5

Unsupervised Learning and Clustering Algorithms

5.1 Competitive learning

The perceptron learning algorithm is an example of *supervised learning*. This kind of approach does not seem very plausible from the biologist's point of view, since a teacher is needed to accept or reject the output and adjust the network weights if necessary. Some researchers have proposed alternative learning methods in which the network parameters are determined as a result of a self-organizing process. In *unsupervised learning* corrections to the network weights are not performed by an external agent, because in many cases we do not even know what solution we should expect from the network. The network itself decides what output is best for a given input and reorganizes accordingly.

We will make a distinction between two classes of unsupervised learning: *reinforcement* and *competitive learning*. In the first method each input produces a reinforcement of the network weights in such a way as to enhance the reproduction of the desired output. Hebbian learning is an example of a reinforcement rule that can be applied in this case. In competitive learning, the elements of the network compete with each other for the "right" to provide the output associated with an input vector. Only one element is allowed to answer the query and this element simultaneously inhibits all other competitors.

This chapter deals with competitive learning. We will show that we can conceive of this learning method as a generalization of the linear separation methods discussed in the previous two chapters.

5.1.1 Generalization of the perceptron problem

A single perceptron divides input space into two disjoint half-spaces. However, as we already mentioned in Chap. 3, the relative number of linearly separable Boolean functions in relation to the total number of Boolean functions converges to zero as the dimension of the input increases without bound. There-

fore we would like to implement some of those not linearly separable functions using not a single perceptron but a collection of computing elements.

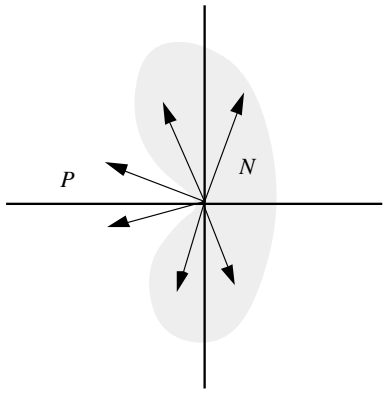


Fig. 5.1. The two sets of vectors P and N

Figure 5.1 shows a two-dimensional problem involving two sets of vectors, denoted respectively P and N. The set P consists of a more or less compact bundle of vectors. The set N consists of vectors clustered around two different regions of space.

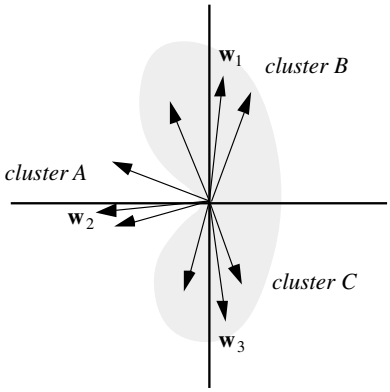


Fig. 5.2. Three weight vectors for the three previous clusters

This classification problem is too complex for a single perceptron. A weight vector $\mathbf{w}$ cannot satisfy $\mathbf{w} \cdot \mathbf{p} \geq 0$ for all vectors $\mathbf{p}$ in P and $\mathbf{w} \cdot \mathbf{n} < 0$ for all vectors $\mathbf{n}$ in N. In this situation it is possible to find three different vectors $\mathbf{w}_1$,$\mathbf{w}_2$ and $\mathbf{w}_3$ which can act as a kind of "representative" for the vectors in each of the three clusters A, B and C shown in Figure 5.2. Each one of

these vectors is not very far apart from every vector in its cluster. Each weight vector corresponds to a single computing unit, which only fires when the input vector is close enough to its own weight vector.

If the number and distribution of the input clusters is known in advance, we can select a representative for each one by doing a few simple computations. However the problem is normally much more general: if the number and distribution of clusters is unknown, how can we decide how many computing units and thus how many representative weight vectors we should use? This is the well-known *clustering problem*, which arises whenever we want to classify multidimensional data sets whose deep structure is unknown. One example ot this would be the number of phonemes in speech. We can transform small segments of speech to n-dimensional data vectors by computing, for example, the energy in each of n selected frequency bands. Once this has been done, how many different patterns should we distinguish? As many as we think we perceive in English? However, there are African languages with a much richer set of articulations. It is this kind of ambiguity that must be resolved by unsupervised learning methods.

5.1.2 Unsupervised learning through competition

The solution provided in this chapter for the clustering problem is just a generalization of perceptron learning. Correction steps of the perceptron learning algorithm rotate the weight vector in the direction of the wrongly classified input vector (for vectors belonging to the positive half-space). If the problem is solvable, the weight vector of a perceptron oscillates until a solution is found.

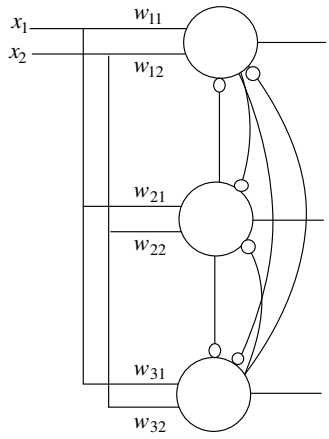


Fig. 5.3. A network of three competing units

In the case of unsupervised learning, the n-dimensional input is processed by exactly the same number of computing units as there are clusters to be individually identified. For the problem of three clusters in Figure 5.2 we could use the network shown in Figure 5.3.

The inputs x_1 and x_2 are processed by the three units in Figure 5.3. Each unit computes its weighted input, but only the unit with the largest excitation is allowed to fire a 1. The other units are inhibited by this active element through the lateral connections shown in the diagram. Deciding whether or not to activate a unit requires therefore *global information* about the state of each unit. The firing unit signals that the current input is an element of the cluster of vectors it represents. We could also think of this computation as being performed by perceptrons with variable thresholds. The thresholds are adjusted in each computation in such a way that just one unit is able to fire.

The following learning algorithm allows the identification of clusters of input vectors. We can restrict the network to units with threshold zero without losing any generality.

