5

Kohonen Self-Organising Networks

5.1 INTRODUCTION

So far we have looked at algorithms that rely on supervised learning techniques. In this chapter we will explore unsupervised learning methods, and in particular Kohonen's self-organising maps. As we have seen with back propagation techniques, supervised learning relies on an external training response (the desired response of the network) being available for each input from the training class. This technique is very useful, and in some ways relates to the human learning process. However in many applications, it would be more beneficial if we could ask the network to form its own classifications of the training data. To do this we have to make two basic assumptions about the network; the first is that class membership is broadly defined as input patterns that share *common features*, the other is that the network will be able to identify common features across the range of input patterns. Kohonen's self-organising map is one such network that works upon these assumptions, and uses unsupervised learning to modify the internal state of the network to model the features found in the training data. We shall explore this idea fully by looking closely at Kohonen's learning algorithm (and the Grossberg ART network in Chapter 7).

5.1.1 The Self-Organisation Concept

Kohonen—a Professor of the Faculty of Information Sciences, University of Helsinki—has worked steadily in the area of neural networks for many years, long before the current surge of interest

erupted in the mid 1980's. He has worked extensively with concepts of associative memory and models for neurobiological activity. His work is characterised by a drive to model the self-organising and adaptive learning features of the brain.

Neurobiologists have long since established that localised areas of the brain, particularly across the cerebral cortex, perform specific functions. Examples might be speech, vision or motion control, each of which can be identified as regions of intense local activity in the brain. More recently, evidence has also been found that suggests even the localised regions may contain further structures which represent the internal mappings of response from sensory organs. A good example is found in the auditory cortex region.

In the auditory cortex it is possible to distinguish a spatial ordering of the neurons which reflects the frequency response of the auditory system. The ordering of the cells within the auditory cortex region trace an almost logarithmic scale of frequency. Low frequencies will generate responses at one end of the cortex region, high frequencies at the opposite extreme. There are arguments for and against the idea of internal neuron mappings of this kind. Those who oppose it argue the case for the so called "Grandmother cell". This idea suggests that individual neurons in the brain are coded to represent a specific concept, for example a specific cell could be responsible for the task of identifying Grandmother. This argument would appear to have little biological justification however—cells in the brain die off at a rather alarming rate for those of us who have passed the first score of our "three score and ten" (typical estimates put the figure at 25000 cells a day). Having an encoding scheme that maps concepts to unique cells cannot be expected to remain reliable with the typical rates of decay of neurons.

The ideas of self-organisation were proposed as early as 1973 by von der Malsburg and followed up in the mid 70's with computer models for self-organisation, by Willshaw and von der Malsburg. Their work was particularly biologically motivated—based on the development of selectively sensitive neurons (i.e. to light intensity and edge orientation) in the visual cortex region. As we discussed earlier in the book, biological learning or adaptation is a chemical

process that modifies the effectiveness of the synaptic connections at the input to the neuron cell. There is little doubt that much of the high-level structure is genetically placed and fixed from birth, but this does not account for our continued experience of learning. There is no simple answer to this question—the biological and physiological issues raised are complex. We recently heard the quote "If the brain was simple enough to be understood—we would be too simple to understand it!". Minsky in his book *Society of Mind* elaborates on this complexity and draws the conclusions that the human brain has over 400 specialised architectures, and is equivalent in capacity to about 200 Connection Machines (Model CM–2). (The book is well worth a read if the neurobiological area of this subject interests you.) The outcome of Kohonen's investigations has been the derivation of a neural network learning algorithm based on these concepts of self-organisation, with very plausible extensions to the biological realm.

5.1.2 An Overview

It has been postulated that the brain uses spatial mapping to model complex data structures internally. Kohonen uses this idea to good advantage in his network because it allows him to perform data compression on the vectors to be stored in the network, using a technique known as *vector quantisation.* It also allows the network to store data in such a way that spatial or topological relationships in the training data are maintained and represented in a meaningful way.

Data compression means that multi-dimensional data can be represented in a much lower dimensional space. Much of the cerebral cortex is arranged as a two-dimensional plane of interconnected neurons but it is able to deal with concepts in much higher dimensions. The implementations of Kohonen's algorithm are also predominantly two dimensional. A typical network is shown in figure 5.1. The network shown is a one-layer two-dimensional Kohonen network. The most obvious point to note is that the neurons are not arranged in layers as in the multilayer perceptron (input, hidden, output) but rather on a flat grid. All inputs connect to every node in the network. Feedback is restricted to lateral interconnections to immedi-

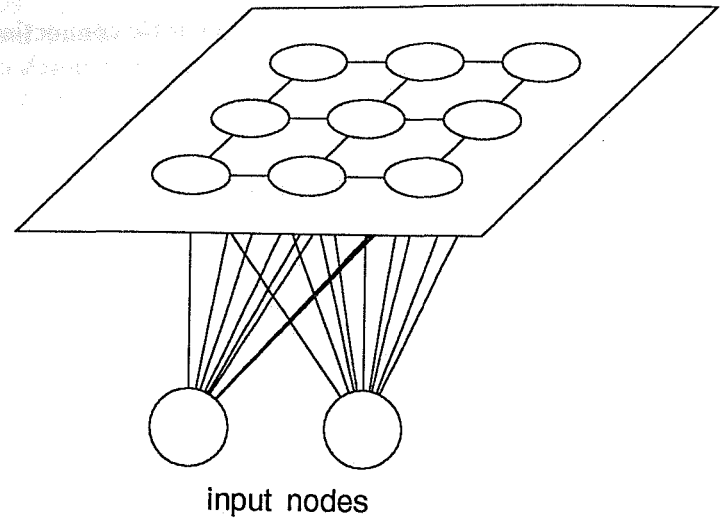


Figure 5.1 A Kohonen feature map. Note that there is only one layer of neurons and all inputs are connected to all nodes.

ate neighbouring nodes. Note too that there is no separate output layer—each of the nodes in the grid is itself an output node.

