Learning Synaptic Clusters for Nonlinear Dendritic Processing

Michael W Spratling*and Gillian M Hayes
Department of Artificial Intelligence,
University of Edinburgh,
5 Forrest Hill, Edinburgh. EH1 2QL. UK.

Abstract

Nonlinear dendritic processing appears to be a feature of biological neurons and would also be of use in many applications of artificial neural networks. This paper presents a model of an initially standard linear node which uses unsupervised learning to find clusters of inputs within which inactivity at one synapse can occlude the activity at the other synapses.

Keywords: Dendritic processing, Higher-order neurons, Invariance, Unsupervised learning.

This paper has not been submitted elsewhere in identical or similar form, nor will it be during the first three months after its submission to Neural Processing Letters.

*Author for correspondence and proofing; telephone: (+44) 131 650 3079, fax: (+44) 131 650 6899, email: mikes@dai.ed.ac.uk

Learning Synaptic Clusters for Nonlinear Dendritic Processing

Abstract

Nonlinear dendritic processing appears to be a feature of biological neurons and would also be of use in many applications of artificial neural networks. This paper presents a model of an initially standard linear node which uses unsupervised learning to find clusters of inputs within which inactivity at one synapse can occlude the activity at the other synapses.

Keywords: Dendritic processing, Higher-order neurons, Invariance, Unsupervised learning.

1 Introduction

Certain applications of artificial neural networks require more powerful computational mechanisms than the standard linear threshold unit. In addition, the response properties of biological neurons display nonlinearities, hence, higher-order units are also required to model such cells. For example:

- To form a topological map of egocentric space (to guide reaching movements) for a robot with movable 'eyes' and 'neck' a node within the map would need to respond, exclusively, to all combinations of gaze direction and neck rotation which bring fixation to the same point in space. Cells in the parietal area of the neocortex respond to a preferred retinal location but are modulated, multiplicatively, by eye and head position [1].
- To form an invariant representation (*e.g.* of an object under visual transformations) requires that sets of input patterns be associated together while being distinguished from other, possibly over-lapping, patterns. Cells in the inferotemporal area of the neocortex are selective to the form of visual stimulus but insensitive to location and scale of the image [2].

There is evidence to suggest that information processing takes place in the dendritic trees of biological neurons (see [3] for a review). Local thresholding of the summed input from clusters of excitatory synapses within the dendritic tree could account for the multiplicative responses of cells [4, 5]. A single such unit is thus as powerful as a multilayer network of linear threshold units, and the outputs of many nonlinear units can be combined together to act as a larger, 'virtual', nonlinear unit [5].

1.1 Sigma-Pi Units

The principal model of a nonlinear neuron is the sigma-pi unit (figure 1(a)), in which the output activation (y) is calculated as the weighted sum of the products of independent sets, or clusters, of input values (x_k) [6, 5]:

$$y = \sum_{j=1}^{M} \left(q_j \prod_{k \in cluster_j} x_k \right) .$$

Hebbian learning, between the cluster's product and the unit's (required) output, can be used to find appropriate values for the synaptic weights (q) [5, 7]. The inputs which form

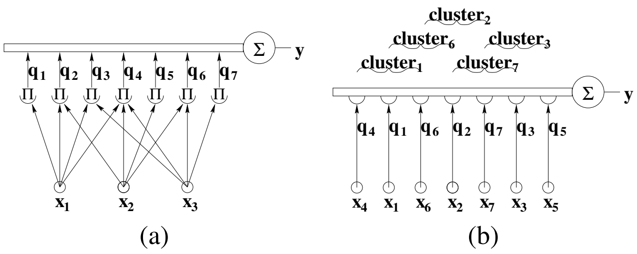


Figure 1: **Models of nonlinear neurons.** (a) A sigma-pi neuron. (b) A clusteron neuron. (c) The proposed model.

a cluster need to be predefined: either all clusters are defined to be the same size, in which case all combinations of m inputs are used (this is unsatisfactory as the appropriate cluster size may not be known before-hand, nor may all the required clusters be of the same size); alternatively all combinations of $\leq m$ inputs are used (which is computationally inhibitive, due to the number of clusters required, if the number of inputs or the maximum cluster size is large). Even using the more tractable first approach the number of parameters required can easily be unreasonably large (*e.g.* for a node to represent all clusters of size 5 from a total of 50 inputs would require over 2.1×10^6 synapses). Noise is also a problem due to the input values being multiplied together, so that noise in a single input value will affect the activation of the entire cluster.

