6

One and Two Layered Networks

6.1 Structure and geometric visualization

In the previous chapters the computational properties of isolated threshold units have been analyzed extensively. The next step is to combine these elements and look at the increased computational power of the network. In this chapter we consider feed-forward networks structured in successive layers of computing units.

6.1.1 Network architecture

The networks we want to consider must be defined in a more precise way in terms of their *architecture*. The atomic elements of any architecture are the computing units and their interconnections. Each computing unit collects the information from n input lines with an *integration function* $\Psi : \mathbb{R}^n \rightarrow \mathbb{R}$. The total excitation computed in this way is then evaluated using an *activation function* $\Phi : \mathbb{R} \rightarrow \mathbb{R}$. In perceptrons the integration function is the sum of the inputs. The activation (also called output function) compares the sum with a threshold. Later we will generalize Φ to produce all values between 0 and 1. In the case of Ψ some functions other than addition can also be considered [454], [259]. In this case the networks can compute some difficult functions with fewer computing units.

Definition 9. *A network architecture is a tuple* (I, N, O, E) *consisting of a set* I *of input sites, a set* N *of computing units, a set* O *of output sites and a set* E *of weighted directed edges. A directed edge is a tuple* (u, v, w) *whereby* $u \in I \cup N$*,* $v \in N \cup O$ *and* $w \in \mathbb{R}$*.*

The input sites are just entry points for information into the network and do not perform any computation. Results are transmitted to the output sites. The set N consists of all computing elements in the network. Note that the edges between all computing units are weighted, as are the edges between input and output sites and computing units.

R. Rojas: Neural Networks, Springer-Verlag, Berlin, 1996

In neural network literature there is an inconsistency in notation that unfortunately has become tradition. The input sites of a network are usually called input units, although nothing is computed here. The output sites of the network are implicit in the construction but not explicitly given. The computing units from which results are read off are called the output units.

Layered architectures are those in which the set of computing units N is subdivided into ℓ subsets $N_1, N_2, \ldots, N_\ell$ in such a way that only connections from units in N_1 go to units in N_2, from units in N_2 to units in N_3, etc. The input sites are only connected to the units in the subset N_1, and the units in the subset N_ℓ are the only ones connected to the output sites. In the usual terminology, the units in N_ℓ are the output units of the network. The subsets N_i are called the *layers* of the network. The set of input sites is called the *input layer*, the set of output units is called the *output layer*. All other layers with no direct connections from or to the outside are called *hidden layers*. Usually the units in a layer are not connected to each other (although some neural models make use of this kind of architecture) and the output sites are omitted from the graphical representation.

A neural network with a layered architecture does not contain cycles. The input is processed and relayed from one layer to the other, until the final result has been computed. Figure 6.1 shows the general structure of a layered architecture.

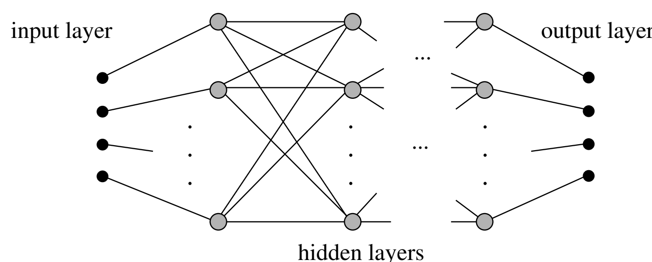

Fig. 6.1. A generic layered architecture

In layered architectures normally all units from one layer are connected to all other units in the following layer. If there are m units in the first layer and n units in the second one, the total number of weights is mn. The total number of connections can become rather large and one of the problems with which we will deal is how to reduce the number of connections, that is, how to *prune* the network.

6.1.2 The XOR problem revisited

The properties of one- and two-layered networks can be discussed using the case of the XOR function as an example. We already saw that a single perceptron cannot compute this function, but a two-layered network can. The

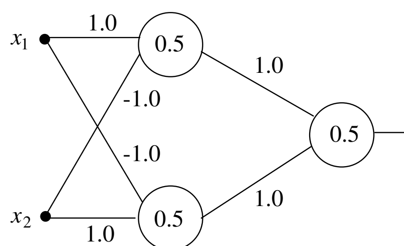

Fig. 6.2. A three-layered network for the computation of XOR

network in Figure 6.2 is capable of doing this using the parameters shown in the figure. The network consists of three layers (adopting the usual definition of the layer of input sites as input layer) and three computing units. One of the units in the hidden layer computes the function $x_1 \wedge \neg x_2$, and the other the function $\neg x_1 \wedge x_2$. The third unit computes the OR function, so that the result of the complete network computation is

$$(x_1 \wedge \neg x_2) \vee (\neg x_1 \wedge x_2).$$

A natural question to ask is how many basically different solutions can be implemented with this network. Different solutions are only those expressed with a different combination of two-dimensional binary functions, not those which differ only in the weights being used and not in the individual functions being computed. The question then is how many different expressions for XOR can be written using only three out of 14 of the 16 possible Boolean functions of two variables (since XOR and ¬XOR are not among the possible building blocks). An exhaustive search for all possible combinations can be made and the solution is the one shown in Figure 6.3.

The notation used in the figure is as follows: since we are considering logical functions of two variables, there are four possible combinations for the input. The outputs for the four inputs are four bits which uniquely distinguish each logical function. We use the number defined by these four bits as a subindex for the name of the functions. The function $(x_1, x_2) \mapsto 0$, for example, is denoted by f_0 (since 0 corresponds to the bit string 0000). The AND function is denoted by f_8 (since 8 corresponds to the bit string 1000), whereby the output bits are ordered according to the following ordering of the inputs: (1,1), (0,1), (1,0), (0,0).

The sixteen possible functions of two variables are thus:

R. Rojas: Neural Networks, Springer-Verlag, Berlin, 1996

$$
\begin{aligned}
f_0(x_1,x_2) &= f_{0000}(x_1,x_2) = 0\\
f_1(x_1,x_2) &= f_{0001}(x_1,x_2) = \neg(x_1 \vee x_2)\\
f_2(x_1,x_2) &= f_{0010}(x_1,x_2) = x_1 \wedge \neg x_2\\
f_3(x_1,x_2) &= f_{0011}(x_1,x_2) = \neg x_2\\
f_4(x_1,x_2) &= f_{0100}(x_1,x_2) = \neg x_1 \wedge x_2\\
f_5(x_1,x_2) &= f_{0101}(x_1,x_2) = \neg x_1\\
f_6(x_1,x_2) &= f_{0110}(x_1,x_2) = x_1 \oplus x_2\\
f_7(x_1,x_2) &= f_{0111}(x_1,x_2) = \neg(x_1 \wedge x_2)
\end{aligned}
$$

$$
\begin{aligned}
f_8(x1,x_2) &= f_{1000}(x_1,x_2) = x_1 \wedge x_2\\
f_9(x1,x_2) &= f_{1001}(x_1,x_2) = x_1 \equiv x_2\\
f_{10}(x1,x_2) &= f_{1010}(x_1,x_2) = x_1\\
f_{11}(x1,x_2) &= f_{1011}(x_1,x_2) = x_1 \vee \neg x_2\\
f_{12}(x1,x_2) &= f_{1100}(x_1,x_2) = x_2\\
f_{13}(x1,x_2) &= f_{1101}(x_1,x_2) = \neg x_1 \vee x_2\\
f_{14}(x1,x_2) &= f_{1110}(x_1,x_2) = x_1 \vee x_2\\
f_{15}(x1,x_2) &= f_{1111}(x_1,x_2) = 1
\end{aligned}
$$

Figure 6.3 shows all solutions found by an exhaustive search. The network of Figure 6.2 corresponds to the function composition

$$(x_1 \wedge \neg x_2) \vee (\neg x_1 \wedge x_2) = f_{14}(f_2(x_1,x_2), f_4(x_1,x_2)).$$

Increasing the number of units in the hidden layer increases the number of possible combinations available. We say that the *capacity* of the network increases. Note the symmetry in the function compositions of Figure 6.3, which is not just a random effect as we show later on.