Algorithm 5.1.1 *Competitive learning*

Let $X = (\mathbf{x}_1, \mathbf{x}_2, \ldots, \mathbf{x}_\ell)$ be a set of normalized input vectors in n-dimensional space which we want to classify in k different clusters. The network consists of k units, each with n inputs and threshold zero.

start: The normalized weight vectors $\mathbf{w}_1, \ldots, \mathbf{w}_k$ are generated randomly.

test: Select a vector $\mathbf{x}_j \in X$ randomly.
Compute $\mathbf{x}_j \cdot \mathbf{w}_i$ for $i = 1, \ldots, k$.
Select $\mathbf{w}_m$ such that $\mathbf{w}_m \cdot \mathbf{x}_j \geq \mathbf{w}_i \cdot \mathbf{x}_j$ for $i = 1, \ldots, k$.
Continue with *update*.

update: Substitute $\mathbf{w}_m$ with $\mathbf{w}_m + \mathbf{x}_j$ and normalize.
Continue with *test*.

The algorithm can be stopped after a predetermined number of steps. The weight vectors of the k units are "attracted" in the direction of the clusters in input space. By using normalized vectors we prevent one weight vector from becoming so large that it would win the competition too often. The consequence could be that other weight vectors are never updated so that they just lie unused. In the literature the units associated with such vectors are called *dead units*. The difference between this algorithm and perceptron learning is that the input set cannot be classified *a priori* in a positive or a negative set or in any of several different clusters.

Since input and weight vectors are normalized, the scalar product $\mathbf{w}_i \cdot \mathbf{x}_j$ of a weight and an input vector is equal to the cosine of the angle spanned by both vectors. The selection rule (maximum scalar product) guarantees that the weight vector $\mathbf{w}_m$ of the cluster that is updated is the one that lies closest to the tested input vector. The update rule rotates the weight vector $\mathbf{w}_m$ in the direction of $\mathbf{x}_j$. This can be done using different learning rules:

- *Update with learning constant* – The weight update is given by

$$\Delta \mathbf{w}_m = \eta \mathbf{x}_j .$$

  The learning constant η is a real number between 0 and 1. It decays to 0 as learning progresses. The *plasticity* of the network can be controlled in such a way that the corrections are more drastic in the first iterations and afterwards become more gradual.

- *Difference update* – The weight update is given by

$$\Delta \mathbf{w}_m = \eta(\mathbf{x}_j - \mathbf{w}_m).$$

  The correction is proportional to the difference of both vectors.

- *Batch update* – Weight corrections are computed and accumulated. After a number of iterations the weight corrections are added to the weights. The use of this rule guarantees some stability in the learning process.

The learning algorithm 5.1.1 uses the strategy known as *winner-takes-all*, since only one of the network units is selected for a weight update. Convergence to a good solution can be accelerated by distributing the initial weight vectors according to an adequate heuristic. For example, one could initialize the k weight vectors with k different input vectors. In this way no weight vector corresponds to a dead unit. Another possibility is monitoring the number of updates for each weight vector in order to make the process as balanced as possible (see Exercise 4). This is called *learning with conscience*.

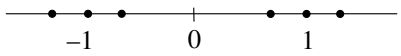
5.2 Convergence analysis

In Algorithm 5.1.1 no stop condition is included. Normally only a fixed number of iterations is performed, since it is very difficult to define the "natural" clustering for some data distributions. If there are three well-defined clusters, but only two weight vectors are used, it could well happen that the weight vectors keep skipping from cluster to cluster in a vain attempt to cover three separate regions with just two computing units. The convergence analysis of unsupervised learning is therefore much more complicated than for perceptron learning, since we are dealing with a much more general problem. Analysis of the one-dimensional case already shows the difficulties of dealing with convergence of unsupervised learning.

5.2.1 The one-dimensional case – energy function

In the one-dimensional case we deal with clusters of numbers in the real line. Let the input set be $\{-1.3, -1.0, -0.7, 0.7, 1.0, 1.3\}$.

Note that we avoid normalizing the input or the weights, since this would make no sense in the one-dimensional case. There are two well-defined clusters



centered at -1 and 1 respectively. This clustering must be identified by the network shown in Figure 5.4, which consists of two units, each with one weight. A possible solution is $\alpha = -1$ and $\beta = 1$. The winning unit inhibits the other unit.

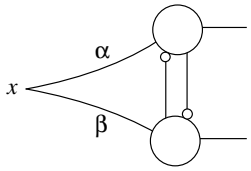


Fig. 5.4. Network for the one-dimensional case

If the learning algorithm is started with α negative and small and β positive and small, it is easy to see that the cluster of negative numbers will attract α and the cluster of positive numbers will attract β. We should expect to see α converge to -1 and β to 1. This means that one of the weights converges to the centroid of the first cluster and the other weight to the centroid of the second. This attraction can be modeled as a kind of force. Let x be a point in a cluster and α_0 the current weight of the first unit. The attraction of x on the weight is given by

$$F_x(\alpha_0) = \gamma(x - \alpha_0)\,,$$

where γ is a constant. Statisticians speak of a "potential" or "inertia" associated with this force. We will call the potential associated with F_x the *energy function* corresponding to the learning task. The energy function for our example is given by

$$E_x(\alpha_0) = \int -\gamma(x - \alpha_0)d\alpha_0 = \frac{\gamma}{2}(x - \alpha_0)^2 + C,$$

where C is an integration constant and the integration is performed over cluster 1. Note that in this case we compute the form of the function for a given α_0 under the assumption that all points of cluster 1 are nearer to α_0 than to the second weight β. The energy function is quadratic with a well-defined global minimum, since in the discrete case the integral is just a summation, and a sum of quadratic functions is also quadratic. Figure 5.5 shows the basins of attraction for the two weights α and β, created by two clusters in the real line which we defined earlier.