To overcome these problems individual clusters can be represented as 'product units' which can learn arbitrary polynomial functions via gradient descent [8]. Each product unit, thus, learns which inputs are relevant to the cluster it represents. However, it is then necessary to use a multilayer network of both summing and product units as well as using supervised learning, and noise remains a problem.

1.2 Clusteron Units

The clusteron (figure 1(b)) is a more biologically inspired attempt to model dendritic cluster sensitivity [9, 3]. The output of a unit is calculated as:

$$y = \sum_{j=1}^{M} \left(q_j x_j \sum_{k=j-m}^{j+m} q_k x_k \right) .$$

The contribution of a synapse j is thus affected by the activity of neighbouring synapses (within radius m of j) on the one-dimensional dendrite. A 'cluster' is thus defined differently in this case. In contrast to sigma-pi units, where each set of inputs have a synaptic weight, in the clusteron each input has a synaptic weight and the term 'cluster' is used to refer to those inputs which can affect the activation received by a particular synapse. All clusters are of a constant, predefined, size and hence suffer from the same problems as a sigma-pi unit with the same restriction (see above), with the additional problem that false clusters are formed (*e.g.* in figure 1(b), having clustered synapses 1 and 6 and synapses 6 and 2, synapses 1 and 2 will also be in a cluster, whether or not this is justified by the input data). Noise in one input will affect the contributions of all the other synapses in the cluster so that the effect of noise is almost as strong as in a sigma-pi unit. Moreover, because there is no way of isolating a cluster from the activations of other synapses in the neighbourhood there is crosstalk and the clusteron is inherently much more noisy. The effect of neighbouring synapses is not conjunctive, and restricting the dendrite to one-dimension limits the number of clusters any one synapse can participate in.

Clustering is determined by the ordering of the synapses along the dendrite, and is formed by randomly swapping the positions of those synapses that have been least involved in firing the unit; this is little better than random search. The apparent huge reduction in the number of parameters in comparison to a sigma-pi unit (*e.g.* a node with 50 inputs requires just 50 synapses) is due to the clustering of the inputs not being explicitly represented and is paid for in the search time required before clusters are formed. The probability of finding the correct clusters becomes small very rapidly with problem size (*e.g.* the probability that within any random ordering of 50 synapses a specific cluster of 5 inputs is present is less than 2.2×10^{-5}).

2 Implementation

The drawbacks of the sigma-pi unit, discussed above, are all practical issues. It is too computationally expensive to implement a unit with all possible clusters, while using a more tractable subset of clusters requires *a priori* knowledge and presumes clusters will be of the same size. (Since the clusteron was designed to explore the effects of cluster sensitivity in models of the dendritic trees of biological neurons it does not consider these practical issues, and suffers from similar limitations.) A solution would be to learn the clustering, assuming that the learning procedure itself was tractable. A minimum set of clusters could then be found and the size of each cluster could be independent.

We introduce a model of dendritic computation which learns clustering (figure 1(c)). In this model any synapse (k) may form, part of, a cluster with any other synapse (j). The degree to which both are in the same cluster is designated by c_{jk} (the 'cluster weight'). The definition of 'cluster' used here is similar to that used in the clusteron; each input has an individual synaptic weight and other inputs in the cluster can affect the activation received by that synapse. The definition differs in that the set of inputs which form a cluster is determined by the strength of the cluster weights rather than by neighbourhood. For an individual synapse, j, all other inputs, k, which can affect the activation at synapse j are its cluster, these will be inputs for which the cluster weight $c_{jk} > \kappa$ (see below, where κ is a constant threshold). The cluster weights define global pairwise interactions between synapses, however the input at synapse j can be affected by all other inputs for which $c_{jk} > \kappa$ so that the cluster can be any size.

