# A Comparison of some Connectionist Compression Schemes

T. D. Gedeon, J.A. Catalan and J. Jin

Department of Information Engineering
School of Computer Science & Engineering
The University of New South Wales
Sydney NSW 2052 AUSTRALIA
E-mail: tom@cse.unsw.edu.au
Facsimile: +61 2 9385 5995

**Abstract** — In this paper we consider connectionist compression schemes using auto-associative networks, and demonstrate the advantages gained by imposing two different constraints on the allowed network weights, and comparison with pruning of the unconstrained auto-associative network.

## I. INTRODUCTION

Most applications of neural networks for image compression have emphasised the degree of compression [1-3]. In many applications the main consideration is the decompressed image quality [4]. We can guarantee a consistent level of functionality of units in the compression layer based on their distinctiveness, and can progressively reduce the size of the compression layer for the desired level of image quality [5].

Pruning of redundant or less important hidden neurons from the popular back-propagation trained neural networks is useful for a host of reasons. For pruning it is necessary to identify hidden neurons with similar functionality. We have used a pruning process based on the behaviour of the hidden neurons in an image processing application to produce a quality driven compression by eliminating the least different hidden neurons. It is also possible to use the computationally cheaper alternative using only the trained weight matrix of the neural networks at each stage of the compression process [6-7].

In this paper we demonstrate the advantages for generalisation performance of constraining weights symmetrically using weight sharing, and by constraining functional symmetry by the use of enhanced back-propagation networks trained bidirectionally [8-9].

## II. PRUNING

The seminal work on pruning trained networks [10] 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. Properties such as *relevance* [11-12], *contribution* [13], *sensitivity* [14], *badness* [15], and *distinctiveness* [16] have been described in detail elsewhere. We will briefly describe *distinctiveness* here.

The *distinctiveness* of hidden units is determined from the unit output activation vector over the pattern presentation set [16]. 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.

Recognising similar pairs of vectors is by calculation of the angle between them in pattern space. As all activations are constrained to the range 0 to 1, the vectors are normalised to 0.5, 0.5 to use the angular range of 0-180° rather than 0-90°. Angular separations of up to about 15° are considered too 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 little worse subsequently. This produces a network with one fewer unit which requires little further training.

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 constant. The discovery of such groups is done by a sorted Gaussian vector pivot on the cumulative rectangular matrix of pattern space vectors. Fortunately, such groups are not common in our experience, and this part of distinctiveness analysis was not used in this paper.

## III. IMAGE COMPRESSION PREVIOUS WORK

A 64 by 64 reduced greyscale image was chosen for our method. The image was broken into 16 non-overlapping 16 by 16 patches froming separate training patterns, 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.

The network architecture was a 256 input, $x$ unit hidden layer, and 256 output network, each of the pixels in the 16 by 16 parts of the image being a single input, and output. The number of bits per pixel were mapped onto the range 0 to 1 as input, and outputs remapped to the gray scale.

The significance of training an auto-associative network for image compression is that the hidden neurons learn a compressed representation for the input patterns. For transmission over a slow network connection, the neural network weights could be transmitted as a fixed initial cost, while subsequently the compressed representation formed by the hidden units could be transmitted using the activation values of the hidden units for the particular image. All neural compression is inherently lossy, hence interest in such work is largely for insights into image compression and into neural networks that can ensue.

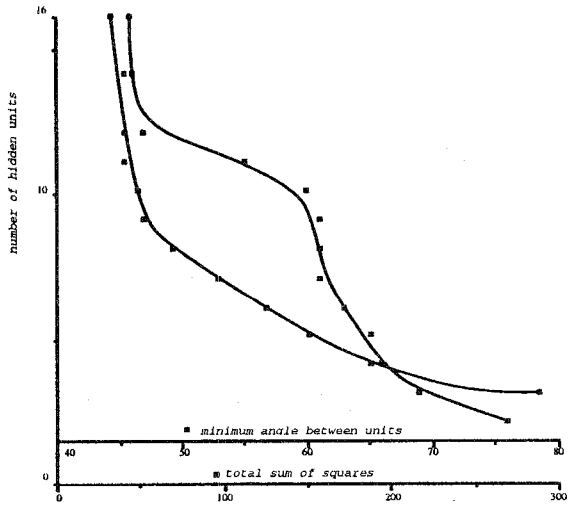


Fig. 1. Units vs. image quality, unit significance

Fig. 1 shows the relationship between the number of hidden units 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 units are removed. Beyond a certain point each further unit removed produces a significant degradation in quality. Fig. 1 also shows the relationship between the number of hidden units and the smallest angle between hidden units. These values are all much higher than the standard 15° we use. It is interesting to note that the removal of the first few units in the vicinity of 60° does not cause a marked reduction in the error measure. Since we are removing significant units at each stage, the network will require retraining. After the removal of a unit, the network is retrained briefly. All such training uses stochastic (pattern) updating of the weights.