Since the attraction of each cluster-point on each of the two weights depends on their relative position, a graphical representation of the energy function must take both α and β into account. Figure 5.6 is a second visualization

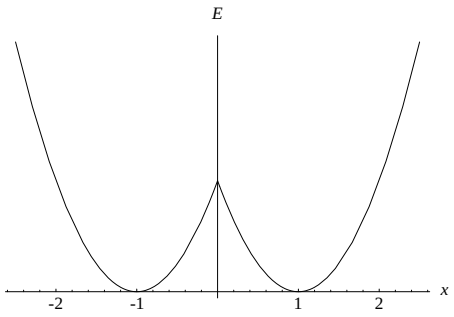


Fig. 5.5. Attractors generated by two one-dimensional clusters

attempt. The vertical axis represents the sum of the distances from α to each one of the cluster-points which lie nearer to α than to β, plus the sum of the distances from β to each one of the cluster-points which lie nearer to β than to α. The gradient of this distance function is proportional to the attraction on α and β. As can be seen there are two global minima: the first at $\alpha = 1, \beta = -1$ and the second at $\alpha = -1, \beta = 1$. Which of these two global minima will be found depends on the weight initialization.

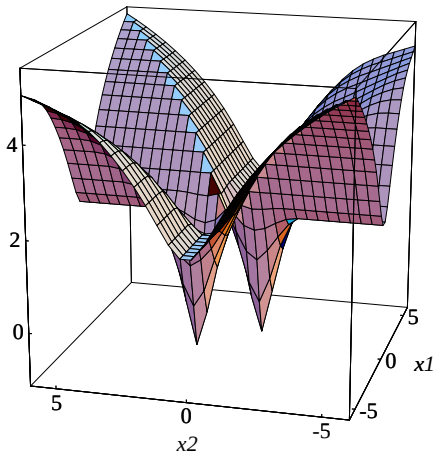


Fig. 5.6. Energy distribution for two one-dimensional clusters

Note, however, that there are also two local minima. If α, for example, is very large in absolute value, this weight will not be updated, so that β converges to 0 and α remains unchanged. The same is valid for β. The dynamics of the learning process is given by *gradient descent* on the energy function. Figure 5.7 is a close-up of the two global minima.

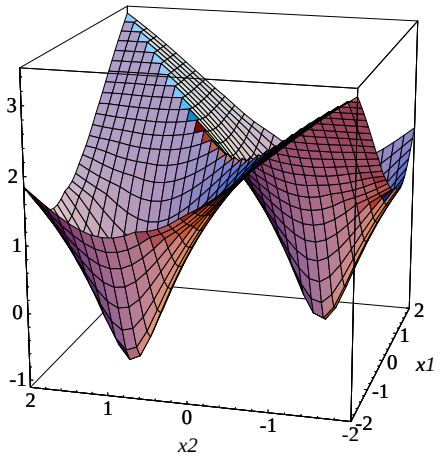


Fig. 5.7. Close-up of the energy distribution

The two local minima in Figure 5.6 correspond to the possibility that unit 1 or unit 2 could become "dead units". This happens when the initial weight vectors lie so far apart from the cluster points that they are never selected for an update.

5.2.2 Multidimensional case – the classical methods

It is not so easy to show graphically how the learning algorithm behaves in the multidimensional case. All we can show are "instantaneous" slices of the energy function which give us an idea of its general shape and the direction of the gradient. A straightforward generalization of the formula used in the previous section provides us with the following definition:

Definition 5. *The energy function of a set* $X = \{\mathbf{x}_1, \ldots, \mathbf{x}_m\}$ *of* n*-dimensional normalized vectors* $(n \geq 2)$ *is given by*

$$E_X(\mathbf{w}) = \sum_{i=1}^{m} (\mathbf{x}_i - \mathbf{w})^2$$

where $\mathbf{w}$ *denotes an arbitrary vector in* n*-dimensional space.*

The energy function is the sum of the quadratic distances from $\mathbf{w}$ to the input vectors $\mathbf{x}_i$. The energy function can be rewritten in the following form:

$$E_X(\mathbf{w}) = m\mathbf{w}^2 - 2\sum_{i=1}^{m} \mathbf{x}_i \cdot \mathbf{w} + \sum_{i=1}^{m} \mathbf{x}_i^2$$

$$
\begin{aligned}
&= m(\mathbf{w}^2 - \frac{2}{m}\mathbf{w} \cdot \sum_{i=1}^{m} \mathbf{x}_i) + \sum_{i=1}^{m} \mathbf{x}_i^2 \\
&= m(\mathbf{w} - \frac{1}{m}\sum_{i=1}^{m} \mathbf{x}_i)^2 - \frac{1}{m^2}(\sum_{i=1}^{m} \mathbf{x}_i)^2 + \sum_{i=1}^{m} \mathbf{x}_i^2 \\
&= m(\mathbf{w} - \mathbf{x}^*)^2 + K.
\end{aligned}
$$

The vector $\mathbf{x}^*$ is the centroid of the cluster $\{\mathbf{x}_1, \mathbf{x}_2, \ldots, \mathbf{x}_m\}$ and K a constant. The energy function has a global minimum at $\mathbf{x}^*$. Figure 5.8 shows the energy function for a two-dimensional example. The first cluster has its centroid at $(-1, 1)$, the second at $(1, -1)$. The figure is a snapshot of the attraction exerted on the weight vectors when each cluster attracts a single weight vector.

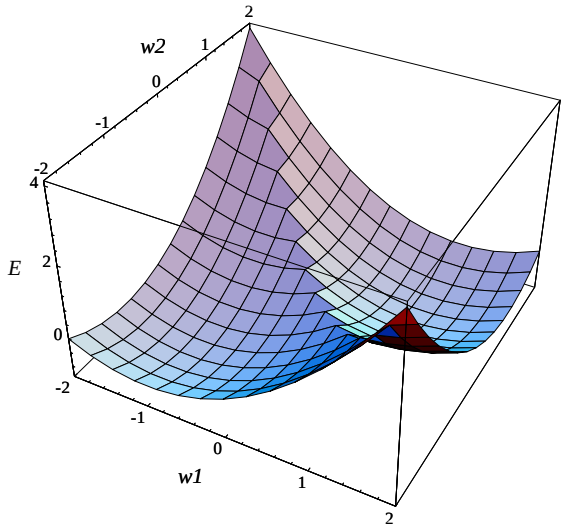


Fig. 5.8. Energy function of two two-dimensional clusters

Statisticians have worked on the problem of finding a good clustering of empirical multidimensional data for many years. Two popular approaches are:

- *k-nearest neighbors* – Sample input vectors are stored and classified in one of ℓ different classes. An unknown input vector is assigned to the class to which the majority of its k closest vectors from the stored set belong (ties can be broken with special heuristics) [110]. In this case a training set is needed, which is later expanded with additional input vectors.
- *k-means* – Input vectors are classified in k different clusters (at the beginning just one vector is assigned to each cluster). A new vector $\mathbf{x}$ is assigned to the cluster k whose centroid $\mathbf{c}_k$ is the closest one to the vector. The centroid vector is updated according to

$$\mathbf{c}_k = \mathbf{c}_k + \frac{1}{n_k}(\mathbf{x} - \mathbf{c}_k),$$

where n_k is the number of vectors already assigned to the k-th cluster. The procedure is repeated iteratively for the whole data set [282]. This method is similar to Algorithm 5.1.1, but there are several variants. The centroids can be updated at the end of several iterations or right after the test of each new vector. The centroids can be calculated with or without the new vector [59].

The main difference between a method like k-nearest neighbors and the algorithm discussed in this chapter is that we do not want to store the input data but only capture its structure in the weight vectors. This is particularly important for applications in which the input data changes with time. The transmission of computer graphics is a good example. The images can be compressed using unsupervised learning to find an adequate *vector quantization*. We do not want to store all the details of those images, only their relevant statistics. If these statistics change, we just have to adjust some network weights and the process of image compression still runs optimally (see Sect. 5.4.2).

5.2.3 Unsupervised learning as minimization problem

In some of the graphics of the energy function shown in the last two sections, we used a strong simplification. We assumed that the data points belonging to a given cluster were known so that we could compute the energy function. But this is exactly what the network should find out. Different assumptions lead to different energy function shapes.

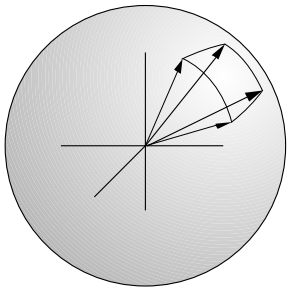


Fig. 5.9. Extreme points of the normalized vectors of a cluster

We can illustrate the iteration process using a three-dimensional example. Assume that two clusters of four normalized vectors are given. One of the clusters is shown in Figure 5.9. Finding the center of this cluster is equivalent to the two-dimensional problem of finding the center of two clusters of four

points at the corners of two squares. Figure 5.10 shows the distribution of members of the two clusters and the initial points a and b selected for the iteration process. You can think of these two points as the endpoints of two normalized vectors in three-dimensional space and of the two-dimensional surface as an approximation of the "flattened" surface of the sphere.

The method we describe now is similar to Algorithm 5.1.1, but uses the Euclidian distance between points as metric. The energy of a point in $\mathbb{R}^2$ corresponds to the sum of the quadratic distances to the points in one of the two clusters. For the initial configuration shown in Figure 5.10 all points above the horizontal line lie nearer to a than to b and all points below the line lie nearer to b than to a.

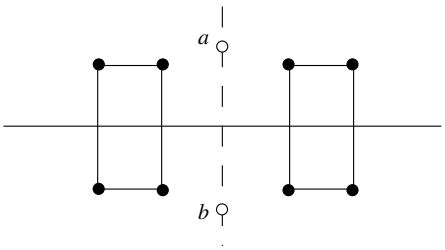


Fig. 5.10. Two cluster and initial points 'a' and 'b'

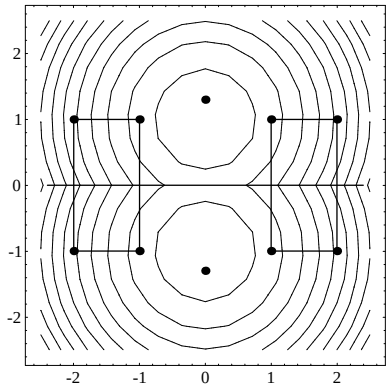


Fig. 5.11. Energy function for the initial points

Figure 5.11 shows the contours of the energy function. As can be seen, both a and b are near to equilibrium. The point a is a representative for the cluster of the four upper points, point b a representative for the cluster of the four lower points. If one update of the position of a and b that modifies the distribution of the cluster points is computed, the shape of the energy function

changes dramatically, as shown in Figure 5.12. In this case, the distribution of points nearer to a than to b has changed (as illustrated by the line drawn between a and b). After several more steps of the learning algorithm the situation shown in Figure 5.13 could be reached, which corresponds now to stable equilibrium. The points a and b cannot jump out of their respective clusters, since the iterative corrections always map points inside the squares to points inside the squares.

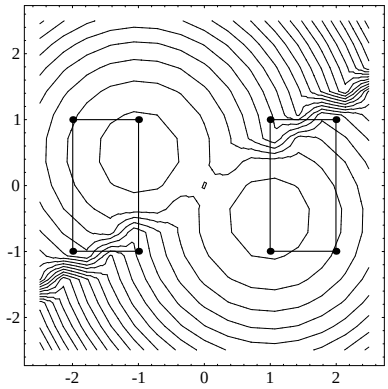


Fig. 5.12. Energy function for a new distribution

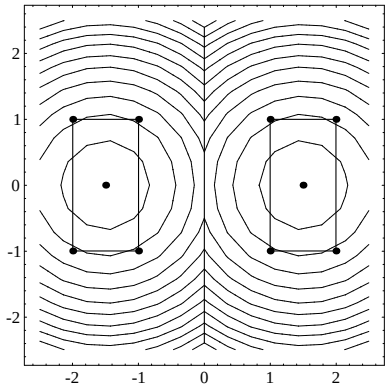


Fig. 5.13. Energy function for the linear separation $x = 0$

5.2.4 Stability of the solutions

The assignment of vectors to clusters can become somewhat arbitrary if we do not have some way of measuring a "good clustering". A simple approach is determining the distance between clusters.