The network of Figure 6.4 can also be used to compute the XOR function. This is not a pure layered architecture, since there are direct connections from the input sites to the output unit. The output unit computes the OR function of the two inputs but is inhibited by the first unit if both inputs are 1.

6.1.3 Geometric visualization

The symmetry of the 16 basic solutions for the XOR problem can be understood by looking at the regions defined in weight space by the two-layered network. Each of the units in Figure 6.2 separates the input space into a closed positive and an open negative half-space. Figure 6.5 shows the linear separations defined by each unit and the unit square. The positive half-spaces have been shaded.

The three regions defined in this way can be labeled with two bits: the first bit is 1 or 0 according to whether this region is included in the positive or negative half-space of the first linear separation. The second bit is 1 or 0 if it is included in the positive or negative half-space of the second linear separation. In this way we get the labeling shown in Figure 6.6.

The two units in the first layer produce the labeling of the region in which the input is located. The point $(1,1)$, for example is contained in the region 00.

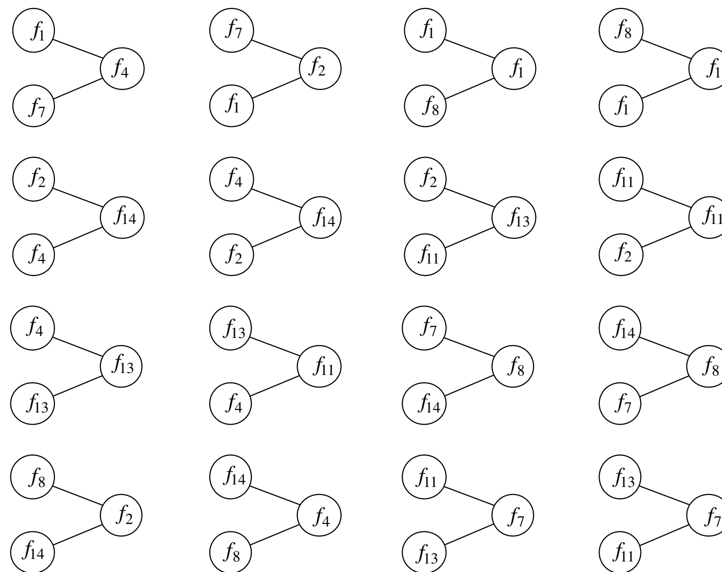

Fig. 6.3. The 16 solutions for the computation of XOR with three computing units

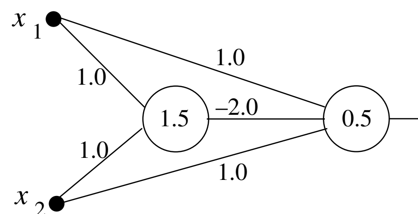

Fig. 6.4. Two unit network for the computation of XOR

This recoding of the input bits makes the XOR problem solvable, because the output unit must only decode three region labels. Only the shaded areas must produce a 1 and this can be computed with the OR function applied to the two bits of the regions labels. This is a general feature of layered architectures: the first layer of computing units maps the input vector to a second space, called classification or feature space. The units in the last layer of the network must decode the classification produced by the hidden units and compute the final output.

We can now understand in a more general setting how layered networks work by visualizing in input space the computations they perform. Each unit in the first hidden layer computes a linear separation of input space. Assume that input space is the whole of $\mathbb{R}^2$. It is possible to isolate a well-defined cluster of points in the plane by using three linear separations as shown in Figure 6.7. Assume that we are looking for a network which computes the

R. Rojas: Neural Networks, Springer-Verlag, Berlin, 1996

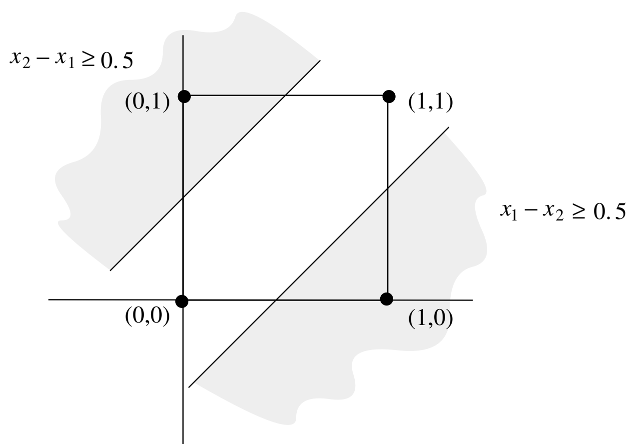

Fig. 6.5. Space separation defined by a two-layered network

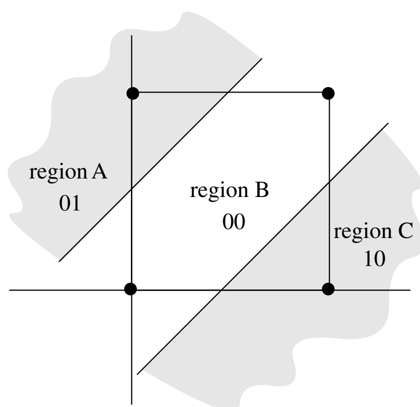

Fig. 6.6. Labeling of the regions in input space

value 1 for the points in the cluster. Three hidden units, and an output unit which computes the AND function of three inputs, can solve this problem. The output unit just decodes the label of the shaded region (111) and produces in this case a 1. Note that, in general, to define a convex cluster in an input space of dimension n at least $n+1$ hidden units are needed.

If the union of two clusters has to be identified and points in them are assigned the value 1, it is possible to use three units in the first hidden layer to enclose the first cluster and another three units in this layer to enclose the second cluster. Two AND units in the second hidden layer can identify when a point belongs to one or to the other cluster. A final output unit computes the OR function of two inputs. Such a network can identify points in the union of the two clusters. In general, any union of convex polytopes in input space can be classified in this way: units in the first hidden layer define the sides of

R. Rojas: Neural Networks, Springer-Verlag, Berlin, 1996

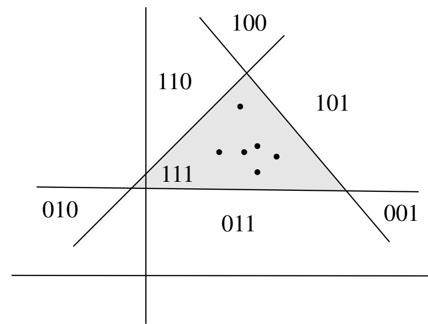

Fig. 6.7. Delimiting a cluster with three linear separations

the polytopes, the units in the second layer the conjunction of sides desired, and the final output unit computes whether the input is located inside one of the convex polytopes.

