Neural Comput & Applic (1993)1:17–22

Neural
Computing
& Applications

Node Splitting: A Constructive Algorithm for Feed-Forward Neural Networks

Mike Wynne-Jones

82 Lower Wyche Road, Malvern Wells, UK

A constructive algorithm is proposed for feed-forward neural networks which uses node-splitting in the hidden layers to build large networks from smaller ones. The small network forms an approximate model of a set of training data, and the split creates a larger, more powerful network which is initialised with the approximate solution already found. The insufficiency of the smaller network in modelling the system which generated the data leads to oscillation in those hidden nodes whose weight vectors cover regions in the input space where more detail is required in the model. These nodes are identified and split in two using principal component analysis, allowing the new nodes to cover the two main modes of the oscillating vector. Nodes are selected for splitting using principal component analysis on the oscillating weight vectors, or by examining the Hessian matrix of second derivatives of the network error with respect to the weights.

Keywords: Adaptive architecture; Construction; Modelling; Neural networks; Pruning

1. Introduction

To achieve good generalisation in neural networks and other techniques for inferring a model from data, we aim to match the number of degrees of freedom of the model to that of the system generating the data. With too small a model we learn an incomplete solution, while too many free parameters capture individual training samples and noise.

Since the optimum size of network is seldom known in advance, there are two alternative ways of finding it. The *constructive algorithm* aims to build an approximate model, and then add new nodes to learn more detail, thereby approaching the optimum network size from below. *Pruning algorithms*, on the other hand, start with a network which is known to be too big, and then cut out nodes or weights which do not contribute to the model. A review of recent techniques [1] has led the author to favour the constructive approach, since pruning still requires an estimate of the optimum size, and the initial large networks can take a long time to train. Constructive algorithms offer fast training of the initial small networks, with the network size and training slowness reflecting the amount of information already learned. The best approach of all would be a constructive algorithm which also allowed the pruning of unnecessary nodes or weights from the network.

The constructive algorithm trains a network until no further detail of the training data can be learned, and then adds new nodes to the network. New nodes can be added with random weights, or with pre-determined weights. Random weights are likely to disrupt the approximate solution already found, and are unlikely to be initially placed in parts of the weight space where they can learn something useful, although encouraging results have been reported in this area [2]. This problem is likely to be accentuated in higher dimensional spaces. Alternatively, weights can be pre-determined by measurements on the performance of the seed network, and this is the approach adopted here. One node is turned into two, each with half the

Original manuscript received 31 January 1992

Correspondence and offprint requests to: Mr M. Wynne-Jones, 82 Lower Wyche Road, Malvern Wells WR14 4ET, UK.

output weight. A divergence is introduced in the weights into the nodes which is sufficient for them behave independently in future training without disrupting the approximate solution already found.

The discussion in this paper refers to any network or modelling technique with an internal representation of the data-generating system. The hidden nodes in a multi layer perceptron (MLP) or radial basis function network provide this internal representation, and the discussion proceeds in the context of MLP hidden nodes.

2. Node-Splitting

A network is trained using standard techniques until no further improvement on training set performance is achieved. Since we begin with a small network, we have an approximate model of the data which captures the dominant properties of the generating system but lacks detail. We now freeze the weights in the network, and calculate the updates which would be made to them, by each separate training

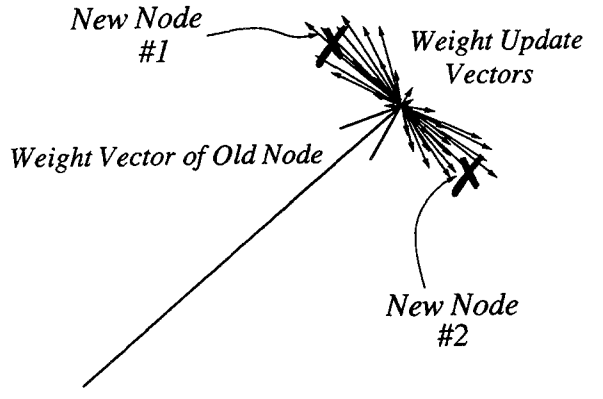


Fig. 1. A hidden node weight vector and the updates proposed by individual training patterns.

pattern. Figure 1 shows the frozen vector of weights into a single hidden node, and the scatter of proposed updates around the equilibrium position.

