# NETWORK REDUCTION TECHNIQUES

**T.D. Gedeon & D. Harris**
Department of Computer Science,
Brunel University

**Abstract**:

Much research has recently been carried out using feed-forward networks of a few layers trained by back-propagation to solve a variety of problems.

Many workers have remarked that to train such networks within reasonable time scales, or to train them at all, it is necessary to include more hidden units than would appear to be required. This is certainly the case for some hand-crafted networks for some simpler problems. These excess units perform no real function in the final product, and may be supportive of significant units, may solve for other functions, or may literally be unused. In any case, they are unnecessary for the post-learning utilisation of the network.

It has been suggested that these excess units could be removed and the reduced system retrained from what is in effect a new starting point. These methods have some efficiency disadvantages, and can not readily deal with all the cases we have identified.

We have made a study of the activity of these hidden units, and have identified a number of classes of excess units, some not previously reported. We can automatically locate them, remove them from the network and automatically adjust the network weights so that the reduced network produces an equivalent solution without the need for further training.

## Introduction:

In this paper we will generally assume a feed-forward network of three layers of processing units. All connections are from units in one level to the subsequent one, with no lateral, backward or multilayer connections. Each unit has a simple weighted connection from each unit in the layer above. For simplicity of discussion only, we have assumed a single output unit separating the input patterns into two classes. The network is trained using a training set of patterns with desired outputs, using back-propagation of error measures (Rumelhart and McClelland, 1986). Most workers now use batch rather than sequential pattern by pattern updating of weights. By back-propagation we mean the general concept of developing the error gradient with respect to the weights, and not restricted to the standard gradient descent method. In the examples we cite here we have used the the basic logistic activation function $y=(1-e^{-x})^{-1}$, again this is not germane to the generality of our method. Training by back-propagation is popular because of its simplicity theoretically, and the ease of use or production of such networks. There is also a wide availability of simulators, often at low cost. The method has been used successfully in a number of disparate areas ranging from speech synthesis (Sejnowski and Rosenberg, 1987), to bond underwriting (Dutta and Shekhar, 1988).

## Disadvantages of back-propagation:

The major disadvantages of the back-propagation method are that it can be slow to train networks, and that the architecture required for a solution to a problem is not currently determinable *a priori*. In practice, it is deciding the number of hidden units which is most

difficult, as the numbers of input and output units can be determined straightforwardly, depending as they do on the number of input features chosen and the number of classes into which the input patterns are to be divided.

We will not discuss algorithmic solutions to disadvantages of back propagation, except to note that while using weight decay, if the weight linking two units has decayed to zero at the end of training, the activation of the unit using this link is ignored and is analogous to the direct architectural removal of the unit.

The problem of deciding network size is likely to remain difficult in that if the number of hidden units required for a minimal solution could readily be calculated, then this would imply that we could decide what internal representations will be required. We would then be able to solve the problem by some method directly using these internal representations, and would not need the training-by-example capabilities of neural networks. That is, the use of neural networks in general is likely to remain most important in areas in which the then current state of knowledge there remain problems which cannot be solved directly.

Many workers have remarked that to train networks successfully or at an effective rate, more hidden units are required than the minimal number. This is certainly the case for some hand crafted networks. Similarly, brute force methods such as eliminating randomly chosen units from trained networks and continuing training can be used at great expense of time to produce network sizes which cannot be trained up from scratch.

The rules of thumb one can glean from comments in the literature, for what to do when your network fails to learn, range from adding one or two units, to 10% extra units, or even to doubling the number of hidden units. When we have a successful network, we would like to reduce it to the minimal size required to solve the problem for two major reasons. Firstly, for increasing the efficiency of the network during actual use, and most importantly, to discover the minimal size so we will be able to make a better estimate of the minimal size the next time we are trying to solve a problem which is in some sense similar.

We will first briefly overview some approaches taken by others in this area before introducing our method for determining the distinctiveness of units.

The seminal work on pruning trained networks (Sietsma and Dow, 1988) uses the outputs of units in a two stage pruning process. In the first stage, units whose inputs are always close to zero are removed. The outputs of the other units are banded to *0* or *1*. The pattern of outputs are compared, and any that are duplicates or inverses by inspection are removed, and the network undergo further training to restore the solution. In the second stage of pruning, unit outputs are again inspected for redundancy with regards to the separation of classes within the presented pattern space. Also, entire layers are examined for the number of classes being coded, and compared with the number of classes needed at the next layer. The example given allows the entire layer to be removed. The resulting overall network can be trained further to again retrieve the solution, but can not learn it if restarted in this topology. Such pruning by inspection is quite difficult even on small examples, so clearly some automatable process would be ideal. A number of workers have made attempts in this direction, and have delineated various properties to determine which units can or should be eliminated.