## IV. SHARED WEIGHT TOPOLOGY

The topology of the shared weight auto-associative network appears the same as the standard feed-forward network, though the meaning of the weighted connections between units. The connection between input $A$ and hidden unit $B$ in Fig. 2 is the same weight as between hidden unit $B$ and output $C$. That is, the number of free parameters is no longer the same as the number of weights.

The simplest implementation of shared weights in a simulator is to back-propagate errors and update weights as in standard backpropagation without weight sharing, and then after the values of the A-B weight and B-C weight have diverged to average their values. This has the unfortunate effect of increasing the computation time required, but is much clearer conceptually than the alternatives, and was the approach we followed.

The significance of the weight sharing is that the space of possible network weight configurations is very much reduced or constrained. In a standard auto-associative network such as described in the previous section, the first layer of weights from the inputs to the hidden units can be seen as implementing a compression function on the input pattern. The second layer of weights from the hidden units to the outputs implement a decompression function on the compressed image. That is, the hidden unit activations are mapped to recreate an approximation of the original image. (It is only an approximation, because as indicated above, neural compression of the nature we do here is inherently lossy.)

In a shared weight network, the constraining of input to hidden and hidden to output weights to be identical is to effectively require that the compression function be invertible. This has two consequences. Firstly, it may make finding a compression-decompression function harder for a particular case. Secondly, if an appropriate function can be found, we would expect it to perform better in general than a standard backpropagation network in that the inverse function is 'the' inverse function, rather than an approximation of the inverse function.

In a standard backpropagation network, the first layer of weights may implement a non-invertible compression function. Then, the second layer could only implement some approximation to the ideal decompression function which does not exist if the compression function was non-invertible.

## V. BIDIRECTIONAL NEURAL NETWORKS

To introduce bidirectional neural networks [8], we must first restate the directionality implicit in the weights in feed-forward neural networks trained using error back-propagation. The weighted links in these networks are from inputs to hidden, towards the outputs only. (Alternatively, we could view the weights as bidirectional, with different weights in both directions, with the weights in the reverse direction being all zeros.) Some (recurrent) models allow weighted links back to units closer to the inputs, however, the direction of flow on activation along the weighted links is still maintained. Such recurrent connections can be modelled by more complex topologies with shared weights, which do not even require such connections to earlier parts of the network, by making the link implicit. As a philosophical comment, note that the errors are propagated backward through the structure though there are no visible structures for this flow of information to take place.

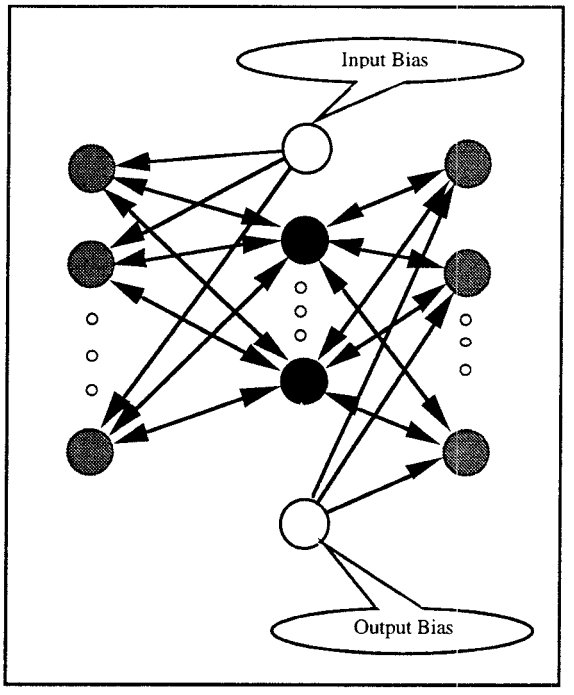


Fig. 2. Bidir. topology showing input / output biases

The practical motivation for the development of this bidirectional model was the recognition that is many situations using a neural network to form a mapping from inputs to outputs is only the first step in a wider process, where the intention is to ask questions such as "given these socio-economic indicators predict a certain GDP values, what modifications of the inputs will produce a higher GDP?" [17].

