7 Topology Preserving Maps

This chapter introduces three negative feedback artificial neural network architectures which perform a vector quantization. Vector quantization is used in signal processing applications to encode a high–dimensional signal in order to minimise processing/transmission costs. The basic aim is to associate with each group of vectors of the raw data a code which uniquely identifies that group. If the vectors of the group are sufficiently alike and the decoded code is sufficiently representative of the group, then the error when the code is used to represent a vector in the group can be made acceptably small. One further feature of the mapping which we desire is that it should retain an accurate representation of the topology of the data space. This is a rather complex feature to specify absolutely accurately so we shall initially content ourselves with a mapping in which nearby points in the data space are mapped to the same or nearby neurons in the coding space while ensuring that nearby neurons in the coding space are decoded to nearby points in data space.

7.1 Background

The most common type of artificial neural networks used to perform a topology preserving vector quantization is that developed by Kohonen. In the Kohonen network, when a data point is presented to the network, a competition takes place between the neurons and one neuron is declared the winner; its weights and those of its neighbours are moved towards the input pattern's values while those of the other neurons are moved further away. The net result is that if the same input pattern or one similar to it is presented again, the same neuron is most likely to win again. It has been shown that, after a suitable training period, the neurons of the second layer form a map of the inputs which preserves some aspect of the topology of the input data. Such nets are known as self-organising nets as there is no teacher input to the net.

We will develop two negative feedback networks which quantise the data in a topology preserving manner but first we review competitive learning and Kohonen's algorithm.

7.1.1 Competitive Learning

The basic mechanism of simple competitive learning is to find a winning unit and update its weights to make it more likely to win in the future should a similar input be given to the network. We first have the activity transfer equation

$$y_i = \sum_j w_{ij} x_j, \forall i \tag{7.1}$$

which is followed by a competition between the output neurons and then

$$\Delta w_{ij} = \eta (x_j - w_{ij}) \tag{7.2}$$

for the winning neuron i. Note that the change in weights is a function of the *difference* between the weights and the input. This rule will move the weights of the winning neuron directly towards the input. If used over a distribution, the weights will tend to the mean value of the distribution since $\Delta w_{ij} \rightarrow 0 \Longleftrightarrow w_{ij} \rightarrow E(x_j)$.

7.1.2 The Kohonen Feature Map

The interest in feature maps stems directly from their biological importance. A feature map uses the "physical layout" of the output neurons to model some feature of the input space. In particular, if two inputs $\mathbf{x}_1$ and $\mathbf{x}_2$ are close together with respect to some distance measure in the input space, then if they cause output neurons y_a and y_b to fire respectively, y_a and y_b must be close together in some layout of the output neurons. Further, we can state that the opposite should hold: if y_a and y_b are close together in the output layer, then those inputs which cause y_a and y_b to fire should be close together in the input space. When these two conditions hold, we have a feature map. Such maps are also called **topology preserving maps**.

Examples of such maps in biology include

- the retinotopic map, which takes input from the retina (at the eye) and maps it onto the visual cortex (back of the brain) in a two–dimensional map.
- the somatosensory map, which maps our touch centres on the skin to the somatosensory cortex.
- the tonotopic map, which maps the responses of our ears to the auditory cortex.

Each of these maps is believed to be determined genetically but refined by usage. e.g. the retinotopic map is very different if one eye is excluded from seeing during particular periods of development.

Hertz et al [73] distinguish between

- those maps which map continuous inputs from single (such as one ear) inputs or a small number of inputs to a map in which similar inputs cause firings on neighbouring outputs (left half of Fig. 7.1)
- with those maps which take in a broad array of inputs and map onto a second array of outputs (right half of Fig. 7.1).

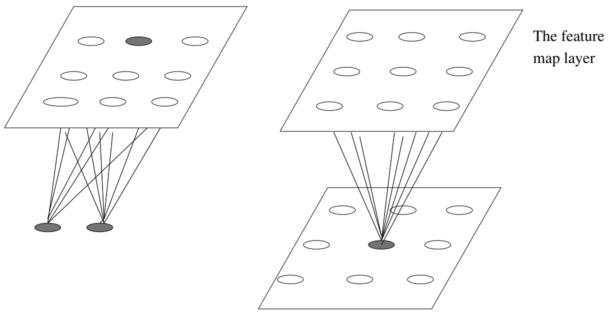


Fig. 7.1. Two types of feature maps: (a) a map from a small number of continuous inputs; (b) a map from one layer spatially arranged to another

Kohonen's algorithm [111] is exceedingly simple - the network is a simple 2-layer network and competition takes place between the output neurons; however, now not only are the weights into the winning neuron updated, but also the weights into its neighbours. Kohonen defined a neighbourhood function $f(i, i^*)$ of the winning neuron i^*. The neighbourhood function is a function of the distance between i and i^*. A typical function is the difference of Gaussians function (Figure 7.2); thus, if unit i is at point $\mathbf{r}_i$ in the output layer then

$$f(i, i^*) = a \exp\left(\frac{-|r_i - r_{i^*}|^2}{2\sigma^2}\right) - b \exp\left(\frac{-|r_i - r_{i^*}|^2}{2\sigma_1^2}\right) \tag{7.3}$$

Notice that a winning neurons' chums – those neurons which are "close" to the winning neuron in the output space – are also dragged out to the input data while neurons further away are pushed slightly in the opposite direction.

The algorithm is:

1. Select at random an input data point.
2. There is a competition among the output neurons. That neuron whose weights are closest to the input data point wins the competition:

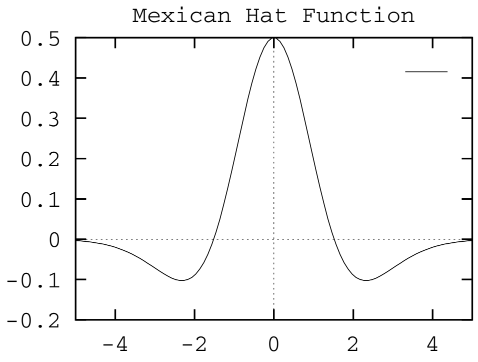


Fig. 7.2. The difference of Gaussian function.

$$\text{winning neuron, } i^* = \arg\min(\| \mathbf{x} - \mathbf{w}_i \|) \tag{7.4}$$

3. Now update all neurons' weights using

$$\Delta w_{ij} = \alpha(x_j - w_{ij}) * f(i, i^*) \tag{7.5}$$

where

$$f(i, i^*) = a \exp\left(\frac{-|r_i - r_{i^*}|^2}{2\sigma^2}\right) - b \exp\left(\frac{-|r_i - r_{i^*}|^2}{2\sigma_1^2}\right) \tag{7.6}$$

4. If not converged, go back to the start.

Kohonen typically keeps the learning rate constant for the first 1000 iterations or so and then slowly decreases it to zero over the remainder of the experiment (a simulation can take 100 000 iterations for self-organising maps). Two–dimensional maps can be created by imagining the output neurons laid out on a rectangular grid (we then require a two-dimensional neighbourhood function) or sometimes a hexagonal grid.

7.2 The Classification Network

Before we introduce a negative feedback topology-preserving mapping, we discuss the following very simple negative feedback competitive network based on nonlinear PCA discussed in Chapter 5:

$$s_i = \sum_{j=1}^{N} w_{ij} x_j \tag{7.7}$$

$$y_i = f(s_i) = f\left(\sum_{j=1}^{N} w_{ij}x_j\right) \tag{7.8}$$

$$e_j = x_j - \sum_{k=1}^{M} w_{kj}y_k \tag{7.9}$$

$$\Delta w_{ij} = \eta_t y_i e_j \tag{7.10}$$

$$= \eta_t f\left(\sum_{k=1}^{N} w_{ik}x_k\right)\left\{x_j - \sum_{l=1}^{M} w_{lj}\sum_{p=1}^{N} w_{lp}x_p\right\} \tag{7.11}$$

We noted that Karhunen and Joutsensalo [101] (in the context of feedforward networks) have shown that (7.11) is an approximation to the rule required to minimise the residuals at the inputs after the negative feedback is returned.

We will use the network described above to classify by using a function $f()$ defined by

$$f(s_i) = 1 \text{ if } i = \arg\max_j s_j \tag{7.12}$$

$$f(s_i) = 0 \text{ otherwise.} \tag{7.13}$$

Therefore we have an extremely simple network, which nevertheless will be shown to be capable of hierarchical classification. Clearly the learning rule is now equivalent to $\Delta w_{ij} = \eta(x_j - w_{ij})$, the standard competive learning rule for the winning neuron, so that

$$\Delta w_{ij} \to 0 \Longleftrightarrow w_{ij} \to E_i(x) \tag{7.14}$$

where $E_i(x)$ is the average x for which neuron i is firing. Thus we can see that the weights will converge to the mean of the data to which the weight responds. Further, we can show that this network is stable, requiring no renormalisation or otherwise bounding of the network weights. As usual the–winner–take–all network can be modeled as a lateral inhibition network (c.f. Chapter 4) of the form used for the PCA network.

