Indicators of Hidden Neuron Functionality: the Weight Matrix versus Neuron Behaviour

T.D. Gedeon
School of Computer Science and Engineering
The University of New South Wales
Sydney 2052 AUSTRALIA
http://www.cse.unsw.edu.au/~tom

Abstract

There are a number of reasons for attempting to differentiate neurons with respect to their functionality in back-propagation neural networks. With only a similarity measure, we can prune redundant or less important hidden neurons, which is useful for a host of reasons, ranging from improvements of generalisation performance, to use as a precursor for rule extraction. With a measure that can numerically rank neurons as to degree of similarity, we can visualise the training process, grow networks by adding neurons when more functionality is required, and distinguish between networks which are stuck in a local minimum from those which are temporarily in a shallow valley in weight space.

In this paper we consider the computationally cheaper alternative of using only the trained weight matrix to determine similarity and numeric measures of neuron functionality.

We contrast the use of neuron behaviour dynamics over the training set as an indication of neuron functionality against the use of the static trained weight matrix in two problems. The first is the use of a similarity measure for pruning in an image processing application to produce a quality driven compression by eliminating the least different hidden neurons. The second problem is the use of a numeric measure on a small problem to demonstrate differentiation between networks which are trapped in local minima, those in a shallow valley and those which learn quickly.

We conclude that the weight matrix is not sufficient for differentiating the functionality of the hidden neurons for these tasks, being essentially the functional equivalence problem which is computationally intractable.

1 Assumptions

In this paper we will generally assume a feed-forward network of three layers of neurons. All connections are from neurons in one layer to the subsequent one, with no lateral, backward or multilayer connections. Each neuron has a simple weighted connection from each neuron in the previous layer. The network is trained as an auto-associator using a training set of input patterns with desired outputs being the same as the inputs, using back-propagation of error measures (Rumelhart et al, 1986). Training by back-propagation is popular because of its simplicity theoretically, and the ease of use and production of such networks.

The method has been used successfully in a number of disparate areas ranging from pattern synthesis (Lewis, 1991), to image compression (Cottrell, et al, 1987, Namphol, et al, 1991, Gedeon and Harris, 1992a).

2 Introduction

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 neurons which is most difficult. Many workers have remarked that to train networks successfully or at an effective rate, more hidden neurons are required than the minimal number. Extra neurons which end up duplicating the functionality of existing neurons do nothing but decrease the speed of the network by increasing its size. Also, significant time is lost for restarting the training process from scratch.

Brute force methods to find minimal size networks by eliminating randomly chosen neurons from trained networks have been used; recently some approaches have emerged delineating properties to choose which neurons should be eliminated.

The seminal work on pruning trained networks (Sietsma and Dow, 1988) uses the outputs of neurons in a two stage pruning process, which operates by inspection. Such pruning by inspection is difficult even on small examples, some automatable process would be ideal.

Properties such as *relevance* (Mozer and Smolenski, 1989, Segee and Carter, 1991), *contribution* (Sanger, 1989), *sensitivity* (Karnin, 1990), *badness* (Hagiwara, 1990), and *distinctiveness* (Gedeon and Harris, 1991a) have been described in detail elsewhere.

We will briefly describe *distinctiveness* here, as it is our own, conceptually one of the simplest, and also forms the basis of comparison for the rest of this work. Note that all of the other methods are at least as computationally expensive as our own method.

The *distinctiveness* of hidden neurons is determined from the neuron output activation vector over the pattern presentation set. That is, for each hidden neuron 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 that particular neuron. This vector represents the functionality of the hidden neuron in (input) pattern space.

In this model, vectors for identical or clone neurons would be the same irrespective of the relative magnitudes of their outputs and will be recognised. On the basis of experience, for normal pruning, angular separations of up to a threshold of 15° are considered too similar and one of the neurons is removed. The weight vector of the neuron which is removed is added to the weight vector of the neuron which remains. With low angular separations, the averaging effect is insignificant and the mapping from weights to pattern space remains adequate in that the error measure is not significantly worse subsequently.

This produces a network with one fewer neuron which requires no further training. Similarly, neurons which have an angular separation over about 165° are complementary, and both can be removed. In this work we are not interested in distinguishing the different forms of similarity (eg complementarity), hence angular separations over 90° are mapped back to the 0° to 90° range, this range linguistically is from *identical* to *orthogonal* in behaviour. This threshold acts as the qualitative similarity measure we require for pruning.

3 Application domain

In the image compression application, we use a feed-forward neural network called an *auto-associative* network, as the same patterns are used as the input and target patterns. The hidden layer consists of fewer neurons than the input layer, thus compressing the image.

As the output layer is the same size as the input layer, it is used to recover the compressed image. Clearly this is a form of lossy compression, with the loss in quality on decompression being dependent on the overall goodness of the (generalised) representation the hidden neurons manage to form of the patterns being compressed.