Fig. 2 illustrates the topology of the network for a bidirectional neural network. The diagram de-emphasises the auto-associative nature of the topology in favour of illustrating the modifications required to allow bidirectional training.

The weights are all now bidirectional and have the same value in both directions. The hidden units are fine as they are, as we allow the bias weights to be used in both directions. The input neurons require a new set of bias weights for use only in the reverse direction.

The network is trained using the error-back-propagation algorithm in one direction at a time. We start in the forward direction and train normally. That is, the training patterns are used so that the input values are applied to the input units, values propagated towards the output, and the target values are compared to the actual outputs and the differences used to modify the weights, with these errors being propagated backwards towards the inputs to potentially modify all the weights. The test patterns are used similarly, with the input values being applied to the input units, and the target values compared to the actual values, though without modifying the weights in the network.

After some number of training events, the direction is reversed. The output neurons are used as inputs. That is, the same patterns from the training data set are used, but this time the 'target' values are applied to the 'output' units, and values propagated towards the 'input' units. The actual values produced by the 'input' units are compared to the desired values of the inputs in the training pattern to derive an error value which is used to modify weights. This error is propagated backwards (in this case towards the 'output' units) potentially modifying all the weights.

The mapping in this image compression experiment is clearly a one-to-one identity mapping. This allows us to avoid discussion of the different sizes of training sets that are ideal for each direction of training, as the input and output dimensions are the same. A simple implementation of bidirectional training for the auto-associative case is to regularly swap the weights connecting the input to the hidden units with the weights connecting the hidden to the output units. Many simulators even provide zero valued place holders for input unit biases, which serve as a handy storage location for the bias weights for the direction not being trained at the time. Clearly we do not need to modify the training or test pattern data files, as the input and target values are identical in an auto-associative task.

## VI. EXPERIMENT DESCRIPTION

This experiment is to compare the quality of compression possible with the three network structures described above.

The basic image used in this experiment is again a 64 x 64 greyscale image, this time the traditional 'Lena' image is used. This has the advantage of continuity with our previous work, keeps neural network training time short as the amount of data is small, and produces images which need to be magnified allowing the effects on pixels to be seen. The Lena image is quite small at full resolution:  and is magnified in subsequent diagrams.

The image is (as in our previous work) cut up into 16 non-overlapping patches. The patches are all sufficiently different as to require the network to learn a suitably general compression function. Again we point out that if the overall objective at this stage was to maximise the degree of compression, there are a number of optimisations we would make, such as to use overlapping patches and so on.

The overall objective is to determine the relationship to the degree and kind of constraints on the weights which will produce good results in the image compression task.

## VII. RESULTS: STANDARD BACK-PROP.

The first nine results are provided below. The network is initially trained for 2,000 epochs, then the distinctiveness of the hidden units is used to select a unit to prune, that unit is removed leaving all other weights unchanged, and the network retrained for 600 epochs. These values were selected based on experience, and on some initial experiments with this task.

IMAGES: STANDARD BACK-PROP.

Fig. 3. Image sequence using back-propagration

The first image is of excellent quality, this decreases over the following images, as more and more hidden units are removed. The last image is of sufficiently low quality that the image is scarcely recognisable, the last row is really provided to illustrate how image quality degrades. This degradation in quality happens sooner than in our previous experiment, which may be due to the less sparse population of the greyscale continuum of values in this case.

We note that cutting images into non-overlapping patches allows ease of exposition but introduces extra edge effects on the output which necessarily impacts on image quality, and does not reflect on the degree of compression available using neural techniques. Hence we will not comment on the degree of compression we can achieve.

## VIII. RESULTS: SHARED WEIGHTS

The network is again trained using the 2,000 / 600 epochs for initial training and retraining after pruning respectively.

The following nine images are similarly produced, using the same initial starting weights as the standard back-propagation example.

IMAGES: SHARED WEIGHTS

Fig. 4. Image sequence using shared weights

The first image is of slightly lower quality than the initial standard back-propagation image. This is not surprising in that the number of weights are the same, but the number of free parameters is halved, which we have explained above can be interpreted as constraints on the function that the network implements.

This function implemented by the shared weights network is more robust, which we can see from the sequence because the degradation in image quality happens more slowly. Thus, we start with a lower quality image, but the final image is better than the standard back-propagation version.