However, the most important property of this network is its ability to perform hierarchical decomposition of classification. Since the neuron's weights are converging to the mean of the distribution to which it is responding *and subtracting this mean* what remains is the difference between the mean and the distribution to which it is responding. This allows subsequent neurons which use exactly the same network learning to converge to subpatterns within the data.

7.2.1 Results

We describe an experiment using the network on artificial data. The data which we use is two dimensional and is shown diagrammatically in Fig. 7.3;

it comprises randomly chosen points from one of five equally probable two dimensional distributions. The distributions have centres (2,2), (2,−2), (−2,2), (−2,−2) and (−3,−3). The horizontal and vertical distances of individual samples from their respective distribution centres are independent and are drawn from a Gaussian distribution of standard deviation 1.

We use an initial network with two inputs – the x and y coordinates – and 10 outputs. Since we only have five classes, this gives us far more power than we need; however, we wish to simulate the situation in which the number of clusters is not known a priori. This task comprises for the network a simple problem – differentiating between the data clusters in the four quadrants – and a more difficult problem – that of differentiating between the two clusters in the 3^{rd} (all negative) quadrant.

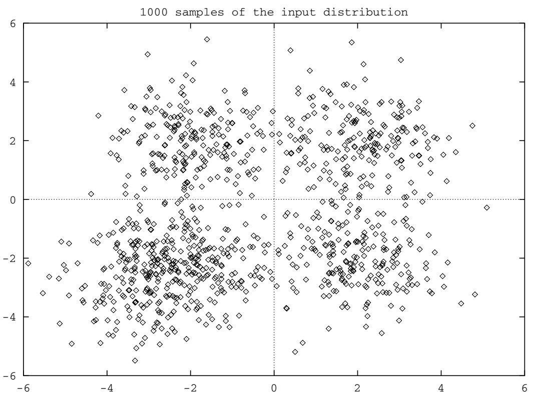


Fig. 7.3. The diagram represents the input data schematically: the data was taken from five independent Gaussian distributions with centres (2,2),(2,−2),(−2,2),(−2,−2) and (−3,−3), each distribution having a standard deviation of 1.

The weights for each neuron after convergence and its number of successes thereafter on 10 000 trials are shown in Table 7.1. These weights were learned in 1000 presentations of the input data (i.e. the network saw approximately 200 samples from each distribution); we have, however, trained the network on 200 000 iterations and found that the weights are stable at these points. The learning rate is annealed from 1 to 0 during the experiment.

Notice that during learning, neuron 9 has clearly been competing for the stewardship of the $(x_1 > 0, x_2 > 0)$ quadrant but has lost out to neuron 6. It is easy in such cases to monitor the success of such a neuron and delete it on testing the rate of success after learning.

Table 7.1. The filters to which the weights of the output neurons converged and the number of successes in 10 000 trials (after convergence).

Neuron	1	2	3	4	5	6	7	8	9	10
x_1-weight	0.00	-2.44	0.00	2.11	0.00	1.94	-1.86	0.00	1.60	0.00
x_2-weight	0.00	-2.47	0.00	-1.98	0.00	1.85	2.16	0.00	0.81	0.00
Successes	0	3992	0	2006	0	2043	1959	0	0	0

Clearly the network is very adept at differentiating between four groups, but, unsurprisingly, the two groups in the $(x_1 < 0, x_2 < 0)$ quadrant are treated as one.

Now the negative feedback network is not only finding the centres of each group, it is also subtracting out these centres because of the negative feedback. We can use this fact in finding subclasses of the sets which our network has already found by continuing the simulation with a second layer of output neurons which will learn on the distribution remaining *after* the first layer of output neurons has subtracted their activations. We could, in this situation, perform a top-down dichotomy of the data by creating layers containing only two output neurons each.

7.2.2 Stochastic Neurons

A network with similar properties can be created using stochastic neurons. We define a winner–take–all network using a roulette-wheel selection procedure: each neuron's probaility of firing is dependent on the sum of its weighted inputs by

$$P(\text{neuron}_j \text{ fires}) = \frac{\exp(s_j)}{\sum_k \exp(s_k)} \tag{7.15}$$

where initially the weights are small ($\approx 10^{-5}$) random numbers. Experiments have confirmed that this network also self-organises in a manner identical to that above to find increasingly refined hierarchies.

7.3 The Scale Invariant Map

We now extend this network to enable it to perform feature mapping by a method similar to that used by Kohonen to change simple competitive networks to feature maps.

Thus, in the learning rule above, we change not only the winning neuron's weights but also the weights of those neurons closest to it. We use a Gaussian mask[1] to model a differential effect: the weights of those closest to the winning

[1] A difference of Gaussian gives the Mexican hat field but was found to be unnecessary in this simulation and indeed more difficult to stabilise.

neuron are updated most highly. We decrease the diameter of the Gaussian during the simulation run. This, as usual [111], focuses the activation on a narrow set of neurons.

Consider a network with N dimensional input data and having M output neurons. Then the activation of the i^{th} output neuron is given by

$$y_i = \sum_{j=1}^{N} w_{ij} x_j \tag{7.16}$$

Now we invoke a competition between the output neurons. We will investigate networks using one of two possible criteria:

Type A: The neuron with greatest activation wins:

$$\text{winner } = \arg\max_i y_i \tag{7.17}$$

Type B: The neuron closest to the input vector wins:

$$\text{winner } = \arg\min_i ||\mathbf{x} - \mathbf{w}_i|| \tag{7.18}$$

In both cases, the winning neuron, the p^{th}, is deemed to be maximally firing (=1) and all other output neurons are suppressed. Its firing is then fed back through the same weights to the input neurons as inhibition,

$$e_j = x_j - w_{pj}.1 \text{ for all } j \tag{7.19}$$

where p is the winning neuron. Now the winning neuron excites those neurons close to it i.e. we have a neighbourhood function $\Lambda(p,j)$ which satisfies $\Lambda(p,j) \leq \Lambda(p,k)$ for all $j,k :\| p-j \| \geq \| p-k \|$ where $\| \, . \, \|$ is the Euclidean norm. In the simulations described in this chapter, we use a Gaussian whose radius is decreased during the course of the simulation. Then simple Hebbian learning gives

$$\begin{aligned} \Delta w_{ij} &= \eta_t \Lambda(p,i).e_j & (7.20)\\ &= \eta_t \Lambda(p,i).(x_j - w_{pj}) & (7.21) \end{aligned}$$

For the p^{th} winning neuron, the network is performing simple competitive learning but note the direct effect the p^{th} output neuron's weight has on the learning of other neurons.

7.3.1 An Example

To illustrate convergence we use, as input data, a two–dimensional vector drawn randomly from the square $\{(x,y) : -1 < x \leq 1, -1 < y \leq 1\}$. With the rules used above we get results such as those shown in Fig. 7.4. The 25 output neurons' weights have organised in such a way that similar (and only similar) input values are mapped onto similar output neurons which is the usual definition of topographic mappings (e.g. [111]). The scale-invariance of the resultant mapping can be seen most clearly in Fig. 7.6 in which we have shown the points which were mapped onto three specific output neurons.

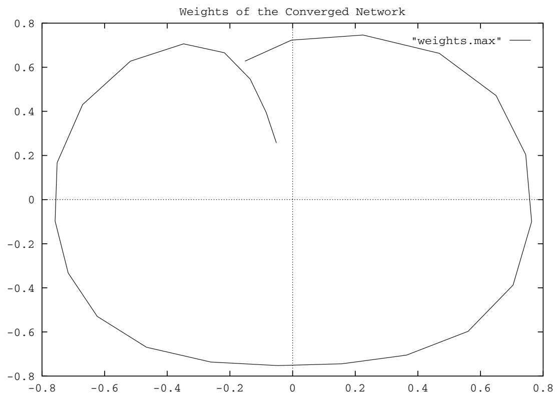


Fig. 7.4. The converged weights from the feature mapping network when the input data is drawn from $\{(x, y) : -1 < x \leq 1, -1 < y \leq 1\}$.

7.3.2 Comparison with Kohonen Feature Maps

It is well known (e.g. [111]) that, given similar input data to that used above, a one-dimensional Self-Organising Map (SOM) will self-organise to spread itself over the square to minimise the expected distance between the code points and the points of the square. We have called the current mapping a scale-invariant feature map since it ignores the magnitude of each input vector and responds solely to the relative proportion of the magnitude of the elements of the input vectors. The Kohonen feature map may be criticised on grounds of biological implausibility in that a single neuron takes responsibility for representing a set of inputs (the "grandmother" cell). However, if we increase the learning rate with the Scale Invariant feature map, an interesting effect comes into play: the mapping winds round upon itself so that each outer neuron (which is currently winning competitions) is backed up by a set of support neurons. The results of such an experiment are shown in Fig. 7.5.

