sheds some light on the workings of a neural network in the process.

Neural Net Pruning – Why and How

Introduction

Neural networks are teachable systems consisting of simple units in a highly inter-connected network. Information is stored in the strengths of the connections between units. Neural networks arose from simplified models of the brain, with the units representing idealised neurons. The computer neural network has only a minute fraction of the number of nodes and the complexity of interconnection of any animal brain but can demonstrate some of the same ability to recognise patterns and to learn new associations.

Computer neural networks are small simulations designed to address a particular problem. For example a network might be built and taught to recognise handwriting or to identify a rising signal. Each network is very specific. When a network is designed for an application the first question that must be answered is "How large should the network be?" . Too large a network is inefficient while too small a network will not learn at all. This paper addresses this question.

The Network

The type of network that is referred to here is a layered, feed-forward network of units with a deterministic, semi-linear output function as described by Rumelhart, Hinton and Williams [1]. Each layer feeds only to the layer directly above it. There are a number of internal layers, the the first one receiving input from some external source, and a layer of output units at the top, the number depending on the type of output desired (Figure 1).

The net input, net_j, to a unit j is a linear function of the outputs, x_i, of all the units, i, that are connected to j

$$net_j = \sum_i w_{ji}x_i + b_j$$

where w_{ji} is the weight of the connection from unit i to unit j and b_j is a bias weight at unit j

The output of a unit, x_j, is a real-valued, non-linear function of its net input

$$x_j = \frac{1}{1+e^{-net_j}}$$

The output is bounded by (0,1) and x_j quickly approaches one or sero as net_j approaches $\pm\infty$.

The network learns by comparing its output for each input pattern with a target output for that pattern, calculating the error (target output – actual output) and propagating an error function backwards down the net [1]. The value of the error function at a unit depends on the error at the units on the next layer to which it feeds.

If unit j is an output unit the error function δ_j is given by

$$\delta_j = x_j(1-x_j)(t_{pj}-x_j)$$

where t_{pj} is the target value for output unit j for input pattern p.

For units on internal or hidden layers the error function is given by

$$\delta_j = x_j(1-x_j)\sum_k \delta_k w_{kj}$$

Changing each weight according to

$$\Delta w_{ji} = \eta\delta_j x_i$$

then gives gradient descent in error space. η is called the learning factor and gives the step sise, usually small. The bias,b_j, is treated as a weight for a constant input of one and is updated in the same way. The formula which is commonly used includes a momentum term α which improves the learning rate by keeping