5.2 THE KOHONEN ALGORITHM

The learning algorithm organises the nodes in the grid into local neighbourhoods that act as feature classifiers on the input data. The topographic map is autonomously organised by a cyclic process of comparing input patterns to vectors "stored" at each node. No training response is specified for any training input. Where inputs match the node vectors, that area of the map is selectively optimised to represent an average of the training data for that class. From a randomly organised set of nodes the grid settles into a feature map that has local representation and is self-organised. The algorithm itself is notionally very simple.

Kohonen Network Algorithm

1. Initialise network

Define $w_{ij}(t)\,(0 \leq i \leq n-1)$ to be the weight from input i to node j at time t. Initialise weights from the n inputs to the nodes to small random values. Set the initial radius of the neighbourhood around node j, $N_j(0)$, to be large.

2. Present input

Present input $x_0(t), x_1(t), x_2(t), \ldots, x_{n-1}(t)$, where $x_i(t)$ is the input to node i at time t.

3. Calculate distances

Compute the distance d_j between the input and each output node j, given by

$$d_j = \sum_{i=0}^{n-1} (x_i(t) - w_{ij}(t))^2$$

4. Select minimum distance

Designate the output node with minimum d_j to be $j*$.

5. Update weights

Update weights for node $j*$ and its neighbours, defined by the neighbourhood size $N_{j*}(t)$. New weights are

$$w_{ij}(t+1) = w_{ij}(t) + \eta(t)(x_i(t) - w_{ij}(t))$$

For j in $N_{j*}(t)$, $\quad 0 \leq i \leq n-1$

The term $\eta(t)$ is a gain term $(0 < \eta(t) < 1)$ that decreases in time, so slowing the weight adaption. Notice that the neighbourhood $N_{j*}(t)$ decreases in size as time goes on, thus localising the area of maximum activity.

6. Repeat by going to 2.

In summary:

- Find the closest matching unit to a training input
- Increase the similarity of this unit, and those in the neighbouring proximity, to the input.

5.2.1 Biological Justification

Is there any biological justification for such a learning rule? As we have seen already, Kohonen has based most of his work on close studies of the topology of the brain's cortex region, and indeed there would appear to be a good deal of biological evidence to support this idea.

We have seen in previous chapters that activation in a nervous cell is propagated to other cells via axon links (which may have an inhibitory or excitatory effect at the input of another cell). However, we have not considered the question of how the axon links are affected by lateral distance from the propagating neuron. A simplified yet plausible model of the effect is illustrated by the Mexican hat function shown in figure 5.2.

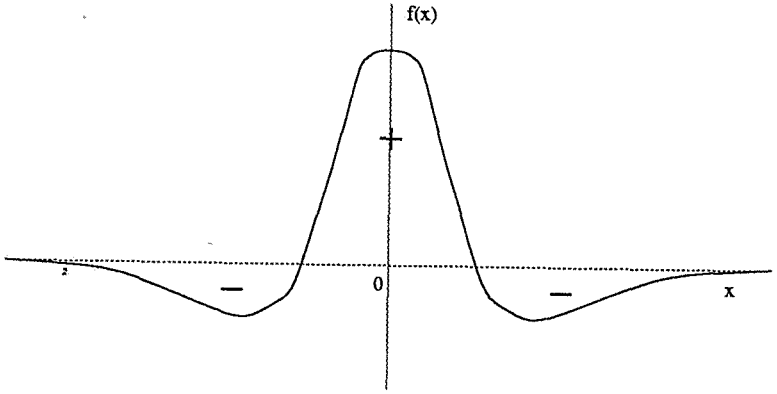


Figure 5.2 The Mexican hat function describes the effect of lateral interconnection.

We can see that cells physically close to the active cell have the strongest links. Those at a certain distance actually switch to inhibitory links. It is this phenomenon to which Kohonen attributes to the development (at least in part) of localised topological mapping in the brain. As we shall see, he has modelled this effect by using only locally interconnected networks and restricting the adaption of weight values to localised "neighbourhoods".

Much of the popularity of this paradigm could be attributed to the fact that it has a very accessible and "natural" feel to it, as we shall hopefully see when we expand the algorithm. Reading through the algorithm, we can see that the learning rule is not complicated. There are no troublesome derivatives to be calculated, as in gradient descent methods. Initially all the connections from the inputs to the nodes are assigned small random weight values. Each node will thus have a unique weight vector, the dimensionality of which is defined by the number of components in the input vector. During the learning cycle, a set of training patterns (a representative subset of the full data set) is shown to the network. The action of the network under the stimuli of these training inputs can be compared to a "winner-take-all" function. A comparison is made between each input pattern, as it is presented, and the weight vectors—the node with the weight vector closest to the input pattern is selected as the "winner". The winning node "claims" the input vector and modifies its own weight vector to align it with the input. The node has now become sensitive to this particular training input and will provide a maximum response from the network if it is applied again after training is completed.

