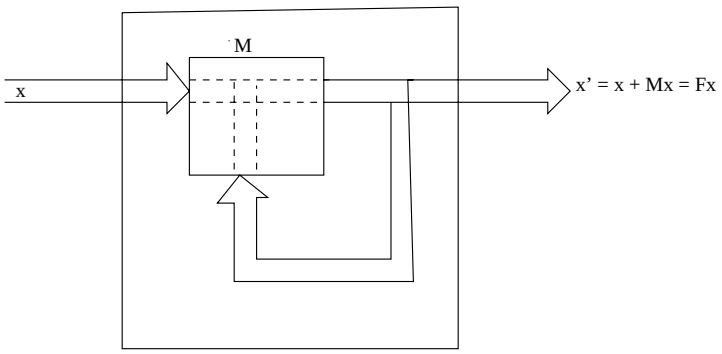

Figure 5.4: The System Model of the Novelty Filter

"It is tentatively assumed $(I-M)^{-1}$ always exists." Kohonen shows that, under fairly general conditions on the sequence of $\mathbf{x}$ and the initial conditions of the matrix M, the values of F always converge to a projection matrix under which the output $\mathbf{x}^{'}$ approaches zero although F does not converge to the zero matrix i.e. F converges to a mapping whose kernel is the subspace spanned by the vectors $\mathbf{x}$. Thus any new input vector $\mathbf{x_1}$ will cause an output which is solely a function of the novel features in $\mathbf{x_1}$.

Thus the "Novelty Filter" knows what patterns it has seen and extracts information from any pattern which is equivalent to patterns which it has seen; the residuals are the "novelties".

5.6 Competitive learning

One of the non-biological aspects of the basic Hebbian learning rule is that there is no limit to the amount of resources which may be given to a synapse. This is at odds with real neural growth in that it is believed that there is a limit on the number and efficiency of synapses per neuron. In other words, there comes a point during learning in which if one synapse is to be strengthened, another must be weakened. This is usually modelled as a competition for resources.

In competitive learning, there is a competition between the output neurons after the activity of each neuron has been calculated and only that neuron which wins the competition is allowed to fire. Such output neurons are often called **winner-take-all** units. The aim of competitive learning is to **categorize** the data by forming **clusters**. However, as with the Hebbian learning networks, we provide no correct answer (i.e. no labelling information) to the network. It must self-organise on the basis of the structure of the input data. The method attempts to ensure that the similarities of instances within a class is as great as possible while the differences between instances of different classes is as great as possible.

Hertz *et al* point out that simple competitive learning leads to the creation of **grandmother** cells, the proverbial neuron which would fire if and only if your grandmother hove in sight. The major difficulty with such neurons is their lack of robustness: if you lose your grandmother cell, will you never again recognise your grannie. In addition, we should note that if we have N grandmother cells we can only recognise N categories whereas if we are using a binary code, we could distinguish between 2^N categories.

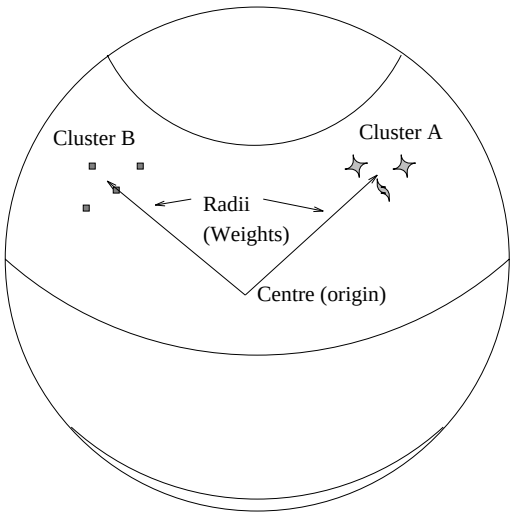

Figure 5.5: The input vectors are represented by points on the surface of a sphere and the lines represent the directions to which the weights have converged. Each is pointing to the mean of the group of vectors surrounding it.