6.2 Counting regions in input and weight space

The construction used in the last section to isolate clusters is not optimal, because no effort is made to "reuse" hyperplanes already defined. Each cluster is treated in isolation and uses as many units as necessary. In general we do not know how many different clusters are contained in the data and besides the clusters do not need to be convex. We must look more deeply into the problem of how many regions can be defined by intersecting half-spaces and why in some cases a network does not contain enough "plasticity" to solve a given problem.

6.2.1 Weight space regions for the XOR problem

Assume that we are interested in finding the weight vectors for a perceptron capable of computing the AND function. The weights w_1, w_2, w_3 must fulfill the following inequalities:

$$\begin{aligned}
&\text{for the point (0,0):} 0 \cdot w_1 + 0 \cdot w_2 + 1 \cdot w_3 < 0, \text{output} = 0\,,\\
&\text{for the point (0,1):} 0 \cdot w_1 + 1 \cdot w_2 + 1 \cdot w_3 < 0, \text{output} = 0\,,\\
&\text{for the point (1,0):} 1 \cdot w_1 + 0 \cdot w_2 + 1 \cdot w_3 < 0, \text{output} = 0\,,\\
&\text{for the point (1,1):} 1 \cdot w_1 + 1 \cdot w_2 + 1 \cdot w_3 \geq 0, \text{output} = 1\,.
\end{aligned}$$

These three inequalities define half-spaces in three-dimensional weight space. The four separating planes go through the origin and are given by the equations:

R. Rojas: Neural Networks, Springer-Verlag, Berlin, 1996

plane 1: $w_3 = 0$
plane 2: $w_2 + w_3 = 0$
plane 3: $w_1 + w_3 = 0$
plane 4: $w_1 + w_2 + w_3 = 0$

Three separating planes in a three-dimensional space define 8 different regions, but four separating planes define only 14 regions. Each region corresponds to one of 14 possible combinations of inequality symbols in the set of four inequalities which defines a Boolean function. Since there are 16 Boolean functions of two variables, two of them cannot be computed with a perceptron. We already know that they are the XOR and ¬XOR functions.

We can visualize the fourteen regions with the help of a three-dimensional sphere. It was shown in Chapter 4 that the four inequalities associated with the four points (1,1), (0,1), (1,0), and (0,0) define a solution polytope in weight space. One way of looking at the regions defined by m hyperplane cuts going through the origin in an n-dimensional space is by requiring that the weight vectors for our perceptron be normalized. This does not affect the perceptron computation and is equivalent to the condition that the tip of all weight vectors should end at the unit hypersphere of dimension n. In this way all the convex regions produced by the m hyperplane cuts define solution regions on the "surface" of the unit sphere. The 14 solution polytopes define 14 solution regions. Each region corresponds to a logical function. Figure 6.8 shows some of them and their labeling. Each label consists of the four output bits for the four possible binary inputs associated with the function. The region 0000, for example, is the solution region for the function $f_{0000} = f_0$. The region 1000 is the solution region for the function $f_{1000} = f_8$, i.e., the AND function.

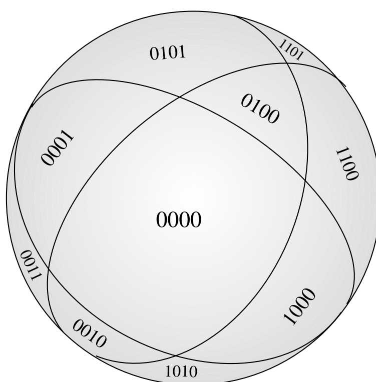

Fig. 6.8. The Boolean sphere

The labeling of neighboring regions separated by a great circle differs in just one bit, as is clear from Figure 6.9. The only regions present are delimited by three of four circles. The AND region (1000) has only the three neighbors

R. Rojas: Neural Networks, Springer-Verlag, Berlin, 1996

0000, 1100 and 0011, because the region 1001 is void. It corresponds to a non-computable function.

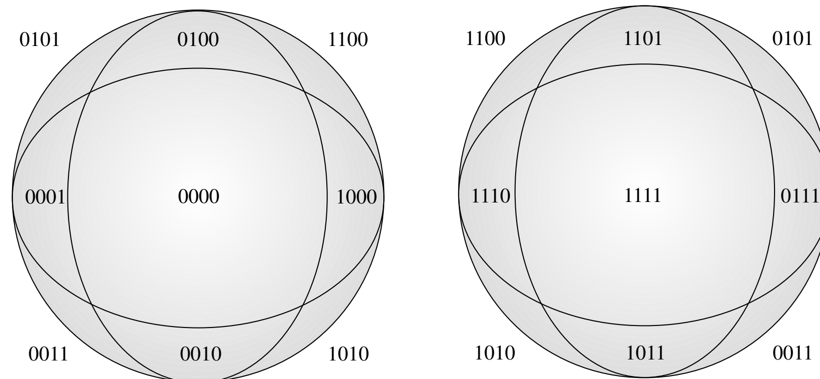

Fig. 6.9. Two opposite sides of the Boolean sphere

6.2.2 Bipolar vectors

Many models of neural networks use bipolar, not binary, vectors. In a bipolar coding the value 0 is substituted by -1. This change does not affect the essential properties of the perceptrons, but changes the symmetry of the solution regions. It is well known that the algebraic development of some terms useful for the analysis of neural networks becomes simpler when bipolar coding is used.

With a bipolar coding the equations for the 4 cutting planes of the three-dimensional Boolean sphere become

$$\begin{aligned}
\text{plane 1:} & -w_1 - w_2 + w_3 = 0 \\
\text{plane 2:} & -w_1 + w_2 + w_3 = 0 \\
\text{plane 3:} & \quad w_1 - w_2 + w_3 = 0 \\
\text{plane 4:} & \quad w_1 + w_2 + w_3 = 0
\end{aligned}$$

All three planes meet at the origin and form symmetric solution polytopes, since the vectors normal to the planes have pairwise scalar products of 1 or -1.

Since the relative sizes of the solution regions on the Boolean sphere represent how difficult it is to learn them, and since our learning algorithm will be asked to learn one of these functions randomly, the best strategy is to try to get regions of about the same relative size. Table 6.1 was calculated using a Monte Carlo method. A normalized weight vector was generated randomly and its associated Boolean function was computed. By repeating the experiment a number of times it was possible to calculate the relative volumes of the

R. Rojas: Neural Networks, Springer-Verlag, Berlin, 1996

solution regions. The table shows that the maximum variation in the relative sizes of the 14 possible regions is given by a factor of 1.33 when bipolar coding is used, whereas in the binary case it is about 12.5. This means that with binary coding some regions are almost one order of magnitude smaller than others. And indeed, it has been empirically observed that multilayer neural networks are easier to train using a bipolar representation than a binary one [341]. The rationale for this is given by the size of the regions in the Boolean sphere. It is also possible to show that bipolar coding is optimal under this criterion.

Table 6.1. Relative sizes of the regions on the Boolean sphere as percentage of the total surface

Coding	*Boolean function number*							
	0	1	2	3	4	5	6	7
binary	26.83	2.13	4.18	4.13	4.17	4.22	0.00	4.13
bipolar	8.33	6.29	6.26	8.32	6.24	8.36	0.00	6.22
Coding	*Boolean function number*							
	8	9	10	11	12	13	14	15
binary	4.28	0.00	4.26	4.17	4.17	4.14	2.07	27.12
bipolar	6.16	0.00	8.42	6.33	8.27	6.31	6.25	8.23

Bipolar coding is still better in n-dimensional space and for a large n, since two randomly selected vectors with coordinates 1 or -1 are orthogonal with a probability near 1. This is so because each component is 1 or -1 with probability 1/2. The expected value of the scalar product is small compared to the length of the two vectors, which is $\sqrt{n}$. Since the separating planes in weight space are defined by these vectors, they also tend to be mutually orthogonal when bipolar coding is used. The expected value of the scalar product of binary vectors, on the other hand, is $n/4$, which is not negligible when compared to $\sqrt{n}$, even for large n.