We can see from the algorithm that the nodes in the neighbourhood N_c of the winning node are also modified. The reason for this is that the network is trying to create *regions* that will respond to a spread of values around the training input. The nodes around the winning node are given a similar alignment, and over the course of the training cycle this settles to an "average" representation of that class pattern. As a consequence, vectors that are close spatially to the training values will still be classified correctly even though the network has not seen them before. This demonstrates the generalisation properties of the network.

The two most central issues to adaptive self-organising learning in a Kohonen network are the weight adaption process and the concept of topological neighbourhoods of nodes. Both of these ideas are very different from the neural networks we have discussed so far, so our description of the workings of the Kohonen network will be based around these key themes.

5.3 WEIGHT TRAINING

As we have already mentioned, there is no derivative process involved in adapting the weights for the Kohonen network. Referring to the algorithm again, we can see that the change in the weight value is proportional to the difference between the input vector and the weight vector:

$$w_{ij}(t+1) = w_{ij}(t) + \eta(t)(x_i(t) - w_{ij}(t))$$

where w_{ij} is the ith component of weight vector to node j, for j in the neighbourhood $N_{j*}(t)$ $(0 \leq i \leq n-1)$.

The unit of proportionality is $\eta(t)$, the learning rate coefficient, where $0 < \eta(t) < 1$. This term decreases the adaption rate with time (where by "time" we mean the number of passes through the training set). We can visualise the training cycle as having two stages. The first stage is creating some form of topological ordering on the map of randomly orientated nodes. The training process attempts to cluster the nodes on the topological map to reflect the range of class types found in the training data. This will be a coarse mapping, where the network is discovering how many classes the map must eventually identify, and where they should lie in relation to each other on the map. These are large scale changes to the orientation of the nodes on the map, so the adaption rate is kept high ($\eta > 0.5$) to allow large weight modifications and hopefully settle into an approximate mapping as quickly as possible. Once a stable coarse representation is found, the nodes within the localised regions of the map are fine-tuned to the input training vectors. To achieve this fine-tuning much smaller changes must be made to the weight vectors at each node, so the adaption rate is reduced as training progresses. Typically the fine-tuning stage will take between 100 and 1000 times as many steps as finding the coarse representation, if a low value of η is used.

Each time a new training input is applied to the network the winning node must first be located; this identifies the region of the feature map that will have its weight values updated. The winning node is categorised as the node that has the closest matching weight vector to the input vector, and the metric that is used to measure

the similarity of the vectors is the Euclidean distance measure. We discussed this metric earlier in Chapter 2. There are, however, a few subtleties to note in implementing the technique in the Kohonen network. The Euclidean norm of a vector is a measure of its magnitude. However, we are not so much interested in the magnitude of the vectors as in finding out how they are orientated spatially. In other words, we will describe two vectors as being similar if they are pointing in the same direction, regardless of their magnitude. The only way that we can ensure that we are comparing the orientation of two vectors, using the Euclidean measure, is to first make sure that all the weight vectors are normalised. Normalising a vector reduces it to a *unity length* vector by dividing it by its magnitude—for a set of vectors in Euclidean space this means that each vector will retain its orientation but will be of a fixed length, regardless of its previous magnitude. The comparison of the weight vectors and the input vector will now be concerned only with the orientation, as required. Another useful advantage of normalising the vectors is that it reduces the training time for the network, because it removes one degree of variability in the weight space. Effectively that means that the weight vectors start in an orientation that is closer to the desired state, thus reducing some of the reorientation time during the training cycle.

5.3.1 Initialising the Weights

A note of caution may be inserted at this point concerning the initialisation of the weight vectors. So far we have suggested that on start-up, the network weights should be set to small, normalised random values. However, this is an over-simplification because if the weight vectors are truly randomly spread, the network may suffer non-convergent or very slow training cycles. The reason for this can be explained fairly intuitively. Typically the input training vectors will fall into clusters over a limited region of the pattern space, corresponding to their class (at least it is hoped that they will, else training will be a difficult process). If the weight vectors, stored at the nodes in the network, are randomly spread then the situa-

tion could quite easily arise where many of the weight vectors are in a very different orientation to the majority of the training inputs. These nodes will not win any of the best-match comparisons and will remain unused in forming the topological map. The consequence of this is that the neighbourhoods on the feature map will be very sparsely populated with trainable nodes, so much so that there may not be enough usable nodes to adequately separate the classes. This will result in very poor classification performance due to the inability of the feature map to distinguish between the inputs.

The optimum distribution for the initial weights is one that gives the network starting clues as to the number of classes in the training data and the likely orientation that each one will be found, but considering that this is often the very information that we are expecting the network to find for us it is a rather impractical proposition. There are, however, methods to approximate such a distribution.

One method is to initialise all the weights so that they are normal and coincident (i.e. with the same value). The training data is modified so that, in the early stages of the training cycle, the vectors are all lumped together in a similar orientation to the start-up state of the nodes. This gives all the nodes in the feature map the same likelihood of being close to the input vectors, and consequently being included in the coarse representation of the map. As training progresses the inputs are slowly returned to their original orientation, but because the coarse mapping is already defined by this stage, the nodes in the feature map will simply follow the modifications made to the input values. A similar technique adds random noise to the inputs in the early stages of training in an attempt to distribute the vectors over a larger pattern space, and thus utilise more nodes.