5.6.1 Simple 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 future should a similar input be given to the network. We first have the activity transfer equation

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

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

$$\Delta w_{ij} = \eta(x_j - w_{ij}), \text{ for the winning neuron i}$$

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 $\Delta w_{ij} \to 0 \iff w_{ij} \to \langle x_j \rangle$, where the angled brackets indicate the ensemble average. We can actually describe this rule as a variant of the Hebb learning rule if we state that y_i =1 for the winning i^{th} neuron and y_i =0 otherwise. Then the learning rule can be written $\Delta w_{ij} = \eta y_i(x_j - w_{ij})$ i.e. a Hebbian learning rule with weight decay. A geometric analogy is often given to aid understanding simple competitive learning. Consider Figure 5.5: we have two groups of points lying on the surface of the sphere and the weights of the network are represented by the two radii. The weights have converged to the mean of each group and will be used to classify any future input to one or other group.

A potential problem with this type of learning is that some neurons can come to dominate the process i.e. the same neuron continues to win all of the time while other neurons (**dead neurons**) never win. While this can be desirable if we wish to preserve some neurons for possible new sets of input patterns it can be undesirable if we wish to develop the most efficient neural network. It pays in this situation to ensure that all weights are normalised at all times (and so already on the surface of the sphere) so that one neuron is not just winning because it happens to be greater in magnitude than the others. Another possibility is **leaky learning** where the winning neuron is updated and so too by a lesser extent are all other neurons.

This encourages all neurons to move to the areas where the input vectors are to be found. The amount of the leak can be varied during the course of a simulation. Another possibility is to have a variable threshold so that neurons which have won often in the past have a higher threshold than others. This is sometimes known as learning with a **conscience**. Finally noise in the input vectors can help in the initial approximate weight learning process till we get approximate solutions. As usual an annealing schedule is helpful. We now consider three important variations on Competitive learning:

1. Learning Vector Quantisation
2. The ART models
3. The Kohonen feature map

5.6.2 Learning Vector Quantisation

In vector quantisation, we create a codebook of vectors which represent as well as possible the vectors of each class. One obvious way to represent vectors is to use the mean of each class of vectors and competitive learning can be used to find the mean. If we use this method we can be sure that there is no vector codebook of equal length which is better able to represent the input vectors.

Kohonen has suggested a *supervised* quantisation method called learning vector quantisation (LVQ) which means that the classes must be known in advance (so that we have a set of labelled input data). We feedforward the activation through the weights as before but now we are in a position to tell if the correct unit has won the competition. If it has, we use the same learning rule as before but if neuron i has won the competition erroneously we move its vector of weights away from the input pattern. Formally the learning rule becomes

$$\begin{aligned} \Delta w_{ij} &= \eta(x_j - w_{ij}) \text{ , if class i is correct} \\ \Delta w_{ij} &= -\eta(x_j - w_{ij}) \text{ , if class i is wrong} \end{aligned}$$

An improved algorithm called LVQ2 only changes the weights negatively if

1. the input vector is misclassified *and*
2. the next nearest neighbour is the correct class *and*
3. the input vector is reasonably close to the decision boundary between the weights

Note that this is a supervised learning method but we include it in this chapter since it is a variant of Competitive learning.

5.6.3 ART Models

There are a number of ART models all developed in response to what Grossberg calls the **stability-plasticity dilemma**. This refers to the conflicting desires for our neural networks to remain stable in the condition to which they have converged (i.e. to retain their memories of what has been learned) while at the same time being receptive to new learning. Typically we ensure stability by reducing the learning rate during the course of a simulation but this has the obvious effect that the network cannot then respond e.g. to changing input distributions.

The ART models are based on having a mixture of top-down and bottom-up competition. The simplest model can be described as:

1. Select randomly an input from the input distribution

2. Propagate its activation forwards
3. Select the output neuron with greatest activation. If the activation is not high enough, create a new neuron and make its weights equal to the normalised input pattern, $\mathbf{x}$. Go back to step 1.
4. Check that the match between the winning neuron and the input pattern is good enough by calculating the ratio $\frac{\mathbf{w}_i.\mathbf{x}}{||\mathbf{x}||}$. If this does not exceed the vigilance parameter, ρ, the winning unit is discarded as not good enough. It is disabled and the competition is carried out again with the same input over all other output neurons (i.e. go back to step 3).
5. Adjust the weights of the winning neuron using the simple competitive learning rule.

Notice how the algorithm solves the stability-plasticity problem: in order that it can continue to learn we assume a set of unused neurons which are created as needed. On the other hand, once a neuron has learned a set of patterns (i.e. is responding maximally to those patterns) it will continue to respond to them and to patterns like them. It is possible to show that all weight changes stop after a finite number of steps.