No *a priori* assumptions are made about the required clustering and so all cluster weights are initially zero ($c_{jk} = 0 \; \forall j, k$). In this condition it is necessary that the node act as a standard linear neuron. As clusters are learnt there should be a transition to nonlinear behaviour with the degree of nonlinearity increasing in proportion to the cluster weight (*i.e.* as c_{jk} increases the confidence that synapse k is part of a cluster with synapse j increases and the value of x_k should have increasing effect on the input at synapse j). An activation function which satisfies these requirements is:

$$y = \sum_{j=1}^{M} q_j min \left(\frac{x_1 + \kappa}{c_{j1}}, \frac{x_2 + \kappa}{c_{j2}}, \ldots \frac{x_{j-1} + \kappa}{c_{j,j-1}}, x_j, \frac{x_{j+1} + \kappa}{c_{j,j+1}}, \ldots \frac{x_M + \kappa}{c_{jM}} \right).$$

Such a node will act as a linear unit until $c_{jk} > \kappa$ for some connection. Once this condition has been reached the input to synapse j can be reduced by low activity in x_k or 'turned off' if input x_k is inactive. In all other situations the activation of the node is simply the weighted sum of the input values.

In the worse case (when $c_{jk} = 1 \; \forall k \in cluster_j$) the effect of noise is as bad as for a sigma-pi unit. However, the cluster weights are generally small, so that only much reduced values of x_k will affect the activation of synapse j. Thus input noise on any one source will only affect the activation at that synapse and not the activation received by other synapses in the cluster resulting in much smaller effect on the output activation.

While it would appear that there are a large number of cluster weights to be learnt the number of parameters required is much less than for a sigma-pi unit (*e.g.* with 50 inputs a node requires 50 synapses plus 2450 cluster weights) with the added benefit that there are no restrictions on cluster size. However, because each synapse has only one cluster weight vector, it can only participate in a single cluster. To represent patterns that require an input to participate in multiple clusters, a 'virtual' unit consisting of multiple, competing, nodes can be used. The individual nodes within a virtual unit use unsupervised competitive learning to divide the problem between them.

The explicit representation of clusters allows them to be formed without search. Since inputs likely to be part of the same cluster will be frequently coactive the clustering weights (c) and the synaptic weights (q) are learnt simultaneously (both sets of weights have zero strength initially):

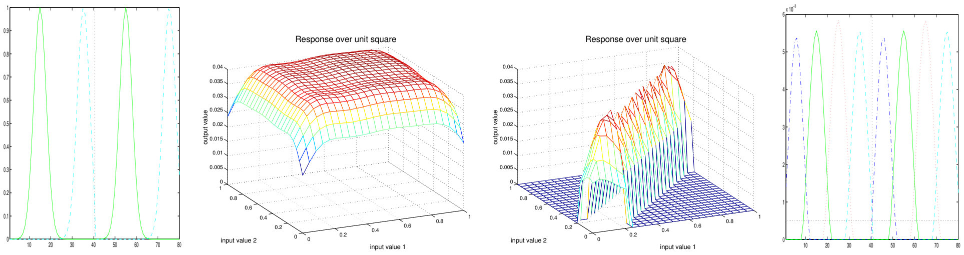
$$\Delta q_j \;= \beta \; lr(y) \otimes lr(x_j)$$