**Relevance**:

The property of *relevance* (Mozer and Smolenski,1989) is based on the idea of comparing how well the network does with the unit in place, versus the situation if the unit was removed. The delineation of this property was reportedly prompted by the difficulties of balancing primary and secondary error terms as required in weight decay. This ideal measure has a problem, in that the quadratic error function ordinarily used in back propagation networks is not suitable for estimating relevance. Thus, the error function must be modified, or a separate back-propagation phase using a different error function must be

introduced. Neither of these alternatives are perfect, modification of the error function may be seen to be less elegant, while a separate back-propagation phase is less efficient. The latter alternative has been used. The unit which has the least relevance is removed.

**Contributions**:

Contribution analysis depends upon storing the weight and hidden unit activations for each combination of input patterns, hidden unit and output unit. This *contribution* (Sanger,1989) is the product of the activation of the hidden unit and the weight from the hidden unit to the output unit. The result is sign corrected so that the measure no longer indicates whether the output unit is stimulated or inhibited, but rather whether it is modified in the correct direction. This method has been tested on a cut down version of NETtalk called micro-NETtalk.

A disadvantage of this method is the large amount of data stored in a three-dimensional array. Using principal component analysis on two dimensional cross-sections of the area can be done on for each hidden unit to produce a set of vectors for each input presentation to reveal the pattern of output units that the hidden unit is responsible for. Similarly, for each output unit the pattern of hidden unit contributions by input presentation can be discovered.

Contribution analysis suffers a weakness in cases of output units which are not activated up until the time of the analysis, in that there is no separation for the analysis to find and hence the contributions will be hidden even though they will probably be the most important factor in properly (in)activating the units. Contribution analysis has not actually been used in pruning, though the author suggests a similarity to the *relevance* measure, and may be usable in that manner.

**Sensitivity**:

Estimation of the *sensitivity* (Karnin, 1990) of the global error function to the removal of a unit can be done by recording the incremental changes to synaptic weights during an epoch of back-propagation. This is similar in concept to *relevance*, although a different error function is not required. The information used in approximating the sensitivity uses terms which are often available during training with back-propagation, such as the weight increments, and the partial derivatives of the error surface. The unit with the lowest sensitivity can be removed.

**Badness**:

The *badness* factor (Hagiwara, 1990) for a hidden unit is the sum of back-propagated error components over all patterns. This is an even simpler measure than *sensitivity*, and requires marginally less computation. The unit with the largest badness factor is removed.

This approach has the advantage of simplicity, and is intrinsically local unlike the three previous properties, but is based on shakier grounds. It is by no means clear that removing the unit with the largest recent historical accumulative error is actually a good idea. This can be seen if we look at the situation from the perspective of what effect significant back-propagated error signals into a unit will have. If the error is low, we may conclude that it is recognising some feature adequately. If the error is high, the unit will be significantly modifying its weights (learning) in response to the error signals. In the middle of these extremes are units which change gradually. It is not obvious that removing a unit doing a lot of learning a will be of benefit - the majority of learning it has done will be lost.

It has often been commented that the last few features often take a disproportionate amount of time to be learnt. It is quite possible, that the unit(s) amassing the highest

badness factor will have been learning these last features. This can be counteracted by only invoking the above mechanism very infrequently. This has been used for example, only when the change in the total sum of the squared error from sampling every 10 to 100 epochs drops below typically 1-5%. In this fashion, the badness factor consists of much less of this aspect of learning. This method is thus limited to being useable only intermittently compared to the other three methods.

**Taxonomy of undesirable units**:

We have identified a number of classes of units which are not necessary to the network and could be removed. We will first describe a number of these classes and then show how we can form a taxonomy which will allow us to use a few related methods to detect all of them.

Classes of undesirable units include:

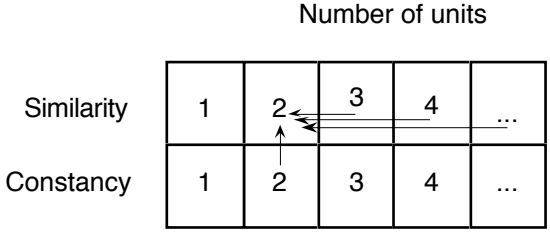
- A unit which performs no function can arise in a number of ways, such as its weights linking to output are all zero or very 'small,' the unit is always off (eg weights and bias all negative), or the unit is always on. This latter case may be supportive of other unit's effects in that some varying factor will be supplied to the output unit, but its effect will be small and should be able to be subsumed by some significant unit.