Note also the interaction between the two criteria:

- The winning neuron is that which is maximally firing
- The vigilance parameter checks whether the winning neuron is actually close enough to the input pattern.

It is the interaction between these two criteria which gives ART its power. We are actually searching through the prototype vectors to find one which is close enough to the current input to be deemed a good match. A local algorithm with top-down (checking vigilance) and bottom-up (finding winning neuron) neurons has been created.

One major difficulty with ART networks is that convergence can be extremely slow for any problem greater than a toy problem.

5.6.4 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* 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 Figure 5.6)

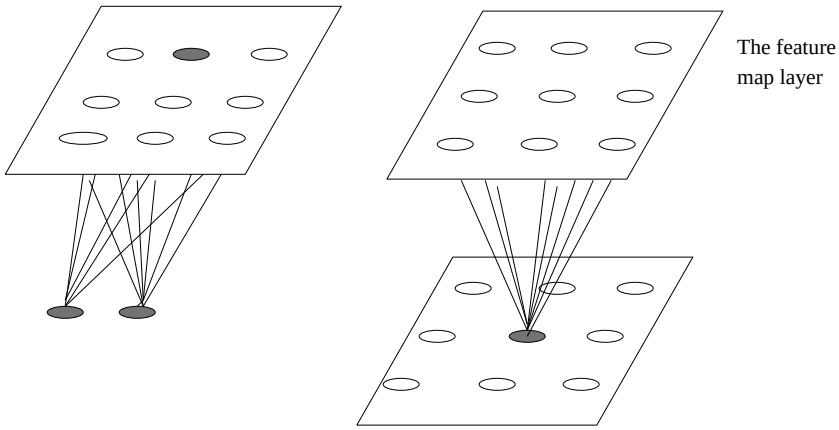

Figure 5.6: Two types of feature maps (a) a map from a small number of continuous inputs (b) a map from one layer spatially arranged to another

- with those maps which take in a broad array of inputs and map onto a second array of outputs (right half of Figure 5.6).

There are several ways of creating feature maps - the most popular is Kohonen's.

Kohonen's algorithm 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; thus if unit i is at point $\mathbf{r}_i$ in the output layer then

$$f(i, i^*) = a \exp(\frac{-|r_i - r_{i^*}|^2}{2\sigma^2}) - b \exp(\frac{-|r_i - r_{i^*}|^2}{2\sigma_1^2}) \tag{5.13}$$

The single Gaussian can be seen in Figure 5.7 while the Mexican Hat function is shown in Figure 5.8. Notice that this means 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 those neurons further away are pushed slightly in the opposite direction.

Results from an example experiment is shown in Figure 5.9. The experiment consists of a neural network with two inputs and twenty five outputs. The two inputs at each iteration are drawn from a uniform distribution over the square from -1 to 1 in two directions e.g typical inputs would be (0.3,0.5), (-0.4,0.9), (0.8,0.8) or (-0.1,-0.5). The algorithm is

1. Select at random an input point (with two values).

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

$$\text{winning neuron, } i^* = \arg\min(\| \mathbf{x} - \mathbf{w_i} \|) \tag{5.14}$$

Use the distance formula to find that distance which is smallest,

$$dist = \sqrt{(x_1 - w_{i1})^2 + (x_2 - w_{i2})^2} \tag{5.15}$$

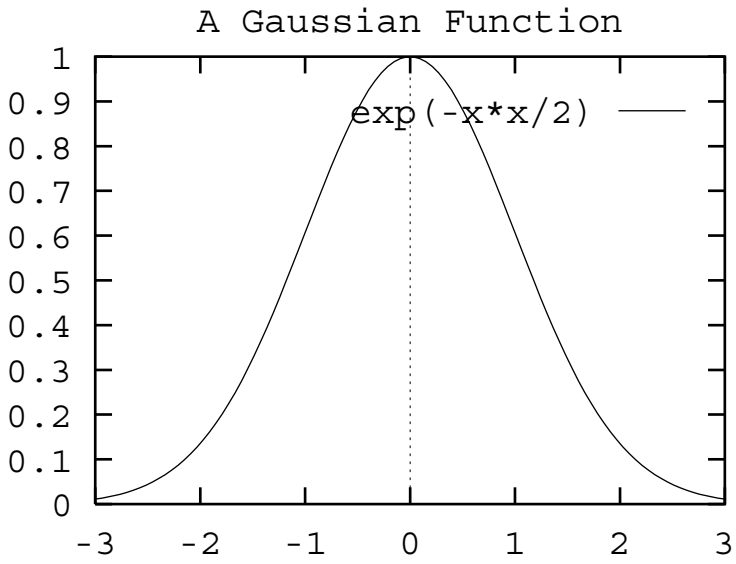

Figure 5.7: The Gaussian function.

