4

Perceptron Learning

4.1 Learning algorithms for neural networks

In the two preceding chapters we discussed two closely related models, McCulloch–Pitts units and perceptrons, but the question of how to find the parameters adequate for a given task was left open. If two sets of points have to be separated linearly with a perceptron, adequate weights for the computing unit must be found. The operators that we used in the preceding chapter, for example for edge detection, used hand customized weights. Now we would like to find those parameters automatically. The *perceptron learning algorithm* deals with this problem.

A learning algorithm is an adaptive method by which a network of computing units self-organizes to implement the desired behavior. This is done in some learning algorithms by presenting some examples of the desired input-output mapping to the network. A correction step is executed iteratively until the network learns to produce the desired response. The learning algorithm is a closed loop of presentation of examples and of corrections to the network parameters, as shown in Figure 4.1.

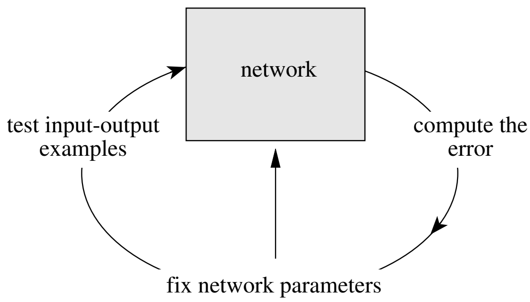

Fig. 4.1. Learning process in a parametric system

In some simple cases the weights for the computing units can be found through a sequential test of stochastically generated numerical combinations. However, such algorithms which look blindly for a solution do not qualify as "learning". A learning algorithm must adapt the network parameters according to previous experience until a solution is found, if it exists.

4.1.1 Classes of learning algorithms

Learning algorithms can be divided into *supervised* and *unsupervised* methods. Supervised learning denotes a method in which some input vectors are collected and presented to the network. The output computed by the network is observed and the deviation from the expected answer is measured. The weights are corrected according to the magnitude of the error in the way defined by the learning algorithm. This kind of learning is also called *learning with a teacher*, since a control process knows the correct answer for the set of selected input vectors.

Unsupervised learning is used when, for a given input, the exact numerical output a network should produce is unknown. Assume, for example, that some points in two-dimensional space are to be classified into three clusters. For this task we can use a classifier network with three output lines, one for each class (Figure 4.2). Each of the three computing units at the output must specialize by firing only for inputs corresponding to elements of each cluster. If one unit fires, the others must keep silent. In this case we do not know a priori which unit is going to specialize on which cluster. Generally we do not even know how many well-defined clusters are present. Since no "teacher" is available, the network must organize itself in order to be able to associate clusters with units.

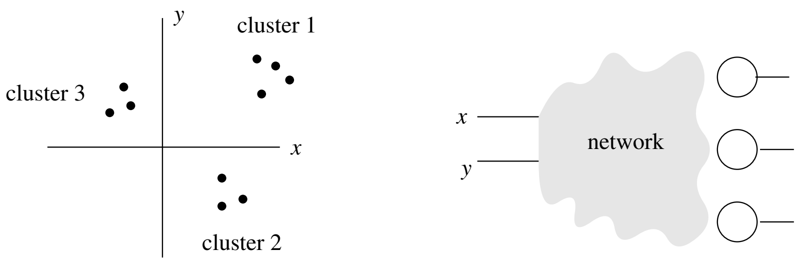

Fig. 4.2. Three clusters and a classifier network

Supervised learning is further divided into methods which use reinforcement or error correction. Reinforcement learning is used when after each presentation of an input-output example we only know whether the network produces the desired result or not. The weights are updated based on this information (that is, the Boolean values *true* or *false*) so that only the input vector can be used for weight correction. In learning with error correction, the magnitude of the error, together with the input vector, determines the magnitude of the corrections to the weights, and in many cases we try to eliminate the error in a single correction step.