## IX. RESULTS: BIDIR. NETWORKS

These nine images are again produced starting from the same initial weights, and the same initial training and retraining regime as in the previous examples.

IMAGES: BIDIRECTIONAL

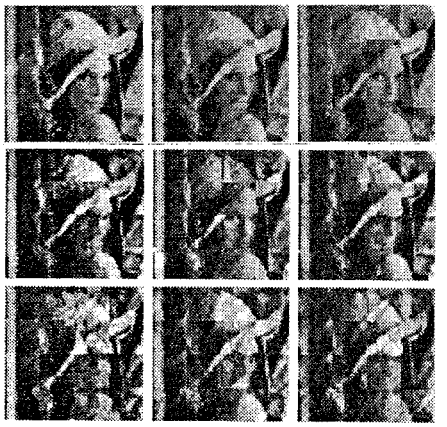

Fig. 5. Image sequence using bidirectional net..

The first image is of even lower quality than when we were using the shared weights method. This was a surprise to us, in that we expected that the bidirectional method results would lie between the standard and the shared weight method results.

This expectation was because the bidirectional method imposes a constraint on the function the neural network implements, but this is weaker than the constraint in weight sharing. We have called this a "functional symmetry" constraint in the bidirectional method, as opposed to the "weight symmetry" constraint in the weight sharing method. The degradation of image quality as we sequentially prune units is very similar to the standard back-propagation method.

## X. COMPARISON OF RESULTS

In the above sections, we have qualitatively compared the different image quality produced by each technique. We now examine the total sum of squares (tss) error measure we have used in our previous experiments.

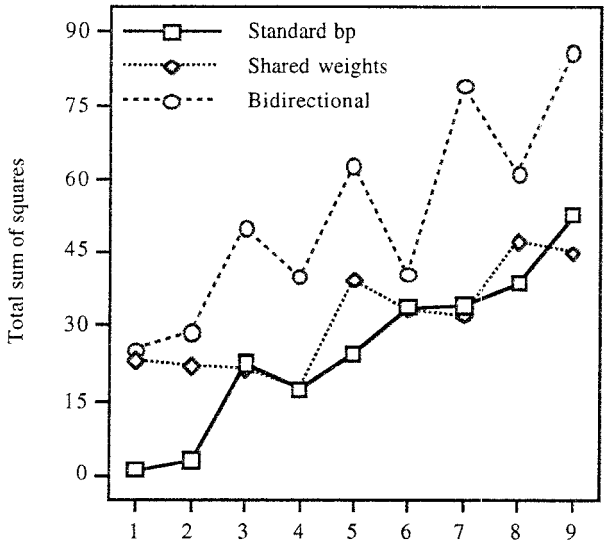


Fig. 6. Comparison of 3 techniques using TSS

The results are in some accord with our qualitative observations, in that the image quality degrades, relatively smoothly, and that the shared weights method starts with a lower quality image and ends up with a similar or slightly better image than the standard back-propagation method.

The oscillation of the bidirectional method is also apparent in the images. This is misleading, however, as the tss value for the fifth image is noticeably higher than for the fourth, and yet the quality of the fifth image is better than that of the fourth.

Hence we must remain careful in any assumption we might make that the total sum of squares values are actually an image quality measure. To further illustrate this point, we have chosen from the values in the above graph the three most similar results, which are for the sixth images.

These are repeated below for ease of comparison:


Fig. 7. Sixth images by technique

The first three images are similar, with the bidirectional training sixth image appearing slightly better, though this may be due to the artefactual smile Lena has acquired! In our previous work, we had found that the angle of the hidden unit which we removed was a better indicator of the image quality.

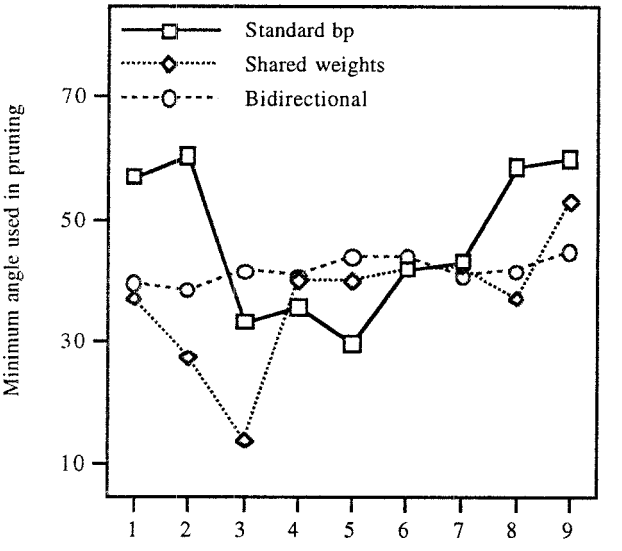


Fig. 8. Comparison of 3 techniques using Angles

The above graph shows that these values can not be said to clearly relate to the image quality process. This may be because the network was not trained for a sufficiently long period. It is also interesting to note that the bidirectional method showed very little change in the minimum angle (which is the one pruned) over the entire process.