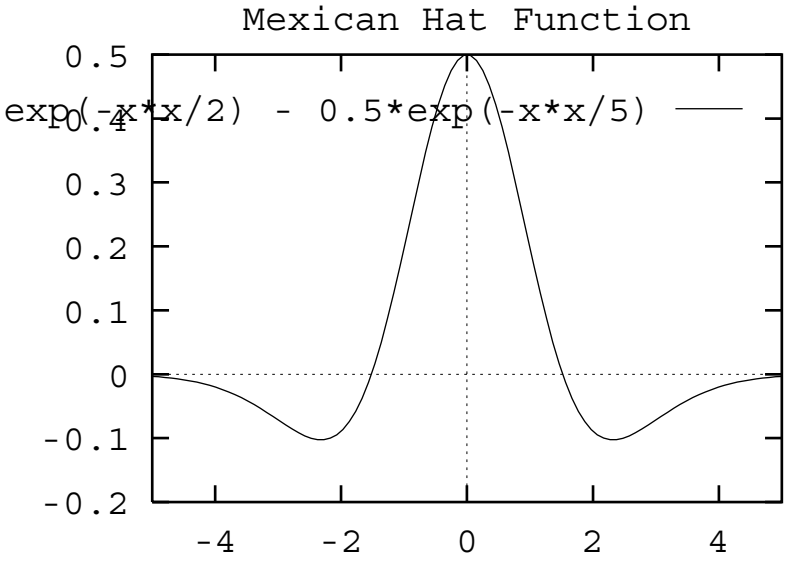

Figure 5.8: The difference of Gaussian function.

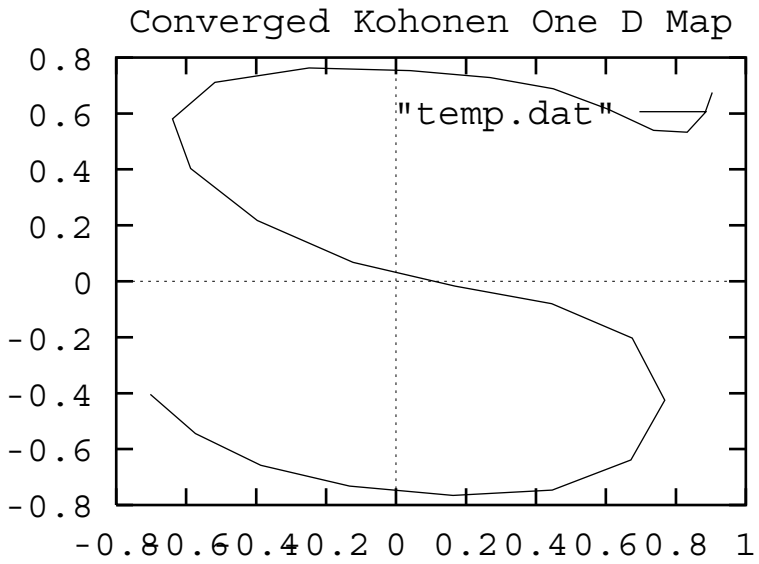

Figure 5.9: A one dimensional mapping of the two dimensional input space

3. Now update all neurons' weights using

$$\Delta w_{ij} = \alpha(x_j - w_{ij}) * f(i, i^*) \tag{5.16}$$

where

$$f(i, i^*) = a \exp(\frac{-|r_i - r_{i^*}|^2}{2\sigma^2}) - b \exp(\frac{-|r_i - r_{i^*}|^2}{2\sigma_1^2}) \tag{5.17}$$

4. 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 (we can be talking about 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 D neighbourhood function) or sometimes a hexagonal grid. An example of such a mapping is shown in Figure 5.10 in which we are making a two dimensional mapping of a two dimensional surface. This is a little redundant and is for illustration only.