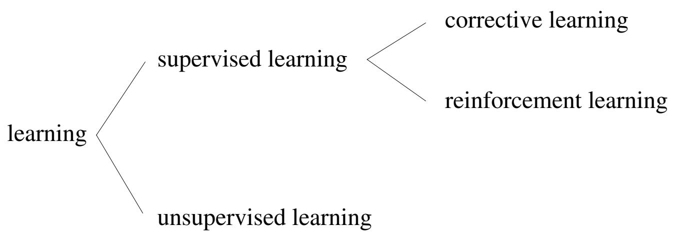

Fig. 4.3. Classes of learning algorithms

The perceptron learning algorithm is an example of supervised learning with reinforcement. Some of its variants use supervised learning with error correction (corrective learning).

4.1.2 Vector notation

In the following sections we deal with learning methods for perceptrons. To simplify the notation we adopt the following conventions. The input $(x_1, x_2, \ldots, x_n)$ to the perceptron is called the input vector. If the weights of the perceptron are the real numbers $w_1, w_2, \ldots, w_n$ and the threshold is θ, we call $\mathbf{w} = (w_1, w_2, \ldots, w_n, w_{n+1})$ with $w_{n+1} = -\theta$ the *extended weight vector* of the perceptron and $(x_1, x_2, \ldots, x_n, 1)$ the *extended input vector*. The threshold computation of a perceptron will be expressed using scalar products. The arithmetic test computed by the perceptron is thus

$$\mathbf{w} \cdot \mathbf{x} \geq \theta\,,$$

if $\mathbf{w}$ and $\mathbf{x}$ are the weight and input vectors, and

$$\mathbf{w} \cdot \mathbf{x} \geq 0$$

if $\mathbf{w}$ and $\mathbf{x}$ are the extended weight and input vectors. It will always be clear from the context whether normal or extended vectors are being used.

If, for example, we are looking for the weights and threshold needed to implement the AND function with a perceptron, the input vectors and their associated outputs are

$$\begin{aligned}(0,0) &\mapsto 0,\\ (0,1) &\mapsto 0,\\ (1,0) &\mapsto 0,\\ (1,1) &\mapsto 1.\end{aligned}$$

If a perceptron with threshold zero is used, the input vectors must be extended and the desired mappings are

$$\begin{aligned}(0,0,1) &\mapsto 0,\\ (0,1,1) &\mapsto 0,\\ (1,0,1) &\mapsto 0,\\ (1,1,1) &\mapsto 1.\end{aligned}$$

A perceptron with three still unknown weights (w_1, w_2, w_3) can carry out this task.

4.1.3 Absolute linear separability

The proof of convergence of the perceptron learning algorithm assumes that each perceptron performs the test $\mathbf{w} \cdot \mathbf{x} > 0$. So far we have been working with perceptrons which perform the test $\mathbf{w} \cdot \mathbf{x} \geq 0$. We must just show that both classes of computing units are equivalent when the training set is finite, as is always the case in learning problems. We need the following definition.

Definition 3. *Two sets A and B of points in an n-dimensional space are called absolutely linearly separable if $n+1$ real numbers $w_1, \ldots, w_{n+1}$ exist such that every point $(x_1, x_2, \ldots, x_n) \in A$ satisfies $\sum_{i=1}^{n} w_i x_i > w_{n+1}$ and every point $(x_1, x_2, \ldots, x_n) \in B$ satisfies $\sum_{i=1}^{n} w_i x_i < w_{n+1}$*

If a perceptron with threshold zero can linearly separate two finite sets of input vectors, then only a small adjustment to its weights is needed to obtain an absolute linear separation. This is a direct corollary of the following proposition.