The network learning rules can be extended to use a higher–dimensional neighbourhood function, though in dimensions higher than two, the resulting map is difficult to view. We have found that higher dimensional maps are more prone to twists such as those well known in the Kohonen SOM [73]. Such twists can take a very long time to untwist. Kohonen's strategy of beginning with a wide neighbourhood function and decreasing its width gradually is the most successful with this problem.

7.3.3 Discussion

As with the Kohonen SOM ([111], page VII), a full analysis of the current mapping is remarkably difficult. Our discussion here is descriptive rather than fully analytical.

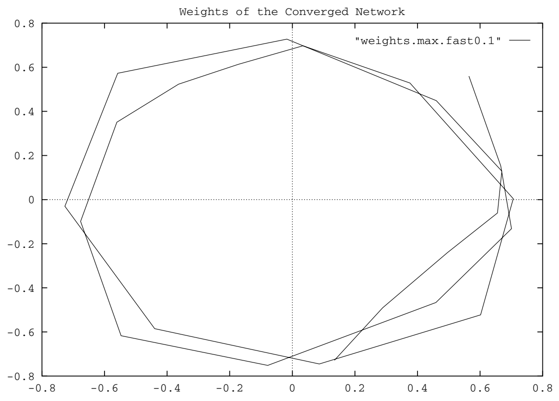


Fig. 7.5. The results when the learning rate is increased 100–fold. Notice that those weights which continue to occupy space within the outer ring of neurons, do not win any competitions. They can be thought of as backups for those neurons which are winning competitions: if one of the winners should fail, there exists a substitute which will step into its shoes.

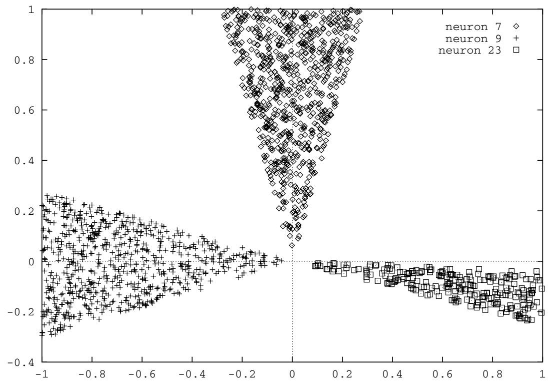


Fig. 7.6. Some points showing the winning areas for specific neurons in 10 000 trials of the fast learning network.

Consider an input distribution which is spherically symmetric and let us use a very simple (though still symmetric) mask in which the neighbourhood function in the weight update rule is given by

$$\begin{aligned} G(i_1) &= 1 \text{ where } i_1 \text{ is the winning neuron} \\ G(i_1+1) = G(i_1-1) &= g, \text{for some g >0} \\ G(j) &= 0, \text{ when j } \neq i_1-1, i_1, i_1+1. \end{aligned}$$

Consider three *consecutively numbered* neurons labelled A, B and C.

Now when neuron A wins the competition, neuron B's weights will also be changed according to:

$$\Delta \mathbf{w}_B = \eta_t . g . (\mathbf{x} - \mathbf{w}_A) \tag{7.22}$$

Note the crucial difference here between this form of learning and the usual competitive learning rules: here B's weights are being directly affected by the centre of gravity of the inputs to the winning neuron next to B. Thus, if the process is to converge,

$$\sum_{\mathbf{x} \in Set_A} g.(\mathbf{x} - \mathbf{w}_A) + \sum_{\mathbf{x} \in Set_B} (\mathbf{x} - \mathbf{w}_B) + \sum_{\mathbf{x} \in Set_C} g.(\mathbf{x} - \mathbf{w}_C) = \mathbf{0} \tag{7.23}$$

where Set_L is the set of points sampled in sector L and $\mathbf{w}_L$ is the weight vector of the neuron winning in sector $L, L = A, B, C$. If we are drawing points randomly from the data set the limit of the above as the number of points tends to infinity is

$$\begin{aligned} \int_{Set_A} g(\mathbf{x} - \mathbf{w}_A) p(Set_A) d\mathbf{x} + \int_{Set_B} (\mathbf{x} - \mathbf{w}_B) p(Set_B) d\mathbf{x} \\ + \int_{Set_C} g(\mathbf{x} - \mathbf{w}_C) p(Set_C) d\mathbf{x} = \mathbf{0} \end{aligned}$$

where $p(Set_L)$ is the probability that a point drawn at random from the input distribution will be in sector L. A solution of this equation is the set of weights which converge to the centre of gravity of each sector where each sector is of equal probability (i.e. of equal angular spread) in each dimension. In this case the mapping created is a maximum entropy mapping. It should be noted that while this is found empirically to be true when the learning rate is small, a high learning rate (Fig. 7.6) disrupts this feature.

Notice also that, while individual input data points within slice A may have the effect of moving B's weights away from A, the mean effect of A's points on the weights into B will be to move them towards A's centre. Points in sector C will be having the opposite effect: if slices A and C are of equal width on either side of B, the effect of each will cancel out the other.

7.3.4 Self-Organisation on Voice Data

We illustrate the converged mapping of the Scale Invariant network on speech data. Typically such networks are trained on data which has been preprocessed by, for example, Fourier transforming the data to the frequency domain and often using the more computationally expensive cepstral coefficients (see e.g. [111]). We, however, wish to test the nework by using as crude voice data as possible as inputs.

The input data to the network then is raw voice data sampled at 8 kHz and subjected to no preprocessing. The network has been tested with 20, 32, 64 and 128 inputs where e.g. 64 inputs represents 64 consecutive inputs from the data stream (equal to 8 ms of data). Each presentation of the data consists of a randomly chosen starting point and the following 63 consecutive inputs. The results have been qualitatively similar for each size of network. We use 25 output neurons with a one–dimensional neighbourhood function and the maximum activation criterion to identify the winning neuron. Consider first one network which is trained on 8 speakers saying the word "far". The trained weights are shown in Fig. 7.7. The top diagram shows the weights into the first three neurons: we can see that:

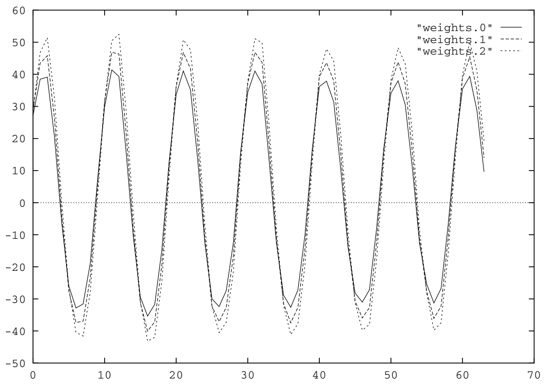
- the neurons are extracting the frequency information from the input data
- the weights into the first neuron differ very little from those into the second and this in turn differs very slightly from those into the third neuron. This is a prerequisite for any network which is claiming to retain neighbourhood relations.

The second half of the diagram shows the weights into three neurons which are not neighbours. Clearly each neuron is extracting the same frequency information from the raw data but has learned to respond to different phases of this data. A full diagram with all 25 output neurons would show a complete coverage of all phases at this frequency.

7.3.5 Vowel Classification

In Fig. 7.8, we show the weights from converged networks which have been trained on different vowel sounds – "far", "pit" and "put". The fact that these are very different can be used to classify vowel sounds into their respective classes.

We now propose a network which will identify any vowel from its raw data samples. We use a network such as shown in Fig. 7.9. During the learning phase, each set of output neurons (Net A, Net B, ...) is trained on a different set of vowel data. Each learns the frequency information associated with that vowel, and individual neurons within that set will respond maximally at any particular time depending on the correspondence between the phase of the signal and the weights into the neuron. However during the vowel identification phase the network is fully connected – each input is connected to all output



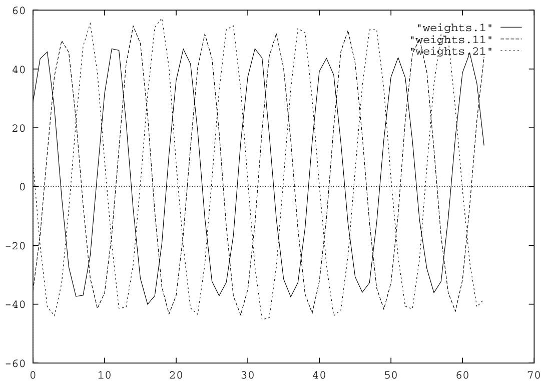


Fig. 7.7. Top diagram: the weights of the first three neurons trained on the vowel sound of "far". Bottom diagram: the weights of the first, eleventh and twenty first neurons on the same data.

neurons in all networks, Net A, Net B, ... When any vowel is presented to the network, the vowel can be identified by noting which of the output networks responds optimally since there is a ripple of activation across that network as particular neurons respond maximally to the particular phase of the inputs. Such a coherent ripple cannot be seen in the other output nets.