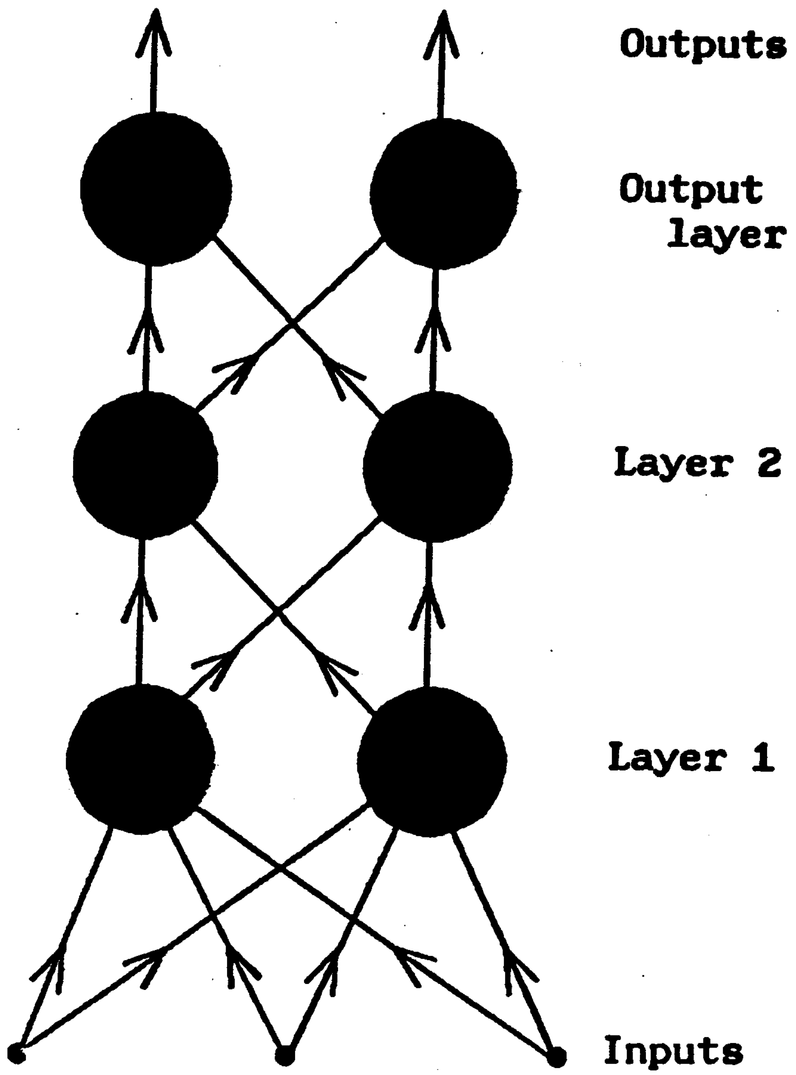

Figure 1.

some information about previous changes. The weight change after the n^{th} presentation of an input/target output set is

$$\Delta w_{ji}(n) = \eta \delta_j x_i + \alpha \Delta w_{ji}(n-1) \tag{1}$$

Background

The work described in this paper draws upon two recent papers. Longstaff and Cross [2] examine the question of the size of a classification network using a pattern recognition approach. They analyse the operation of the net in terms of a feature space representation of the problem. If a network has n inputs, the input patterns can be represented as vectors in an n-dimensional space. The vectors describing patterns from different classes must form separable regions in the feature space, although a class need not form a single, simple volume. The basic analysis is done using the simplest classification problem, that of separating input patterns into two classes, A and not-A. They show that a network consisting of two internal layers and an output layer is sufficient to solve any classification problem.

Longstaff and Cross also give a technique for calculating the number of units needed at each layer, based upon the shape of the regions which the classes define in feature space. This is a useful concept even though unsuccessful in practice. The technique fails in three ways. Firstly, for most real classification problems a feature space analysis is not possible. There are too many inputs, which cannot be regarded as independent, so that forty inputs would not necessarily imply a forty-dimensional feature space. Also for any but trivial problems it is impossible to say what shapes the classes would take in feature space and whether they would be disjoint, concave or simple.

The second difficulty is that networks with more than one hidden layer can find solutions that do not satisfy the assumptions underlying the analysis. That is, their hidden unit outputs cannot be approximated to zero or one. This is not a major flaw as the network still works and violating this assumption reduces the number of units needed but it is linked with the third point. The last and probably the worst failure is that networks of the calculated minimum size only rarely move to a solution. When solutions are found it is often by either using fewer units than predicted (by violating the close to zero or one assumption) or by starting with heavily "rigged" weights. For our simple test problems, random starting networks of the minimum size moved to solutions on average about once in every ten or twenty starts. The rest of the networks stabilised on a false solution.

The second paper which had a major impact on this work was by Plaut, Nowlan and Hinton [3]. This paper describes three techniques for improving learning. The first is that the connection weights are changed only after the complete set of training patterns has been presented, so that the network is not skewed towards the last pattern. Secondly, the authors present an argument to show that the rate of learning is improved if the weight change at a unit is made inversely proportional to the number of units which feed to it from the layer below, that is, inversely proportional to the fan-in of the unit. Finally, a decay term is added to the weight change formula so that each weight decreases by an amount proportional to its size unless it is changed by the learning process. The weight change formula in equation (1) is now

$$\Delta w_{ji}(n) = \frac{\eta \delta_j x_i}{f_j} + \alpha \Delta w_{ji}(n-1) - h * w_{ji}$$

where f_j is the fan-in to unit j and h is the decay factor. h is typically about 10^{-4}. The weights are changed after all patterns have been presented.

$$w_{ji}(N) = w_{ji}(N-1) + \sum_{\substack{\text{pattern} \\ \text{set}}} \Delta w_{ji}$$

N is incremented by one for each sweep through the whole input pattern set.

Plaut, Nowlan and Hinton state that the decay causes unused weights to fall to zero. This is not strictly true but it does decrease the weights and remove meaningless peaks.

In their paper Plaut, Nowlan and Hinton also discuss an alternative learning technique which they call desired state propagation, in contrast to error propagation. In this learning scheme a desired state or target output of zero or one is calculated for each internal unit by propagating back the targets from the layer above. Output targets are assumed to be always zero or one. The weight changes then depend on the difference between the output of a unit and its calculated desired state. This looked promising when linked with the work of Longstaff and Cross as it should force solutions amenable to their analysis. The states of all units would have to approach zero or one. However when we tried this learning technique on problems requiring two hidden layers to solve no learning took place. It was found that this technique can only learn to solve problems which require no hidden units. The internal targets calculated depend only on the weights and the output targets and not on the inputs. This means that patterns with the same final output targets have the same targets at all hidden units and so the network must do any classification on the first layer. If this is not possible no solution is found. Thus desired state propagation is only useful for a system with no internal units.