Proposition 7. *Two finite sets of points, A and B, in n-dimensional space which are linearly separable are also absolutely linearly separable.*

Proof. Since the two sets are linearly separable, weights $w_1, \ldots, w_{n+1}$ exist for which, without loss of generality, the inequality

$$\sum_{i=1}^{n} w_i a_i \geq w_{n+1}$$

holds for all points $(a_1, \ldots, a_n) \in A$ and

$$\sum_{i=1}^{n} w_i b_i < w_{n+1}$$

for all points $(b_1, \ldots, b_n) \in B$. Let $\varepsilon = \max\{\sum_{i=1}^{n} w_i b_i - w_{n+1} | (b_1, \ldots, b_n) \in B\}$. It is clear that $\varepsilon < \varepsilon/2 < 0$. Let $w' = w_{n+1} + \varepsilon/2$. For all points in A it holds that

$$\sum_{i=1}^{n} w_i a_i - (w' - \frac{1}{2}\varepsilon) \geq 0.$$

This means that

$$\sum_{i=1}^{n} w_i a_i - w' \geq -\frac{1}{2}\varepsilon > 0 \Rightarrow \sum_{i=1}^{n} w_i a_i > w'. \tag{4.1}$$

For all points in B a similar argument holds since

$$\sum_{i=1}^{n} w_i b_i - w_{n+1} = \sum_{i=1}^{n} w_i b_i - (w' - \frac{1}{2}\varepsilon) \leq \varepsilon,$$

and from this we deduce

$$\sum_{i=1}^{n} w_i b_i - w' \leq \frac{1}{2}\varepsilon < 0. \tag{4.2}$$

Equations (4.1) and (4.2) show that the sets A and B are absolutely linearly separable. If two sets are linearly separable in the absolute sense, then they are, of course, linearly separable in the conventional sense. □

4.1.4 The error surface and the search method

A usual approach for starting the learning algorithm is to initialize the network weights randomly and to improve these initial parameters, looking at each step to see whether a better separation of the training set can be achieved. In this section we identify points $(x_1, x_2, \ldots, x_n)$ in n-dimensional space with the vector $\mathbf{x}$ with the same coordinates.

Definition 4. *The open (closed) positive half-space associated with the n-dimensional weight vector $\mathbf{w}$ is the set of all points $\mathbf{x} \in \mathbb{R}^n$ for which $\mathbf{w} \cdot \mathbf{x} > 0$ ($\mathbf{w} \cdot \mathbf{x} \geq 0$). The open (closed) negative half-space associated with $\mathbf{w}$ is the set of all points $\mathbf{x} \in \mathbb{R}^n$ for which $\mathbf{w} \cdot \mathbf{x} < 0$ ($\mathbf{w} \cdot \mathbf{x} \leq \mathbf{0}$).*

We omit the adjectives "closed" or "open" whenever it is clear from the context which kind of linear separation is being used.

Let P and N stand for two finite sets of points in $\mathbb{R}^n$ which we want to separate linearly. A weight vector is sought so that the points in P belong to its associated positive half-space and the points in N to the negative half-space. The *error* of a perceptron with weight vector $\mathbf{w}$ is the number of incorrectly classified points. The learning algorithm must minimize this error function $E(\mathbf{w})$. One possible strategy is to use a local greedy algorithm which works by computing the error of the perceptron for a given weight vector, looking then for a direction in weight space in which to move, and updating the weight vector by selecting new weights in the selected search direction. We can visualize this strategy by looking at its effect in weight space.

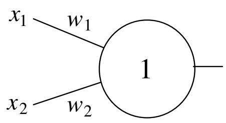

Fig. 4.4. Perceptron with constant threshold

Let us take as an example a perceptron with constant threshold $\theta = 1$ (Figure 4.4). We are looking for two weights, w_1 and w_2, which transform the perceptron into a binary AND gate. We can show graphically the error function for all combinations of the two variable weights. This has been done in Figure 4.5 for values of the weights between -0.5 and 1.5. The solution region is the triangular area in the middle. The learning algorithm should reach this region starting from any other point in weight space. In this case, it is possible to descend from one surface to the next using purely local decisions at each point.