7.4 The Subspace Map

Recognition of patterns subject to transformations such as translation, rotation and scaling has been a difficult problem in artificial perception. The projection of objects on the retina is variable in position, size, orientation, luminance, etc., yet we are still able to recognise objects with great ease, re-

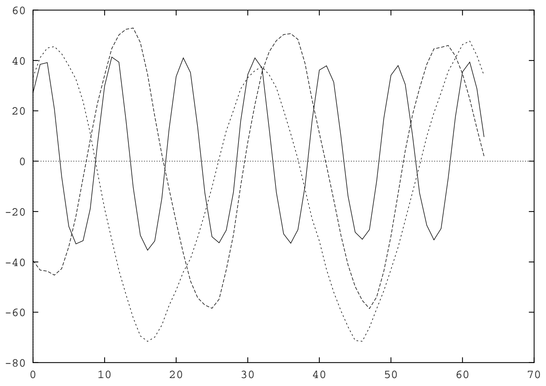


Fig. 7.8. A comparison of network weights when three different vowels are used for input data.

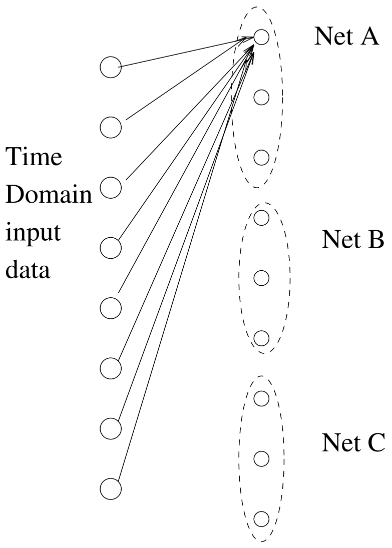


Fig. 7.9. Each group of output neurons comprises a single network trained on examples of a single vowel. Each output neuron responds maximally to a single type of input data.

gardless of such variations. A neural network solution to this problem has been proposed by Kohonen et al. [111] called the Adaptive Subspace Self-Organising Map (ASSOM). The motivation for the ASSOM is that if different samples can be derived from each other by means of a linear transformation, then they span a common linear subspace of the input space and projecting them onto this subspace can then filter out their differences. Using common linear subspaces as filters also motivates the method presented in this Section.

By constructing a SOM where each module defines a subspace, we can observe the organisation of clusters of different subspaces that each provides a different invariant filter. The neighbourhood relations in the map cause neighbouring modules to represent similar subspaces.

The map consists of an array of modules, normally in one or two dimensions, although higher-dimensional maps may be used. Each module of the map consists of the same arbitrary number of nodes which defines the dimensionality of the subspace represented by the module and a single output which is a quadratic function of the outputs of these nodes. (See Fig. 7.10).

Within each module, weights are updated using the Subspace Algorithm (either Oja's feedforward version (Chapter 2) or the negative feedback implementation (Chapter 3)), and each module learns the principal subspace of a subset of the data set. All of the nodes capture a local linear subspace but, collectively, they represent a nonlinear manifold. There is also a single output for each module which is quadratic sum of the activations of other nodes. The learning process requires the use of "episodes" of inputs, meaning that input vectors are presented to the network in batches and the weights are updated after a complete batch of inputs have been fed forward.

7.4.1 Summary of the Training Algorithm

The orthogonal projection of an input vector $\mathbf{x}$, onto a subspace $\mathcal{L}^{(i)}$ gives a projection $\hat{\mathbf{x}}^{(i)}$ and a residual $\tilde{\mathbf{x}}^{(i)}$ where $\mathbf{x} = \hat{\mathbf{x}}^{(i)} + \tilde{\mathbf{x}}^{(i)}$. Therefore, when data is fed forward in the network, each module, i, encodes the data as a vector $\mathbf{y}^{(i)}$ which is a lower–dimensional representation of the projected vector $\hat{\mathbf{x}}^{(i)}$. In order to select a module whose subspace most closely approximates a local part of the data set, it is necessary to consider several data points, and this leads to the use of episodes of data.

An episode $\mathcal{S}$ consists of a set of consecutive time instants and the set of inputs $\mathbf{x}(t_p)$ is taken from these sampling instants $t_p \in \mathcal{S}$. These vectors are projected onto the subspaces represented by the modules. For each episode of inputs, a representative winning node, c_r, is chosen and this is held constant for the duration of the episode. Since each node of the map represents a subspace $\mathcal{L}^{(i)}$ of the input space, the selection of the winning module, c_r, can be based on how closely the orientation of its subspace $\mathcal{L}^{(c)}$ approximates the orientation of the data in the episode. The representative winner is chosen to be the module with the maximum squared projections (7.24) which is equivalent to minimising the residuals since $\|\mathbf{x}\|^2 = \|\tilde{\mathbf{x}}\|^2 + \|\hat{\mathbf{x}}\|^2$:

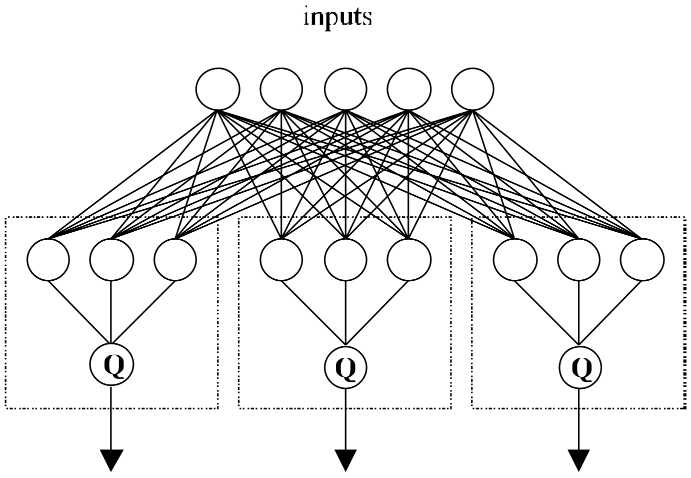


Fig. 7.10. The Subspace Map. Each module of the map (enclosed in dotted boxes) performs principal component analysis on a subset of data from the training set. There is also a single quadratic output in each module that is activated by the sum of squared outputs of the PCA nodes.

$$c_r = \arg\max_i \left\{ \sum_{t_p \in \mathcal{S}} \| \hat{x}^{(i)}(t_p) \|^2 \right\} \tag{7.24}$$

If the input space and subspaces have N and M dimensions, respectively, then an orthogonal projection onto a subspace gives a vector y of dimensionality M. The j^{th} output of $\mathcal{L}^{(i)}$ is denoted by $y_j^{(i)}$ and is calculated by a weighted sum of its inputs (7.25). The activation of the quadratic output is then calculated by (7.26):

$$y_j^{(i)} = \sum_{k=1}^{N} w_{kj}^{(i)} x_k \tag{7.25}$$

$$Q^{(i)} = \sum_j = 1^M y_j^{(i)^2} \tag{7.26}$$

The weights of the winning module and its neighbouring modules are then updated using the Subspace Algorithm (7.27) with a radially decaying neighbourhood function to ensure that nodes that are close to the winner will be trained with a higher learning rate than those further away. The neighbourhood of a winning module is the set $\mathcal{N}^{(c_r)}$ of modules whose weights are updated. This set may include every module in the map; however, significant savings in computing time can be achieved if a threshold is added to the neighbourhood function such that modules are changed only if they are close to the winner. Those that are further away will have very low learning rates due to the neighbourhood function and it is acceptable to exclude them from training:

$$\Delta w_{jk}^{(i)} = h_{ci}(t)\alpha(t)\left(x_k y_j^{(i)} - y_j^{(i)} \sum_l w_{kl}^{(i)} y_l^{(i)}\right) \tag{7.27}$$

Therefore, for every module, $i \in \mathcal{N}^{(c_r)}$, the weights are updated to shift towards a principal subspace of $x(t_p \in \mathcal{S})$. This shift most strongly affects the winning module and the neighbourhood function determines the effect on its neighbouring modules. Competitive learning ensures that each module is trained on a subset of the training data and therefore, the weight vectors of each module form a principal subspace of a different class of input patterns (local PCA).

The neighbourhood function that we used $h_{ci}(t)$ is a Gaussian or difference of Gaussians (Mexican hat) with a width that is decreased in time; however, there are other neighbourhood functions that may be equally suitable. This function may be used with a narrowing threshold so that the number of modules in the neighbourhood of the winner, $\mathcal{N}^{(cr)}$ is reduced.

7.4.2 Training and Results

We discuss two examples of using the map to create filters on data which may be of biological significance – sound data and artificial video data emulating moving objects.

Formation of Wavelet Filters from Raw Sound Data

The network was trained on raw speech data and formation of a mapping was observed that extracted phase and frequency information from voiced phonemes. In the experiments of Kohonen et al. [111], the ASSOM was used to generate wavelet filters from raw speech data and we now show that the subspace map presented will also form filters that are phase invariant.