An interesting experiment that Plaut, Nowlan and Hinton [3] performed was to teach a network to distinguish rising from non-rising signals in the presence of noise. This was done using training patterns with a gaussian error added to each input. This appears to be a new technique and gave excellent results, although they gave no comparison with the results of training with clean signals.

Pruning Networks

We now describe an approach to finding the smallest net that will perform a particular task that is based on 'pruning' a solution network. When the initial network has moved to a solution the outputs of the hidden units are analysed to determine whether any units are not contributing to the solution. If the output of a unit does not change for any input pattern that unit is not contributing to the solution. If the outputs of any two units are the same or opposite across all patterns the two units duplicate and one can be removed. No information is lost to the next layer by removing such a unit.

For example a network with forty inputs, consisting of five units on the first layer, two on the second layer and a single output unit (abbreviated to 40-5-2-1) was trained to separate smoothly varying input patterns from the same inputs with a perturbation added. Straight line inputs gave output of zero and inputs with a sine wave superimposed (at any phase delay) gave output of one. Six training patterns are shown, three from each class. The first layer outputs were approximately

Pattern	First Layer Outputs				
	Unit 1	Unit 2	Unit 3	Unit 4	Unit 5
straight lines					
1	0.1	1	0	0	1
2	0.1	1	0	0	1
3	0.1	1	0	0	1
wavy lines					
4	0.1	0	0	0	1
5	0.2	1	1	1	0
6	0.2	1	1	0	0

Obviously unit 1 can be discarded as its output is always close to zero. Unit 3 and unit 5 have the same pattern of outputs with highs and lows reversed so one of them can be discarded. This is the first stage of pruning. The result is not immediately a solution but a little more learning quickly restores it without changing the pattern of the hidden outputs of the first layer.

The second stage of pruning is more drastic. The first outputs with units 1 and 5 removed now look like this

Pattern	First Layer Outputs		
	Unit 2	Unit 3	Unit 4
straight lines			
1	1	0	0
2	1	0	0
3	1	0	0
wavy lines			
4	0	0	0
5	1	1	1
6	1	1	0

No unit duplicates another but it can be seen that while units 2 and 3 are essential, removing unit 4 would have no effect on the ability to separate the two classes. If unit 3 were removed the outputs on the first layer for pattern 6 would be the same as the outputs of the straight line patterns. However if unit 4 is cut out all patterns in the 'straight' class still look different from all the patterns in the 'wavy' class. Unit 4 can be cut out and the network has now been pruned from 40-5-2-1 to 40-2-2-1. This first layer now reduces the number of different patterns received by the second layer from six to three. All 'straight' inputs have been reduced to a single pattern.

Examining the second layer outputs reveals that the second layer does not reduce the number of different patterns.

Pattern	Second Layer Outputs	
	Unit 1	Unit 2
straight lines		
1	0	1
2	0	1
3	0	1
wavy lines		
4	0.6	1
5	0	0
6	0	0

The output unit is still receiving three different patterns. One pair of inputs (0,1) to the output unit corresponds to all the straight lines and two pairs of inputs (.6,1 and 0,0) indicate wavy lines. This is the same situation as holds after the first layer.

Thus the second layer is not contributing to the solution and the entire layer can be removed. This is the third and most radical stage of the pruning. The resulting 40-2-1 network is of course not a solution as the outputs of the first layer are not the same as the second layer but with training it quickly becomes one.

The small size of this final network came as a surprise because learning with networks larger than this had been unsuccessful. A 40-2-1 network starting from random initial weights was certainly not able to learn to solve this problem. The extra units in the 40-5-2-1 network were needed for learning to occur but once a solution was found they could be removed without seriously damaging the solution.

Experimenting with Noise

(a) Performance of Pruned Networks

The obvious next question is how pruning affects the performance of the network. Of particular interest is the question of whether reducing the size of the network affects its sensitivity to noise. To examine this a 120-8-4-3 network was taught to separate sinusoidal input patterns according to frequency. All the input patterns were sine waves with amplitude 1.0 . Three training classes were used, each containing four patterns with the same frequency but with different phases. The target outputs were (0,0,1) for an input pattern consisting of 3 cycles, (0,1,0) for a pattern consisting of 6 cycles of a sine wave and (1,0,0) for a 9 cycle input. All patterns were clean, with no noise added. The network was then cut back to 120-4-3 without impairing its performance.