In the illustrated example there is one clear direction of oscillation. This might be caused by two clusters of data within a class, each trying to use the node in its own area of the input space, or by a decision boundary pulled clockwise by some patterns and anticlockwise by others. If the oscillation is strong, either in its exhibition of a clear principal direction or in comparison with other nodes in the same layer, then the node is split in two. The new nodes are placed one standard deviation of the oscillation either side of the old position. While this divergence gives the nodes a push in the right direction, allowing them to continue to diverge in later training, the overall effect on the network is small. Experiments have shown that in most cases there is very little degradation in performance immediately after the split.

The direction and size of oscillation are calculated by principal component analysis of the weight updates. By a traditional method, we are required to make a covariance matrix of the weight updates for the weight vector into each node:

$$\mathbf{C} = \sum_{p} \delta\mathbf{w}\, \delta\mathbf{w}^T \qquad (1)$$

where p is the number of patterns. The matrix is then decomposed to a set of eigenvalues and eigenvectors; the largest eigenvalue is the variance of oscillation, and the corresponding eigenvector is its direction. Suitable techniques for performing this decomposition include Singular Value Decomposition and Householder Reduction [3]. A much more suitable way of calculating the principal components of a stream of continuous measurements, such as weight updates, is iterative estimation. An estimate is stored for each required principal component vector, and the estimates are updated using each sample [4, 5]. By Oja's method [4], the scalar product of the current sample vector with each current estimate of the eigenvectors is used as a matching coefficient M. The matching coefficient is used to re-estimate the eigenvalues and eigenvectors, in conjunction with a gain term γ, which decays as the number of patterns seen increases. The eigenvectors are updated by a proportion of γM of the current sample, and the eigenvalues by γM^2. The trace (sum of eigenvalues) can also be estimated simply as the mean of the traces (sum of diagonal elements) of the individual sample covariance matrices. The algorithm is given in full in Table 1.

The principal component vectors are renormalised and orthogonalised after every few updates. This algorithm is of order n, the number of eigenvalues required, for the re-estimation, and $O(n^2)$ for the orthogonalisation; the matrix decomposition method can take exponential time, and is always much slower in practice. In the node-splitting application, only the first principal component and trace are required, making the algorithm faster still.

The principal components approach to node splitting is an improvement over Hanson's *Meiosis Networks* [6], where stochastic weights led to a natural criterion of splitting weights whose sampling distribution exhibited a large variance after prolonged training. In this scheme, the magnitude of divergence was well defined, but the direction was not. This caused most splits to slip straight back to

Table 1. Iterative eigenvalue and eigenvector estimation algorithm based on Oja's stochastic gradient ascent [4]

1.	Pick initial guesses $\mathbf{u}_0^1 \ldots \mathbf{u}_0^p$ for the columns of matrix $\mathbf{u}_0$, which is the initial estimate of the eigenvectors. They must be linearly independent; the identity matrix is a reasonable initial estimate. Define the eigenvalue estimates $\sigma_0^1 \ldots \sigma_0^p$ and the initial trace estimate τ_0.
2.	Set iteration counter $k = 0$ to start the iteration loop.
3.	For each eigenvalue - eigenvector pair i. (a) Compute the *matching coefficient* M_{k+1}^i between sample $\mathbf{v}^{k+1}$ and the eigenvalue estimates: $M_{k+1}^i = \mathbf{v}_{k+1}{}^T \mathbf{u}_k^i$. (b) Update the eigenvector estimates: $\mathbf{u}_{k+1}^i = \mathbf{u}_k^i + \gamma_{k+1}^{u_i} M_{k+1}^i \mathbf{v}_{k+1}$. (c) Update the eigenvalue estimates: $\sigma_{k+1}^i = (1-\gamma_{k+1}^{\sigma})\sigma_k^i + \gamma_{k+1}^{\sigma} M_{k+1}^i{}^2$. (d) Update the trace estimate: $\tau_{k+1} = (1-\gamma_{k+1}^{\tau})\tau_k + \gamma_{k+1}^{\tau}$ trace $(\mathbf{v}_{k+1}\mathbf{v}_{k+1}{}^T)$
4.	Perform the Gram-Schmidt orthogonalisation of vectors $\mathbf{u}_{k+1}^1 \ldots \mathbf{u}_{k+1}^p$. Normalise them to unit length, and order the eigenvalue-eigenvector pairs in descending eigenvalue order.
5.	Increment k by one, and go to step 3. Repeat for several passes through the data set until convergence is reached.