In the case of image compression, the degree of compression desired could be used to determine a hidden layer size. Given the earlier observations, however, we should use a somewhat larger size initially. Nevertheless, for the measure of the degree of compression to be meaningful, any neurons with redundant functionality after training must be removed. Otherwise, results are not comparable between different training regimes, as well as inconsistent using the same regime, given that with different initial random weights, a different number of functionally useless neurons may result.

4 Neuron behaviour in repeated pruning – image compression

A 64 by 64 image with 4 bits per pixel was chose-n for testing our method. The image was broken into 16 non-overlapping 16 by 16 images, as shown in Figure 1. Each 16 by 16 image is a separate training pattern.

The image was broken into non-overlapping pieces so as to allow for generalisation to have clearly occurred – since there is little obvious similarity between the pieces, and particularly the large areas of white space around the edges of the image could be expected to interfere. This allowed a single image to be used, thus simplifying the discussion.

Note the tall profile of the pixels, due to using a subset of ASCII varying in image density to represent gray levels. The image was captured by the author off the electronic net in the U.K. – the picture originated in the U.S. and had been distributed worldwide.

The network architecture used was a 256 input, *x* neuron hidden layer, and 256 output network, each pixel in the 16 by 16 parts of the image is a single input, and output. The number of bits per pixel were mapped onto the 0 to 1 range as input, and the output values remapped to the simulated gray scale. The initial value of *x* was 16.

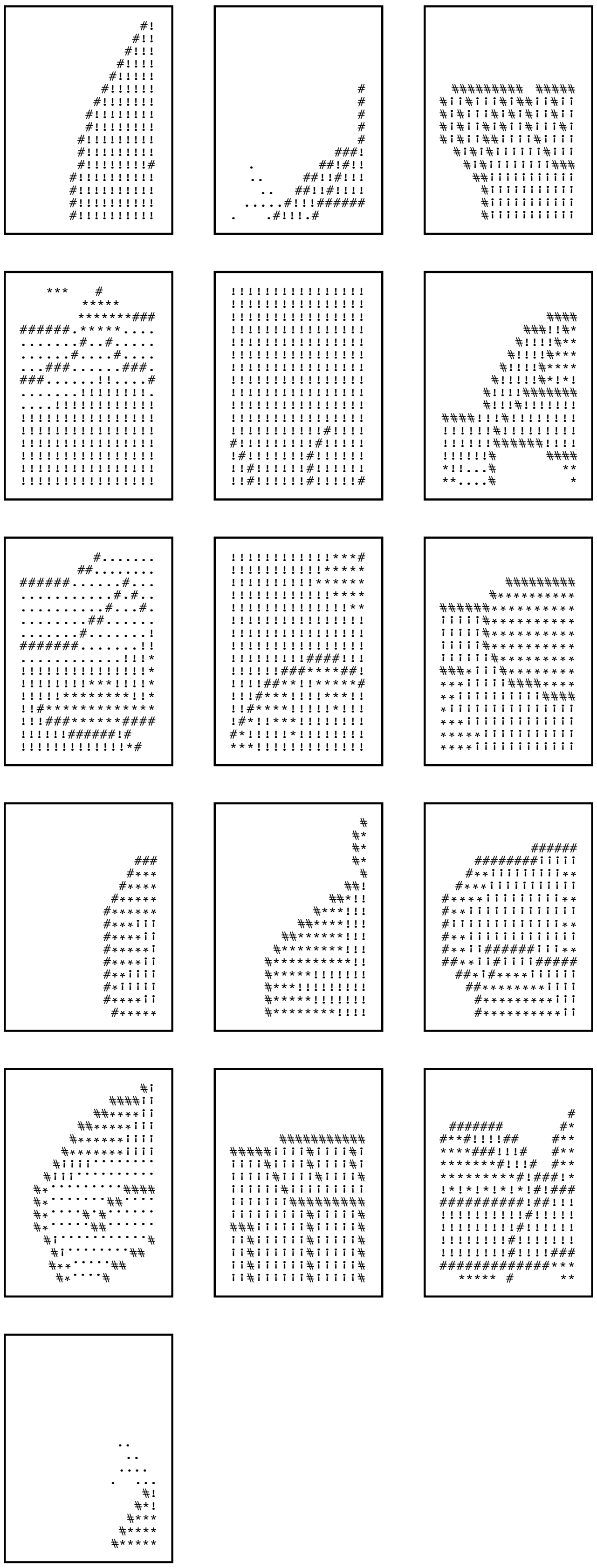
Figure 1: Broken up image

This gives a minimum compression ratio of 16 to 1, which is reasonable given the (deliberately) generalised network architecture used.

The optimum network training time was found using keep one out training and the number of epochs averaged. A final network was initially trained for the optimum number (200) of presentations of all 16 image patches. The only pre-processing done was to recognise that images of objects have white space around them, so patches with significant white space were flipped horizontally / vertically as required to place the most optically dense corner of a patch at the bottom right.