$$\Delta c_{jk} = \beta' \; lr(y) \otimes lr(x_j) \otimes lr(x_k).$$

Synaptic weight change is a function of the pre- and post-synaptic activity at that synapse. Cluster weight change is a function of the post-synaptic activity and the pre-synaptic activity at both synapses connected by that cluster weight. $lr(v) = \frac{\left(\frac{v}{\tilde{v}}-1\right)}{\sum \left|\frac{v}{\tilde{v}}-1\right|}$, this is qualitatively similar to $lr(v) = (v - \tilde{v})$, where $\tilde{v}$ is the average of the past activity of v. Hence, this learning rule is a variant of the covariance rule but constrained not to allow positive weight increases when both pre- and post-synaptic activities are below threshold ($\otimes$ is multiplication when either or both terms are positive and gives zero otherwise). For synaptic weight changes the value of $\left(\frac{v}{\tilde{v}} - 1\right)$ is constrained to be positive, whilefor cluster weight learning it can be either positive or negative. Hence, cluster weights can decreases as well as increase resulting in the formation of cluster weights being more conservative than the formation of synaptic weights. In addition, c for any synapse which has zero synaptic weight (*i.e.* $q_j = 0$) is reset so that $c_{jk} = 0 \; \forall k$. Thus, cluster weights are smaller than synaptic weights and only inputs which form strong synaptic weights can form clusters with other inputs. This prevents false clusters being found and allows the node to realign its receptive field if the input distribution changes.

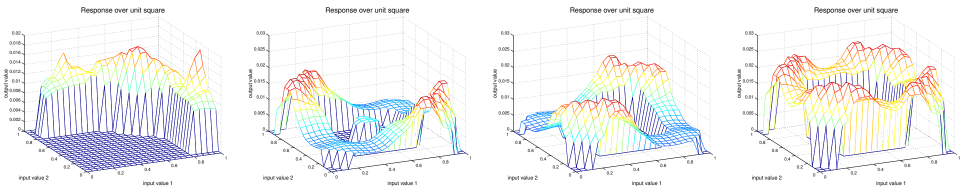
3 Results

As a simple example (applied to a continuous, real valued, input distribution) consider the problem of learning to represent (using one node) points along the leading diagonal of the unit square of the 2-dimensional plane. Each of the two coordinates of the input space were represented by a population coded array providing inputs to the node (figure 2(a)). Within each of these arrays the coordinate of a data point was represented by a Gaussian activity distribution centred on the coordinate value. Training data consisted of uniformly distributed randomly selected points along the diagonal. Since this training data spans the entire range of each coordinate all synaptic weights become equally strong, and hence, without using any dendritic interaction (by keeping $c_{jk} = 0 \; \forall j, k$) the node responds, after training, equally to any point within the unit square (figure 2(b)). However, when the learning of cluster weights is allowed they are formed between synapses representing corresponding coordinate values (figure 2(d)). Hence, a node using dendritic interaction responds to test data along the leading diagonal only (figure 2(c)). Similar results can be generated for learning to represent arbitrary curves. Figure 3(a) shows the response of a node trained to represent the circumference of a quadrant of a circle. To represent the entire circumference of a circle would require a synapse representing a particular x-coordinate (or y-coordinate) to form clusters with inputs representing two distinct y-coordinate (x-coordinate) values. As mentioned above, since each synapse has only a single cluster weight vector it can only take part in a single cluster, but the output of multiple, competing, nodes can act as a larger virtual unit. Figures 3(b) and 3(c) show the response of two nodes which compete to represent the circumference of a circle. It can be seen that each node has come to represent diagonally opposite quadrants of the circle, where each synapse need only take part in a single cluster. The sum of these two responses represents the response of a virtual unit composed of these two nodes and is shown in figure 3(d).



(a) (b) (c) (d)

Figure 2: **Learning to represent points along the leading diagonal of a 2-dimensional input space.** (a) Training data consisted of two population coded input arrays, representing x- and y-coordinates on the 2d plane. This data was supplied to the node such that the first 80 synapses received the population coded representation of the x-coordinate, and the second 80 synapses received the y-coordinate value. Input activations representing two points along the diagonal are shown (each in a different line style). (b) and (c) The response of the node after training. The strength of the response of the node which is caused by input data representing points across the unit square, is shown for (b) a linear node, and, (c) a nonlinear node. (d) The cluster weights that have been learnt for four synapses (cluster weights for each synapse are in a different line style). The horizontal line shows the threshold (κ).



(a) (b) (c) (d)

Figure 3: **Learning to represent points along a curved line of a 2-dimensional input space.** Each figure shows the response of a nonlinear node after training. The strength of the response of the node which is caused by input data representing points across the unit square is shown. (a) For a single node trained with data along the circumference of a quadrant of a circle of radius 1 and centre at the origin. (b) and (c) The response of two, competing nodes, trained with data along the circumference of a circle. (d) The sum of the responses in (b) and (c) representing the response of the virtual nonlinear unit composed of these two nodes.