It is also possible to attach a threshold value to each node, which "monitors" the degree of success or failure that a node has in being selected as best-match. If a node is regularly being selected, it will temporarily have its threshold raised. This reduces its chance of being voted best-match and allows redundant nodes to be used in forming the features of the map.

The most often used technique, however, and the one quoted by Kohonen, is one that we have already mentioned in passing—that of

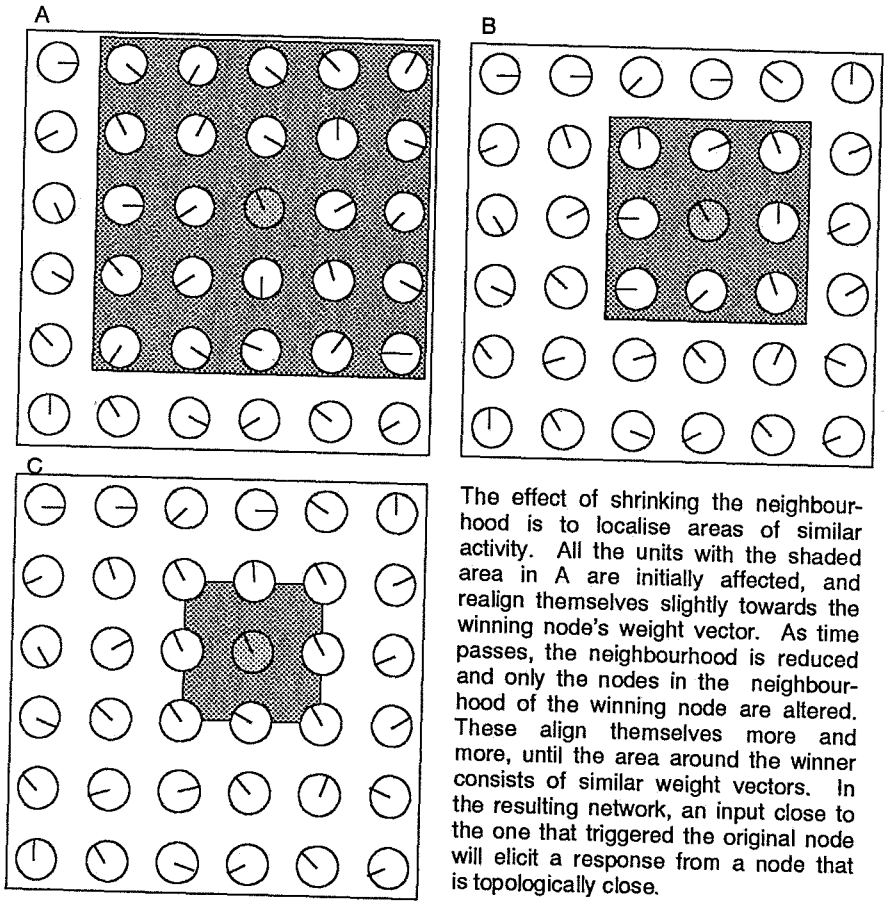
local neighbourhoods around each node. We will now explain how this maximises the use of all the nodes in the network and promotes topological grouping of nodes.

5.4 NEIGHBOURHOODS

In order to model the Mexican hat function for the lateral spread of activation in interconnected nodes, Kohonen introduces the idea of topological neighbourhoods. This is a dynamically changing boundary that defines how many nodes surrounding the winning node will be affected with weight modifications during the training process. Initially each node in the network will be assigned a large neighbourhood (where "large" can imply every node in the network). When a node is selected as the closest match to an input it will have its weights adapted to tune it to the input signal. However, all the nodes in the neighbourhood will also be adapted by a similar amount. As training progresses the size of the neighbourhood is slowly decreased to a predefined limit. To appreciate how this can force clusters of nodes that are topologically related, consider the sequence of diagrams shown in figure 5.3 that represents the topological forming of the feature clusters during a training session. For clarity, we shall show the formation of just one cluster which is centred about the highlighted node.

In A, the network is shown in its initialised state, with random weight vectors and large neighbourhoods around each node. The arrows within each node can be thought of as a spatial representation of the orientation of each node's weight vector. Training commences as previously described; for each training input the best-match node is found, the weight change is calculated and all the nodes in the neighbourhood are adjusted.

In B we can see the network after many passes through the training set. The highlighted region of the map is beginning to form a specific class orientation based around the highlighted node. The neighbourhood size has also shrunk so that weight modifications now have a smaller field of influence.



The effect of shrinking the neighbourhood is to localise areas of similar activity. All the units with the shaded area in A are initially affected, and realign themselves slightly towards the winning node's weight vector. As time passes, the neighbourhood is reduced and only the nodes in the neighbourhood of the winning node are altered. These align themselves more and more, until the area around the winner consists of similar weight vectors. In the resulting network, an input close to the one that triggered the original node will elicit a response from a node that is topologically close.

Figure 5.3 Training a localised neighbourhood.

The fully trained network is shown in C. The neighbourhoods have shrunk to a predefined limit of four nodes, and the nodes within the region have all been adapted to represent an average spread of values about the training data for that class.