Figure 5.14 shows two clusters of vectors in a two-dimensional space. On the left side we can clearly distinguish the two clusters. On the right side we have selected two weight vectors $\mathbf{w}_1$ and $\mathbf{w}_2$ as their representatives. Each weight vector lies near to the vectors in its cluster, but $\mathbf{w}_1$ lies *inside* the cone defined by its cluster and $\mathbf{w}_2$ *outside*. It is clear that $\mathbf{w}_1$ will not jump outside the cone in future iterations, because it is only attracted by the vectors in its cluster. Weight vector $\mathbf{w}_2$ will at some point jump inside the cone defined by its cluster and will remain there.

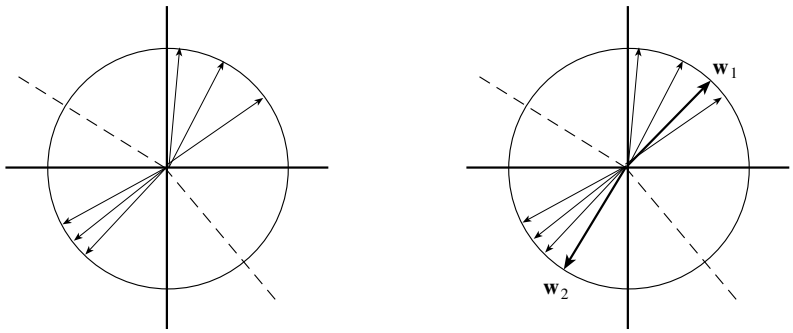


Fig. 5.14. Two vector clusters (left) and two representative weight vectors (right)

This kind of distribution is a *stable solution* or a solution in *stable equilibrium*. Even if the learning algorithm runs indefinitely, the weight vectors will stay by their respective clusters.

As can be intuitively grasped, stable equilibrium requires clearly delimited clusters. If the clusters overlap or are very extended, it can be the case that no stable solution can be found. In this case the distribution of weight vectors remains in *unstable equilibirum*.

Definition 6. *Let P denote the set $\{\mathbf{p}_1, \mathbf{p}_2, \ldots, \mathbf{p}_m\}$ of n-dimensional $(n \geq 2)$ vectors located in the same half-space. The* cone *K defined by P is the set of all vectors $\mathbf{x}$ of the form $\mathbf{x} = \alpha_1\mathbf{p}_1 + \alpha_2\mathbf{p}_2 + \cdots + \alpha_m\mathbf{p}_m$, where $\alpha_1, \alpha_2, \ldots, \alpha_m$ are positive real numbers.*

The cone of a cluster contains all vectors "between" the cluster. The condition that all vectors are located in the same half-space forbids degenerate cones filling the whole space.

The *diameter* of a cone defined by normalized vectors is proportional to the maximum possible angle between two vectors in the cluster.

Definition 7. *The* angular diameter φ *of a cone* K*, defined by normalized vectors* $\{\mathbf{p}_1, \mathbf{p}_2, \ldots, \mathbf{p}_m\}$ *is*

$$\varphi = \sup\{\arccos(\mathbf{a} \cdot \mathbf{b}) | \forall \mathbf{a}, \mathbf{b} \in K, \text{ with } \|\mathbf{a}\| = \|\mathbf{b}\| = 1\},$$

where $0 \leq \arccos(\mathbf{a} \cdot \mathbf{b}) \leq \pi$.

A sufficient condition for stable equilibrium, which formalizes the intuitive idea derived from the example shown in Figure 5.14, is that the angular diameter of the cluster's cone must be smaller than the distance between clusters. This can be defined as follows:

Definition 8. *Let* $P = \{\mathbf{p}_1, \mathbf{p}_2, \ldots, \mathbf{p}_m\}$ *and* $N = \{\mathbf{n}_1, \mathbf{n}_2, \ldots, \mathbf{n}_k\}$ *be two non-void sets of normalized vectors in an* n*-dimensional space* ($n \geq 2$) *that define the cones* K_P *and* K_N *respectively. If the intersection of the two cones is void, the* angular distance *between the cones is given by*

$$\psi_{PN} = \inf\{\arccos(\mathbf{p} \cdot \mathbf{n}) | \mathbf{p} \in K_P, \mathbf{n} \in K_N, \text{ with } \|\mathbf{p}\| = \|\mathbf{n}\| = 1\},$$

where $0 \leq \arccos(\mathbf{p} \cdot \mathbf{n}) \leq \pi$*. If the two cones intersect, the angular distance between them is zero.*

It is easy to prove that if the angular distance between clusters is greater than the angular diameter of the clusters, a stable solution exists in which the weight vectors of an unsupervised network lie in the cluster's cones. Once there, the weight vectors will not leave the cones (see Exercise 1).

In many applications it is not immediately obvious how to rank different clusterings according to their quality. The usual approach is to define a *cost function* which penalizes too many clusters, and favors less but more compact clusters [77]. An extreme example could be identifying each data point as a cluster. This should be forbidden by the optimization of the cost function.

5.3 Principal component analysis

In this section we discuss a second kind of unsupervised learning and its application for the computation of the *principal components* of empirical data. This information can be used to reduce the dimensionality of the data. If the data was coded using n parameters, we would like to encode them using fewer parameters and without losing any essential information.

5.3.1 Unsupervised reinforcement learning

For the class of algorithms we want to consider we will build networks of *linear associators*. This kind of unit exclusively computes the weighted input as result. This means that we omit the comparison with a threshold. Linear associators are used predominantly in associative memories (Chap. **??**).

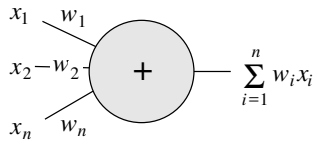


Fig. 5.15. Linear associator

Assume that a set of empirical data is given which consists of n-dimensional vectors $\{\mathbf{x}_1, \mathbf{x}_2, \ldots, \mathbf{x}_m\}$. The first principal component of this set of vectors is a vector $\mathbf{w}$ which maximizes the expression

$$\frac{1}{m} \sum_{i=1}^{m} \|\mathbf{w} \cdot \mathbf{x}_i\|^2,$$

that is, the average of the quadratic scalar products. Figure 5.16 shows an example of a distribution centered at the origin (that is, the centroid of the data lies at the origin). The diagonal runs in the direction of maximum variance of the data. The orthogonal projection of each point on the diagonal represents a larger absolute displacement from the origin than each one of its x_1 and x_2 coordinates. It can be said that the projection contains more information than each individual coordinate alone [276]. In order to statistically analyze this data it is useful to make a coordinate transformation, which in this case would be a rotation of the coordinate axis by 45 degrees. The information content of the new x coordinate is maximized in this way. The new direction of the x_1 axis is the direction of the principal component.