6.2.3 Projection of the solution regions

Another way of looking at the solution regions in the surface of the Boolean sphere is by projecting them onto a plane. A kind of stereographic projection can be used in this case. The stereographic projection is shown in Figure 6.10. From the north pole of the sphere a line is projected to each point P on the surface of the sphere and its intersection with the plane is the point in the plane associated with P. This defines a unique mapping from the surface of the sphere to the plane, adopting the convention that the north pole itself is mapped to a point at infinity.

R. Rojas: Neural Networks, Springer-Verlag, Berlin, 1996

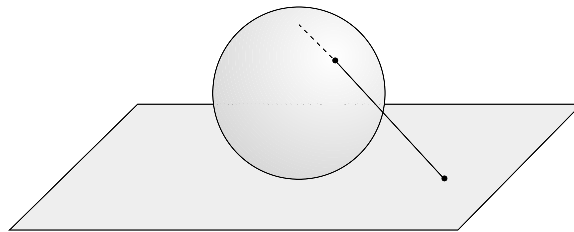

Fig. 6.10. Stereographic projection

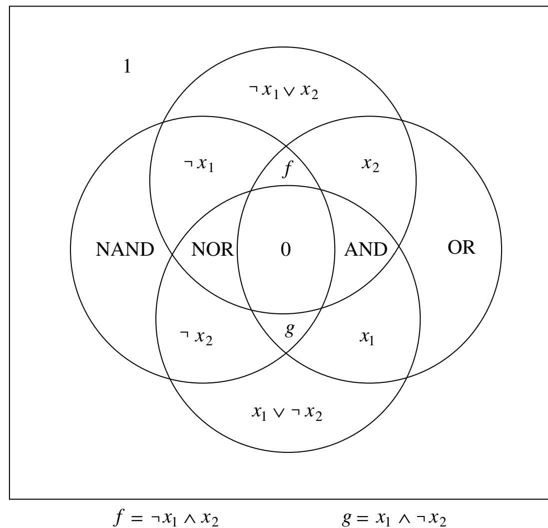

Fig. 6.11. Projection of the solution regions of the Boolean sphere

The stereographic projection projects circles on the sphere (which do not touch the north pole) to ellipses. The four cuts produced by the four separating hyperplanes define four circles on the surface of the Boolean sphere, and these in turn four ellipses on the projection plane. Since we are not interested in the exact shape of these projections, but in the regions they define on the plane, we transform them into circles. Figure 6.11 shows the result of such a projection when the center of region 1111 is chosen as the north pole.

Instead of repeating the four-bit labeling of the regions in Figure 6.11 the expressions for the logical functions "contained" in each region are written explicitly. It is obvious that the number of regions cannot be increased, because four circles on a plane cannot define more than 14 different regions. This result is related to the *Euler characteristic* of the plane [63].

The symmetry of the solution regions is made more evident by adopting a stylized representation. Only the neighborhood relations are important for

R. Rojas: Neural Networks, Springer-Verlag, Berlin, 1996

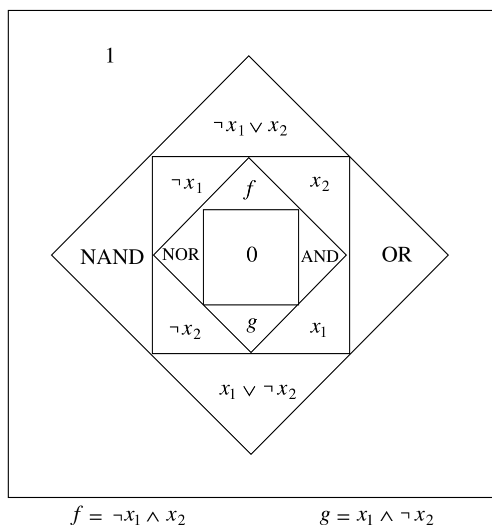

Fig. 6.12. Stylized representation of the projected solution regions

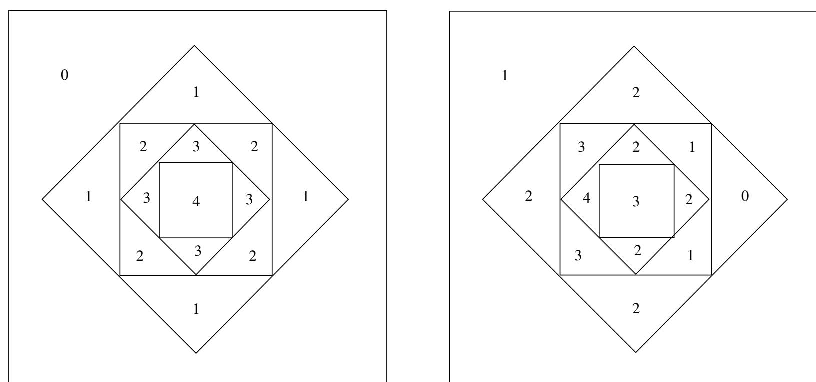

Fig. 6.13. Error functions for the computation of f_{1111} and $x_1 \vee x_2$

our discussion, so we transform Figure 6.11 into Figure 6.12. We can see that functions f and $\neg f$ are always located in symmetrical regions of the space.

The number of neighbors of each region is important if an iterative algorithm is used which, starting from a randomly chosen point, goes from one region to another, trying to minimize the classification error, as the perceptron learning algorithm does. The error function for a given Boolean function of two variables can be represented with this kind of diagram. Figure 6.13 shows the values of the error in each region when looking for the parameters to compute the function f_{1111}.

There is a global maximum and a global minimum only. The first is the solution region for f_{0000}, the second, the solution region for f_{1111}. Starting

R. Rojas: Neural Networks, Springer-Verlag, Berlin, 1996

from a randomly selected point, a possible strategy is to greedily descend the error function. From each region with error 1 there is one path leading to a region with error 0. From regions with error two, there are two alternatives and from regions with error three, three possible paths.

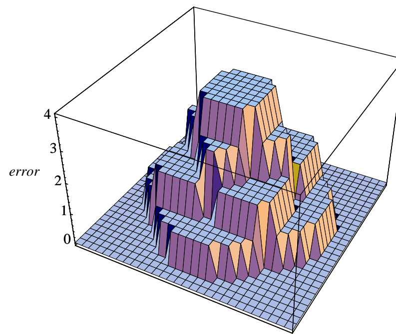

Fig. 6.14. The error function for f_{1111}

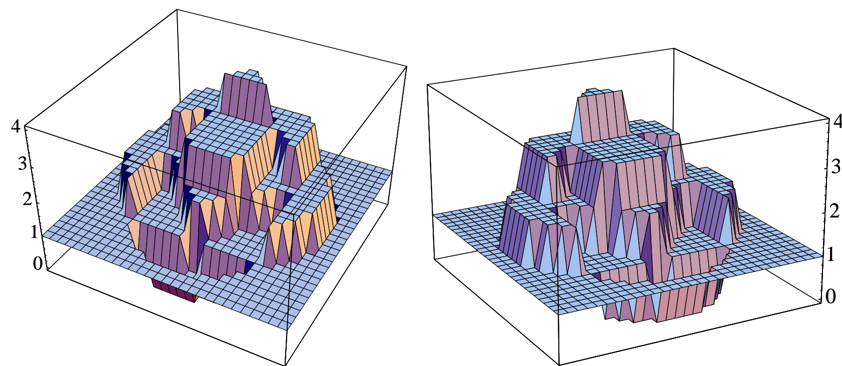

Fig. 6.15. Two perspectives of the error function for OR

The error distribution in weight space in shown in Figure 6.13. The global maximum lies in the solution region for ¬XOR. From each region there is a path which leads to the global minimum.