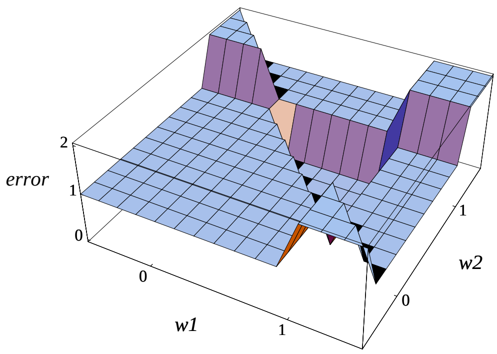

Fig. 4.5. Error function for the AND function

Figure 4.6 shows the different regions of the error function as seen from "above". The solution region is a triangle with error level 0. For the other regions the diagram shows their corresponding error count. The figure illustrates an iterative search process that starts at $\mathbf{w}_0$, goes through $\mathbf{w}_1$, $\mathbf{w}_2$, and finally reaches the solution region at $\mathbf{w}^*$. Later on, this visualization will help us to understand the computational complexity of the perceptron learning algorithm.

The optimization problem we are trying to solve can be understood as descent on the error surface but also as a search for an inner point of the solution region. Let $N = \{(0,0), (1,0), (0,1)\}$ and $P = \{(1,1)\}$ be two sets of points to be separated absolutely. The set P must be classified in the positive

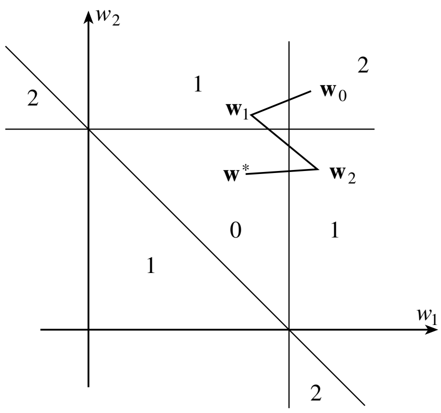

Fig. 4.6. Iteration steps to the region of minimal error

and the set N in the negative half-space. This is the separation corresponding to the AND function.

Three weights w_1, w_2, and $w_3 = -\theta$ are needed to implement the desired separation with a generic perceptron. The first step is to extend the input vectors with a third coordinate $x_3 = 1$ and to write down the four inequalities that must be fulfilled:

$$(0, 0, 1) \cdot (w_1, w_2, w_3) < 0 \tag{4.3}$$

$$(1, 0, 1) \cdot (w_1, w_2, w_3) < 0 \tag{4.4}$$

$$(0, 1, 1) \cdot (w_1, w_2, w_3) < 0 \tag{4.5}$$

$$(1, 1, 1) \cdot (w_1, w_2, w_3) > 0 \tag{4.6}$$

These equations can be written in the following simpler matrix form:

$$\begin{pmatrix} 0 & 0 & -1 \\ -1 & 0 & -1 \\ 0 & -1 & -1 \\ 1 & 1 & 1 \end{pmatrix} \begin{pmatrix} w_1 \\ w_2 \\ w_3 \end{pmatrix} > \begin{pmatrix} 0 \\ 0 \\ 0 \\ 0 \end{pmatrix}. \tag{4.7}$$

This can be written as

$$\mathbf{A}\mathbf{w} > \mathbf{0}.$$

where $\mathbf{A}$ is the 4×3 matrix of Equation (4.7) and $\mathbf{w}$ the weight vector (written as a column vector). The learning problem is to find the appropriate weight vector $\mathbf{w}$.

Equation (4.7) describes all points in the interior of a convex polytope. The sides of the polytope are delimited by the planes defined by each of the inequalities (4.3)–(4.6). Any point in the interior of the polytope represents a solution for the learning problem.

We saw that the solution region for the AND function has a triangular shape when the threshold is fixed at 1. In Figure 4.7 we have a three-dimensional view of the whole solution region when the threshold (i.e., w_3) is allowed to change. The solution region of Figure 4.5 is just a cut of the solution polytope of Figure 4.7 at $w_3 = -1$. The shaded surface represents the present cut, which is similar to any other cut we could make to the polytope for different values of the threshold.

