3

Weighted Networks – The Perceptron

3.1 Perceptrons and parallel processing

In the previous chapter we arrived at the conclusion that McCulloch–Pitts units can be used to build networks capable of computing any logical function and of simulating any finite automaton. From the biological point of view, however, the types of network that can be built are not very relevant. The computing units are too similar to conventional logic gates and the network must be completely specified before it can be used. There are no free parameters which could be adjusted to suit different problems. Learning can only be implemented by modifying the connection pattern of the network and the thresholds of the units, but this is necessarily more complex than just adjusting numerical parameters. For that reason, we turn our attention to weighted networks and consider their most relevant properties. In the last section of this chapter we show that simple weighted networks can provide a computational model for regular neuronal structures in the nervous system.

3.1.1 Perceptrons as weighted threshold elements

In 1958 Frank Rosenblatt, an American psychologist, proposed the *perceptron*, a more general computational model than McCulloch–Pitts units. The essential innovation was the introduction of numerical weights and a special interconnection pattern. In the original Rosenblatt model the computing units are threshold elements and the connectivity is determined stochastically. Learning takes place by adapting the weights of the network with a numerical algorithm. Rosenblatt's model was refined and perfected in the 1960s and its computational properties were carefully analyzed by Minsky and Papert [312]. In the following, Rosenblatt's model will be called the *classical perceptron* and the model analyzed by Minsky and Papert the *perceptron*.

The classical perceptron is in fact a whole network for the solution of certain pattern recognition problems. In Figure 3.1 a projection surface called the

retina transmits binary values to a layer of computing units in the projection area. The connections from the retina to the projection units are deterministic and non-adaptive. The connections to the second layer of computing elements and from the second to the third are stochastically selected in order to make the model biologically plausible. The idea is to train the system to recognize certain input patterns in the connection region, which in turn leads to the appropriate path through the connections to the reaction layer. The learning algorithm must derive suitable weights for the connections.

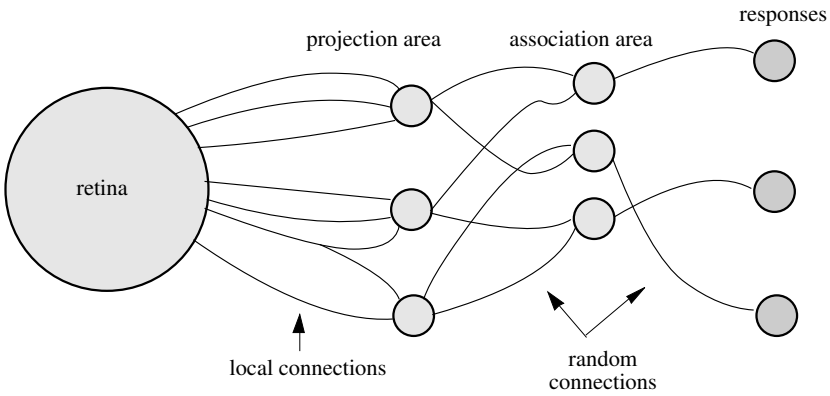


Fig. 3.1. The classical perceptron [after Rosenblatt 1958]

Rosenblatt's model can only be understood by first analyzing the elementary computing units. From a formal point of view, the only difference between McCulloch–Pitts elements and perceptrons is the presence of weights in the networks. Rosenblatt also studied models with some other differences, such as putting a limit on the maximum acceptable fan-in of the units.

Minsky and Papert distilled the essential features from Rosenblatt's model in order to study the computational capabilities of the perceptron under different assumptions. In the model used by these authors there is also a retina of pixels with binary values on which patterns are projected. Some pixels from the retina are directly connected to logic elements called predicates which can compute a single bit according to the input. Interestingly, these predicates can be as computationally complex as we like; for example, each predicate could be implemented using a supercomputer. There are some constraints however, such as the number of points in the retina that can be simultaneously examined by each predicate or the distance between those points. The predicates transmit their binary values to a weighted threshold element which is in charge of reaching the final decision in a pattern recognition problem. The question is then, what kind of patterns can be recognized in this massively parallel manner using a single threshold element at the output of the network? Are there limits to what we can compute in parallel using unlimited processing power

for each predicate, when each predicate cannot itself look at the whole retina? The answer to this problem in some ways resembles the speedup problem in parallel processing, in which we ask what percentage of a computational task can be parallelized and what percentage is inherently sequential.