To demonstrate the behaviour of this network, it has been trained on artificial sound data containing a range of frequencies between 200 Hz and 10000 Hz. This data was high-pass filtered by taking the differences between successive samples. The training vector, consisted of 64 consecutive samples of sound recorded at 8kHz giving an input window duration of 8.75 ms. The one-dimensional map consisted of 24 modules, each of which had a subspace dimensionality of two. Eight sets of inputs that were adjacent in time formed a single episode. Representative winners were selected using the maximum energy criterion.

Each input window was multiplied by a Gaussian weighting function with a full width half maximum, FWHM(t), of eight samples and this was increased to 20 during training. The initial learning rate was 0.008 and was reduced by 0.0005 after every 20 episodes.

Fig. 7.11 shows the weights of the converged map after 300 training episodes. At one end of the map the low frequencies have been learned and there is a smooth progression to the high frequencies at the opposite end, therefore the frequency of an input pattern corresponds to the position of the winning module. Each module has orthonormal basis vectors as a result of the principal subspace learning algorithm. Since the vectors are always orthogonal to the vectors, one can be considered to be a sine wavelet and the other a cosine wavelet. Sinusoidal oscillation is therefore eliminated from the square of the wavelet amplitude transform (A),

$$A^2 = F_c^2(t,\omega) + F_s^2(t,\omega) \tag{7.28}$$

and the organised subspaces are therefore approximately invariant to time shifts. When using the ASSOM, this property has to be forced by periodically re-orthonormalising the vectors.

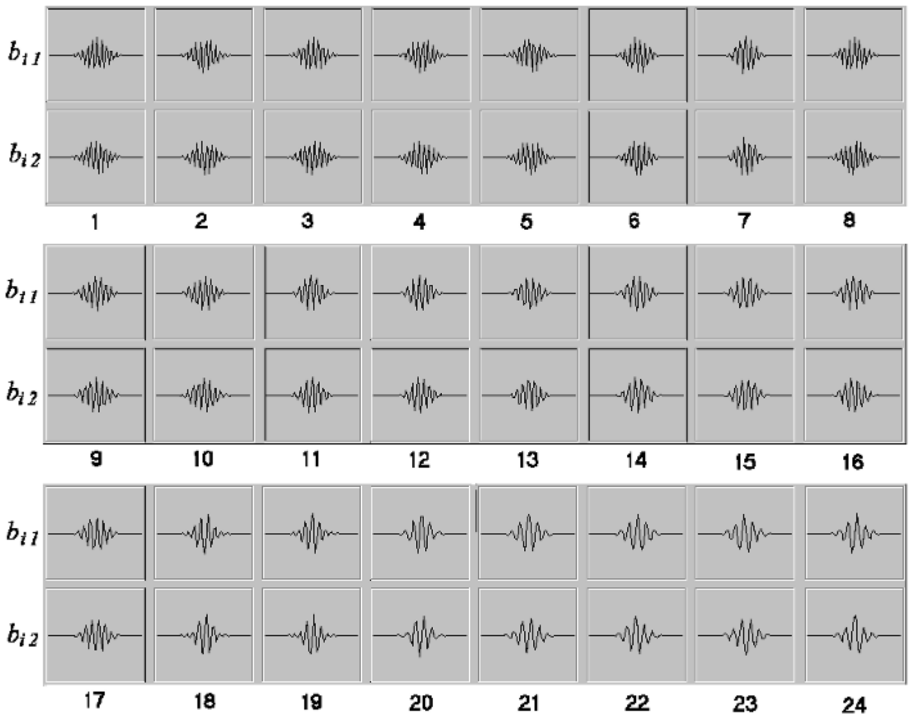


Fig. 7.11. The weights of the converged subspace map trained on sound data.

Invariant Image Filters

The network was trained on the bars data set (Chapter 5) but with two differences:

1. There is only a single bar in each image.
2. Eight different bar orientations were used (oriented at regular intervals between 0 and $\frac{7\pi}{8}$ radians from the vertical orientation.) In each episode, the orientation of the bar was held constant while its position was varied. A Gaussian weighting function was used at the inputs with a full width half maximum (FWHM) of sixteen pixels. This was to ensure that the orientation filters were centred on the sampling lattice.

A trained map is shown in Fig. 7.12. The diagram shows that each weight vector has positive and negative values in areas close to the centre of the sampling lattice and that weights of similar magnitudes are arranged along one direction. Pairs of weight vectors in the same module share the same direction. It is interesting to note that these vectors look similar to Gabor–type wavelets since they are localised and they exhibit orientation preferences. The Subspace Algorithm guarantees that trained weight vectors are orthogonal and therefore, within each module, the subspaces defined by the orthogonal Gabor–type wavelet pairs eliminate sinusoidal oscillations and are approximately invariant to translations. This is further discussed in the following section.

It is also important to point out that modules that are close together in the map have captured similar orientations and therefore the map has been smoothly ordered by the training process.

After training, the network was tested on the same data used for training but using only a single image in each feed-forward operation i.e. without using episodes of input vectors. The node with the highest activation in the quadratic output node was then selected as the winner (7.29) and the position of this node in the map shows the detected orientation of the bar in the inputs:

$$c_r = \arg\max_i Q^{(i)} = \arg\max_i \left\{ \sum_j {y_j^{(i)}}^2 \right\} \tag{7.29}$$

Fig. 7.13 shows the results from orientation selection using the trained network. In this example, the same module has the highest activation for every bar with the exception of two. These two correspond to lines that are far from the centre of the filter and reliability of the classification is therefore lower. (See Section 7.4.3.) No module has been activated significantly more than any other for these bars and they can be regarded as unclassified data rather than misclassified. The other bars clearly activated one group of modules significantly more than others and their orientations have therefore been successfully classified.

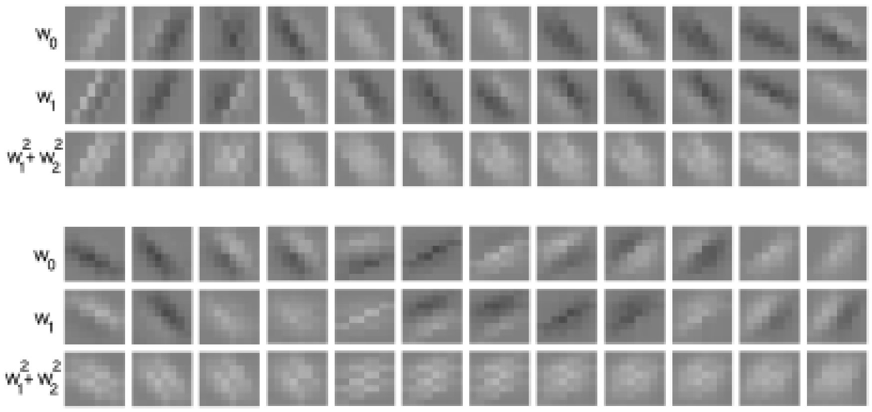


Fig. 7.12. The weights of a trained map. White pixels indicate positive weights and black pixels indicate negative. The weight vectors within each module show the same orientation and neighbouring modules have similar orientations. The vectors showing the sum of squared weights are approximately Gaussian and this confirms that the sinusoidal oscillations in the individual weight vectors can be eliminated (see Section 7.4.3).

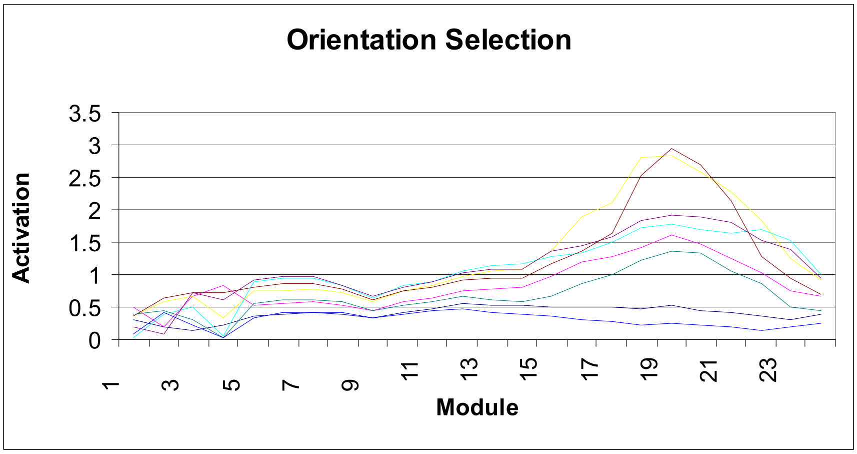


Fig. 7.13. Module activations plotted against the module number for several bars with the same orientation but with different positions. The winning module is in position 19 for every bar position except the outermost two.