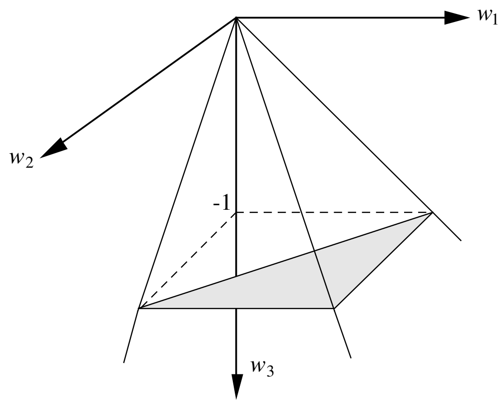

Fig. 4.7. Solution polytope for the AND function in weight space

We can see that the polytope is unbounded in the direction of negative w_3. This means that the absolute value of the threshold can become as large as we want and we will still find appropriate combinations of w_1 and w_2 to compute the AND function. The fact that we have to look for interior points of polytopes for the solution of the learning problem, is an indication that linear programming methods could be used. We will elaborate on this idea later on.

4.2 Algorithmic learning

We are now in a position to introduce the perceptron learning algorithm. The training set consists of two sets, P and N, in n-dimensional extended input space. We look for a vector $\mathbf{w}$ capable of absolutely separating both sets, so that all vectors in P belong to the open positive half-space and all vectors in N to the open negative half-space of the linear separation.

Algorithm 4.2.1 *Perceptron learning*

start: The weight vector $\mathbf{w}_0$ is generated randomly,
set $t := 0$

test: A vector $\mathbf{x} \in P \cup N$ is selected randomly,
if $\mathbf{x} \in P$ and $\mathbf{w}_t \cdot \mathbf{x} > 0$ go to *test*,
if $\mathbf{x} \in P$ and $\mathbf{w}_t \cdot \mathbf{x} \leq 0$ go to *add*,
if $\mathbf{x} \in N$ and $\mathbf{w}_t \cdot \mathbf{x} < 0$ go to *test*,
if $\mathbf{x} \in N$ and $\mathbf{w}_t \cdot \mathbf{x} \geq 0$ go to *subtract*.

add: set $\mathbf{w}_{t+1} = \mathbf{w}_t + \mathbf{x}$ and $t := t + 1$, goto *test*

subtract: set $\mathbf{w}_{t+1} = \mathbf{w}_t - \mathbf{x}$ and $t := t + 1$, goto *test*

This algorithm [312] makes a correction to the weight vector whenever one of the selected vectors in P or N has not been classified correctly. The perceptron convergence theorem guarantees that if the two sets P and N are linearly separable the vector $\mathbf{w}$ is updated only a finite number of times. The routine can be stopped when all vectors are classified correctly. The corresponding test must be introduced in the above pseudocode to make it stop and to transform it into a fully-fledged algorithm.

4.2.1 Geometric visualization

There are two alternative ways to visualize perceptron learning, one more effective than the other. Given the two sets of points $P \in \mathbb{R}^2$ and $N \in \mathbb{R}^2$ to be separated, we can visualize the linear separation in input space, as in Figure 4.8, or in extended input space. In the latter case we extend the input vectors and look for a linear separation through the origin, that is, a plane with equation $w_1 x_1 + w_2 x_2 + w_3 = 0$. The vector normal to this plane is the weight vector $\mathbf{w} = (w_1, w_2, w_3)$. Figure 4.9 illustrates this approach.