Figures 6.14 and 6.15 show the error functions for the functions f_{1111} and f_{1110} (OR). The global maxima and minima can be readily identified.

6.2.4 Geometric interpretation

We analyzed the surface of the Boolean sphere in so much detail because it gives us a method to interpret the functioning of networks with a hidden layer. This is the problem we want to analyze now.

Consider a network with three perceptrons in the hidden layer and one output unit. An input vector is processed and the three hidden units produce a new code for it using three bits. This new code is then evaluated by the output unit. Each unit in the hidden layer separates input space into two half-spaces. In order to simplify the visualization, we only deal with normalized vectors. Each division of input space is equivalent to a subdivision of the unit sphere which now represents vectors in input space. Figure 6.16 shows a stylized graphical representation of this idea. The input vector has three components.

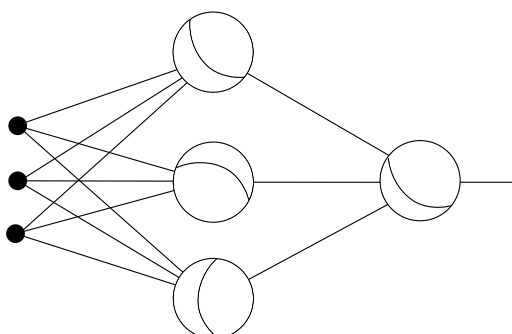

Fig. 6.16. Stylized representation of the input space separations

The separation of input space can be summarized in a single unit sphere in three-dimensional space (Figure 6.17). The three units in the hidden layer produce the labeling of three bits for each region on the sphere. The output unit decodes the three bits and, according to the region, computes a 1 or a 0.

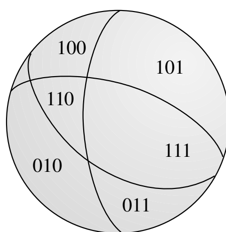

Fig. 6.17. Labeling of the regions in input space

R. Rojas: Neural Networks, Springer-Verlag, Berlin, 1996

This kind of representation leads to an important idea: the "shattering" of an input space by a class of concepts. In computational learning theory we are interested in dealing with elements from an input space which can be arranged into subsets or classes. If a subset S of an input space X is given, and its points are assigned the value 1 or 0, we are interested in determining the "concept" which can correctly classify this subset of X. In the case of perceptrons the input space is $\mathbb{R}^n$ and the concepts are half-spaces. If the positive elements of subset S (the elements with associated value 1) are located in one half-space, then it is said to be learnable, because one of our concepts (i.e., a half-space) can correctly classify the points of S. One important question is, what is the maximum number of elements of an input space which can be classified by our concepts. In the case of perceptrons with two inputs, this number is three. We can arrange three points arbitrarily in $\mathbb{R}^2$ and assign each one a 0 or a 1, and there is always a way to separate the positive from the negative examples. But four points in general position cannot be separated and the XOR function illustrates this fact.

What kind of shatterings (divisions of input space) are produced by a network with two units in the hidden layer? This question is easier to answer by considering the surface of the unit sphere in input space. In general the two units in the hidden layer divide the surface of the sphere into four regions. The output unit assigns a 1 or a 0 to each region. Figure 6.18 shows the possible shatterings. Regions in which the input is assigned the value 1 have been shaded and regions in which the input is mapped to zero are shown in white. There are sixteen possible colorings, but two of them are impossible because the output unit cannot decode the XOR function. Our network can only produce 14 different shatterings.

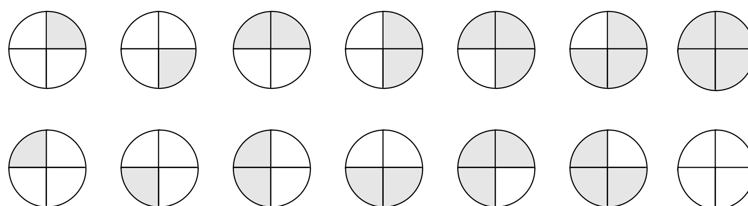

Fig. 6.18. Coloring of the regions in input space

The XOR problem can be solved with one of the shatterings of Figure 6.18 and, in general, any four points in input space can be divided arbitrarily into a positive and a negative class using two hidden units (two dividing lines). However, eight points in input space cannot be divided using only two lines. Consider the example of Figure 6.19. In this case the points at the corners of the square belong to the positive class, the points in the middle of each edge to the negative class. It is not difficult to see that no combination of two separating lines can divide input space in such a way as to separate

R. Rojas: Neural Networks, Springer-Verlag, Berlin, 1996

both classes. In this case we say that the shattering produced by the class of concepts represented by our network does not cover all possible subsets of eight points and not every Boolean function defined on these eight points is *learnable*.

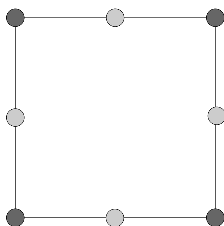

Fig. 6.19. Example of a non-learnable concept for two linear separations

The maximum number of points d of an input space X which can be shattered by a class of concepts C is called the Vapnik-Chervonenkis dimension of the class of concepts C. We will come back to this important definition after learning how to count the threshold functions.

6.3 Regions for two layered networks

We now proceed to formalize the intuitive approach made possible by the graphical representation and examine especially the problem of counting the number of solution regions for perceptron learning defined by a data set.

6.3.1 Regions in weight space for the XOR problem

We can now deal with other aspects of the XOR problem and its solution using a network of three units. Since nine parameters must be defined (two weights and a threshold per unit), weight space is nine-dimensional. We already know that there are sixteen different solutions for the XOR problem with this network, but what is the total number of solution regions for this network?