As a second example (applied to a discontinuous, binary valued, input space) consider the problem of trying to distinguish (using a single layer of two nodes) horizontal and vertical lines on an 8 by 8 grid. Learning is unsupervised with the two nodes competing to represent inputs. Since there is no overlap between lines of the same orientation an extra constraint is required to cause them to be classified together. One such constraint is to increase the probability that contiguous inputs are of the same orientation (figure 4(a)), and to use a learning rule that biases ('sensitises') the previously active node to remain active. Although a similar method has previously been proposed as a mechanism for learning invariance to input translations [10] it was used with (two layers of) linear units in which case it is only possible to learn invariances if there is no overlap between patterns

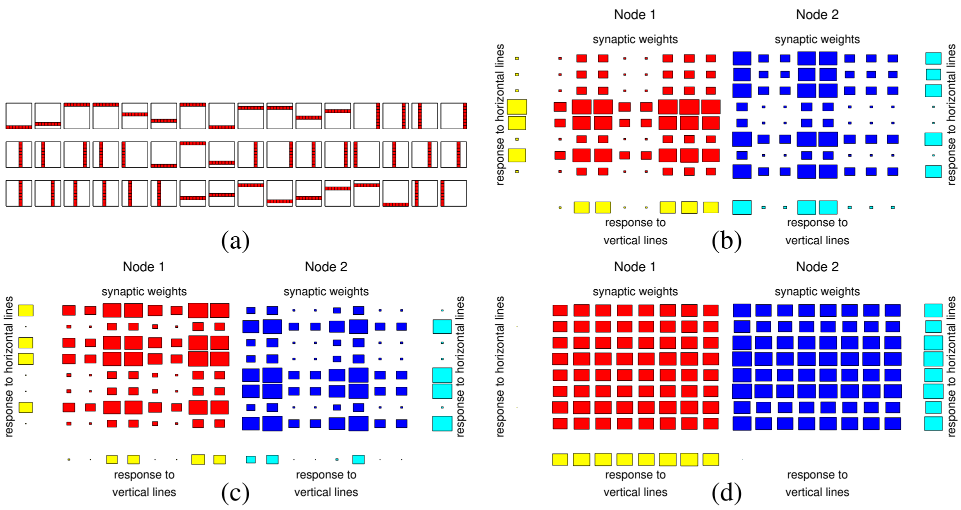


Figure 4: **Learning to represent horizontal and vertical lines.** Two nodes receive input from an 8 by 8 array of binary inputs. (a) Training data consists of one only of the 16 possible horizontal or vertical lines being active at any one time with short sequences of lines at the same orientation (a new, random, orientation is chosen with probability 0.25 at each iteration). The strengths of the synaptic weights connecting grid points to each node are illustrated by squares proportional in size to the weight. Similarly, the strength of the (mean) activation of each node on presentation of each input pattern is illustrated by squares proportional in size to this activation. For (b) linear nodes with sensitisation, (c) nonlinear nodes without sensitisation, and, (d) nonlinear nodes with sensitisation.

[11] and the node will incorrectly respond if the input consists of partial patterns from different translational positions. Hence, such a method, with linear units, could not solve the horizontal/vertical lines problem. Instead, the competition between the nodes causes them to divide the weight array arbitrarily so that each represents some horizontal and some vertical lines (figure 4(b)). A similarly arbitrary division of the input space also occurs when dendritic interaction is used but without sensitisation of the competition (figure 4(c)). Since both sets of lines activate all points in the grid a node representing horizontal lines would have equal weights to all inputs, and a node representing vertical lines would have a similar weight array. Thus, the weight space alone can not be used to distinguish the line's orientation. However, using dendritic interaction together with sensitisation nodes can learn to exclusively represent either vertical or horizontal lines (figure 4(d)), both nodes learn similar weight arrays, but the cluster weights distinguish between line orientations.