7.4.3 Discussion

Assuming that the trained weight matrices are close approximations to orthogonal wavelet pairs, then we can simplify the analysis by considering one-dimensional cosine and sine wavelets respectively defined by:

$$\psi_c(t,\omega) = e^{-\frac{\omega^2 t^2}{2\sigma^2}} \cos(\omega t), \psi_s(t,\omega) = e^{-\frac{\omega^2 t^2}{2\sigma^2}} \sin(\omega t) \tag{7.30}$$

where t is time, ω is the angular frequency and σ is the variance of the Gaussian weighting component. An input signal, $\mathbf{x}$, is used and the output, Q, is given by :

$$Q = \sum_i (x_i \psi_c(i,\omega))^2 + (x_i \psi_s(i,\omega))^2 \tag{7.31}$$

$$= \sum_i \left(x_i e^{-\frac{\omega^2 i^2}{2\sigma^2}} \cos(\omega i)\right)^2 + \left(x_i e^{-\frac{\omega^2 i^2}{2\sigma^2}} \sin(\omega i)\right)^2 \tag{7.32}$$

$$= \sum_i x_i^2 e^{-\frac{\omega^2 i^2}{\sigma^2}} \left(\sin^2(\omega i) + \cos^2(\omega i)\right) \tag{7.33}$$

$$= \sum_i x_i^2 e^{-\frac{\omega^2 i^2}{\sigma^2}} \tag{7.34}$$

Therefore, sinusoidal oscillations are eliminated from (7.33) leaving only the Gaussian component, and the subspace defined by the orthogonal wavelets gives time–shift invariance. In the case of two dimensional complex Gabor wavelets, using x and y in place of the time variable t, Q is given by:

$$Q = \sum_x \sum_y e^{-\frac{\omega^2 (x^2+y^2)}{\sigma^2}} \tag{7.35}$$

Consider a bar in the input data to be a plane, v, perpendicular to the $x - y$ plane with orientation θ_1. When this plane coincides with a Gabor function with orientation θ_2, the result is a curve. The integral of this curve indicates the difference between the orientations of the plane and the Gabor filter. This is effectively the same as calculating the weighted sum of an input vector where the input is a straight line in visual data and the weight matrix is similar to a complex Gabor function. The integral is described by (7.36):

$$\int e^{\frac{-\omega^2 v^2}{2\sigma^2}} \cos(v \sin(\theta_1 - \theta_2)) dv \tag{7.36}$$

This integral is always at its highest absolute value when $\theta_1 = \theta_2$. For straight lines with orientation θ_2, some positions may give a zero value for the integral in (7.36) since the wavelet amplitude can be zero along that line. However, an orthogonal wavelet will coincide with a maximum absolute value along the same straight line and the sum of squares of these products eliminates any sinusoidal oscillations, as previously explained. Therefore, a pair of orthogonal Gabor wavelet filters will give the highest response to straight lines with orientation similar to that of the wavelet, regardless of the line position.

7.5 The Negative Feedback Coding Network

The various algorithms describing learning within the negative feedback network in previous chapters have been shown to extract the maximum information from sets of stochastic data. The next obvious question is to decide what a network should do with such information when it has been extracted. Some form of coding would be helpful in classifying such data.

The negative feedback coding network was developed in appreciation of the way in which Carlson [21] amended the basic network of Rubner and Schulten [158], a PCA network, in order to create a coding network.

While we have wished to emulate his success, we have the continuing design ethos based on the retention of as many of the attractive features of the basic negative feedback network as possible – those of simplicity, homogeneity, locality of information use and parallelism.

Our aim is to create a network which will take a set of raw data and code it so that different sections of the data are coded differently and such that data which have the greatest similarity are most alike in codes i.e. a topology-preserving network. A binary code is easiest to implement with a simple threshold. Since we require several bits for each codeword, we suggest that each input be connected to a set of coding output neurons. Raw data at the $\mathbf{x}$ input is converted to a binary coded vector $\mathbf{y}$ at the coding output neurons. There is only one major difference between this network and those investigated previously: each output neuron has a threshold above which its weighted inputs must sum in order to force a positive firing; if the threshold is not reached, the negative feedback will have a negative activation. For simplicity in exposition, we will consider only a scalar input, x. In detail the algorithm is:

- Set the residual at time 0 to be equal to the input, i.e. $e(0) = x$.
- For each output neuron,
  1. Calculate $w_i e(i)$ and set
  $$\begin{aligned} y_i &= 1 \text{ if } w_i e > 0 \\ &= -1 \text{ if } w_i e < 0 \end{aligned}$$
  2. Calculate the new residual
  $$e(i+1) = e(i) - v_i y_i$$
  3. Update weights w_i and v_i according to the same rule as previously, i.e. a simple Hebbian learning rule.
  $$\begin{aligned} w_i &= w_i + \eta e y_i \\ v_i &= v_i + \eta e y_i \end{aligned}$$

Note that the y_i values are either 1 or -1; changing the weights is the sole method of learning in the system to ensure that appropriate codes are found.

Each output neuron, in turn, receives a weighted sum (in this case of only 1) of the e values; however, each output neuron has a threshold above which its activation will have a positive value and below which the activation will be negative. We choose the threshold for *all* output neurons to be 0.

We wish to emphasise that we have not programmed a threshold nor any specific function which monitors the variance of the data and adjusts the network's response appropriately. We have retained an extremely simple network with only a single modification to the previous network.

7.5.1 Results

A typical set of results is shown in Table 7.2. These results are for a 6–output–neuron network which is learning from a set of x–values generated from a uniform distribution[2] between 2 and 4. The network used a learning rate of 0.01 and ran for 10 000 iterations.

Table 7.2. A section of coding for vectors produced by a 1–input 6–output neuron network for input data from a uniform distribution between 2 and 4. Learning rate = 0.01, number of trials = 10 000. We have replaced −1 with 0 to highlight the binary nature of the code.

Decimal		Decimal	
2.0	1 0 0 0 0 0	3.0	1 0 1 1 1 1
2.1	1 0 0 0 0 1	3.1	1 1 0 0 0 1
2.2	1 0 0 0 1 1	3.2	1 1 0 0 1 1
2.3	1 0 0 1 0 0	3.3	1 1 0 1 0 0
2.4	1 0 0 1 1 0	3.4	1 1 0 1 1 0
2.5	1 0 1 0 0 0	3.5	1 1 0 1 1 1
2.6	1 0 1 0 0 1	3.6	1 1 1 0 0 1
2.7	1 0 1 0 1 1	3.7	1 1 1 0 1 0
2.8	1 0 1 1 0 0	3.8	1 1 1 1 0 0
2.9	1 0 1 1 1 0	3.9	1 1 1 1 1 0

Table 7.3. The weights which the above network learned using only simple Hebbian learning

output neuron	1	2	3	4	5	6
Weight (w)	3.005	0.505	0.254	0.123	0.063	0.031

Several points are worth noting:

- First the coding seems fairly inefficient in that the first figure is always 1. This is due to our insistence that all means are zero. Thus the first code element is always 1 for inputs > 0 (see Section 7.5.2).

[2] We use a uniform distribution here to make it clear why each weight has converged to the actual value to which it has converged.

- If we wish a code where the first output neuron performs maximum discrimination, (i.e. in the above example, all inputs less than 3 would be coded as -1, all inputs > 3 would be coded as $+1$) we would use a threshold which will also learn; a rule such as

$$\theta_j = \theta_j + \alpha y_j$$

  where θ_j is the threshold for the j^{th} output neuron is an entirely local rule and easy to implement; however, in keeping with our design philosophy of maintaining simplicity, we have not implemented that here.
- The code is topology preserving – similar inputs have similar outputs (see Section 7.5.4).
- Experiments have shown that larger networks have no difficulty in providing more detailed codes and require only a slight increase in time as each element of the coding is done on the error remaining after the previous output neurons have performed their coding.
- A topological feature map using the negative feedback network has one major advantage over e.g. a Kohonen feature map: it can easily be re-implemented to show a hierarchy of subfeatures (see Section 7.5.6) by adding a new level of coding output neuron.
- Lower–valued digits are automatically coded more slowly and hence are less prone to an unusual input, an outlier, creating large changes.

7.5.2 Statistics and Weights

We will investigate what the weights are actually learning by considering their values at convergence. In general, we are using simple Hebbian learning, so

$$\Delta w_i = \eta e(i) y_i$$

where the subscript denote the i^{th} output neuron and $e(i)$ denotes the value of e at time i. The proof that $v_i = w_i$ is similar to that shown in Chapter 3 and will not be repeated here. We investigate two interesting cases before looking at the general case:

1. A zero mean symmetric distribution.
   - Consider w_1, the weight to the first output neuron. Then,

$$\begin{aligned}\Delta w_1 &= \eta e(1) y_1 \\ &= \eta (x - w_1 y_1) y_1 \\ &= \eta (x y_1 - w_1) \end{aligned} \tag{7.37}$$

   since $y_1^2 = 1$. So at convergence

$$E(\Delta w_1) = 0 \Longleftrightarrow E(w_1) = E(x y_1) \tag{7.38}$$

With a zero mean symmetric distribution, $y_1 = -1$ when x is negative and $y_1 = +1$ when x is positive; therefore $y_1 x = |x|$ and so

$$w_1 = \overline{|x|}$$

at convergence i.e. w_i converges to the expected value of the absolute value of the input data, i.e. the mean absolute value. This has the effect of mapping the two halves of the distribution to a tighter (bipolar) distribution which is mostly contained within the interval $[-\overline{|x|}, \overline{|x|}]$.