## XI. CONCLUSION

The advantages for generalisation for both shared weights and bidirectional training probably derive from the reduction in free parameters, and the faster training of the input to hidden weights. That is, for both techniques these weights receive stronger feedback than under the standard back-propagation topology / training algorithm.

## XII. ACKNOWLEDGEMENT

The support from the Philippines Department of Science and Technology is gratefully acknowledged.

## XIII. REFERENCES


[1] G. Cottrell, P. Munro and D. Zipser, "Learning internal representations of gray scale images," *Proceedings 9th Annual Cognitive Science Society Conference*, Seattle, pp. x-y, 1987.

[2] M.K. Fleming and W. Cottrell, "Categorisation of Faces Using Unsupervised Feature Extraction," *Proceedings International Joint Conference on Neural Networks*, vol. 2, pp. 65-70, 1990.

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

[4] T.D. Gedeon and D. Harris "Finding Small Compression Layers," *Proceedings DICTA*, Melbourne, 1991.

[5] T.D. Gedeon and D. Harris "Progressive Image Compression," *Proceedings International Joint Conference on Neural Networks*, vol. 4, pp. 403-407, Baltimore, 1992.

[6] T.D. Gedeon "Indicators of Hidden Neuron Functionality: Static versus Dynamic Assessment," *Proceedings International Conference on Neural Networks and Expert Systems*, pp. 26-29, 1995.

[7] T.D. Gedeon "Indicators of Hidden Neuron Functionality: the Weight Matrix versus Neuron Behaviour," *Australasian Journal of Intelligent Information Processing Systems*, vol. 3, no. 2, pp. 1-9, 1996.

[8] A.F. Nejad and T.D. Gedeon "BiDirectional MLP Neural Networks," *Proceedings International Symposium on Artificial Neural Networks*, pp. 308-313, Taiwan, 1994.

[9] A.F. Nejad and T.D. Gedeon "Bidirectional Neural Networks Reduce Generalisation Error," in Mira, J and Sandoval, F, (eds.), *From Natural to Artificial Neural Computation*, pp. 543-550, Springer Verlag, Lecture Notes in Computer Science, vol. 930, 1995.

[10] J. Sietsma and R.F. Dow "Neural net pruning - why and how," *Proceedings International Joint Conference on Neural Networks*, vol. 1, pp. 325-333, 1988.

[11] M.C. Mozer and P. Smolenski "Using relevance to reduce network size automatically," *Connection Science*, vol. 1, pp. 3-16, 1989.

[12] B.E. Segee, BE, & Carter, MJ, "Fault Tolerance of Pruned Multilayer Networks," *Proceedings International Joint Conference on Neural Networks*, vol. 2, pp. 447-452, Seattle, 1991.

[13] D. Sanger "Contribution analysis: a technique for assigning responsibilities to hidden units in connectionist networks," *Connection Science*, vol. 1, pp. 115-138, 1989.

[14] E.D. Karnin "A simple procedure for pruning back-propagation trained neural networks," *IEEE Transactions on Neural Networks*, vol. 1, pp. 239-242, 1990.

[15] M. Hagiwara "Novel back propagation algorithm for reduction of hidden units and acceleration of convergence using artificial selection," *Proceedings International Joint Conference on Neural Networks*, vol. 1, pp. 625-630, 1990.

[16] T.D. Gedeon and D. Harris "Network Reduction Techniques," *Proceedings International Conference on Neural Networks Methodologies and Applications*, vol. 1, pp. 119-126, San Diego, 1991.

[17] T.D. Gedeon and R.P. Good "Interactive modelling of a neural network model of GDP," *Proceedings International Conference on Modelling and Simulation*, pp. 355-360, Perth, 1993.

[18] T.D. Gedeon and H. Turner "Extracting Contextual if-then Rules from a Feedforward Neural Network," *Proceedings Brazil-Japan Joint Symposium on Fuzzy Systems*, pp. 113-122, Manaus, 1994.