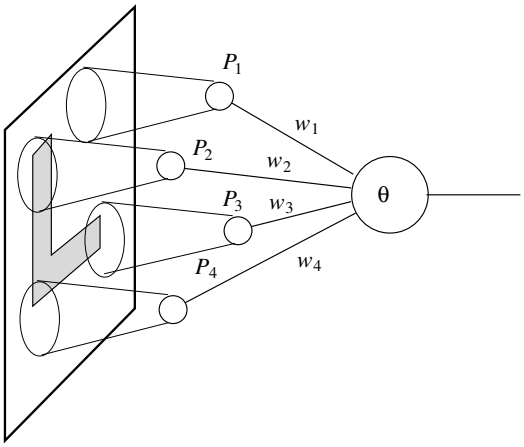


Fig. 3.2. Predicates and weights of a perceptron

Figure 3.2 illustrates the model discussed by Minsky and Papert. The predicates P_1 to P_4 deliver information about the points in the projection surface that comprise their *receptive fields*. The only restriction on the computational capabilities of the predicates is that they produce a binary value and the receptive field cannot cover the whole retina. The threshold element collects the outputs of the predicates through weighted edges and computes the final decision. The system consists in general of n predicates $P_1, P_2, \ldots, P_n$ and the corresponding weights $w_1, w_2, \ldots, w_n$. The system fires only when $\sum_{i=1}^{n} w_i P_i \geq \theta$, where θ is the threshold of the computing unit at the output.

3.1.2 Computational limits of the perceptron model

Minsky and Papert used their simplified perceptron model to investigate the computational capabilities of weighted networks. Early experiments with Rosenblatt's model had aroused unrealistic expectations in some quarters, and there was no clear understanding of the class of pattern recognition problems which it could solve efficiently. To explore this matter the number of predicates in the system is fixed, and although they possess unbounded computational power, the final bottleneck is the parallel computation with a single threshold element. This forces each processor to *cooperate* by producing a partial result pertinent to the global decision. The question now is which problems can be solved in this way and which cannot.

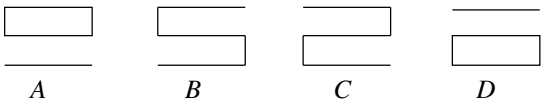
The system considered by Minsky and Papert at first appears to be a strong simplification of parallel decision processes, but it contains some of the most important elements differentiating between *sequential* and *parallel* processing. It is known that when some algorithms are parallelized, an irreducible sequential component sometimes limits the maximum achievable speedup. The mathematical relation between speedup and irreducible sequential portion of an algorithm is known as *Amdahl's law* [187]. In the model considered above the central question is, are there pattern recognition problems in which we are forced to analyze sequentially the output of the predicates associated with each receptive field or not? Minsky and Papert showed that problems of this kind do indeed exist which cannot be solved by a single perceptron acting as the last decision unit.

The limits imposed on the receptive fields of the predicates are based on realistic assumptions. The predicates are fixed in advance and the pattern recognition problem can be made arbitrarily large (by expanding the retina). According to the number of points and their connections to the predicates, Minsky and Papert differentiated between

- Diameter limited perceptrons: the receptive field of each predicate has a limited diameter.
- Perceptrons of limited order: each receptive field can only contain up to a certain maximum number of points.
- Stochastic perceptrons: each receptive field consists of a number of randomly chosen points

Some patterns are more difficult to identify than others and this structural classification of perceptrons is a first attempt at defining something like complexity classes for pattern recognition. Connectedness is an example of a property that cannot be recognized by constrained systems.

Proposition 6. *No diameter limited perceptron can decide whether a geometric figure is connected or not.*



Proof. We proceed by contradiction, assuming that a perceptron can decide whether a figure is connected or not. Consider the four patterns shown above; notice that only the middle two are connected.

Since the diameters of the receptive fields are limited, the patterns can be stretched horizontally in such a way that no single receptive field contains points from both the left and the right ends of the patterns. In this case

we have three different groups of predicates: the first group consists of those predicates whose receptive fields contain points from the left side of a pattern. Predicates of the second group are those whose receptive fields cover the right side of a pattern. All other predicates belong to the third group. In Figure 3.3 the receptive fields of the predicates are represented by circles.