Like the ART algorithms, Kohonen feature maps can take a long while to converge. Examples of the converged mappings which a Kohonen SOFM can find are shown in Figure 5.10. We can see that the neurons have spread out evenly when the input distribution is uniform over a rectangular space but with the less regular distribution in the right half of the figure, there are several dead neurons which are performing no useful function - they will respond to areas of the input space which are empty.

5.7 Applications

The SOFM has been used in modelling features of the human brain such as visual maps and somatosensory maps. In terms of engineered applications some of interest are

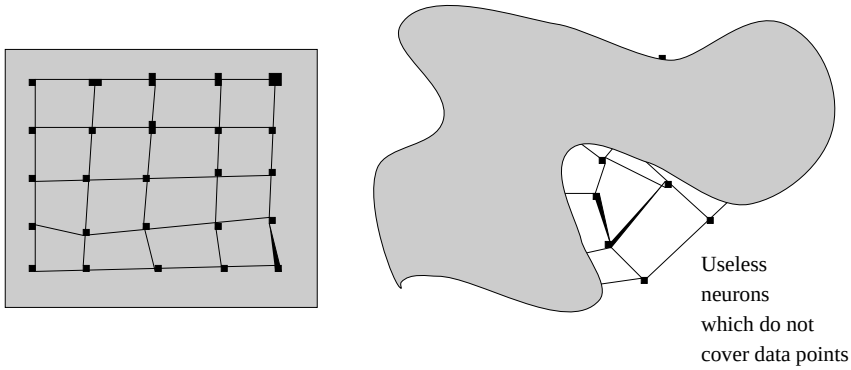

Figure 5.10: A 5*5 feature map which conforms well to a set of inputs drawn from a rectangular distribution but less well to a less regular distribution

1. Vector quantisation in which we wish to identify all vectors close to a particular prototypical vector of a particular class with the prototype. This can be useful e.g. in terms of data compression since if we transmit the code for the prototype vector (i.e. the information that it is prototype 3 rather than 2 or 4) we may be giving enough information for the vector to be reconstructed at the receiving location with very little loss of information. This can be very useful e.g. in image reconstruction where the bandwidth necessary to fully reconstruct an image may be prohibitively high.
2. Control of robot arms. Inverse kinematics problems can be extremely computationally expensive to solve. A SOFM allows an unsupervised learning of a set of standard positions of the robot end-receptors in terms of the positions of the individual angles and lengths of levers which make up a robot arm.
3. Classification of e.g. clouds from raw data.

5.8 Exercises

1. Hebbian learning is sometimes defined in terms of a covariance-type rule:

$$\Delta w_{ij}^p = \eta(y_j^p - \langle y_j \rangle)(x_i^p - \langle x_i \rangle) + k \tag{5.18}$$

   where Δw_{ij}^p is the change in the weight w_{ij} with respect to the p^{th} input pattern, y_j^p is the output of the j^{th} output neuron when the p^{th} input pattern is presented and $\langle y_j \rangle$ is the expected value of the j^{th} output neuron over all patterns. By expanding Equation 5.18, compare the parameters with the standard Hebbian rule. (Objectives 1, 2).

2. (Nikovski, 1995) Compute the covariance matrix C of the following two vectors:

$$\begin{aligned} \mathbf{x}_1 &= (3, 4) \\ \mathbf{x}_2 &= (12, 5) \end{aligned}$$

   Calculate the eigenvalues and corresponding eigenvectors of the covariance matrix. Why is only one of them not zero? What is the direction of the first principal component? How many principal components are there for only these two vectors $\mathbf{x}_1$ and $\mathbf{x}_2$. (Objective 2).

3. A conscience mechanism is a mechanism which limits the number of patterns to which a neuron can respond. (A successful neuron develops a conscience and so stops winning so much). Discuss. (Objective 4).

4. (Haykin, p435) It is said that the SOFM algorithm based on competitive learning lacks any tolerance against hardware failure, yet the algorithm is error-tolerant in that a small perturbation applied to the input vector causes the output to jump from the winning neuron to a neighbouring one. Discuss the implications of these statements. (Objectives 4, 5)

5. Discuss the effect of the Mexican hat function on the formation of the activity bubble in the SOFM. Show that the function can be calculated as the sum of two Gaussians. (Objectives 4, 5)

6. Prove that the simple competitive learning rule can be derived as an error descent rule for the function

$$E^P = \frac{1}{2}\sum_j (w_{kj} - x_j^P)^2 \tag{5.19}$$

where k is the winning neuron. (Objective 4)