Using supervised learning (as opposed to unsupervised learning, as used above) neurons which learn synaptic clusters can be applied to learning any nonlinear function that other nonlinear nodes or multilayer networks of linear threshold units can be applied to. In cases where multiple units act as a virtual unit, supervision specifies the required output of the virtual unit, rather than the outputs of individual nodes, and the nodes within the virtual unit compete (as in unsupervised learning) to divide the problem between them. A standard problem is to detect the parity of a binary input vector. Since the synaptic weights in this model are constrained not to be negative, it is necessary to use an encoding of the input vector containing each bit and its complement (such an encoding would be required to solve this problem by any other neural network with the same restriction on synaptic weights). The two bit parity problem (or XOR problem) can be solved using a single nonlinear node. A linear unit, trained in the same way, is unable to fully solve this problem. The mean response, after training, for nodes trained on this problem is shown in table 1. Two linear units are still unable to solve the XOR problems, while two nonlinear units (acting as a virtual unit) produce a good solution (table 2). For parity problems involving more bits each input needs to form a part of more than one cluster, so that solutions can only be found using multiple nodes acting as a larger virtual unit. Hence, using one unit, either linear or nonlinear, fails to solve the 3 bit parity problem (table 3, the results for linear and nonlinear nodes are identical as the conflicting requirements for the cluster weights in the nonlinear case cause all cluster weights to remain below κ). The response of two competing nodes to patterns of three bits are shown in table 4, together with the summed response representing the activation of the virtual node formed by these two nodes. It can be seen that a virtual unit of two nonlinear nodes has begun to solve the problem, while a virtual unit of linear units fails.

input pattern		parity	response of one nonlinear unit	response of one linear unit
0	0	0	*0.0*	*0.0134*
0	1	1	*0.0133*	*0.0133*
1	0	1	*0.0135*	*0.0135*
1	1	0	*0.0*	*0.0134*

Table 1: **Learning XOR problem with one node.** The first two columns give the input patterns. The last two columns give the mean mean output responses of nodes after training. The fourth column is the response of a nonlinear node. The fifth column is the response of a linear node.

input pattern		parity	response of two nonlinear units			response of two linear units		
0	0	0	0.0	0.0	*0.0*	0.0099	0.0083	*0.0182*
0	1	1	0.0	0.0224	*0.0224*	0.0	0.0245	*0.0245*
1	0	1	0.0230	0.0	*0.0230*	0.0242	0.0	*0.0242*
1	1	0	0.0	0.0	*0.0*	0.0089	0.0094	*0.0183*

Table 2: **Learning XOR problem with two nodes.** The first two columns give the input patterns. The last six columns give the mean output responses of nodes after training. The fourth and fifth columns are the responses of two nonlinear nodes, and the sixth column is the summed response of these two nodes. The seventh and eighth columns are the responses of two linear nodes, and the ninth column is the summed response of these two nodes.

input pattern			parity	response of one nonlinear unit	response of one linear unit
0	0	0	0	*0.0581*	*0.0581*
0	0	1	1	*0.0588*	*0.0588*
0	1	0	1	*0.0582*	*0.0582*
0	1	1	0	*0.0583*	*0.0583*
1	0	0	1	*0.0584*	*0.0584*
1	0	1	0	*0.0591*	*0.0591*
1	1	0	0	*0.0584*	*0.0584*
1	1	1	1	*0.0587*	*0.0587*

Table 3: **Learning the 3 bit parity problem with one node.** The first three columns give the input patterns. The last two columns give the mean output responses of nodes after training. The fifth column is the responses of a nonlinear nodes. The sixth column is the responses of a linear node.

input pattern			parity	response of two nonlinear units			response of two linear units		
0	0	0	0	0.0019	0.0407	*0.0425*	0.0	0.0710	*0.0710*
0	0	1	1	0.0	0.0732	*0.0732*	0.0	0.0718	*0.0718*
0	1	0	1	0.0	0.0719	*0.0719*	0.0	0.0708	*0.0708*
0	1	1	0	0.0059	0.0374	*0.0433*	0.0040	0.0646	*0.0686*
1	0	0	1	0.0716	0.0	*0.0716*	0.0705	0.0	*0.0705*
1	0	1	0	0.0407	0.0022	*0.0429*	0.0709	0.0	*0.0709*
1	1	0	0	0.0407	0.0019	*0.0425*	0.0708	0.0	*0.0708*
1	1	1	1	0.0721	0.0	*0.0721*	0.0711	0.0	*0.0711*