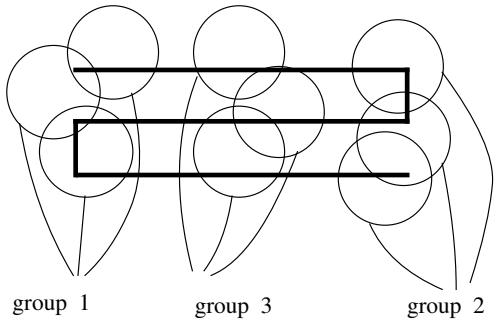


Fig. 3.3. Receptive fields of predicates

All predicates are connected to a threshold element through weighted edges which we denote by the letter w with an index. The threshold element decides whether a figure is connected or not by performing the computation

$$S = \sum_{P_i \in \text{group 1}} w_{1i} P_i + \sum_{P_i \in \text{group 2}} w_{2i} P_i + \sum_{P_i \in \text{group 3}} w_{3i} P_i - \theta \geq 0.$$

If S is positive the figure is recognized as connected, as is the case, for example, in Figure 3.3.

If the disconnected pattern A is analyzed, then we should have $S < 0$. Pattern A can be transformed into pattern B without affecting the output of the predicates of group 3, which do not recognize the difference since their receptive fields do not cover the sides of the figures. The predicates of group 2 adjust their outputs by $\Delta_2 S$ so that now

$$S + \Delta_2 S \geq 0 \Rightarrow \Delta_2 S \geq -S.$$

If pattern A is transformed into pattern C, the predicates of group 1 adjust their outputs so that the threshold element receives a net excitation, i.e.,

$$S + \Delta_1 S \geq 0 \Rightarrow \Delta_1 S \geq -S.$$

However, if pattern A is transformed into pattern D, the predicates of group 1 cannot distinguish this case from the one for figure C and the predicates of group 2 cannot distinguish this case from the one for figure B. Since the predicates of group 3 do not change their output we have

$$\Delta S = \Delta_2 S + \Delta_1 S \geq -2S,$$

and from this

$$S + \Delta S \geq -S > 0.$$

The value of the new sum can only be positive and the whole system classifies figure D as connected. Since this is a contradiction, such a system cannot exist. □

Proposition 6 states only that the connectedness of a figure is a global property which cannot be decided locally. If no predicate has access to the whole figure, then the only alternative is to process the outputs of the predicates sequentially.

There are some other difficult problems for perceptrons. They cannot decide, for example, whether a set of points contains an even or an odd number of elements when the receptive fields cover only a limited number of points.

3.2 Implementation of logical functions

In the previous chapter we discussed the synthesis of Boolean functions using McCulloch–Pitts networks. Weighted networks can achieve the same results with fewer threshold gates, but the issue now is which functions can be implemented using a single unit.

3.2.1 Geometric interpretation

In each of the previous sections a threshold element was associated with a whole set of predicates or a network of computing elements. From now on, we will deal with perceptrons as isolated threshold elements which compute their output without delay.

Definition 1. *A simple perceptron is a computing unit with threshold θ which, when receiving the n real inputs $x_1, x_2, \ldots, x_n$ through edges with the associated weights $w_1, w_2, \ldots, w_n$, outputs 1 if the inequality $\sum_{i=1}^{n} w_i x_i \geq \theta$ holds and otherwise 0.*

The origin of the inputs is not important, whether they come from other perceptrons or another class of computing units. The geometric interpretation of the processing performed by perceptrons is the same as with McCulloch–Pitts elements. A perceptron separates the input space into two half-spaces. For points belonging to one half-space the result of the computation is 0, for points belonging to the other it is 1.

Figure 3.4 shows this for the case of two variables x_1 and x_2. A perceptron with threshold 1, at which two edges with weights 0.9 and 2.0 impinge, tests the condition

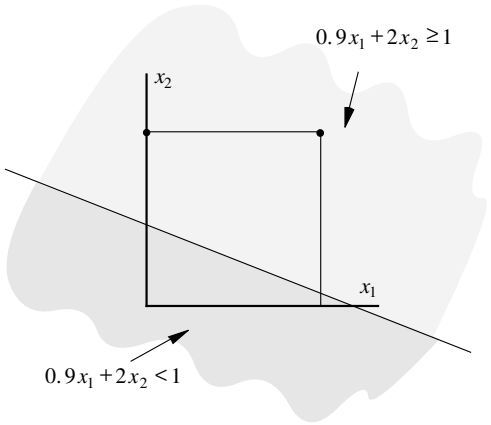


Fig. 3.4. Separation of input space with a perceptron

$$0.9x_1 + 2x_2 \geq 1.$$

It is possible to generate arbitrary separations of input space by adjusting the parameters of this example.

In many cases it is more convenient to deal with perceptrons of threshold zero only. This corresponds to linear separations which are forced to go through the origin of the input space. The two perceptrons in Figure 3.5 are equivalent. The threshold of the perceptron to the left has been converted into the weight $-\theta$ of an additional input channel connected to the constant 1. This extra weight connected to a constant is called the *bias* of the element.