The training algorithm will produce clusters for all the class types found in the training data. The ordering of the clusters on the map, and the convergence times for training are dependent on the way the training data is presented to the network. Once the network has self-organised the internal representation the clusters on the feature map can be labelled to indicate their class so that the network can be used to classify unknown inputs. Note that the network forms the internal features without supervision, but the classification labelling must be done by hand, once the network has been fully trained.

5.5 REDUCING THE NEIGHBOURHOOD

We have already stressed that the neighbourhood size is reduced with time during the training sequence. But how quickly do we reduce it and to what final size? Unfortunately there are no hard and fast rules for adaptive training algorithms of this nature and some experimentation will be required in individual applications. However Kohonen does stress that his method is not one that is brittle—that is, small changes in system parameters do not reflect gross divergence of training results—and also suggests some rules of thumb as a starting point for intuitive tweaking!

We have explained that the adaption rate must be reduced during the training cycle so that weight changes are made more and more gradual as the map develops. This ensures that clusters form accurate internal representations of the training data as well as causing the network to converge to a solution within a predefined time limit. In typical applications Kohonen suggests that the adaption rate be a linearly decreasing function with the number of passes through the training set.

Training is effected not only by the adaption rate and the rate at which the neighbourhood is reduced, but also by the shape of the neighbourhood boundary. The example we used earlier, in figure 5.3,

only discussed the possibility of using a square neighbourhood—however, that is not to say that we cannot define a circular or even a hexagonal region, and these may provide optimal results in some cases. As with the adaptation rate, however, it is preferable to start with neighbourhoods fairly wide initially and allow them to decrease slowly with the number of training passes.

5.5.1 Point Density Functions

For those who prefer a more mathematical definition of what is happening during the training cycle we can explain the clustering phenomenon using probability density functions. A probability density function is a statistical measure that describes the data distribution in the pattern space. For any given point in the pattern space, the probability density function will define a value for the likelihood of finding a vector at that point. Given a pattern space with a known probability density function (i.e. we know how the patterns are spread across the pattern space) it can be shown that the map will order itself such that the point density of the nodes in the feature map will tend to approximate the probability density function of the pattern space (if a representative subset of the data is chosen to train the network). To visualise what this means in practice, consider figure 5.4.

The network is being trained on data from a uniformly distributed pattern space within the two-dimensional outer frame—in other words, the patterns are evenly distributed across the rectangular region. A training set is selected from by choosing independent and random points in the pattern space—the randomness will ensure that a good representation of the total pattern space is provided. The sequence of diagrams represents the state of the weight vectors for various stages of the training cycle. The weight vectors are two dimensional ((X_1, X_2)), and the value of the weight vector defines a point in the weight space. The diagrams are plotted by drawing lines between the points defined by the weights of neighbouring nodes. These plots, then, depict the spatial relationship of the nodes in the weight space (two dimensional in this case). The final state

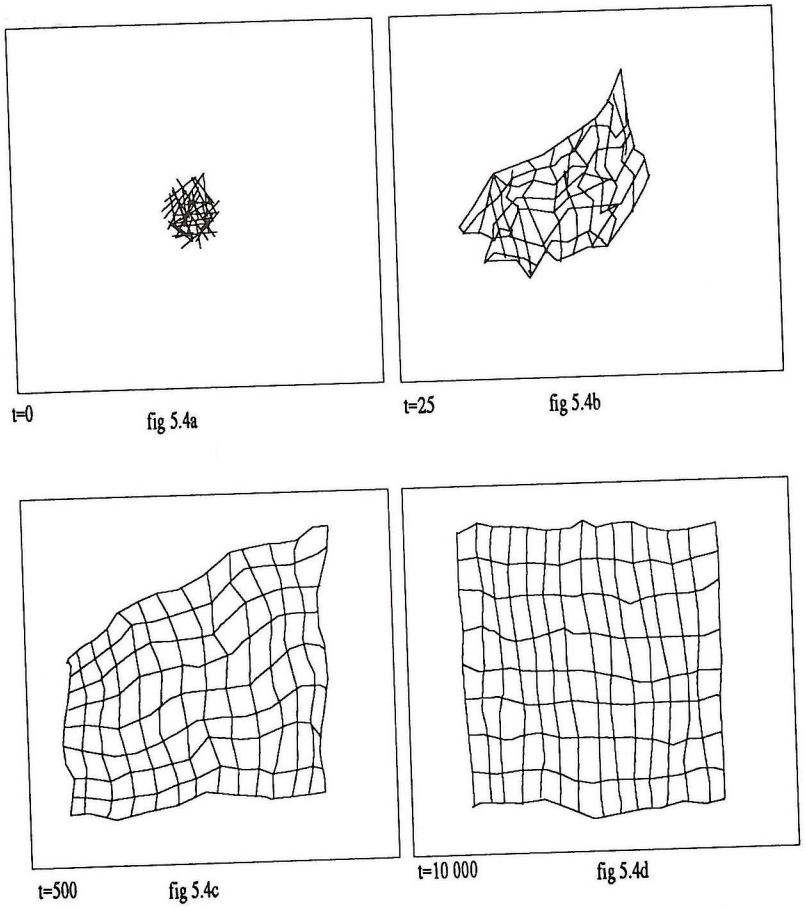


Figure 5.4 Representation of the development of the spatial ordering of the weight vectors.

of the network shown in figure 5.4 shows how the weight vectors have ordered themselves to represent the distribution of the pattern space. The nodes have been optimally ordered to span the pattern space as accurately as possible, given the constraint that there is a limited number of nodes to map the much larger space.