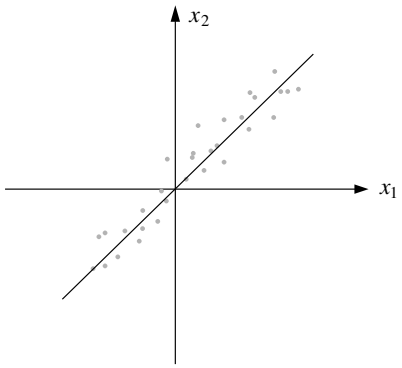


Fig. 5.16. Distribution of input data

The *second* principal component is computed by subtracting from each vector $\mathbf{x}_i$ its projection on the first principal component. The first principal component of the residues is the second principal component of the original data. The second principal component is orthogonal to the first one. The

third principal component is computed recursively: the projection of each vector onto the first and second principal components is subtracted from each vector. The first principal component of the residues is now the third principal component of the original data. Additional principal components are computed following such a recursive strategy.

Computation of the principal components makes a reduction of the dimension of the data with minimal loss of information feasible. In the example of Figure 5.16 each point can be represented by a single number (the length of its projection on the diagonal line) instead of two coordinates x_1 and x_2. If we transmit these numbers to a receiver, we have to specify as sender the direction of the principal component and each one of the projections. The transmission error is the difference between the real and the reconstructed data. This difference is the distance from each point to the diagonal line. The first principal component is thus the direction of the line which minimizes the sum of the deviations, that is the optimal fit to the empirical data. If a set of points in three-dimensional space lies on a line, this line is the principal component of the data and the three coordinates can be transformed into a single number. In this case there would be no loss of information, since the second and third principal components vanish. As we can see, analysis of the principal components helps in all those cases in which the input data is concentrated in a small region of the input space.

In the case of neural networks, the set of input vectors can change in the course of time. Computation of the principal components can be done only adaptively and step by step. In 1982 Oja proposed an algorithm for linear associators which can be used to compute the first principal component of empirical data [331]. It is assumed that the distribution of the data is such that the centroid is located at the origin. If this is not the case for a data set, it is always possible to compute the centroid and displace the origin of the coordinate system to fulfill this requirement.

Algorithm 5.3.1 *Computation of the first principal component*

start: Let X be a set of n-dimensional vectors.
The vector $\mathbf{w}$ is initialized randomly ($\mathbf{w} \neq \mathbf{0}$).
A learning constant γ with $0 < \gamma \leq 1$ is selected.

update: A vector $\mathbf{x}$ is selected randomly from X.
The scalar product $\phi = \mathbf{x} \cdot \mathbf{w}$ is computed.
The new weight vector is $\mathbf{w} + \gamma\phi(\mathbf{x} - \phi\mathbf{w})$.
Go to *update*, making γ smaller.

Of course, a stopping condition has to be added to the above algorithm (for example a predetermined number of iterations). The learning constant γ is chosen as small as necessary to guarantee that the weight updates are not too abrupt (see below). The algorithm is another example of unsupervised learning, since the principal component is found by applying "blind" updates

to the weight vector. Oja's algorithm has the additional property of automatically *normalizing* the weight vector $\mathbf{w}$. This saves an explicit normalization step in which we need global information (the value of each weight) to modify each individual weight. With this algorithm each update uses only local information, since each component of the weight vector is modified taking into account only itself, its input, and the scalar product computed at the linear associator.

5.3.2 Convergence of the learning algorithm

With a few simple geometric considerations we can show that Oja's algorithm must converge when a unique solution to the task exists. Figure 5.17 shows an example with four input vectors whose principal component points in the direction of $\mathbf{w}$. If Oja's algorithm is started with this set of vectors and $\mathbf{w}$, then $\mathbf{w}$ will oscillate between the four vectors but will not leave the cone defined by them. If $\mathbf{w}$ has length 1, then the scalar product $\phi = \mathbf{x} \cdot \mathbf{w}$ corresponds to the length of the projection of $\mathbf{x}$ on $\mathbf{w}$. The vector $\mathbf{x} - \phi\mathbf{w}$ is a vector normal to $\mathbf{w}$. An iteration of Oja's algorithm attracts $\mathbf{w}$ to a vector in the cluster. If it can be guaranteed that $\mathbf{w}$ remains of length 1 or close to 1, the effect of a number of iterations is just to bring $\mathbf{w}$ into the middle of the cluster.

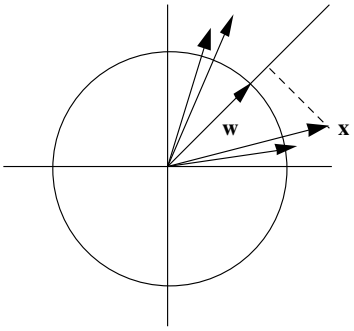


Fig. 5.17. Cluster of vectors and principal component

We must show that the vector $\mathbf{w}$ is automatically normalized by this algorithm. Figure 5.18 shows the necessary geometric constructions. The left side shows the case in which the length of vector $\mathbf{w}$ is greater than 1. Under these circumstances the vector $(\mathbf{x} \cdot \mathbf{w})\mathbf{w}$ has a length greater than the length of the orthogonal projection of $\mathbf{x}$ on $\mathbf{w}$. Assume that $\mathbf{x} \cdot \mathbf{w} > 0$, that is, the vectors $\mathbf{x}$ and $\mathbf{w}$ are not too far away. The vector $\mathbf{x} - (\mathbf{x} \cdot \mathbf{w})\mathbf{w}$ has a negative projection on $\mathbf{w}$ because

$$(\mathbf{x} - (\mathbf{x} \cdot \mathbf{w})\mathbf{w}) \cdot \mathbf{w} = \mathbf{x} \cdot \mathbf{w} - \|\mathbf{w}\|^2 \mathbf{x} \cdot \mathbf{w} < 0.$$