Using the *distinctiveness* angular measures, we progressively reduce the size of the hidden layer. Beyond a certain point, the neurons we remove are significantly distinct – we are trading image quality for degree of compression.

Figure 2 shows the relationship between the number of hidden neurons and the total sum of squares error measure which can be taken as a rough indicator of the image quality. Initially there is little drop in quality as neurons are removed. Subsequently, each further neuron removed produces a significant degradation in quality.

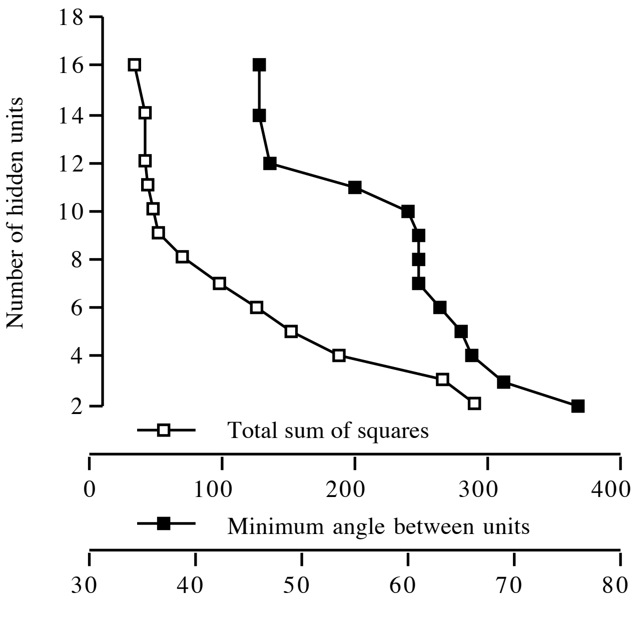

Figure 2: Number of neurons versus image quality and neuron significance

Figure 2 also shows the relationship between the number of hidden neurons and the smallest angle between hidden neurons.

These values are all much higher than the standard 15° we use for pruning redundant neurons. It is interesting to note that the removal of the first few neurons in the vicinity of 60° does not cause a marked reduction in the error measure. These neurons are the equivalent of the secondary backup neurons we introduced for increasing network damage resistance in (Gedeon and Harris, 1991b). Since we are removing significant neurons at each stage, the network will require retraining. After the removal of a neuron, the network is trained for 200 epochs.

Number of units	Compression ratio	Image quality
16	16	excellent
14	18	
12	21	
11	23	good
10	26	ok
9	28	
8	32	
7	37	
6	43	just recog.
5	51	not recog.
4	64	
3	85	nothing

Table 1: Qualitative assessment of image quality

Table 1 lists the qualitative judgements of image quality for the different levels of compression which derive from the number of hidden units.

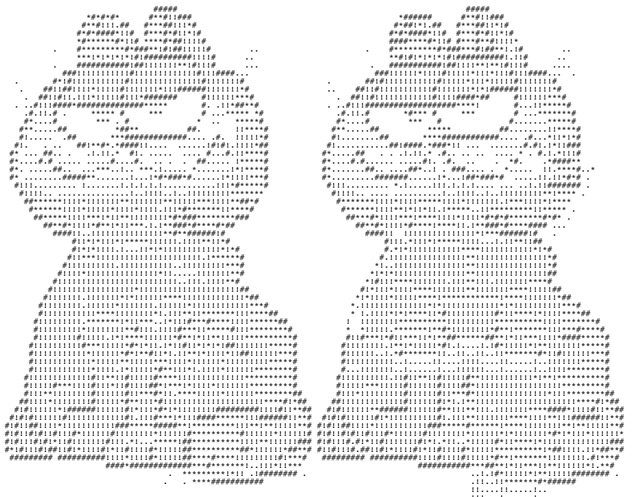

Figure 3: 16 hidden units compression 16 — 10 hidden units compression 26

We can see that the picture is clearly recognisable in the 16 and 10 neuron cases. The image is acceptable in the 7 neuron case at a compression ratio of 37 to 1, and only marginally recognisable in the 6 neuron case, which is at a compression ratio of 43 to 1.

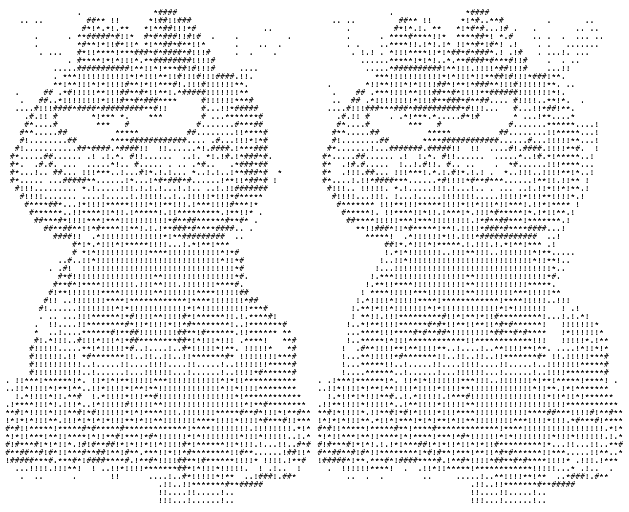

Figure 4: 7 hidden units compression 37 — 6 hidden units compression 43

The use of overlapping patches, or training on a number of similar entire images (Fleming and Cottrell, 1990) would boost the compression ratio of both stages of our process.