5.6 LEARNING VECTOR QUANTISATION (LVQ)

Despite the fact that the Kohonen network is an unsupervised self-organising learning paradigm, Kohonen does in fact make use of a supervised learning technique. This he describes as learning vector quantisation. This is worth mentioning because it amounts to a method for fine-tuning a trained feature map to optimise its performance in altering circumstances. A typical situation may be that we wish to add new training vectors to improve the performance of individual neighbourhoods within the map.

The way this is achieved is by selecting training vectors (x) with known classification, and presenting them to the network to examine cases of misclassification. Again, a best-match comparison is performed at each node and the winner is noted (n_w). The weight vector of the winning node is then modified according to the following criteria.

For a correctly classified input:

$$n_w(t+1) = n_w(t) + \eta(t)[x(t) - n_w(t)]$$

For an incorrect classification:

$$n_w(t+1) = n_w(t) - \eta(t)[x(t) - n_w(t)]$$

The term $\eta(t)$ controls the rate of adaptation, and performs the same function as it did in the learning cycle. The application study that follows shows how this method may be used to add new users to a speech recognition system by optimising the phoneme classifiers on a feature map.

5.7 THE PHONETIC TYPEWRITER

Perhaps one of the best ways of demonstrating the value of an idea is by its successful application. Kohonen has applied his feature map algorithm to the time honoured problem of speech recognition. This is perhaps an ideal application for feature map techniques. The problem is one of classifying phonemes in real time. Why is it so ideally suited to the Kohonen method? The phonemes form a small classification set with class samples showing subtle variations between them. This implies that only a small number of feature detectors need to be formed in the topological map, each of which has many nodes in its neighbourhood tuned over a limited range. That is not to say that phoneme classification is a trivial problem—far from it! Speech recognition is a complex pattern recognition task. Our own human recognition of speech works at several levels of perception. Apart from the fundamental interpretation of the speech waveform, much of the recognition is done at levels applying context, inference, extrapolation, parsing and syntactic rules. We even perform these functions reliably in considerably noisy environments. If you don't believe that statement, think about the cocktail party scenario. We are capable of understanding and holding a conversation in the midst of the general buzz of discussion. We are able to ignore the noise of conversation around us, and yet, if our name is mentioned in conversation elsewhere in the room we are very likely to pick it out (and be worried by it!).

From a signal processing perspective the speech waveform is also ill-defined and complex. Speech phonemes vary in signal strength and shape from speaker to speaker. Even in an individual speaker, phonemes vary in the context of the words that they are formed in, and invariably the spectral signals of the different phonemes overlap for much of their waveform. A great deal of effort has been expended over a sustained period to try to create accurate phoneme classifiers using conventional techniques. The simplicity of the solution we are about to describe serves to show the particular merit of Kohonen's technique.

The driving goal of Kohonen's work was to build a phonetic typewriter—that is a typewriter that could type from dictation. This is

perhaps easier in his native tongue of Finnish than it would be in other languages (Finnish being a phonetic language), but it was still quite a complex task. Kohonen has approached the problem applying a mixture of the best of many techniques—he is quick to point out that neural networks are not a universal panacea for all aspects of a data processing problem. The system that he devised is shown schematically in the following figure, figure 5.5.

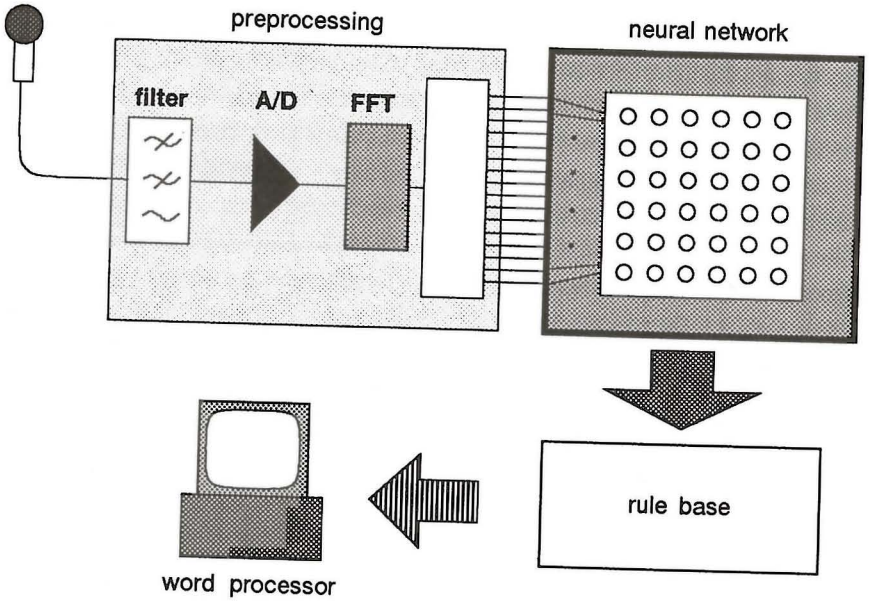


Figure 5.5 A schematic circuit of Kohonen's neural based phonetic typewriter.