Let w_1, w_2, w_3 be the weights for the first unit, w_4, w_5, w_6 the weights for the second unit and w_7, w_8, w_9 the weights for the output unit. Let x_1 and x_2 denote the components of the input vector and y_1 and y_2 the outputs of the hidden units. These inputs for each unit define a set of separating hyperplanes in weight space. The set of equations for the two hidden units is

$$\begin{array}{ll}
0 \cdot w_1 + 0 \cdot w_2 + 1 \cdot w_3 = 0 & 0 \cdot w_4 + 0 \cdot w_5 + 1 \cdot w_6 = 0 \\
0 \cdot w_1 + 1 \cdot w_2 + 1 \cdot w_3 = 0 & 0 \cdot w_4 + 1 \cdot w_5 + 1 \cdot w_6 = 0 \\
1 \cdot w_1 + 0 \cdot w_2 + 1 \cdot w_3 = 0 & 1 \cdot w_4 + 0 \cdot w_5 + 1 \cdot w_6 = 0 \\
1 \cdot w_1 + 1 \cdot w_2 + 1 \cdot w_3 = 0 & 1 \cdot w_4 + 1 \cdot w_5 + 1 \cdot w_6 = 0
\end{array}$$

R. Rojas: Neural Networks, Springer-Verlag, Berlin, 1996

For the output unit there are as many cutting hyperplanes as there are (y_1, y_2)-combinations produced by the hidden units. The form of the equation for each plane is:

$$y_1 w_7 + y_2 w_8 + 1 \cdot w_9 = 0.$$

The separating hyperplanes of the first unit are orthogonal to the separating hyperplanes of the second unit and of the output unit. The first four cuts can generate at most 14 different regions (as in the case of the simple perceptron). The cuts defined by the second unit are also at most 14 and the same happens with the output unit. The maximum total number of regions we get by combining these orthogonal cuts is $14 \cdot 14 \cdot 14$, that is, 2744 polytopes.

This result can be interpreted as meaning that the number of solution regions defined at the surface of the nine-dimensional Boolean sphere is 2744. Sixteen of them are solutions for the XOR problem and 16 for ¬XOR problem. The XOR problem can be solved with this network because the number of solution regions in weight space was increased enormously. Is it possible to proceed as in perceptron learning, by descent on the error function of the network, in order to find an appropriate set of parameters? What is the shape of this error function? Before answering these questions we turn to a topological problem.

6.3.2 Number of regions in general

The capacity of a unit or network depends on the dimension of weight space and the number of cuts with separating hyperplanes. The general question to be answered is: how many regions are defined by m cutting hyperplanes of dimension $n - 1$ in an n-dimensional space? We consider only the case of hyperplanes going through the origin but otherwise in *general position*. This means that the intersection of $\ell \leq n$ hyperplanes is of dimension $n - \ell$.

The two-dimensional case is simple: m lines going through the origin define at most $2m$ different regions. Each new line can only go through the cone defined by two previous lines, dividing its two sides in two and adding two new regions in this way.

The three-dimensional case with one, two, or three cuts is simple too. Each cut increases the number of regions by a factor 2. In general: n cuts with $(n - 1)$-dimensional hyperplanes in n-dimensional space define 2^n different regions.

The three-dimensional case with four cutting hyperplanes can be solved by projecting on dimension two. The three-dimensional input space is first cut three times with planes in general position. The fourth cutting plane intersects the three previous planes at three different lines. These three lines define a maximum of six regions on the fourth separating hyperplane. This means that the fourth cutting hyperplane divides at most six of the eight existing regions. After the cut with the fourth hyperplane there are six new regions which, added to the eight old ones, gives a total of 14. The general case can be solved using a similar argument.

Proposition 9. *Let $R(m,n)$ denote the number of different regions defined by m separating hyperplanes of dimension $n-1$, in general position, in an n-dimensional space. We set $R(1,n)=2$ for $n \geq 1$ and $R(m,0)=0, \forall m \geq 1$. For $n \geq 1$ and $m > 1$*

$$R(m,n) = R(m-1,n) + R(m-1,n-1).$$

Proof. The proof is by induction on m. When $m = 2$ and $n = 1$ the formula is valid. If $m = 2$ and $n \geq 2$ we know that $R(2,n) = 4$ and the formula is valid again:

$$R(2,n) = R(1,n) + R(1,n-1) = 2 + 2 = 4.$$

Now $m+1$ hyperplanes of dimension $n-1$ are given in n-dimensional space and in general position ($n \geq 2$). From the induction hypotheses it follows that the first m hyperplanes define $R(m,n)$ regions in n-dimensional space. The hyperplane $m+1$ intersects the first m hyperplanes in m hyperplanes of dimension $n-2$ (since all are in general position). These m hyperplanes divide the $(n-1)$-dimensional space into $R(m,n-1)$ regions. After the cut with the hyperplane $m+1$, exactly$R(m,n-1)$ new regions have been created. The new number of regions is therefore $R(m+1,n) = R(m,n) + R(m,n-1)$ and the proof by induction is complete. □

This result can be represented using a table. Each column of the table corresponds to a different dimension of input space and each row to a different number of separating hyperplanes. The table shows some values for $R(m,n)$.

	dimension				
m \ n	0	1	2	3	4
1	0	2	2	2	2
2	0	2	4	4	4
3	0	2	6	8	8
4	0	2	8	14	16
5	0	2	10	22	30

Fig. 6.20. Recursive calculation of $R(m,n)$

It follows from the table that $R(m,n) = 2^m$ whenever $m \leq n$. This means that the number of regions increases exponentially until the dimension of the space puts a limit to this growth. For $m > n$ the rate of increase becomes polynomial instead of exponential. For $n = 2$ and $n = 3$ we can derive analytic expressions for the polynomials: $R(m,2) = 2m$ and $R(m,3) = m^2 - m + 2$. The following proposition shows that this is not accidental.

R. Rojas: Neural Networks, Springer-Verlag, Berlin, 1996

Proposition 10. *For $n \geq 1$, $R(m,n)$ is a polynomial of degree $n-1$ on the variable m.*

Proof. The proof follows from induction on n. We denote with $P(a,b)$ a polynomial of degree b on the variable a. The polynomial was explicitly given for $n=2$. For dimension $n+1$ and $m=1$ we know that $R(1,n+1)=2$. If $m>1$ then

$$R(m,n+1) = R(m-1,n+1) + R(m-1,n).$$

Since $R(m-1,n)$ is a polynomial of degree $n-1$ in the variable m it follows that

$$R(m,n+1) = R(m-1,n+1) + P(m,n-1).$$

Repeating this reduction $m-1$ times we finally get

$$\begin{aligned} R(m,n+1) &= R(m-(m-1),n+1) + (m-1)P(m,n-1) \\ &= 2 + (m-1)P(m,n-1) \\ &= P(m,n) \end{aligned}$$

$R(m,n+1)$ is thus a polynomial of degree n in the variable m and the proof by induction is complete. □

A useful formula for $R(m,n) is$

$$R(m,n) = 2\sum_{i=0}^{n-1} \binom{m-1}{i}.$$

The validity of the equation can be proved by induction. It allows us to compute $R(m,n)$ iteratively [271]. Note that this formula tells us how many regions are formed when hyperplanes meet in a general position. In the case of Boolean formulas with Boolean inputs, the hyperplanes in weight space have binary or bipolar coefficients, that is, they do not lie in a general position. The number of regions defined in a 4-dimensional weight space by 8 hyperplanes is, according to the formula, 128. But there are only 104 threshold functions computable with a perceptron with three input lines (and therefore four parameters and eight possible input vectors). The number $R(m,n)$ must then be interpreted as an *upper bound* on the number of logical functions computable with binary inputs.

It is easy to find an upper bound for $R(m,n)$ which can be computed with a few arithmetical operations [457]:

$$R(m,n) < 2\frac{m^n}{n!}.$$

Table 6.2 shows how these bounds behave when the number of inputs n is varied from 1 to 5 [271]. The number of threshold functions of n inputs is denoted in the table by $T(2^n,n)$.