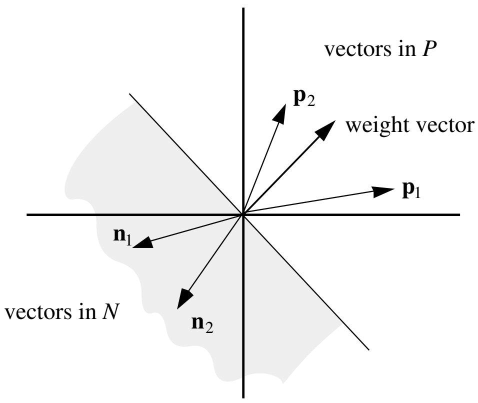

Fig. 4.8. Visualization in input space

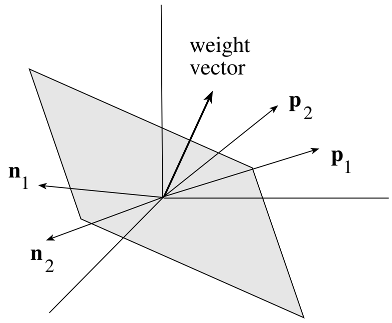

Fig. 4.9. Visualization in extended input space

We are thus looking for a weight vector $\mathbf{w}$ with a positive scalar product with all the extended vectors represented by the points in P and with a negative product with the extended vectors represented by the points in N. Actually, we will deal with this problem in a more general way. Assume that P and N are sets of n-dimensional vectors and a weight vector $\mathbf{w}$ must be found, such that $\mathbf{w} \cdot \mathbf{x} > 0$ holds for all $x \in P$ and $\mathbf{w} \cdot \mathbf{x} < 0$ holds for all $x \in N$.

The perceptron learning algorithm starts with a randomly chosen vector $\mathbf{w}_0$. If a vector $\mathbf{x} \in P$ is found such that $\mathbf{w} \cdot \mathbf{x} < 0$, this means that the angle between the two vectors is greater than 90 degrees. The weight vector must be rotated in the direction of $\mathbf{x}$ to bring this vector into the positive half-space defined by $\mathbf{w}$. This can be done by adding $\mathbf{w}$ and $\mathbf{x}$, as the perceptron learning algorithm does. If $\mathbf{x} \in N$ and $\mathbf{w} \cdot \mathbf{x} > 0$, then the angle between $\mathbf{x}$ and $\mathbf{w}$ is less than 90 degrees. The weight vector must be rotated away from $\mathbf{x}$. This is done by subtracting $\mathbf{x}$ from $\mathbf{w}$. The vectors in P rotate the weight vector in one direction, the vectors in N rotate the negative weight vector in another. If a solution exists it can be found after a finite number of steps. A good initial heuristic is to start with the average of the positive input vectors minus the average of the negative input vectors. In many cases this yields an initial vector near the solution region.

In perceptron learning we are not necessarily dealing with normalized vectors, so that every update of the weight vector of the form $\mathbf{w} \pm \mathbf{x}$ rotates the weight vector by a different angle. If $\mathbf{x} \in P$ and $\|\mathbf{x}\| \gg \|\mathbf{w}\|$ the new weight vector $\mathbf{w} + \mathbf{x}$ is almost equal to $\mathbf{x}$. This effect and the way perceptron learning works can be seen in Figure 4.10. The initial weight vector is updated by adding $\mathbf{x}_1$, $\mathbf{x}_3$, and $\mathbf{x}_1$ again to it. After each correction the weight vector is rotated in one or the other direction. It can be seen that the vector $\mathbf{w}$ becomes larger after each correction in this example. Each correction rotates the weight vector by a smaller angle until the correct linear separation has been found. After the initial updates, successive corrections become smaller and the algorithm "fine tunes" the position of the weight vector. The learning rate, the rate of change of the vector $\mathbf{w}$, becomes smaller in time and if we