5 Output weights in repeated pruning – image compression

We now repeat the calculation of vector angles using weights on outputs instead of activations as the vector components (Gedeon, 1995).

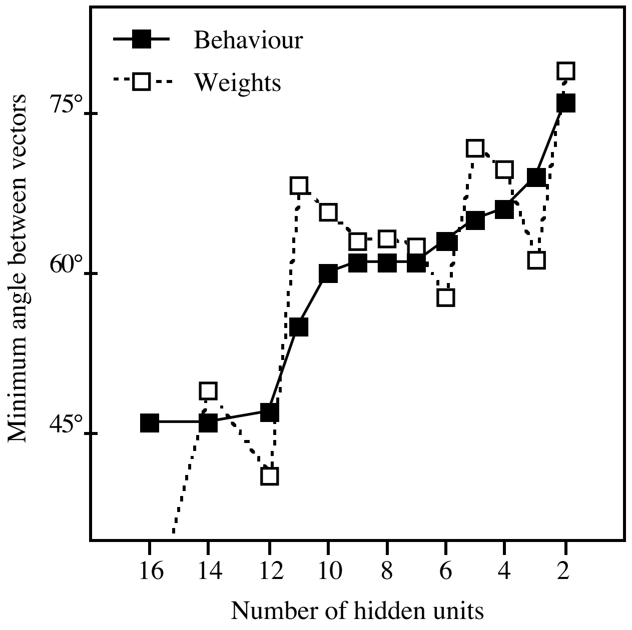

Figure 5: Minimum angles during pruning process

Figure 5 shows the result of using the weights linking the hidden neurons and the 256 neuron output layer. These are normalised between the global maxima and minima found, and then used to form 256 dimensional vectors. The angles between these vectors are then calculated as in the *distinctiveness* approach described earlier using neuron behaviour.

Clearly, the weight matrix measure is coarsely approximating the behaviour of the hidden neurons over the experiment progressively reducing the number of neurons.

6 Which weights?

A further issue deserving investigation is which weights to use in the process of forming the vectors and determining angles. We have used the hidden neuron to output weights in the previous section as being intuitively the most similar to the use of hidden neuron output activations. There are a number of possible arguments for the use of other weights, for example that all the weights connecting to or from a neuron should be used as these embody the full encoding of a specific neuron's algorithm. Also, output weights are not useful alone if there is only a single output, as the angles then map to two values only, being 0° and 180°. The following figure contrasts the use of all of the weights and the use of just input weights (including the bias), with the output weights and actual neuron behaviour used previously.

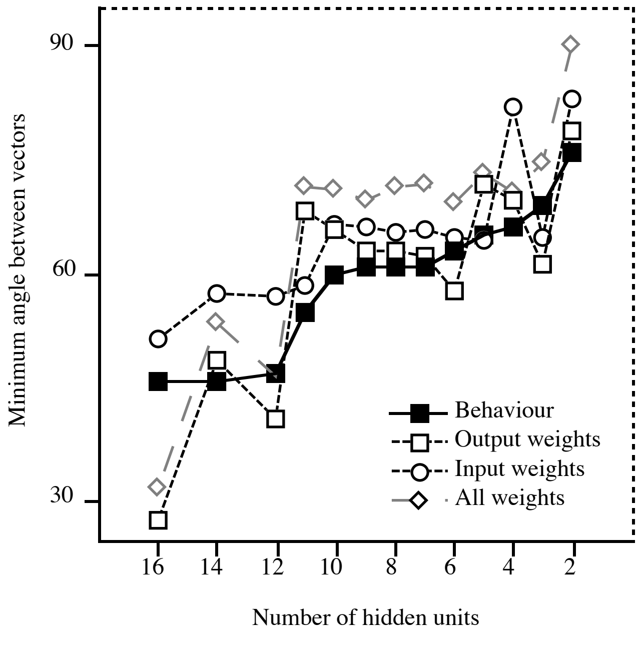

Figure 6: Use of various weight subsets

The neuron angles calculated using input weights in general show less variation than the angles from the output weights. The latter appear more evenly distributed about the actual neuron behaviour angles.