2. A positive, compact distribution.
By compact, we mean a distribution which contains no holes.
   - Consider w_1; as before, we can show that, at convergence,

$$E(\Delta w_1) = 0 \Longleftrightarrow E(w_1) = E(xy_1) \tag{7.39}$$

   With this distribution, $y_1 = +1$ for all input values of x. So

$$w_1 = \overline{x}$$

   i.e. the mean value of the input data
   - Now consider w_2, the weight to the second output neuron. Then,

$$\begin{aligned}\Delta w_2 &= \eta e(2) y_2 \\ &= \eta(e(1) - w_2 y_2) y_2 \\ &= \eta(e(1) y_2 - w_2)\end{aligned} \tag{7.40}$$

   since $y_2^2 = 1$. But

$$\begin{aligned}e(1) &= x - w_1 y_1 \\ &= x - \overline{x}\end{aligned}$$

   Now if $x > \overline{x}$, i.e. $x > w_1$, then $y_2 = 1$ while if $x < \overline{x}$, i.e. $x < w_1$, then $y_2 = -1$ Thus w_2 at convergence equals $\overline{|x - \overline{x}|}$. Therefore, at convergence, w_2 is bisecting the residual area of the distribution after w_1 has subtracted out the mean. So w_2 for this distribution is performing the task which w_1 performed for the symmetric distribution.

3. In general, for values of x drawn from any distribution

$$\begin{aligned}\Delta w_i &= \eta e(i) y_i \\ &= \eta(e(i-1) - w_i y_i) y_i \\ &= \eta(e(i-1) y_i - w_i)\end{aligned}$$
$$\text{Therefore, } w_i \rightarrow E(e(i-1)y_i) \text{ at convergence}$$

and if $e(i-1) > 0$ then $y_i = 1$
while if $e(i-1) < 0$ then $y_i = -1$.
Therefore, at convergence, $w_i = \overline{|e(i-1)|}$

In general, the y values correspond to the coding taking place while the e values represent the error after the coding has taken place.

The coding of a uniform distribution is shown in Figure 7.2. We use a uniform distribution this time to make it easy to corroborate that the network is performing an efficient coding; note from Table 7.3 that $w_1 = \overline{|x|} = \overline{x}$ and $w_2 = \overline{|x - \overline{x}|}$, etc..

7.5.3 Reconstruction Error

We show that, if we use the vectors produced by this method to reconstruct the original vectors, we can make the expected absolute reconstruction error from a final code (sometimes called the mean quantization error) arbitrarily small by simply adding new coding output neurons. We will devote a chapter (Chapter 8) to discussing optimising the learning rules of the negative feedback network with respect to the probability density functions of reconstruction errors or residuals. Let us assume that we cannot i.e that there exists an $\epsilon > 0$ such that all mean absolute reconstruction errors are greater than ϵ. We note from the above that this is equivalent to showing that the value of $w_i > \epsilon$ for all i.

Note that for a finite distribution, the maximum possible error after the first coding is

$$E_{Max} = \max(|x_{Max} - \overline{|x|}|, |x_{Min} - \overline{|x|}|, \overline{|x|})$$

where x_{Max}, x_{Min} is the largest (resp. smallest) possible member of the distribution. Thus E_{Max} is finite.

Consider a particular input x. From the results in the last section, because the system is creating the coding at y and subtracting the weighted coding at e, the value $e(i)$ is simply the error between the code found by the first i coding output neurons and the input x after i codings have taken place. Therefore $\overline{|e(i)|}$ is the mean absolute error after i codings of the input. Now,

$$\begin{aligned} e(i) &= e(i-1) - w_i y_i \\ &= e(i-1) - \overline{|e(i-1)|} y_i \end{aligned}$$

If $e(i-1) < 0$, $y_i = -1$ and

$$\begin{aligned} e(i) &= e(i-1) + \overline{|e(i-1)|} \\ &= -|e(i-1)| + \overline{|e(i-1)|} \end{aligned}$$

while if $e(i-1) > 0$, $y_i = +1$ and

$$\begin{aligned} e(i) &= e(i-1) - \overline{|e(i-1)|} \\ &= |e(i-1)| - \overline{|e(i-1)|} \end{aligned}$$

Therefore the amplitude of $e(i)$ is the difference between the absolute value of $e(i-1)$ and the mean absolute value of $e(i-1)$.

Thus

$$\begin{aligned}|e(i)| &= |\{|e(i-1)| - \overline{|e(i-1)|}\}| \\ &= |\{|e(i-1)| - w_i\}| \\ &< ||e(i-1)| - \epsilon|\end{aligned}$$

since all terms are positive. Therefore the absolute error at each stage is decreasing by more than ϵ. Therefore the mean absolute error is also decreasing by at least ϵ at each stage. Now the initial maximum error is E_{Max} which is finite and the absolute error is decreasing by a finite amount each time and so cannot remain above ϵ for all time.

Therefore, we can make the quantization error arbitrarily small by continuing the coding for a sufficient number of output neurons.

7.5.4 Topology Preservation

In stating that we have a topology preserving coding, we mean that similar inputs should be projected onto similar outputs and similar outputs should be the representations of similar inputs. This is only approximately true, in general, of feature maps using neural nets e.g. a Kohonen [110] map attempts to project the input space onto a network in such a way that the most essential neighbourhood relationships between data in the input space are preserved. Yet input data can be constructed which do not permit a 2-D (or 3-D) mapping to adequately represent all topological equivalences in the data. We give an intuitive notion of topology preservation here; a more formal proof is given in the next section.

Consider an n-unit coding of a set of input values. Let a particular point x be represented by $\mathbf{y}_i$ where $\mathbf{y}_i = \{y_{i1}, y_{i2}, ..., y_{in}\}$. The subscript i denotes an ordering of the $\mathbf{y}$ values (i.e. of the coding) such that $\mathbf{y}_i < \mathbf{y}_j$ for all $i < j$.

Then any input $x + \Delta x$ is represented by the vector $\mathbf{y}_i$ or by $\mathbf{y}_{i-1}$ or $\mathbf{y}_{i+1}$ for all Δx such that $|\Delta x| < w_n$, the n^{th} weight. i.e. similar inputs are represented by similar outputs.

Now consider two distinct input values, x and u, which are represented by the same $\mathbf{y}_j$. Then

$$\begin{aligned}x &= \sum_{i=1}^{n} w_i y_{ji} + \Delta x \\ u &= \sum_{i=1}^{n} w_i y_{ji} + \Delta u\end{aligned}$$

where Δx and Δu are the errors in the representations after the first n codings have taken place. Therefore,

$$\begin{aligned}x - u &= \Delta x - \Delta u \\ &\leq 2w_n\end{aligned}$$

From the previous section, we know that the value of w_n can be made arbitrarily small and so a single code can be made to represent only similar values. Clearly a similar argument will show that if the values, x and u, are represented by contiguous codes, the values x and u can only be at most $3w_n$ apart. Thus, similar codes represent similar values.

7.5.5 Approximate Topological Equivalence

We will use the results from Section 7.5.3 in this proof: recall that for every $\epsilon > 0$, there exists an n such that $w_n < \epsilon$.

Let M be the metric space defined by that (sub)set of the real numbers defined by the probability distribution of the raw data and the metric, d, defined by the usual Euclidean distance metric.

Let M_1 be the metric space defined by the set of codes (of the real numbers in M) and the metric, d_1, defined by the usual Euclidean distance metric (now calculating in binary).

It is not possible to prove that there is a topology-preserving function which maps M to a particular value of M_1; however, it is possible to prove that there exists a mapping from the set M to a member of the family $\{M_1^1, M_1^2, M_1^3, ..., M_1^n, ...\}$ where M_1^n is the coding which has length n (i.e. formed by using n coding output neurons). In other words, we can make our mapping as close to a topology preserving mapping as possible by choosing n appropriately.

We shall create an ordering, $\{C_i\}$ of the codes in M_1^n based on the size of their binary values. Thus $C_i < C_j$ for $i < j$. Note that for this coding on M_1^n, if $f(x)$ is the function which codes the inputs i.e. $f : M \rightarrow M_1$, then $f^{-1}(C_i) - f^{-1}(C_{i-1}) = w_n$, the weight of the n^{th} level of the coding. Note also that the greatest distance between values coded by the same code is also w_n.