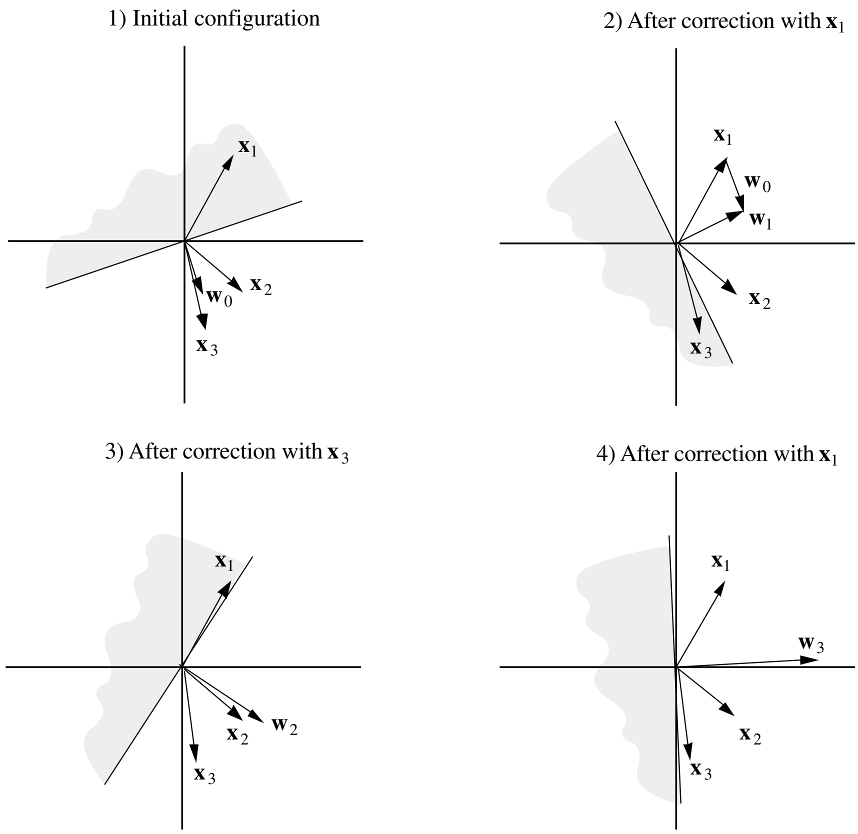

Fig. 4.10. Convergence behavior of the learning algorithm

keep on training, even after the vectors have already been correctly separated, it approaches zero. Intuitively we can think that the learned vectors are increasing the "inertia" of the weight vector. Vectors lying just outside of the positive region are brought into it by rotating the weight vector just enough to correct the error.

This is a typical feature of many learning algorithms for neural networks. They make use of a so-called *learning constant*, which is brought to zero during the learning process to consolidate what has been learned. The perceptron learning algorithm provides a kind of automatic learning constant which determines the degree of adaptivity (the so-called *plasticity* of the network) of the weights.

4.2.2 Convergence of the algorithm

The convergence proof of the perceptron learning algorithm is easier to follow by keeping in mind the visualization discussed in the previous section.

Proposition 8. *If the sets P and N are finite and linearly separable, the perceptron learning algorithm updates the weight vector $\mathbf{w}_t$ a finite number of times. In other words: if the vectors in P and N are tested cyclically one after the other, a weight vector $\mathbf{w}_t$ is found after a finite number of steps t which can separate the two sets.*

Proof. We can make three simplifications without losing generality:

i) The sets P and N can be joined in a single set $P' = P \cup N^-$, where the set N^- consists of the negated elements of N.
ii) The vectors in P' can be normalized, because if a weight vector $\mathbf{w}$ is found so that $\mathbf{w} \cdot \mathbf{x} > 0$ this is also valid for any other vector $\eta\mathbf{x}$, where η is a constant.
iii) The weight vector can also be normalized. Since we assume that a solution for the linear separation problem exists, we call $\mathbf{w}^*$ a normalized solution vector.

Assume that after $t+1$ steps the weight vector $\mathbf{w}_{t+1}$ has been computed. This means that at time t a vector $\mathbf{p}_i$ was incorrectly classified by the weight vector $\mathbf{w}_t$ and so $\mathbf{w}_{t+1} = \mathbf{w}_t + \mathbf{p}_i$.

The cosine of the angle ρ between $\mathbf{w}_{t+1}$ and $\mathbf{w}^*$ is

$$\cos\rho = \frac{\mathbf{w}^* \cdot \mathbf{w}_{t+1}}{\|\mathbf{w}_{t+1}\|} \tag{4.8}$$