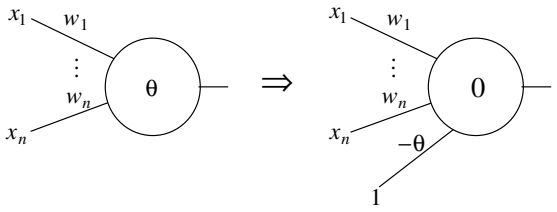


Fig. 3.5. A perceptron with a bias

Most learning algorithms can be stated more concisely by transforming thresholds into biases. The input vector $(x_1, x_2, \ldots, x_n)$ must be extended with an additional 1 and the resulting $(n+1)$-dimensional vector $(x_1, x_2, \ldots, x_n, 1)$ is called the *extended input vector*. The extended weight vector associated with this perceptron is $(w_1, \ldots, w_n, w_{n+1})$, whereby $w_{n+1} = -\theta$.

3.2.2 The XOR problem

We can now deal with the problem of determining which logical functions can be implemented with a single perceptron. A perceptron network is capable of computing any logical function, since perceptrons are even more powerful than unweighted McCulloch–Pitts elements. If we reduce the network to a single element, which functions are still computable?

Taking the functions of two variables as an example we can gain some insight into this problem. Table 3.1 shows all 16 possible Boolean functions of two variables f_0 to f_{15}. Each column f_i shows the value of the function for each combination of the two variables x_1 and x_2. The function f_0, for example, is the zero function whereas f_{14} is the OR-function.

Table 3.1. The 16 Boolean functions of two variables

x_1	x_2	f_0	f_1	f_2	f_3	f_4	f_5	f_6	f_7	f_8	f_9	f_{10}	f_{11}	f_{12}	f_{13}	f_{14}	f_{15}
0	0	0	1	0	1	0	1	0	1	0	1	0	1	0	1	0	1
0	1	0	0	1	1	0	0	1	1	0	0	1	1	0	0	1	1
1	0	0	0	0	0	1	1	1	1	0	0	0	0	1	1	1	1
1	1	0	0	0	0	0	0	0	0	1	1	1	1	1	1	1	1

Perceptron-computable functions are those for which the points whose function value is 0 can be separated from the points whose function value is 1 using a line. Figure 3.6 shows two possible separations to compute the OR and the AND functions.

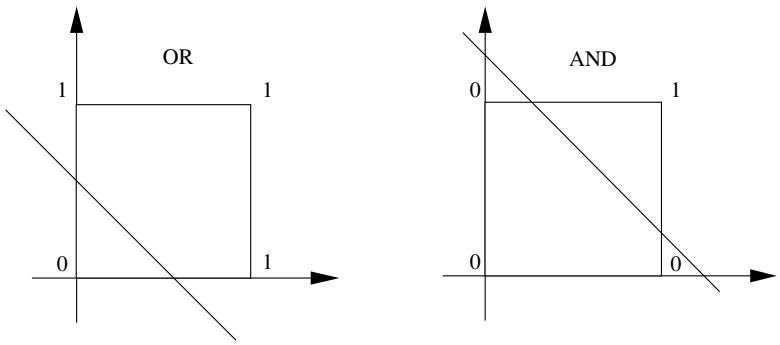


Fig. 3.6. Separations of input space corresponding to OR and AND

It is clear that two of the functions in the table cannot be computed in this way. They are the function XOR and identity (f_6 and f_9). It is intuitively evident that no line can produce the necessary separation of the input space. This can also be shown analytically.

Let w_1 and w_2 be the weights of a perceptron with two inputs, and θ its threshold. If the perceptron computes the XOR function the following four inequalities must be fulfilled:

$$\begin{array}{llll} x_1 = 0\ x_2 = 0 & w_1 x_1 + w_2 x_2 = 0 & \Rightarrow & 0 < \theta \\ x_1 = 1\ x_2 = 0 & w_1 x_1 + w_2 x_2 = w_1 & \Rightarrow & w_1 \geq \theta \\ x_1 = 0\ x_2 = 1 & w_1 x_1 + w_2 x_2 = w_2 & \Rightarrow & w_2 \geq \theta \\ x_1 = 1\ x_2 = 1 & w_1 x_1 + w_2 x_2 = w_1 + w_2 & \Rightarrow & w_1 + w_2 < \theta \end{array}$$

Since θ is positive, according to the first inequality, w_1 and w_2 are positive too, according to the second and third inequalities. Therefore the inequality $w_1 + w_2 < \theta$ cannot be true. This contradiction implies that no perceptron capable of computing the XOR function exists. An analogous proof holds for the function f_9.