Note: The gain terms $\gamma_{k+1}^{u_i}, \gamma_{k+1}^{\sigma}$, and γ_{k+1}^{τ} decay with time, so that all patterns have equal influence on the final estimates. In practice, small-eigenvalue eigenvectors converge less quickly, and hence require larger gain than large-eigenvalue eigenvectors. The convergence can be accelerated further by causing the eigenvector gains to decay more slowly than linearly with time. Hence: $\gamma_{k+1}^{u_i} = \frac{0.5 \times 2^i}{\sqrt{k+1}}$, $\gamma_{k+1}^{\sigma} = \frac{1.0}{k+1}$, and $\gamma_{k+1}^{\tau} = \frac{1.0}{k+1}$.

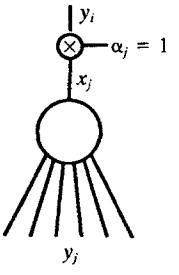
the original node's position without any improvement in the network performance.

3. Selecting Nodes for Splitting

Node splitting is carried out in the direction of maximum variance of the scatter plot of weight updates proposed by individual training samples. The hidden layer nodes most likely to benefit from splitting are those for which the non-spherical nature of the scatter plot is most pronounced. In later implementations this criterion was measured by comparing the largest eigenvalue with the sum of the eigenvalues, both these quantities being calculated by the iterative method. This is less simple in cases where there are a number of dominant directions of variance; the scatter plot might, for example, be a four dimensional disk in a ten dimensional space, and hence present the possibility of splitting one node into eight. These more complicated splits will be the subject of further research.

An alternative approach in determining the need of nodes to be split, in comparison with other nodes in the same layer, is to use the second derivatives of the network error with respect to a parameter of the nodes which is normalised across all nodes in a given layer of the network. Such a parameter was proposed by Mozer and Smolensky [7]: a multiplicative gating function is applied to the outputs of the nodes, with its gating parameter set to one. This is shown in Fig. 2. Small increments in this parameter can be used to characterise the error surface around the unity value, with the result that derivatives are normalised across all nodes in a given layer of the network. Mozer and Smolensky [7] replaced the sum squared error criterion with a modulus error criterion to preserve non-zero gradients close to the local minimum reached in training; we prefer to characterise the true error surface by means of second derivatives, which can be calculated by repeated use of the chain rule (backpropagation). Backpropagation of second derivatives has previously been reported in Le Cun *et al.* [8] and Heading [9].

Since a high curvature error minimum in the



Back-propagation: $\frac{\partial E}{\partial y_i} = \frac{\partial E dy_j \partial x_j}{\partial y_j dx_j \partial y_i} = \frac{\partial E dy_j}{\partial y_i dx_j} W_{ij}$

Sensitivity: $\frac{\partial E}{\partial \alpha_i} = \frac{\partial E dy_j \partial x_j}{\partial y_j dx_j \partial \alpha_i} = \frac{\partial E dy_j}{\partial y_j dx_j} W_{ij} Y_i$

Since $x_j = \sum_i \alpha_i y_i w_{ij}$

Fig. 2. Multiplicative gating function added to node output. With α set to 1, there is no effect on the network. Derivatives with respect to α can be used to determine the sensitivity of the network error to the node.

space of the gating parameter for a particular node indicates steep gradients surrounding the minimum, it is these nodes which exhibit the greatest instability in their weight-space position. In the weight space, if the curvature is high only in certain directions, we have the situation in Fig. 1, where the node is oscillating and is in need of splitting. If the curvature is high in all directions in comparison with other nodes, the network is highly sensitive to changes in the node or its weights, and again it will benefit from splitting.