- Two or more units may be too similar, in that they produce the same output for each pattern.

- Groups of units together having no function can be two units complementary to each other in that for any pattern their outputs are inverses of each other; alternatively three or more units may produce no effect.

- Groups of units may together produce a constant effect across the pattern set.

All of these classes of units, along with useful units, can be described in terms of a two-dimensional categorisation array. The horizontal dimension is the number of units involved, while the vertical dimension is labelled with the items similarity, and constancy.




The similarity row of our categorisation array consists of the cases: one unit similar to itself, two units similar to each other, three units all similar, etc. The first of these cases is that of the useful units, in that they are similar only to themselves, and thus serve a useful function. The rest of the cases involve increasing numbers of unnecessary units, all but one of which can be removed.

The constancy row of the categorisation array consists of the cases: one unit producing a constant effect, two units together producing a constant effect, three units together producing a constant effect etc. Since a zero effect is also constant, the first case includes not only units always turned on, or off, but also those units with insignificant effects as well. For the zero effect alternative, the second case consists of a pair of complementary units, in the sense that the same information is conveyed to the next

layer by both units, but with opposite sign so that they nullify each other's effects. The rest of the cases consist of increasing numbers of units which jointly have no or constant effects. All of the cases in this category are such that all the units can be removed, with biases needing to be modified suitably when the constant effect is non-zero.

Computationally, the third and later cases for the similarity row collapse into the second case, if we test for the similarity of all pairs of units. For the constancy row, the first case is readily discovered, while the second case can be discovered by the same means as the similarity of pairs of units, as we shall see in the discussion of our method of distinctiveness analysis. The discovery of three or more units with constant or zero effects is more difficult. A brute force approach on $n$ units would require of the order of $2^n$ trials of removing a group of units and testing the network for degradation of performance.

**Distinctiveness**:

The *distinctiveness* of hidden units is determined from the unit output activation vector over the pattern presentation set. That is, for each hidden unit we construct a vector of the same dimensionality as the number of patterns in the training set, each component of the vector corresponding to the output activation of the unit. This vector represents the functionality of the hidden unit in (input) pattern space. In this model, vectors for clone units would be identical irrespective of the relative magnitudes of their outputs and be recognised. Units with short activation vectors in pattern space are recognised as insignificant and can be removed.

The recognition of similarity of pairs of vectors is done by the calculation of the angle between them in pattern space. Since all activations are constrained to the range 0 to 1, the vector angle calculations are normalised to 0.5, 0.5 to use the angular range of 0-180° rather than 0-90°. Units with exactly identical functionality are uncommon, however angular separations of up to about 15° are considered sufficiently similar and one of them is removed. The weight vector of the the unit which is removed is added to the weight vector of the unit which remains. With low angular separations as above, the averaging effect is insignificant and the mapping from weights to pattern space remains adequate in that the error measure is no worse subsequently.

This produces a network with one fewer unit which requires no further training. A number of individually redundant units can be removed simultaneously in this fashion. Similarly, units which have an angular separation over about 165° can be seen to be effectively complementary, and both can be removed.

The following table shows a set of six hidden units with two pairs of similar units, with the units in one pair being anti-similar or complementary to either of the units in the other pair.

| | Hidden Units | | | | | | | |
|---|---|---|---|---|---|---|---|---|
| Pattern | 1 | 2 | 3 | 4 | 5 | 6 | Result | Target |
| p.000 | 0.330 | 1.000 | 0.910 | 0.582 | 0.593 | 0.381 | 0.000 | 0.000 |
| p.001 | 0.091 | 0.994 | 0.339 | 0.101 | 0.834 | 0.839 | 0.000 | 0.000 |
| p.002 | 0.098 | 0.994 | 0.348 | 0.130 | 0.874 | 0.868 | 0.000 | 0.000 |
| p.003 | 0.022 | 0.488 | 0.026 | 0.012 | 0.960 | 0.982 | 0.025 | 0.000 |
| p.004 | 0.094 | 0.994 | 0.325 | 0.128 | 0.853 | 0.849 | 0.000 | 0.000 |
| p.005 | 0.021 | 0.489 | 0.024 | 0.012 | 0.952 | 0.979 | 0.025 | 0.000 |
| p.006 | 0.023 | 0.488 | 0.025 | 0.016 | 0.965 | 0.984 | 0.025 | 0.000 |
| p.007 | 0.005 | 0.006 | 0.001 | 0.001 | 0.990 | 0.998 | 0.805 | 1.000 |
| p.008 | 0.117 | 0.994 | 0.339 | 0.132 | 0.875 | 0.870 | 0.000 | 0.000 |
| p.009 | 0.026 | 0.488 | 0.025 | 0.012 | 0.960 | 0.983 | 0.025 | 0.000 |
| p.010 | 0.029 | 0.488 | 0.026 | 0.016 | 0.971 | 0.986 | 0.025 | 0.000 |