Correlation Matrix for Variables: $X_1 \dots X_4$

	behav	out wts	in wts	all wts
behav	1			
out wts	.83	1		
in wts	.84	.77	1	
all wts	.9	.96	.76	1

The correlation matrix shows that the full weight matrix is better correlated to the actual behaviour than our intuition or subjective appraisal of the graphs in Figure 6 revealed.

We will not investigate the use of the input weights only further here, and just note that we have elsewhere used the weights on the input connections to the hidden neurons successfully to differentiate between significant inputs and relatively less significant inputs (Wong, Gedeon and Taggart, 1995).

In the next section we will therefore use the full weight matrix.

7 Weights & neuron behaviour versus network performance

In previous sections we have discussed the use of the distinctiveness technique on individual neuron behaviour for pruning in a practical application to compress images, and investigated the use of different subsets of the weight matrix to substitute for the computationally relatively complex and inherently dynamic measure of actual behaviour.

In this section we will extend the comparison of the use of the weight matrix versus neuron behaviour to an examination of entire network performance.

In previous work (Gedeon and Harris, 1992b) we have used the neuron behaviour to classify networks being trained on the same problem into three categories:

i) networks which get stuck in a local minimum;
ii) networks which are in a shallow valley in weight space, and will learn more eventually; and
iii) networks which learn quickly.

Of course the latter is easy to distinguish on the basis of decreasing total sum of squares values, but the first two are not statistically separable on that basis. These descriptions are teleological, and not useful during training a network.

What is visible at that time is that the total sum of squares measure is not changing significantly over a number of epochs of training, and hence the network seems to be stuck on a 'plateau'.

Externally, both *i* and *ii* above appear identical, and we would like some indication which of these will learn more without waiting the potentially infinite time required if we judged only by the total sum of squares network error measure.

Distinctiveness analysis has provided some useful insights into the nature of plateaux during network training as demonstrated graphically in Figures 7, 8 and 9.

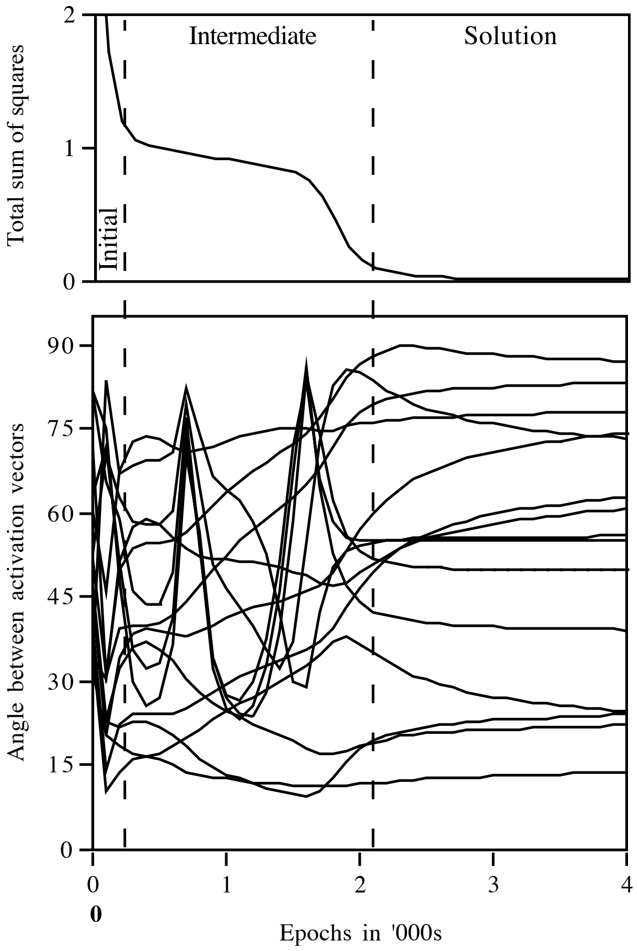

Figure 7: Network with no plateau

In Figure 7, the angles between pattern vectors vary significantly during the plateau phase. This indicates that the plateau phase will end, and thus it is worthwhile to continue training. The figure has been labelled with 'Initial', 'Plateau', and 'Solution' phases, as well as an 'Edge' phase which is a transition between a plateau and a solution.