- First consider the mapping $f : M \rightarrow M_1$.
  Then take any point $c \in M$. $\forall \epsilon > 0$, we require to prove that $\exists$ a $\delta > 0$ such that
  $$d_1(f(c), f(x)) < \epsilon$$
  $\forall x \in M : d(c, x) < \delta$. Choose the coding M_1^n such that $w_n < \frac{1}{2}\epsilon$. Let $\delta = w_n$. Let f map c to class C_i, the i^{th} class of M_1^n. Then for all x such that $d(c, x) < \delta = w_n$, $f(x)$ is either in class C_i or in one of its neighbours C_{i-1} or C_{i+1}. So $f(x)$ is within $2w_n$ of $f(c)$ i.e. $f(x) \in (f(c) - \epsilon, f(c) + \epsilon)$, when $d(x, c) < 2w_n$, i.e.
  $$d_1(f(c), f(x)) < \epsilon \text{ when } d(c, x) < \delta$$
- Now consider the mapping $f : M_1 \rightarrow M$.
  Then take any point $C_i \in M_1$. Given any $\epsilon > 0$, we must prove that
  $$\exists \delta > 0 : d(f(C_i), f(x)) < \epsilon, \forall x \in M : d_1(C_i, x) < \delta$$

Choose n such that $w_n < \frac{1}{2}\epsilon$; this defines the actual representative of M_1 as M_1^n. We chose δ to be equal to 1. Then $\forall x \in (C_i - \delta, C_i + \delta)$ is equivalent to $x \in C_{i-1}, C_i$ or C_{i+1}.
Now the maximum distance between the values which code to C_{i-1} or C_{i+1} and those which map to C_i is $2w_n$ i.e. two times the remaining error after the n^{th} coding. i.e. $d(f(x), f(c_1)) \leq 2w_n < \epsilon$ for all x in the declared interval.

Now any particular negative feedback coding network is not truly topology preserving; however, it can be made arbitrarily close to such a network by increasing the length of the coding. Therefore any particular coding is performing an approximate topology-preserving mapping.

We define an ϵ_0*-topology preserving network* as a negative feedback network in which for all points $c \in M$, $\forall \epsilon > \epsilon_0, \exists \delta > 0 : d_1(f(c), f(x)) < \epsilon, \forall x \in M : d(c, x) < \delta$ where $f(x), f(c)$ are the binary codes.

Note that this is equivalent to defining $w_n = \frac{1}{2}\epsilon_0$. Then each network in the sequence of negative feedback networks of increasing discrimination is an ϵ_0-topology preserving network with the value of ϵ_0 defined as $2w_n$ for the specific mapping M_1^n.

7.5.6 A Hierarchical Feature Map

We now propose the complete network composed of two separate negative feedback networks, rather like the EPP network in Chapter 6. The first half of the network is the basic negative feedback network described in Section 3.3 which will perform PCA; the right half of the network is the negative feedback coding network described in Section 7.5. The first section will extract the maximum information from the raw input data i.e. data will be projected onto those directions which contain maximum information; the second section will code the data along each dimension independently. All parts of the system use unsupervised learning. Our learning rule continues to be simple Hebbian learning with no weight decay or clipping of weights.

Since the network is topology preserving in each direction, it can be shown to be topology preserving in the space spanned by these directions. Therefore the construction can be viewed as forming a feature map which is topology preserving in the major information directions of the data. While it is possible to create continuous multidimensional maps which are topologically different from their projection onto this subspace, each such anomaly will tend to swing the principal components in the direction of the anomaly suggesting that such anomalies can at most be a minor part of the input data. Further, as will be shown, augmenting this type of feature map to take account of such features is a simple process.

The inherently modular nature of the network allows us to consider the effects of augmenting the network as a purely local process. This modular nature is a direct consequence of the Principal Component Analysis performed by the first section which leads to orthogonal input vectors for the second section.

Augmenting a Map

The desire to augment a map may be brought about by two circumstances:

1. The map is too crude since too little information has been extracted from the original input data.
   To extract more information from the raw data, we must find a new Principal component along which to project the data. Therefore, we must create a new data extraction output neuron i.e. in the central layer of the network. Now, by adding our new output neuron at the end of the learning process described in (3.22), (3.23) and (3.24), we are not disturbing the learning in any of the other directions which have already been found. Therefore the new direction can be found without disturbing the principal components already found; therefore the existing codes are not disturbed and the coding of the new dimension can be done independently of the existing codes.
2. The map is too crude since too little discrimination has taken place in a particular direction
   Note again that the modular nature of the map allows the discrimination in each direction to be modified independently of that in the other directions. Further in the coding within a particular direction, we may simply add a new coding output neuron into the network and, provided it learns after all the others have learned, it will simply learn to bisect the remaining information after the others have subtracted their activations. In other words, a new coding output neuron will simply provide increased discrimination within that direction and will affect neither the coding in other directions nor the existing coding in its direction.

Experimental results have confirmed this analysis.

Note that the potential for improving a map after it has been constructed is an improvement on the Kohonen map whose parameters must be specified in advance.

7.5.7 A Biological Implementation

It is well known that biological neurons are not accurately modelled by either simple linear summation neurons or by neurons which have step functions as activation functions. Instead a degree of nonlinearity of response has been found which is usually modeled by a sigmoidal function.

Using tanh() as an activation function for the output neurons allows a unified model of the above two types of output neurons to be created:

- tanh() is approximately linear in its middle section. We may adjust the range of middle section over which it is linear by using the parameter λ in $y_i = \tanh(\lambda \sum_j w_{ij} x_j)$. To get a large linear section requires a small value of λ. Experimental results have confirmed that $\lambda = 0.1$ is sufficiently small to approximate a linear function with which Principal Components can be found as before.
- tanh() may be made more dichotomous by adjusting the parameter, λ, upwards. A value of $O(10)$ is sufficient to create an output neuron which performs a smoothed coding of the input data (but see below). The larger the value of λ, the more step–like the function becomes.

Therefore a single type of output neuron with different λ parameters can perform both the information extraction and the information coding described above. Both sets of output neurons will have an activation function, $\tanh(\lambda \sum_j w_{ij} x_j)$. The sole difference is that output neurons in the first layer have the parameter $\lambda = 0.1$, while those in the second have the parameter $\lambda = 10$. The first value is, of course, dependent on the distribution of the input data and is based on distributions with single figure standard deviation and mean; the second depends on the amount of discrimination (length of the code) required.

Using such an activation function with the information extracting output neurons has several implications:

1. The output values, $\mathbf{y}$, at these output neurons are all in the range $(-1,1)$
2. Thus the weights in the first part of the network will grow much larger than previously
3. The learning rate in this network can be made much larger than before as the $\mathbf{y}$ values are constrained to this small range. Previous networks used a learning rate of $O(0.0001)$; with a tanh() activation function a learning rate of $O(0.1)$ is possible.

However, there is a drawback to the use of this activation function in the coding layer: with the values stated, codes for the first three coding output neurons in each direction agree with those found with the step function network. However, for the later coding output neurons a lack of discrimination develops leading to very imprecise codes. By the 6^{th} coding output neuron, we have lost the precision necessary for a topology-preserving network.

7.6 Conclusion

Three methods of creating topology-preserving feature maps have been presented: the first depends on an introduction of competition to the existing negative feedback network; the second used subspaces of clusters of the data; the third is a two stage process involving projection of the input data onto the directions of maximum information followed by a discrimination process within each direction.

The Scale Invariant network ignores the magnitude of the input vectors and quantises only on the directional information. This is a useful quantisation when only the relative proportion of each of several input data streams in important not the actual magnitude of the signal. Our original experiments with speech data were because humans can decode speech at a range of different volumes.

The Subspace map envisages a stream of data and was motivated by Kohonen's ASSOM. In this type of mapping, we wish to be able to decode a mapping while ignoring transformations which are not material to the mapping. Thus we may build in translation or rotation invariance into a code. The conjecture for both the ASSOM and the Subspace map of this chapter is that certain typical transformations of a data set define a low–dimensional manifold in which a coded object may be found. With these type of maps, we often find filters which perform wavelet–like operations.

In the coding network, the simple negative feedback network performs the projection while the negative feedback coding network performs the discrimination. Both use only simple Hebbian learning with no normalisation, weight decay or clipping of weights. The sole difference between the two sets of output neurons is that the coding output neurons have a threshold which must be achieved before their activation becomes positive. An attempt to produce a biologically feasible unification of the two types of output neurons was partially successful in that a single type of output neuron with different values of a parameter in its activation function has been shown to be capable of performing both tasks necessary to produce the feature map; the single addition of an activation function $\tanh(\lambda x)$ allowed us to dispense with the threshold in the coding output neurons.

We have, in other contexts [123], made a comparison of different types of topology-preserving networks and found, unsurprisingly, that the value of different coding networks depends on the use that will be made of the coding. Thus it is not possible to say that e.g. the Scale Invariant map is better than the Subspace map in a general sense. One can only be compared with another with respect to the specific use which will be made of the coding.