At the other end of the scale of curvature sensitivity, a node or weight with very low curvature is one to which the network error is quite insensitive, and the parameter is a suitable candidate for pruning. This scheme has previously been used for weight pruning by Le Cun *et al.* [8], and offers the potential for an integrated system of splitting and pruning – a truly adaptive network architecture.

4. Applying the Sensitivity Measure to Input Nodes

In addition to using the gating parameter sensitivity to select nodes for pruning, Mozer and Smolensky [7] mention the possibility of using it on the input nodes to indicate those inputs to which the classification is most sensitive. This has been implemented in our system with the second derivative sensitivity measure, and applied to a large financial classification problem supplied by Thorn EMI Research. The analysis was carried out on the 78-dimensional data, and the input sensitivities varied over several orders of magnitude. The inputs were grouped into four sets according to sensitivity, and MLPs of 10 hidden nodes were trained on each subset of the data. While the low sensitivity groups failed to learn anything at all, the higher sensitivity groups quickly attained a reasonable classification rate. Identification of useless inputs leads to greatly increased training speed in future analysis, and can yield valuable economies in future data collection.

5. Evaluation in Multi Layer Perceptron Networks

Despite promising results from initial evaluations, further testing showed that the splitter technique was often unable to improve on the performance of the network used as a seed for the first split. These tests were carried out on a number of different classification problems, where large numbers of hidden nodes were already known to be required, and with a number of different splitting criteria.

Prolonged experimentation and consideration of this failure lead to the hypothesis that a split might be made to correct some misclassified patterns in one region of the input space but, owing to the long range effects of MLP decision boundaries, the changed positions of the planes might cause a much greater number of misclassifications elsewhere. These would tend to cause the newly created nodes to slip back to the position of the node from which they were created, with no overall benefit. This explanation was verified by means of some specially designed data splits. The splitter technique was then re-implemented in a gaussian mixture modelling system, which uses a network of localised receptive fields, and hence does not have the long range effects which occurred in the multi layer perceptron.

6. Implementation of the Splitter in a Gaussian Mixture Model

The Gaussian Mixtures Model [10, 11] is a clustering algorithm which attempts to model the distribution of points in a data set. It consists of a number of multivariate gaussian distributions in different positions in the input space, and with different variances in different directions. The responses of these receptive fields (bumps) are weighted and summed together; the weights are calculated to satisfy the probability density function constraint that the responses should sum to one over the data set. For the experiments on node splitting, the variance was constrained to be the same in all directions for a particular bump, leading to a model which is a sum of weighted spherical gaussian distributions of different sizes and in different positions. This greatly speeds the training of the model, but would cause insufficiency in a model where the number of nodes was not automatically determined.

The model is trained by maximisation of the likelihood of the model fitting the data, which leads to a set of learning rules for re-estimating first the weights, then the centre positions of the receptive fields, then their variances [10]. For the splitter, a small model is trained until nothing more can be learned, and the parameters are frozen. The training set is run through once more, and the updates are calculated which each pattern attempts to make to the centre position of each receptive field. The variance update need not be calculated in this phase, making it quicker than the standard training phase. The first principal component and trace of these

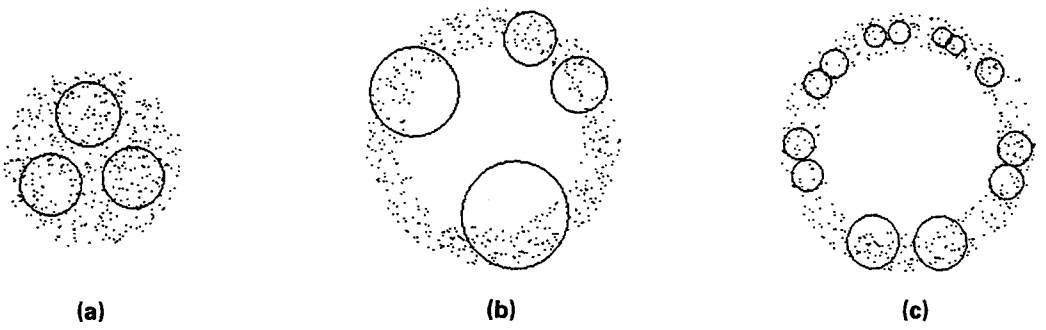


Fig. 3. Gaussian mixture model with spherical receptive fields applied to a circle and surrounding annulus. Solid ring is one standard deviation of gaussian receptive field. **(a)** Modelling uniform circular distribution; **(b)** surrounding annulus after first split; **(c)** annulus after many splits.

updates are calculated by the iterative method, and any nodes for which the principal component variance is a large proportion of the trace is split in two.