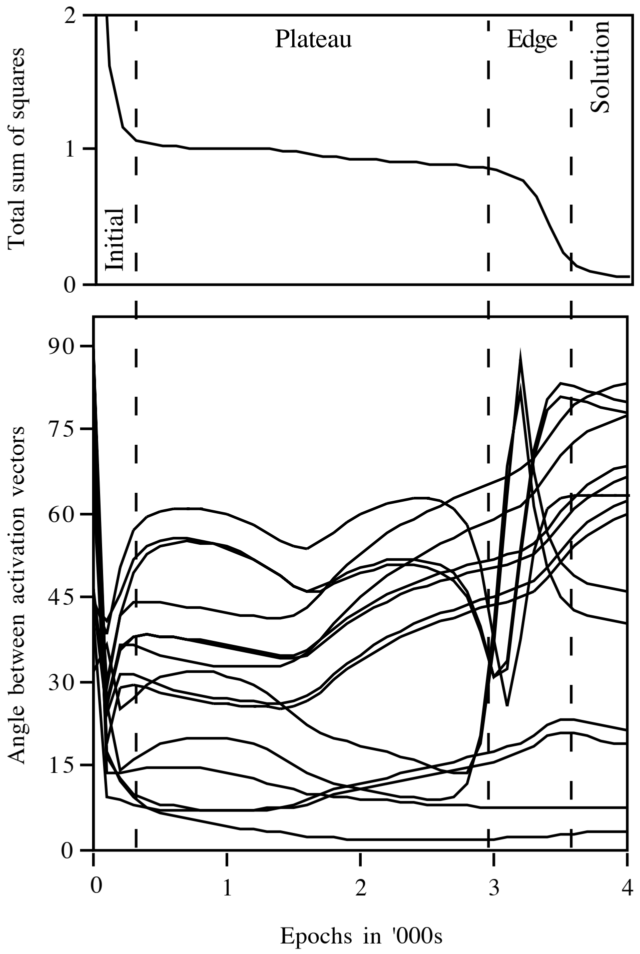

Figure 8: Network with a plateau that ends

To demonstrate the contrast, Figure 8 shows the behaviour of a network for the same problem which does not demonstrate a plateau phase, with the total sum of squares error measure decreasing continually during training.

Although there is some slowing down of this error measure curve in the early part of the 'intermediate' phase, there is continued decrease readily detected from the total sum of squares values.

Figure 8 during the 'Intermediate' stage has very considerable changes in angular measure of pattern vectors, similar to that in the 'Edge' phase in Figure 7.

Also note that in Figure 8 we can see the (usual) cessation of change in the angles between pattern vectors in the 'solution' phase.

Figure 9 demonstrates a third network for the same problem which encounters a (local) minimum. Alternatively, we can say it never gets off the plateau to continue learning. Note the consistently small amounts of change in the activation pattern angles (distinctiveness) at all stages.

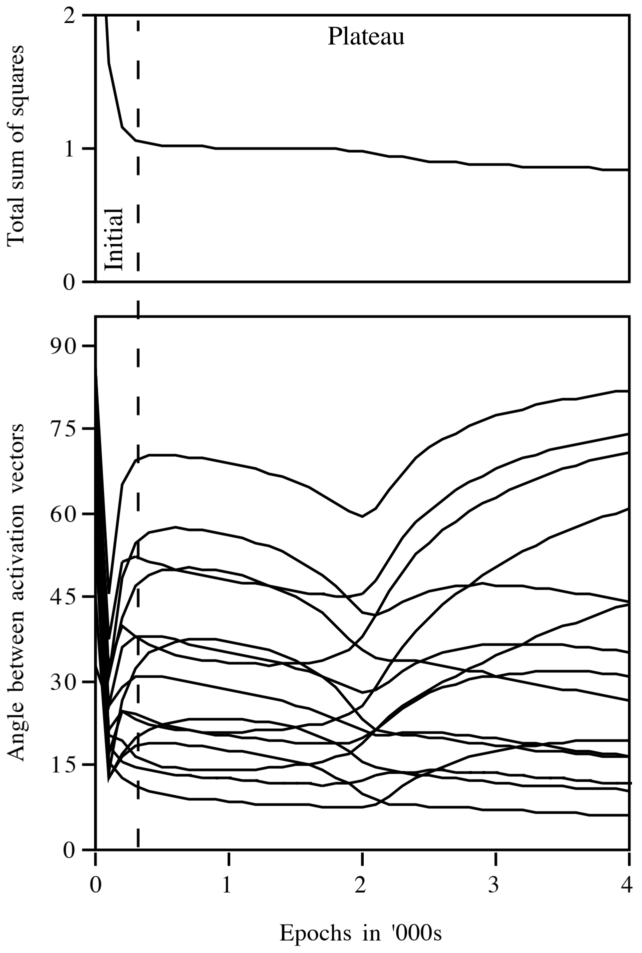

Figure 9: Network with plateau and no solution phase

The point at which there is a clear dip in the activation angle curves is not (statistically significantly) detectable on the total sum of squares curve. In all the cases we have observed, this dip is followed 'soon' by a period of notable change in angles, or the network will not learn and is stuck.

The diagrams in Figures 7 to 9 show that there is a significant difference in the angular separation of unit vectors during a plateau phase and during a minimum or solution phase. Except for very early on in the minimum or solution phase, the vector angles stop changing. This is in contrast with the significant amount of activity during a plateau phase which will eventually cease. That is, by examining the change in vector angles it is possible to deduce whether the network is in a plateau or solution phase. We can also note that the changes in vector angles during the plateau phase indicate that the network is continuing to learn even though this is not reflected in the total sum of squares error measure.