For the expression in the numerator we know that

$$\begin{aligned}\mathbf{w}^*\mathbf{w}_{t+1} &= \mathbf{w}^* \cdot (\mathbf{w}_t + \mathbf{p}_i)\\ &= \mathbf{w}^* \cdot \mathbf{w}_t + \mathbf{w}^* \cdot \mathbf{p}_i\\ &\geq \mathbf{w}^* \cdot \mathbf{w}_t + \delta\end{aligned}$$

with $\delta = \min\{\mathbf{w}^* \cdot \mathbf{p} \mid \forall \mathbf{p} \in P'\}$. Since the weight vector $\mathbf{w}^*$ defines an absolute linear separation of P and N we know that $\delta > 0$. By induction we obtain

$$\mathbf{w}^* \cdot \mathbf{w}_{t+1} \geq \mathbf{w}^* \cdot \mathbf{w}_0 + (t+1)\delta. \tag{4.9}$$

On the other hand for the term in the denominator of (4.8) we know that

$$\begin{aligned}\|\mathbf{w}_{t+1}\|^2 &= (\mathbf{w}_t + \mathbf{p}_i) \cdot (\mathbf{w}_t + \mathbf{p}_i)\\ &= \|\mathbf{w}_t\|^2 + 2\mathbf{w}_t \cdot \mathbf{p}_i + \|\mathbf{p}_i\|^2\end{aligned}$$

Since $\mathbf{w}_t \cdot \mathbf{p}_i$ is negative or zero (otherwise we would have not corrected $\mathbf{w}_t$ using $\mathbf{p}_i$) we can deduce that

$$\begin{aligned}\|\mathbf{w}_{t+1}\|^2 &\leq \|\mathbf{w}_t\|^2 + \|\mathbf{p}_i\|^2\\ &\leq \|\mathbf{w}_t\|^2 + 1\end{aligned}$$

because all vectors in P have been normalized. Induction then gives us

$$\|\mathbf{w}_{t+1}\|^2 \leq \|\mathbf{w}_0\|^2 + (t+1). \tag{4.10}$$

From (4.9) and (4.10) and Equation (4.8) we get the inequality

$$\cos\rho \geq \frac{\mathbf{w}^* \cdot \mathbf{w}_0 + (t+1)\delta}{\sqrt{\|\mathbf{w}_0\|^2 + (t+1)}}$$

The right term grows proportionally to $\sqrt{t}$ and, since δ is positive, it can become arbitrarily large. However, since $\cos\rho \leq 1$, t must be bounded by a maximum value. Therefore, the number of corrections to the weight vector must be finite. □

The proof shows that perceptron learning works by bringing the initial vector $\mathbf{w}_0$ sufficiently close to $\mathbf{w}^*$ (since $\cos\rho$ becomes larger and ρ proportionately smaller).

4.2.3 Accelerating convergence

Although the perceptron learning algorithm converges to a solution, the number of iterations can be very large if the input vectors are not normalized and are arranged in an unfavorable way.

There are faster methods to find the weight vector appropriate for a given problem. When the perceptron learning algorithm makes a correction, an input vector $\mathbf{x}$ is added or subtracted from the weight vector $\mathbf{w}$. The search direction is given by the vector $\mathbf{x}$. Each input vector corresponds to the border of one region of the error function defined on weight space. The direction of $\mathbf{x}$ is orthogonal to the step defined by $\mathbf{x}$ on the error surface. The weight vector is displaced in the direction of $\mathbf{x}$ until it "falls" into a region with smaller error.

We can illustrate the dynamics of perceptron learning using the error surface for the OR function as an example. The input $(1,1)$ must produce the output 1 (for simplicity we fix the threshold of the perceptron to 1). The two weights w_1 and w_2 must fulfill the inequality $w_1 + w_2 \geq 1$. Any other combination produces an error. The contribution to the total error is shown in Figure 4.11 as a step in the error surface. If the initial weight vector lies in the triangular region with error 1, it must be brought up to the verge of the region with error 0. This can be done by adding the vector (1,1) to $\mathbf{w}$. However, if the input vector is, for example, $(0.1, 0.1)$, it should be added a few times before the weight combination (w_1, w_2) falls to the region of error 0. In this case we would like to make the correction in a single iteration.

These considerations provide an improvement for the perceptron learning algorithm: if at iteration t the input vector $\mathbf{x} \in P$ is classified erroneously, then we have $\mathbf{w}_t \cdot \mathbf{x} \leq 0$. The error δ can be defined as