Now we have to think about the result of many iterations of this type. The vector $\mathbf{x} - (\mathbf{x} \cdot \mathbf{w})\mathbf{w}$ has a component normal to $\mathbf{w}$ and another pointing in the negative direction of $\mathbf{w}$. Repeated iterations will bring $\mathbf{w}$ into the middle of the cluster of vectors, so that in the average case the normal components later cancel. However, the negative component does not cancel if $\mathbf{w}$ is still of a length greater than 1. The net effect of the iterations is therefore to position $\mathbf{w}$ correctly, but also to make it smaller. However, care must be taken to avoid making $\mathbf{w}$ *too small* or even reversing its direction in a single iteration. This can be avoided by setting the learning constant γ as small as necessary. It also helps to normalize the training vectors before the algorithm is started. If the vector $\mathbf{x}$ has a positive scalar product ϕ with $\mathbf{w}$, we would like also the new weight vector to have a positive scalar product with $\mathbf{x}$. This means that we want the following inequality to hold

$$\mathbf{x} \cdot (\mathbf{w} + \gamma\phi(\mathbf{x} - \phi\mathbf{w}) > 0.$$

This is equivalent to

$$\gamma(\|\mathbf{x}\|^2 - \phi^2) > -1,$$

and if γ is positive and small the inequality can always be satisfied regardless of the sign of $\|\mathbf{x}\|^2 - \phi^2$.

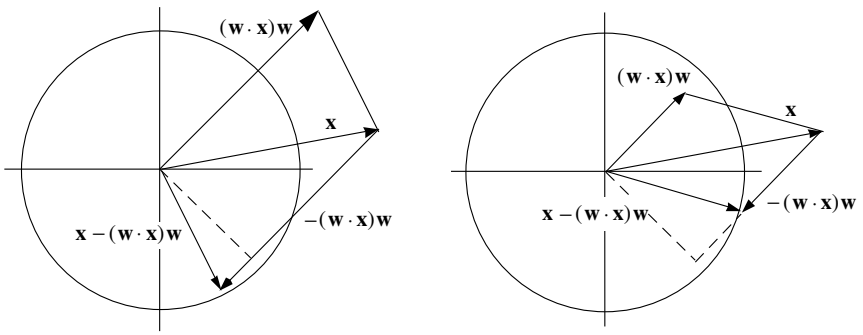


Fig. 5.18. The two cases in Oja's algorithm

The right side of Figure 5.18 shows what happens when the vector $\mathbf{w}$ has a length smaller than 1. In this case the vector $\mathbf{x} - (\mathbf{x} \cdot \mathbf{w})\mathbf{w}$ has a positive projection on $\mathbf{w}$. The normal component of this vector will bring $\mathbf{w}$ into the center of the cluster after repeated iterations. The positive projection of this vector on $\mathbf{w}$ has the net effect of making $\mathbf{w}$ larger. Combining both cases, we can deduce that the net effect of the algorithm is to bring $\mathbf{w}$ into the middle of the cluster, while at the same time the length of $\mathbf{w}$ oscillates around 1 (for a small enough γ). Once this state is achieved, and if the task has a unique solution, $\mathbf{w}$ just oscillates around its equilibrium position and equilibrium length. Note, however, that these are considerations about the *expected value*

of **w**. If the clusters are sparsely populated or if the data vectors are of very different lengths, the actual value of **w** can differ significantly at each iteration from its average value.

It can be shown that the first principal component of a set of vectors is equivalent to the direction of the longest eigenvector of their correlation matrix [225] and that Oja's algorithm finds approximately this direction.

5.3.3 Multiple principal components

Sanger proposed a network architecture capable of finding the first m principal components of a data set [387]. The idea is to compute the first principal component using a linear associator and then to subtract the projection of the data on this direction from the data itself. The residues are processed by the next linear associator in the chain. This unit computes the second principal component of the original data. Figure 5.19 shows the structure of the network. The figure shows the connections associated with vector $\mathbf{w}_1$ as an arrow in order to simplify the wiring diagram.

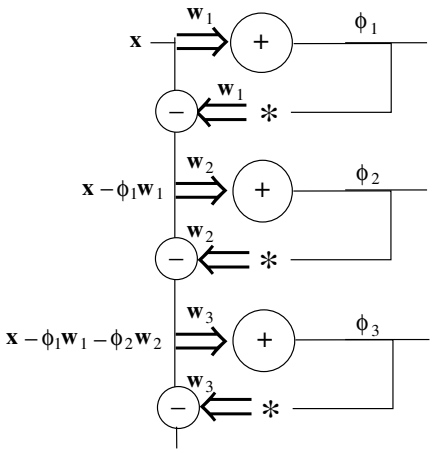


Fig. 5.19. Network for the iterative computation of the first three principal components

The network weights are found using Oja's algorithm at each level of the network. It must be guaranteed that the individual weight vectors (which appear twice in the network) are kept consistent. The three linear associators shown in the figure can be trained simultaneously, but we can stop training the last unit only after the weight vectors of the previous units have stabilized.

Other authors have proposed alternative network topologies to compute the first m principal components of empirical data [332, 381], but the essential idea is roughly the same.

5.4 Some applications

Vector quantization and unsupervised clustering have found many interesting applications. In this section we discuss two of them.

5.4.1 Pattern recognition

Unsupervised learning can be used in all cases where the number of different input clusters is known *a priori*, as is the case with optical character recognition. Assume that letters in a document are scanned and centered in 16×16 projection screens. Each letter is coded as a vector of dimension 256, assigning the value 1 to black pixels and -1 to white pixels. There is some noise associated with each image, so that sometimes pixels are assigned the false bit. If the number of errors per image is not excessive, then the input vectors for the same letter cluster around a definite region in input space. These clusters can be identified by a network of competitive units.