The three diagram shown are representative of the most recognisable classes of results from a number of experiments using a variety of training procedures. We have observed this kind of behaviour in networks of different sizes, and solving different problems.

The weight distinctiveness graphs do not provide such clearly discernible patterns as observed with the activation distinctiveness graphs. The following figures 10 to 12 parallel figures 7 to 9, and illustrate this statement. In each figure, the leftmost graph is the weight distinctiveness graph corresponding to the appropriate activation distinctiveness graph. The rightmost graph is another weight distinctiveness graph which is significantly different. In both cases, the total sum of square graphs are included to aid comparisons.

Thus, Figure 10 demonstrates 2 networks with plateau phases, which eventually learn.

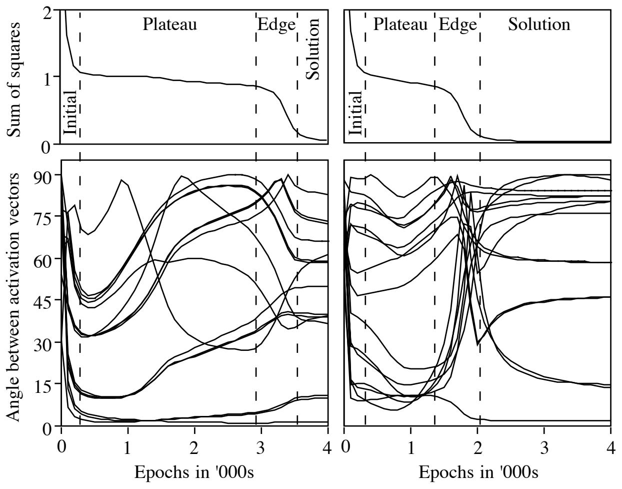

Figure 10: Weight distinctiveness of networks with plateau and solution phases

The rightmost part of Figure 10 demonstrates a weight distinctiveness graph which is quite similar to the activation distinctiveness graphs for networks with plateaux.

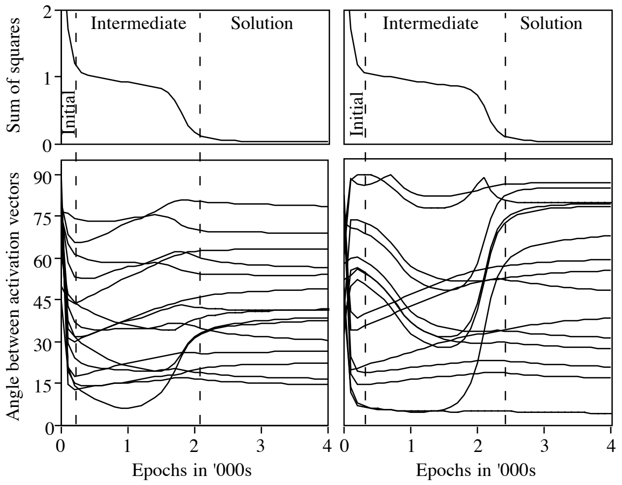

Figure 11: Weight distinctiveness of networks with no plateau phase, and a solution phase

The preceding Figure 11, shows two networks which learn without a plateau phase.

The following Figure 12, demonstrates two networks which never learn.

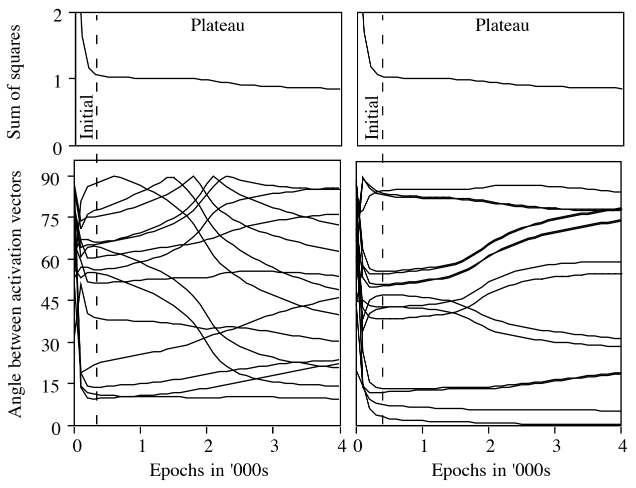

Figure 12: Weight distinctiveness of networks with only a plateau phase, and no solution phase

Note that both Figures 10 and 12 showed a second, rightmost, graph in which the form of the curves was quite similar to those we have identified as characteristic of (one of) the three forms of activation distinctiveness curves. This was to illustrate that in some cases the graphs contain patterns similar to those we have identified. In both of these cases, we could have chosen results from other runs to display which have graphs which bear no discernible relationship.