Table 4: **Learning the 3 bit parity problem with two nodes.** The first three columns give the input patterns. The last six columns give the mean output responses of nodes after training. The fifth and sixth columns are the responses of two nonlinear nodes, and the seventh column is the summed response of these two nodes. The eighth and ninth columns are the responses of two linear nodes, and the tenth column is the summed response of these two nodes.

4 Conclusions

This paper suggests that learning clusters of inputs, for nonlinear dendritic processing, provides a solution to some of the practical limitations of other models. The resulting neurons are no more powerful than other models with nonlinear input functions (nor more powerful than a multilayer network of linear threshold units). However, this method does have the advantage of being more tractable than other implementations of nonlinear neurons. It provides unsupervised, and efficient, learning of dendritic clusters, without the prespecification of cluster size; allowing clusters to be of independent and arbitrary size. False clusters are not formed. Inactive synapses can block the activity of all other synapses in the cluster, but otherwise clustering has no effect on the activity. This allows the method to work with population coded inputs, and makes it relatively insensitive to noise in the input values.

While the spatial ordering of the synapses within the dendritic tree may be critical for a biological neuron, it does not mean that an artificial neuron must also use the ordering of the synapses to represent clustering. Instead, we have used 'weights' to represent the degree of clustering between synapses. All sense of locality is thus lost, and this model does not attempt to represent local interactions within a fixed dendritic tree. In this sense it is not a biologically plausible model of nonlinear dendritic processing. However, for the static ordering of the synapses on a dendritic tree to be formed it must be learnt. The clustering weights of this model might therefore be interpreted as representing the mutual attraction of axons innervating a 3-dimensional dendritic tree.

References

[1] R. A. Anderson, G. K. Essich, and R. M. Siegal. Encoding of spatial location by posterior parietal neurons. *Science*, 230:456–8, 1985.

[2] M. J. Tovee, E. T. Rolls, and P. Azzopardi. Translation invariance in the responses to faces of single neurons in the temporal visual cortical areas of the alert macaque. *Journal of Neurophysiology*, 72(3):1049–60, 1994.

[3] B. W. Mel. Information processing in dendritic trees. *Neural Computation*, 6:1031–85, 1994.

[4] J. A. Feldman and D. H. Ballard. Connectionist models and their properties. *Cognitive Science*, 6:205–54, 1982.

[5] B. W. Mel. The sigma-pi column: a model of associative learning in cerebral cortex.

Technical Report CNS Memo 6, Computation and Neural Systems Program, California Institute of Technology, 1990.

[6] D. E. Rumelhart, J. L. McClelland, and The PDP Research Group, editors. *Parallel Distributed Processing: Explorations in the Microstructures of Cognition. Volume 1: Foundations*. MIT Press, 1986.

[7] B. W. Mel and C. Koch. Sigma-pi learning: on radial basis functions and cortical associative learning. In D. S. Touretzsky, editor, *Advances in Neural Information Processing Systems 2*, pages 474–81. Morgan Kaufmann, 1990.

[8] R. Durbin and D. E. Rumelhart. Product units: a computationally powerful and biologically plausible extention to backpropagation networks. *Neural Computation*, 1: 133–42, 1990.

[9] B. W. Mel. The clusteron: toward a simple abstraction for a complex neuron. In J. E. Moody, S. J. Hanson, and R. P. Lippmann, editors, *Advances in Neural Information Processing Systems 4*, pages 35–42. Morgan Kaufmann, 1992.

[10] P. Földiák. Learning invariance from transformation sequences. *Neural Computation*, 3:194–200, 1991.

[11] S. Becker. Learning to categorize objects using temporal coherence. In S. J. Hanson, J. D. Cowan, and C. L. Giles, editors, *Advances in Neural Information Processing Systems 5*, pages 361–8. Morgan Kaufmann, 1993.