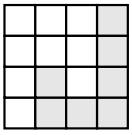


Fig. 5.20. A digital "J" (4×4 matrix)

If only 26 different letters are to be recognized, our network needs only 26 different units. The weights of each unit can be initialized randomly and an unsupervised learning algorithm can be used to find the correct parameters. There are conflicts with letters having a similar shape (for example O and Q), which can be solved only by pulling the clusters apart using a higher scanning resolution.

5.4.2 Image compression

Assume that a picture with 1024×1024 pixels is to be transmitted. The number of bits needed is 2^{20}. However, images are not totally random. They have a structure which can be analyzed with the purpose of compressing the image before transmission. This can be done by dividing the picture in 128×128 fields of 8×8 pixels. Each field can contain any of 2^{64} different possible patterns, but assume that we decide to classify them in 64 different classes. We start an unsupervised network with 64 outputs and train it to classify all 8×8 patterns found in the image. Note that we do not train the network with all 2^{64} different possible patterns, but only using those that actually appear in the picture. The units in the network classify these patterns in clusters. The weight vectors of the 64 units are then transmitted as a

representative of each cluster and correspond to a certain pattern identified by the corresponding unit. The 64 weight vectors are the *codebook* for the transmission.

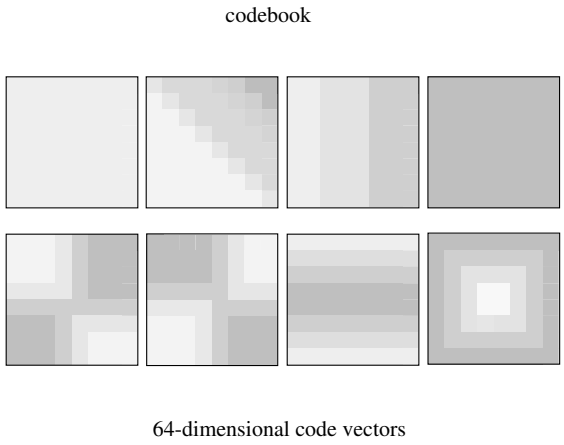


Fig. 5.21. Part of a codebook for image compression

The transmission of the image can begin once the sender and receiver have agreed on a codebook. For each of the 128×128 fields in the picture, the sender transmits the name of the code vector nearer to the 8×8 content of the field. Since there are only 64 codebook vectors, the sender has to transmit 6 bits. The compression ratio achieved for the transmission is $64/6 = 10.66$. The image reconstructed by the sender contains some errors, but if a good clustering has been found, those errors will be minimal. Figure 5.21 shows 8 of the weight vectors found by unsupervised learning applied to a picture. The weight vectors are represented by gray values and show the kind of features identified by the algorithm. Note that there are weight vectors for units which recognize vertical or horizontal lines, white or black regions as well as other kinds of structures. The compression ratio can be modified by using more or less codebook vectors, that is, more or less clusters.

Scientists have long speculated about the possible mechanisms of information compression in the human brain. Many different experiments have shown that the connection pattern of the brain is determined by genetic factors, but also by experience. There is an interplay between "nature and nurture" that leads to the gradual modification of the neural connections. Some classical results were obtained analyzing the visual cortex of cats. It is now well known that if a cat is blind in one eye, the part of the visual cortex normally assigned to that eye is reassigned to the other one. Some other experiments, in which cats were kept in a special environment (lacking for example any kind of horizontal lines) have shown that feature detectors in the brain specialize

to identify only those patterns present in the real world. These results are interpreted by assuming that genetic factors determine the *rules* by which unsupervised learning takes place in the neural connections, whereas actual experience shapes the feature detectors. Some regions of the visual cortex are in fact detectors which identify patterns of the kind shown in Figure 5.21.

5.5 Historical and bibliographical remarks

Frank Rosenblatt was the first to experiment with unsupervised learning in networks of threshold elements [382]. He interpreted the ability of the networks to find clusters as a kind of mechanism similar to the abstraction power of the human mind, capable of forming new concepts out of empirical material.

After Rosenblatt there were not many other examples of unsupervised networks until the 1970s. Grossberg and von der Malsburg formulated biologically oriented models capable of self-organizing in an unsupervised environment [168, 284]. Theoretical models of synaptic plasticity were developed later by Sejnowski and Tesauro [397] and Klopf [249]. The Nobel prizewinner Gerald Edelman developed a whole theory of "neuronal selection" with which he tried to explain the global architecture of the brain [124]. Using large computer simulations he showed that the "market forces", that is, competition at all levels of the neuronal circuits, can explain some of their characteristics.

Vector quantization has become a common tool for the transmission and analysis of images and signals. Although the resulting assignment problem is known to be computationally expensive (*NP*-hard indeed, see Chap. 10), many heuristics and approximate methods have been developed that are now being used [159]. Codebooks have also been applied to the digital transmission of voice signals over telephone lines.

Principal component analysis has been a standard statistical technique since it was first described by Pearson and Hotelling between 1901 and 1903 [225]. The methods were discussed analytically but only found real application with the advent of the computer. Traditional methods of principal component analysis proceed by finding eigenvalues of the correlation matrix. Adaptive methods, of the kind introduced in the 1970s, do not assume that the data set is fixed and constant, but that it can change in the course of time [387]. The algorithms of Oja and Linsker can be used in these cases. The possible applications of such an adaptive principal component analysis are real-time compression of signals, simplification and acceleration of learning algorithms for neural networks [404], or adaptive pattern recognition [208].

Exercises

1. Prove that if m weight vectors lie in the cones of m clusters such that their angular diameters are smaller that the minimum angular distance

between clusters, unsupervised learning will not bring any of the weight vectors out of its respective cone.

2. Implement an optical character recognizer. Train a classifier network of competitive units with scanned examples of 10 different letters. Test if the network can identify noisy examples of the same scanned letters.
3. Train the classifier network using Oja's algorithm.
4. How can dead units be avoided in a network of competitive units? Propose two or three different heuristics.
5. Compute a codebook for the compression of a digitized image. Compare the reconstructed image with the original. Can you improve the reconstructed image using a Gaussian filter?
6. A network can be trained to classify characters using a decision tree. Propose an architecture for such a network and a training method.