The algorithm is quick to converge, and is slowed down only a little by the overhead of computing the principal component and trace. Figure 3 shows the application of the gaussian mixture splitter to modelling a circle and an enclosing annulus; in Fig. 3(a) there is no dominant principal component direction in the data covered by the receptive field of each node (shown at one standard deviation by a circle), while in Fig. 3(b) three nodes are clearly insufficient to model the annulus, and one has just undergone a split. Figure 3(c) shows the same data set and model a little later in training after a number of splits have taken place. The technique has been evaluated on a number of other simple problems, with no negative results to date.

7. Conclusions

The splitter technique based on taking the principal component of the influences on hidden nodes in a network has been shown to be useful, in the multi layer perceptron, in only a very limited number of cases.

The split in this kind of network corresponds to a hinge in the decision boundary, which corrects the errors for which it was calculated, but usually causes far more errors in other parts of the input space. This problem does not occur in networks of localised receptive fields such as radial basis functions or gaussian mixture distributions, where it appears to work very well. Further studies will include splitting nodes into more than two, in cases where there is more than one dominant principal component, and applying node-splitting to different modelling algorithms, to gaussian mixtures in hidden markov models for speech recognition, and to adaptive meshing in finite element analysis.

The analysis of the sensitivity of the network error to individual nodes gives an ordered list which can be used for both splitting and pruning in the same network, although splitting does not generally work in the MLP. This measure has also been used in the input layer, to identify which network inputs are more or less useful in the classification task.[1]

[1] This work was carried out in the Research Initiative in Pattern Recognition, where the author represented Thorn EMI Research.

References

1. Wynne-Jones M. Constructive algorithms and pruning: Improving the multi layer perceptron. In: Vichnevetsky R, Miller JJH, editors. Proceedings of the 13th IMACS World Congress on Computation and Applied Mathematics; 1991 July; Dublin: 747–750
2. Ash T. Dynamic node creation in backpropagation networks. La Jolla (CA): Institute for Cognitive Science, UCSD; 1989 Feb. Technical Report 8901
3. Press WH, Flannery BP, Teukolsky SA, Vetterling WT. Numerical Recipes in C: The Art of Scientific Computing. Cambridge: Cambridge University Press, 1986
4. Oja E. Subspace Methods of Pattern Recognition. Section 3.2. Letchworth: Research Studies Press, 1983
5. Sanger T, Optimal unsupervised learning in a single-layer feedforward neural network. Neural Networks 1989; 2: 459–473
6. Hanson SJ, Meiosis Networks. In: Touretzky DS, editor. Advances in Neural Information Processing Systems 2. San Mateo, CA: Morgan Kaufmann, 1990 Apr: 533–541
7. Mozer MC, Smolensky P. Skeletonization: a technique for trimming the fat from a neural network. In: Touretzky DS, ed. Advances in Neural Information

Processing Systems 1. San Mateo, CA: Morgan Kaufmann, 1989 Apr: 107–115
8. Le Cun Y, Denker JS, Solla SA. Optimal Brain Damage. In: Touretzky DS, editor. Advances in Neural Information Processing Systems 2. San Mateo, CA: Morgan Kaufmann, 1990 Apr: 598–605
9. Heading AJR. An analysis of noise tolerance in multilayer perceptrons. Malvern, UK: DRA Electronics Division, Research Note SP4 122
10. Huang D, Ariki Y, Jack MA. Hidden Markov models for speech recognition, Edinburgh: Edinburgh University Press, 1990
11. Bridle JS, Cox SJ. Recnorm: Simultaneous normalisation and classification applied to speech recognition. In: Lippmann RP, Moody JE, Touretzky DS, eds. Advances in Neural Information Processing Systems 3. San Mateo, CA: Morgan Kaufmann, 1991 Sept: 234–240