The neural network is only dealing with one part of the total task. The system is not totally "neural"—in fact the neural network

is only being used in the critical stage of classifying the phonemes. This amounts to vector quantisation of the spectral speech signal. Kohonen was striving for a real-time commercial system—this meant that where conventional computing techniques provide optimal solutions, they were adopted within the system. This is most clearly seen in the front-end signal processing stage.

5.7.1 Front-end Preprocessing

The front-end processing is an essential element to any neural network technique. This point cannot be over-stressed. Any neural network paradigm will perform poorly if given non-representative or inadequate training data. Neural networks do provide a novel method of abstracting feature information into a distributed encoding. They do not, however, by-pass the critical stage in any pattern recognition task of adequately defining the salient and characteristic features of the data. Kohonen's system relies on standard digital signal processing techniques to extract the phoneme spectral data from the voice input. From a microphone input the speech waveform is fed into a 5.3 kHz low pass filter driving a 12-bit A/D converter (at a sampling rate of 13.03 kHz). A 256 point Fast Fourier Transform (FFT) is computed on the digital data from the A/D at 9.83 ms intervals to capture the spectral content of the phonemes. Kohonen uses the FFT technique because it shows the clustering properties of the spectral component better than more conventional coding methods, and thus provides a more useful representation on which the classifier can train. It is also a fast, reliable and well supported technique. The output of the FFT is filtered and made logarithmic before the information is grouped into a fifteen component continuous pattern vector. The information represented in this vector is the instantaneous power in one of fifteen frequency bands ranging from 200 Hz to 5 kHz. Before being applied to the network as input the components have the signal average removed and are then normalised to a constant length. Kohonen also uses a sixteenth vector component to represent other information about the signal. He chose to use this to represent the rms value of the speech signal.

In the preprocessing stage Kohonen has quantised the voice input to a 16-bit feature vector. The feature vector is a short time slice of the speech waveform. These features were used to train the network. It is important to note that the network was not trained on phoneme data, but only the time-sliced speech waveforms. The nodes in the network, however, become sensitised to the phoneme data because the network inputs are centred around phonemes. The network is able to find these phonemes in the training data without them being explicitly defined. The clusters that are formed during training must then be labelled afterwards by hand. This involves presenting isolated phoneme samples to the network and finding the region of maximum response on the topographical map. In Kohonen's experiments 50 samples of each test phoneme were used to label the network after it was trained on voice data. A typical topological feature map is shown in figure 5.6. It shows the trained network with the labelling attached—Kohonen describes it as a phonotopic map.

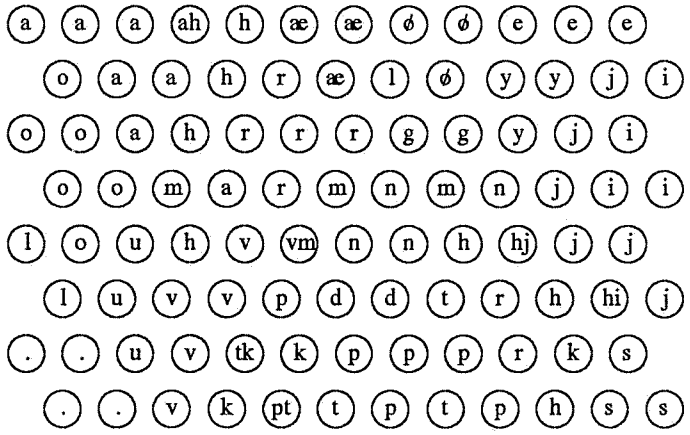


Figure 5.6 A phoneme feature map. Kohonen calls this a "phonotopic map".

The map classifies the more readily defined phonemes—that is, those with relatively stable and predictable speech waveforms. In

practice most phonemes have a much longer duration than the sampling rate used of 9.38 ms. The true duration is typically in the order of 40–400 ms. Consequently the classification of a phoneme is made on the basis of several consecutive inputs. Kohonen classifies the phonemes over a number of inputs using simple heuristic rules. One of these relies on the fact that many phonemes have spectra with a unique stationary state by which they can be identified. Alternatively, a sequence of inputs may be monitored—if the phonotopic map's response is constant for a number of consecutive inputs then those inputs correspond to a single phoneme.

It is also possible to visualise the speech waveform as a dynamic trace across the phonotopic map. This is shown in figure 5.7. The steps represent input samples at 9.38 ms intervals. The trace shows the stationary states of the spoken phonemes converging at localised points on the map. Kohonen does not perform classification using these traces, but he does suggest that they provide a new way of visualising phonetic strings, that may be of use in applications such as teaching aids for the deaf.

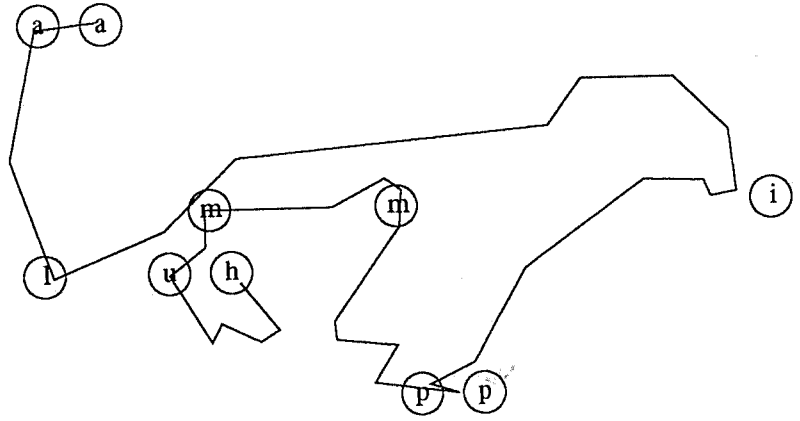


Figure 5.7 The map shows the phonetic trace of the Finnish word "humpilla" across the map.