| | | | | | | | | |
|---|---|---|---|---|---|---|---|---|
| p.011 | 0.006 | 0.006 | 0.001 | 0.001 | 0.991 | 0.998 | 0.805 | 1.000 |
| p.012 | 0.027 | 0.489 | 0.024 | 0.016 | 0.965 | 0.984 | 0.025 | 0.000 |
| p.013 | 0.006 | 0.006 | 0.001 | 0.001 | 0.990 | 0.998 | 0.805 | 1.000 |
| p.014 | 0.006 | 0.006 | 0.001 | 0.002 | 0.992 | 0.998 | 0.805 | 1.000 |
| p.015 | 0.001 | 0.000 | 0.000 | 0.000 | 0.998 | 1.000 | 0.816 | 0.000 |

The above network still has not converged as can be seen from the significant discrepancy between the result and target values for pattern number 15. It is possible to discover the pairs of similar units by inspection of the above table, but would be prohibitively difficult for much larger numbers of patterns or units. The vector angles for the above table are shown below:

| Pair of units | Vector angle |
|---|---|
| 1 2 | 81.8 |
| 1 3 | 25.2 |
| 1 4 | 8.4 |
| 1 5 | 176.5 |
| 1 6 | 169.9 |
| 2 3 | 63.0 |
| 2 4 | 77.8 |
| 2 5 | 100.8 |
| 2 6 | 103.7 |
| 3 4 | 18.0 |
| 3 5 | 157.7 |
| 3 6 | 163.7 |
| 4 5 | 173.7 |
| 4 6 | 177.5 |
| 5 6 | 7.3 |

As mentioned above, a further category of undesirable units is also discovered and included in the distinctiveness analysis. Groups of three or more units which together have no effect, or two or more units with a constant effect can be recognised. That is, in the pattern space, the sum of their vectors is zero, or a constant.

For finding this further category, the pattern space vectors are banded to 0, 0.5, and 1.0 adaptively. We use the two largest edges in differences in unit output activations, rather than fixed values such as the obvious 0.35, 0.65 boundaries. In practice we have seen examples of units which by inspection are identical in functionality but not magnitude of output activation, but would band totally differently due to the differences in magnitudes.

The discovery of such groups is done by a sorted Gaussian vector pivot on the cumulative rectangular matrix of pattern space vectors. This produces the reduced row echelon matrix. For the case of groups of units with jointly no effect, the entire group can be removed. If the joint effect is constant, a bias can replace the entire group. Because of the inaccuracies incurred by banding and subsequent use of this information, it may be necessary to retrain the network. Fortunately, such groups are not common, therefore retraining is not often required.

**Conclusion**:

We have identified a number of classes of excess, or undesirable units, and collected them into a comprehensive taxonomy. We have demonstrated how we can automatically locate them, and remove them. With the network weights being adjusted appropriately, we can create reduced networks which produce a solution equivalent to that of the original trained network without the need for further training.

**References**:

Dutta, S, Shekhar,S, “Bond rating: A non-conservative application of neural networks,” IEEE Int Conf on Neural Networks, vol. II, pp. 443-450, 1988.

Hagiwara, M, “Novel back propagation algorithm for reduction of hidden units and acceleration of convergence using artificial selection,” IJCNN, vol. I, pp. 625-630, 1990.

Karnin, ED, “A simple procedure for pruning back-propagation trained neural networks,” IEEE Transactions on Neural Networks, vol 1., pp. 239-242, 1990.

Mozer, MC, Smolenski, P, “Using relevance to reduce network size automatically,”, Connection Science, vol. 1, pp. 3-16, 1989.

Rumelhart, DE, Hinton, GE, Williams, RJ, “Learning internal representations by error propagation,” in Rumelhart, DE, McClelland, “Parallel distributed processing,” Vol. 1, MIT Press, 1986.

Sanger, D, “Contribution analysis: a technique for assigning responsibilities to hidden units in connectionist networks”, Connection Science, vol. 1, pp. 115-138, 1989.

Sejnowski, TJ, Rosenberg, CR, “Parallel networks that learn to pronounce English text,” Complex Systems, vol. 1, pp. 145-168, 1987.

Sietsma,J, Dow, RF, “Neural net pruning - why and how,” IJCNN, vol. I, pp. 325-333, 1988.