8 Discussion and Conclusions

We have shown that there is significant correlation between the minimum vector angles for weights and actual neuron activations over a pruning process. Nevertheless, the individual points along the curves show significant deviation, and only coarsely approximating the behaviour of the hidden neurons over the experiment progressively reducing the number of neurons. This is due to the use of the static weight matrix, as it ignores the smoothing effect of the transfer function of the hidden neurons. We may also speculate that there may be many possible weight matrix configurations giving rise to similar neuron behaviour over the range of patterns likely to be encountered.

These alternative matrix configurations for different neurons may explain the erratic weight curve in Figure 5. That is, the actual function implementation within the black box is different. Note that the implementation is subtly different over time within the same network, as shown by the distances between the two curves in Figure 5.

We have also examined the use of the weight matrix for determining whether the weight distinctiveness measure can substitute for the activation distinctiveness in distinguishing between networks which are in a shallow valley in weight space versus those stuck in a local minimum. From an observational viewpoint, we have described these are plateaux which eventually lead to a solution and plateaux which do not lead to a solution. We have demonstrated on a case by case basis that there is significantly less consistency in results using the weight distinctiveness measure.

We can conclude that the weight distinctiveness on the static weight matrix is not sufficiently fine an instrument for differentiating the functionality of the hidden neurons for these tasks, and we are forced to continue the use of computationally somewhat more expensive approaches which use measures based on the dynamics of network behaviour.

We can conclude this because we have used the same instrument (distinctiveness analysis) on both the true functionality of the neural network black box being its behaviour, and on its innards, being the static weight matrix.

Obviously the static weight matrix together with the training patterns contains all of the information which provides the dynamic behaviour of the network, the issue is thus whether we need to run the network and observe its behaviour or can we use its static properties. We have concluded here that the former is necessary. In extracting explanations and rules for neural network conclusions and categorisations we have also found that we require the dynamic behaviour (Gedeon and Turner, 1993).

Alternative approaches attempting to discover whether two networks are identical in functionality whether by algorithmic means or by using another neural network would be unlikely to succeed given our observations that even in a single network where the observed functionality changes smoothly as more neurons are removed, but the same analysis of the weight matrix shows discontinuous changes in the same time period.

Further investigation is required as to the size of subset of the training set which will still provide a reasonable prediction of the functionality of the hidden neurons in a network.

References

Cottrell, G, Munro, P, & Zipser, D, “Learning internal representations of gray scale images,”

Gedeon, TD, Harris, D, “Network Reduction Techniques,” *Proc. Int. Conf. on Neural Networks Methodologies and Applications*, AMSE, vol. 1, pp. 119-126, San Diego, 1991a. *Proc. 9th Ann. Cog. Sci. Soc. Conf.*, Seattle, pp. x-y, 1987.

Fleming, MK & Cottrell, W, “Categorisation of Faces Using Unsupervised Feature Extraction,” *IJCNN*, vol. 2, pp. 65-70, 1990.

Gedeon, TD “Indicators of Hidden Neuron Functionality: the Weight Matrix versus Neuron Behaviour,” *ANNES'95 International Conference on Neural Networks and Expert Systems*, pp. 26-29, 1995.

Gedeon, TD and Harris, D, “Creating Robust Networks,” *IJCNN*, pp. 2553-2557, Singapore, 1991b.

Gedeon, TD and Harris, D “Progressive Image Compression,” *Proceedings International Joint Conference on Neural Networks*, vol. 4, pp. 403-407, Baltimore, 1992a.

Gedeon, TD and Harris, D “Hidden Units in a Plateau,” *Proceedings 1st International Conference on Intelligent Systems*, pp. 391-395, World Scientific, Singapore, 1992b.

Gedeon, TD and Turner, H “Explaining student grades predicted by a neural network,” *Proceedings International Joint Conference on Neural Networks*, pp. 609-612, Nagoya, 1993.

Hagiwara, M, “Novel back propagation algorithm for reduction of hidden units and acceleration of convergence using artificial selection,” *IJCNN*, vol. 1, 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.

Lewis, J, Probing the Critic: Approaches to Connectionist Pattern Synthesis, *Proceedings International Joint Conference on Neural Networks*, vol. 1, pp. 85-89, Seattle, 1991.

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

Namphol, A, Arozullah, M, & Chin, S, “Higher Order Data Compression with Neural Networks, *Proceedings International Joint Conference on Neural Networks*, vol. 1, pp. 55-59, Seattle, 1991.

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.

Segee, BE, & Carter, MJ, “Fault Tolerance of Pruned Multilayer Networks,” *IJCNN*, vol. 2, pp. 447-452, Seattle, 1991.

Sietsma, J, & Dow, RF, “Neural net pruning - why and how,” *IJCNN*, vol. 1, pp. 325-333, 1988.

Wong, PM, Gedeon, TD and Taggart, IJ “An Improved Technique in Porosity Prediction: A Neural Network Approach,” *IEEE Transactions on Geoscience and Remote Sensing*, vol. 33, n. 4, pp. 971-980, 1995.