5.7.2 Auxiliary Maps

The "plosive" phonemes (e.g. b/t/g) have very transient spectra characterised by a high burst of initial energy followed by a period of comparative silence. Kohonen found that the standard phonotopic map did not perform very well at classifying this type of phoneme. His solution was to use auxiliary maps (called transient maps) to classify just the plosive type phonemes. The auxiliary maps were trained on the spectra of the plosive phonemes. The results of such a simple modification to the map was an overall improvement in the recognition accuracy of six to seven per cent.

5.7.3 Post Processing

The last stage of the phonetic typewriter is the translation from the phonetic transcription to orthographic. It is here that the errors from the classification stage must be corrected. The majority of errors are caused by an effect known as coarticulation. Coarticulation is the variation in the pronunciation of a phoneme that is caused by the context of the neighbouring phonemes. To deal with this effect, Kohonen has adopted a rule based system that constructs the correct grammar from the phonetic translation. The rule base is large—typically 15000–20000 rules and deals primarily with context sensitivity of phonemes. It would be impractical to attempt to define rules to account for coarticulation without considering context. The rule base would be prohibitively large if it were to deal with all permutations and it could not cope with the contradictory cases so often found in a language. Kohonen's rule base has been developed from actual example speech data and its correct phonetic transcription. Much like the neural network stage the rules have been derived from example rather than explicitly.

The grammar rule base has been implemented efficiently using hash coding (a software technique for content addressable memory) and operates in near real time—even for a large rule base. The output of the rule base is contextually corrected phonetic strings that

can produce orthographic text to drive a word processor environment.

5.7.4 Hardware Implementation

The system has been designed with standard digital hardware. The host computer is an IBM PC/AT with two auxiliary DSP coprocessor boards. Both coprocessors are based on the TMS32010 Digital Signal Processor. One board is responsible for the preprocessing of the speech signal, the other is performing the feature map classification. Post processing is done by the host PC. Even using such standard hardware (much faster DSP's are now available) the recognition system performs at an almost true rate of speech; only a slight pause is required between words.

5.7.5 Performance

The performance figures that follow are quoted by Kohonen from his experiments with the system. Performance figures are always difficult to analyse without understanding the full context of the tests. However, it is worth quoting those for the system as they are fairly indicative of the usefulness of adopting a neural based solution in this type of application.

Correct classification of phonemes from the phonotopic map stage varies between 80 and 90 per cent depending upon speaker and the text. The system accuracy after correction by the grammar rule base increases to between 92 and 97 per cent. This figure is measured by letter accuracy on the orthographic output and is for an unlimited vocabulary.

A performance issue that must also be considered is the flexibility of the system in adapting to new users. Kohonen's system is particularly amenable to the addition of new speakers. They are added using the supervised learning technique, learning vector quantisation, that was described earlier in the chapter. Fine-tuning a map to a new speaker typically requires 100 words and can be completed, according to Kohonen, in ten minutes.

5.7.6 Conclusion

Hopefully, working through this application example has brought two main issues forward. The first is an indication of how self-organising networks may be used in practice. The second is an appreciation of how neural networks may be embedded at the system level. There has been much hysteria concerning the application of neural network techniques and many exaggerated claims for their performance. Neural networks are far from being a universal panacea for all computing situations, but Kohonen's system level approach shows how the strengths of neural techniques (parallelism, generalisation, noise tolerance) may be used in conjunction with conventional techniques to create very powerful computing tools.

Summary

- Kohonen nets are self-organising, with similar inputs mapped to nearby nodes.
- All the nodes are in one two-dimensional layer.
- "Mexican hat" function of lateral excitation and inhibition.
- Neighbourhood of interactions decreases with time.
- Successfully implemented to produce a phonetic typewriter.

FURTHER READING

1. *Self Organisation and Associative Memory*, third edition. T. Kohonen. Springer-Verlag, 1990. A tutorial introduction to concepts of associative memory and neural networks, with a discussion of self-organisation principles.

2. *Parallel Models of Associative Memory*, second edition. G. E. Hinton & J. A. Anderson. Lawrence Erlbaum Associates, 1989. Collected writings on models of memory and parallel processing—a useful discussion of cognitive and connectionist issues.

3. Self-Organisation of Orientation Sensitive Cells in the Striate Cortex. C. von der Malsburg. In *Kybernetik*, 14, pages 85–100, 1973. An original paper proposing the concept of self-organisation.

4. How patterned neural connections can be set up by self-organisation. D. J. Willshaw & C. von der Malsburg. In *Proc. R. Soc. London*, B. 194, pages 431–445, 1976. Models for the development of self-organised biological networks.

5. Competition and Cooperation in Neural Nets. S. Amari & M. A. Arbib. In *Lecture Notes in Biomath., Vol. 45*, Springer-Verlag (Berlin), 1982.