R. Rojas: Neural Networks, Springer-Verlag, Berlin, 1996

Table 6.2. Comparison of the number of Boolean and threshold functions of n inputs and two different bounds

n	2^{2^n}	$T(2^n, n)$	$R(2^n, n)$	$\lfloor 2^{n^2+1}/n! \rfloor$
1	4	4	4	4
2	16	14	14	16
3	256	104	128	170
4	65,536	1,882	3,882	5,461
5	4.3×10^9	94,572	412,736	559,240

6.3.3 Consequences

Two important consequences become manifest from the analysis just performed.

- *First consequence.* The number of threshold functions in an n-dimensional space grows polynomially whereas the number of possible Boolean functions grows exponentially. The number of Boolean functions definable on n Boolean inputs is 2^{2^n}. The number of threshold functions is a function of the form $2^{n(n-1)}$, since the 2^n input vectors define at most $R(2^n, n)$ regions in weight space, that is, a polynomial of degree $n-1$ on 2^n. The percentage of threshold functions in relation to the total number of logical functions goes to zero as n increases.
- *Second consequence.* In networks with two or more layers we also have learnability problems. Each unit in the first hidden layer separates input space into two halves. If the hidden layer contains m units and the input vector is of dimension n, the maximum number of classification regions is $R(m, n)$. If the number of input vectors is higher, it can happen that not enough classification regions are available to compute a given logical function. Unsolvable problems for all networks with a predetermined number of units can easily be fabricated by increasing the number of input lines into the network.

Let us give an example of a network and its computational limits. The network consists of two units in the hidden layer and one output unit. The extended input vectors are of dimension n. The number of weights in the network is $2n+3$. The number of different input vectors is 2^{n-1}. The number of regions N in weight space defined by the input vectors is bounded by

$$N \leq R(2^{n-1}, n) \cdot R(2^{n-1}, n) \cdot R(4, 3).$$

This means that N is bounded by a function of order $2^{2(n-1)^2}$. Since the number of functions F definable on n inputs is 2^{2^n}, there is a value of n which guarantees $F > N$. Some of the Boolean functions are therefore not

R. Rojas: Neural Networks, Springer-Verlag, Berlin, 1996

computable with this network. Such unsolvable problems can always be found by just overloading the network capacity. The converse is also true: to solve certain problems, the network capacity must be increased if it is not sufficiently high.

6.3.4 The Vapnik–Chervonenkis dimension

Computation of the number of regions defined by m hyperplanes in n-dimensional weight space brings us back to the consideration of the Vapnik-Chervonenkis dimension of a class of concepts and its importance for machine learning.

Assume that a linear separation of the points in the unit square is selected at random by an opponent and we have to find the parameters of the separating line (Figure 6.21). We can select points in the unit square randomly and ask our opponent for their classification, that is, if they belong to the positive or negative half-space. We refine our computation with each new example and we expect to get better and better approximations to the right solution.

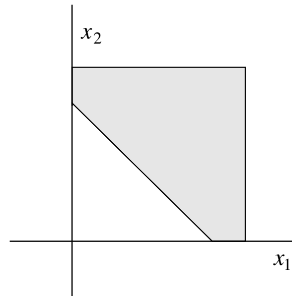

Fig. 6.21. Linear separation of the unit square

Figure 6.22 shows how more examples reduce the range of possible linear separations we can choose from. The linear separations defined by the lines ℓ_1 and ℓ_2 are compatible with the selected points and their classification. All other lines between ℓ_1 and ℓ_2 are also compatible with the known examples. The margin of error, however, is larger in the first than in the second case. Both extreme lines ℓ_1 and ℓ_2 converge asymptotically to the correct linear separation.

If a perceptron is trained with a randomly selected set of examples and tested with another set of points, we call the expected number of correct classifications the *generalization* capability of the perceptron. Generalization becomes better if the training set is larger. In other types of problem in which another class of concepts is used, it is not necessarily so. If, for example, a polynomial of arbitrary degree is fitted to a set of points, it can well happen that the function is *overfitted*, that is, it learns the training set perfectly

R. Rojas: Neural Networks, Springer-Verlag, Berlin, 1996

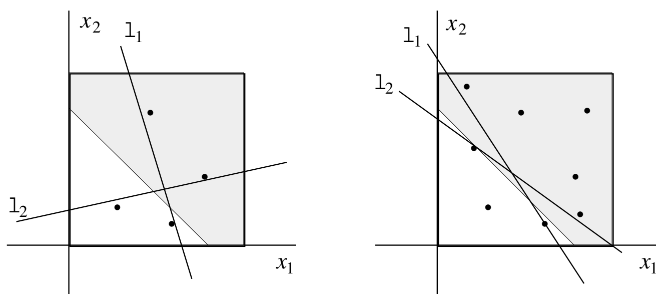

Fig. 6.22. Linear separations compatible with the examples

but interpolates unknown points erroneously. Figure 6.23 shows how this can happen. The learned function oscillates excessively in order to accommodate all points in the training set without error but is very different from the unknown function, which is smoother. More points do not reduce the error as long as our class of concepts consists of all polynomials of arbitrary degree. The only possibility of profiting from more examples is to reduce the size of the search space by, for example, putting a limit on the degree of the acceptable polynomial approximations.

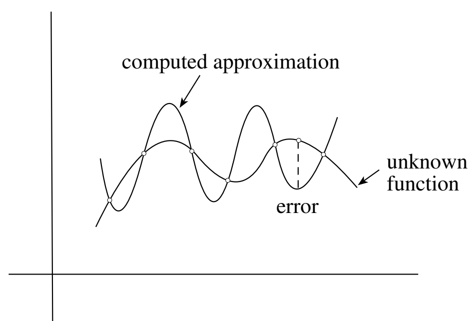

Fig. 6.23. Overfitting a polynomial approximation

A desirable property of the learning process is that it converges to the unknown function with high probability. Additional examples and the minimization of the classification error should bring us monotonically closer to the solution of the problem. We demand that the absolute value of the difference of the approximating function and the unknown function at any point in the input space be less than a given $\varepsilon > 0$.

Neural network's learning consists of approximating an unknown function g with a network function f. Let π_f be the probability that the network function f computes the correct classification of a point chosen randomly in input space. Let ν_f stand for the empirical error rate measured by sampling the

R. Rojas: Neural Networks, Springer-Verlag, Berlin, 1996

input space and testing the classification computed by the network function f. The learning algorithm should guarantee the uniform convergence of ν_f to π_f. In this way, using ν_f as the learning criterion correctly reflects the effective success rate π_f of the network function.

Vapnik and Chervonenkis [437] found a condition for the uniform convergence of ν_f to π_f. They proved the following inequality

$$Pr[\sup_{f \in F} |\nu_f - \pi_f| > \varepsilon] \leq 4\phi(2N)e^{-\varepsilon^2 N/8},$$

which states that the probability that ν_f and π_f of the network function f chosen from the model (the set of computable network functions F) differ by more than ε is smaller than $4\phi(2N)e^{-\varepsilon^2 N/8}$. The variable N stands for the number of examples used and $\phi(2N)$ is the number of binary functions in search space which can be defined over $2N$ examples.

Note that the term $e^{-\varepsilon^2 N/8}$ falls exponentially in the number of examples N. In this case ν_f can come exponentially closer to π_f as long as the number of binary functions definable on $2N$ examples does not grow exponentially. But remember that given a set of points of size $2N$ the number of possible binary labelings is 2^{2N}. The Vapnik-Chervonenkis dimension measures how many binary labelings of a set of points can be computed by one member of a class of concepts. As long as the class of concepts is only capable of covering a polynomial number of these labelings, $\phi(2N)$ will not grow exponentially and will not win the race against the factor $e^{-\varepsilon^2 N/8}$.