The networks were then tested for noise resistance by adding a random number from a normal distribution about zero with standard deviation of 1.0 to each input. An output was deemed incorrect if it differed from the target value by more than 0.25 . The result of a trial was classed as a disaster if any output differed from the target by 0.5 or more. This is equivalent to 'forcing' each output to zero or one and then rating the result. Note that failures include disasters, so that the success rate of the 120-8-4-3 network is 55%. Each pattern was presented 10,000 times, making 120,000 trials in all. The results were

Network	Failures	Disasters
120-8-4-3	45%	29%
120-4-3	11%	2%

Pruning the network has improved its performance enormously. The conclusion is that redundant units actively detract from a network's ability to recognise distorted inputs.

(b) Learning with Noise

Does the noise resistance of a network increase if it is trained with noise? The full 120-8-4-3 network was given more training using input patterns with random noise added. The noise was generated randomly for each input rather than reading patterns with noise pre-added from a file so that the network would not be able to simply learn a particular pattern of distortions.

Training improved the performance of the network many times over the performance of both the original network and the pruned network

Network	Failures	Disasters
120-8-4-3 trained with noisy inputs	1.7%	0.2%

When the pattern of hidden outputs for the noisy-trained network was compared with the original network it could be seen that the network had changed. A previously useless unit now gave a unique pattern of outputs and more decision making was being done at the second layer. In the original network the second layer could be first cut to three units and then discarded entirely but after training with noise each unit of the second layer was unique and essential.

(c) Pruning After Noisy Training

First stage pruning, the removal of repetitive units, was performed on the noisy-trained network, bringing it down to 120-5-4-3. No pruning could be performed on the second layer and if the layer was cut out no solution could be found.

Second stage pruning could still be performed on the first layer. Removing any two of the five remaining units continued to give different first layer output patterns for each class, although no unit duplicated another. The fact that the noisy-trained network could be cut back to 120-3-4-3 while the original, clean-trained network was cut back to 120-4-3 demonstrates that the network learned to overcome noise by deferring more decision making to the second layer.

Testing these reduced networks with a large sample of noisy inputs showed that removing redundant units again improved performance but further pruning degraded the network's performance with noise. The results of all tests are summarised below.

Network	Failures	Disasters
Original 120-8-4-3	45%	29%
Pruned to 120-4-3	11%	2%
Noisy trained 120-8-4-3	1.7%	0.2%
Pruned stage 1 to 120-5-4-3	0.6%	0.1%
Pruned to min. 120-3-4-3	2.0%	0.6%

Learning with noise produced a 10-fold improvement in correctly classifying noisy patterns. Removing repetitive units from this network halved the error rate again, giving peak performance. Removing more units gave a network that correctly classified clean signals but gave slightly degraded performance with noisy signals. The implication is that extra units are identifying different features of the input classes that can be used at the next layer to make a more reliable classification.

Conclusion

Several important conclusions can be drawn from these results. The question, "How large does a network need to be?" does not have simple answer. A network needs more units to learn a solution than are needed to implement the solution. Longstaff and Cross give guidelines to the sise of network needed to implement a solution but for any but the simplest problems a larger network is needed to find the solution. Once a network has learnt a solution to a problem it can be pruned to the minimum sise. This improves the noise resistance of the network and reduces the processing time required to give a result. For a neural net simulation running on a von Neumann computer, the processing time is proportional to the number of units in the net.

The noise resistance improves when useless or repetitive units are removed (stage 1 pruning). Removing more units degrades the performance with distorted inputs, so redundancy in the network does aid the stability of the solution.

Finally, using distorted or noisy signals in the training set improves considerably the performance of the network. The network learns to use more features of the inputs and to combine them at a later stage. In our example a 20-fold reduction in the number of failures was achieved compared to the best (smallest) network taught only with clean signals.

For the best possible performance a network should be first taught using noisy signals and then pruned of any repetitive units.

The authors would like to thank Mr Frank May for his encouragement and criticism, from which this work has greatly benefited.

REFERENCES

[1] D.E. Rumelhart, G.E. Hinton and R.J.Williams, "Learning Internal Representations by Error Propagation", published in Rumelhart & McClelland (Eds.), "Parallel Distributed Processing: Explorations in the Microstructure of Cognition. Vol. 1:Foundations". Bradford Books/MIT Press 1985.

[2] I.D. Longstaff and J.F. Cross, "A Pattern Recognition Approach to Understanding the Multi-Layer Perceptron", RSRE Memorandum No. 3936, 1986.

[3] D.C.Plaut, S.J. Nowlan and G.E. Hinton, "Experiments on Learning by Back Propagation", Carnegie-Mellon Univ., Pittsburgh, PA, 1986