Vapnik and Chervonenkis showed that $\phi(2N)$ is bounded by $N^d + 1$, where d is the VC-dimension of the class of concepts. If its VC-dimension is finite we call a class of concepts *learnable*. Perceptron learning is learnable because the VC-dimension of a perceptron with n weights (including the threshold) is finite.

In the case of a perceptron like the one computing the linear separation shown in Figure 6.21, if the empirical success rate $\nu_f = 1$ the probability that it differs from π_f by more than ε is

$$Pr[\sup_{f \in F}(1 - \pi_f) > \varepsilon] \leq 4R(2N, n)e^{-\varepsilon^2 N/8}, \tag{6.1}$$

since the number of labelings computable by a perceptron on $2N$ points in general position is equal to $R(2N, n)$, that is, the number of solution regions available in weight space. Since $R(2N, n)$ is a polynomial of degree $n-1$ in the variable N and the term $e^{-\varepsilon^2 N/8}$ goes exponentially to zero, the generalization error margin falls exponentially to zero as the number of examples increases.

6.3.5 The problem of local minima

One of the fundamental problems of iterative learning algorithms is the existence of local minima of the error function. In the case of a single perceptron

R. Rojas: Neural Networks, Springer-Verlag, Berlin, 1996

the error function has a single global minimum region. This is not so with more complex networks.

In the case of the network of three units used in this chapter to compute all solutions for the XOR problem, there are four classes of regions in weight space with associated error from 0 to 4, that is, any subset of the four input vectors can be correctly classified or not. Using an exhaustive search over all possible regions in weight space, it can be shown that there are no spurious local minima in the error function. A path can always be found from regions with error greater than zero to any other region with smaller error.

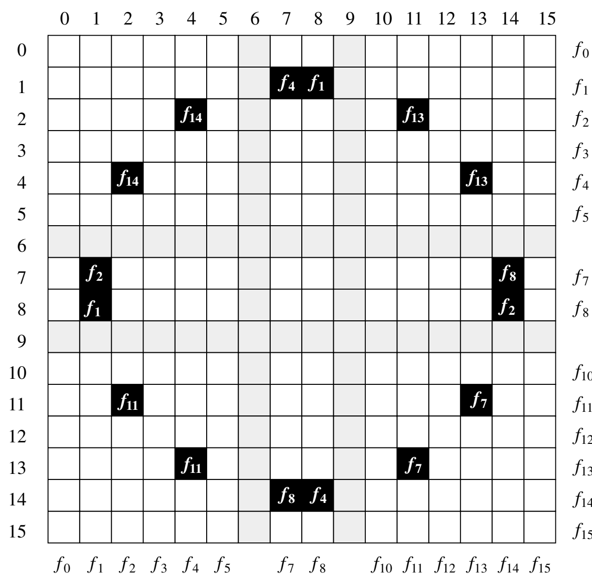

Fig. 6.24. Distribution of the solutions for XOR

Figure 6.24 shows a diagram of the distribution of the solution regions for the XOR problem. Since each of the units in the hidden layer can compute 14 Boolean functions and they are independent, and since the associated cuts in weight space are orthogonal, we can divide the surface of the nine-dimensional Boolean sphere into 14×14 regions. Each one of these is subdivided by the 14 regions defined by the output unit. The column labeling of the diagram corresponds to the function computed by the first hidden unit, the row labeling to the function computed by the second hidden unit. The dark regions indicate where a solution region for the XOR problem can be found and which function is computed by the output unit. The shadowed columns and rows correspond to the XOR and $\neg$XOR functions, which are not computable by each of the hidden units. The diagram shows that the solution regions are distributed symmetrically on the surface of the Boolean sphere. Although this simple

R. Rojas: Neural Networks, Springer-Verlag, Berlin, 1996

diagram cannot capture the whole complexity of the neighborhood relations in a nine-dimensional space, it gives a pretty good idea of the actual distribution of solutions. The symmetrical distribution of the solution regions is important, because we can start randomly from any point in weight space and a solution is never very far away. The problem, however, is deciding in what direction to start the search. We will deal with this problem in the next chapter by generalizing the kind of activation functions acceptable in each unit.

6.4 Historical and bibliographical remarks

Although networks with several layers of computing units were proposed right at the beginning of the development of neural network models, the problem which limited their applicability was that no reliable learning method was known. Rosenblatt experimented with a kind of learning in which the error at the output was propagated to elements in the first layers of computing units.

Another important problem is the location and number of local minima of the error function which can lead the learning algorithm astray. Hecht-Nielsen [186] and Poston et al. [349] have discussed the structure of the error function. Others, like Hush et al. [206], developed similar visualization methods to explore the shape of the error function.

Threshold functions were studied intensively in the 1960s and the bounds on the number of threshold functions given in this chapter were derived at that time. It is possible to characterize any threshold function of n inputs uniquely by a set of $n+1$ parameters, as was shown by Chow and by Dertouzos [401]. The Chow coefficients correspond to the centroid of the vertices with function value 1 on the n-dimensional binary hypercube.

Much research has been done on the topological properties of polytopes and figures on spheres [388]. Nilsson [329] and others studied the importance of the number of regions defined by cutting hyperplanes relatively early. The number of regions defined by cuts in an n-dimensional space was studied in more general form by Euler [29]. The relation between learnability and the Euler characteristic was studied by Minsky and Papert [312].

In the 1970s it became clear that Vapnik and Chervonenkis' approach provided the necessary tools for a general definition of "learnable problems". Valiant [436] was one of the first to propose such a model-independent theory of learning. In this approach the question to be answered is whether the search space in the domain of learnable functions can be restricted in polynomial time. The VC-dimension of the class of concepts can thus help to determine learnability. The VC-dimension of some network architectures has been studied by Baum [44]. Some authors have studied the VC-dimension of other interesting classes of concepts. For example, the VC-dimension of sparse polynomials over the reals, that is polynomials with at most t monomials, is linear in t and thus this class of concepts can be uniformly learned [238]. Such sparse polynomials have a finite VC-dimension because they do

R. Rojas: Neural Networks, Springer-Verlag, Berlin, 1996

not have enough plasticity to shatter an infinite number of points, since their number of different roots is bounded by $2t - 1$.

Exercises

1. Consider the Boolean functions of two arguments. Write a computer program to measure the relative sizes of the 14 solution regions for perceptron learning.
2. Figure 6.19 shows that eight points on the plane can produce non-learnable concepts for two linear separations. What is the *minimum* number of points in $\mathbb{R}^2$ which can produce a non-learnable concept using two linear separations?
3. Write a computer program to test the validity of equation (6.1) for a linear separation of the type shown in Figure 6.21.
4. Consider a perceptron that accepts complex inputs x_1, x_2. The weights w_1, w_2 are also complex numbers, and the threshold is zero. The perceptron fires if the condition $\text{Re}(x_1 w_1 + x_2 w_2) \geq \text{Im}(x_1 w_1 + x_2 w_2)$ is satisfied. The binary input 0 is coded as the complex number $(1, 0)$ and the binary input 1 as the number $(0, 1)$. How many of the logical functions of two binary arguments can be computed with this system? Can XOR be computed?
5. Construct a non-learnable concept in $\mathbb{R}^2$ for three linear separations.