3.3 Linearly separable functions

The example of the logical functions of two variables shows that the problem of perceptron computability must be discussed in more detail. In this section we provide the necessary tools to deal more effectively with functions of n arguments.

3.3.1 Linear separability

We can deduce from our experience with the XOR function that many other logical functions of several arguments must exist which cannot be computed with a threshold element. This fact has to do with the geometry of the n-dimensional hypercube whose vertices represent the combination of logic values of the arguments. Each logical function separates the vertices into two classes. If the points whose function value is 1 cannot be separated with a linear cut from the points whose function value is 0, the function is not perceptron-computable. The following two definitions give this problem a more general setting.

Definition 2. *Two sets of points A and B in an n-dimensional space are called 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 $\sum_{i=1}^{n} w_i x_i \geq w_{n+1}$ and every point $(x_1, x_2, \ldots, x_n) \in B$ satisfies $\sum_{i=1}^{n} w_i x_i < w_{n+1}$*

Since a perceptron can only compute linearly separable functions, an interesting question is how many linearly separable functions of n binary arguments there are. When $n = 2$, 14 out of the 16 possible Boolean functions are linearly separable. When $n = 3$, 104 out of 256 and when $n = 4$, 1882 out of 65536 possible functions are linearly separable. Although there has been extensive research on linearly separable functions in recent years, no formula for

expressing the number of linearly separable functions as a function of n has yet been found. However we will provide some upper bounds for this number in the following chapters.

3.3.2 Duality of input space and weight space

The computation performed by a perceptron can be visualized as a linear separation of input space. However, when trying to find the appropriate weights for a perceptron, the search process can be better visualized in weight space. When m real weights must be determined, the search space is the whole of $\mathbb{R}^m$.

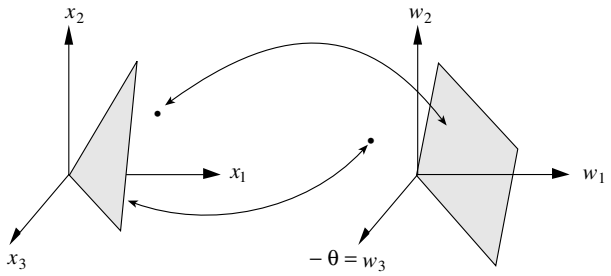


Fig. 3.7. Illustration of the duality of input and weight space

For a perceptron with n input lines, finding the appropriate linear separation amounts to finding $n+1$ free parameters (n weights and the bias). These $n+1$ parameters represent a point in $(n+1)$-dimensional weight space. Each time we pick one point in weight space we are choosing one combination of weights and a specific linear separation of input space. This means that every point in $(n+1)$-dimensional weight space can be associated with a hyperplane in $(n+1)$-dimensional extended input space. Figure 3.7 shows an example. Each combination of three weights, w_1, w_2, w_3, which represent a point in weight space, defines a separation of input space with the plane $w_1x_1 + w_2x_2 + w_3x_3 = 0$.

There is the same kind of relation in the inverse direction, from input to weight space. If we want the point x_1, x_2, x_3 to be located in the positive half-space defined by a plane, we need to determine the appropriate weights w_1, w_2 and w_3. The inequality

$$w_1x_1 + w_2x_2 + w_3x_3 \geq 0$$

must hold. However this inequality defines a linear separation of weight space, that is, the point (x_1, x_2, x_3) defines a cutting plane in weight space. Points in one space are mapped to planes in the other and vice versa. This complementary relation is called *duality*. Input and weight space are dual spaces and we

can visualize the computations done by perceptrons and learning algorithms in any one of them. We will switch from one visualization to the other as necessary or convenient.

3.3.3 The error function in weight space

Given two sets of patterns which must be separated by a perceptron, a learning algorithm should automatically find the weights and threshold necessary for the solution of the problem. The *perceptron learning algorithm* can accomplish this for threshold units. Although proposed by Rosenblatt it was already known in another context [10].

Assume that the set A of input vectors in n-dimensional space must be separated from the set B of input vectors in such a way that a perceptron computes the binary function f_w with $f_w(x) = 1$ for $x \in A$ and $f_w(x) = 0$ for $x \in B$. The binary function f_w depends on the set w of weights and threshold. The *error function* is the number of false classifications obtained using the weight vector w. It can be defined as:

$$E(w) = \sum_{x \in A} (1 - f_w(x)) + \sum_{x \in B} f_w(x).$$

This is a function defined over all of weight space and the aim of perceptron learning is to minimize it. Since $E(w)$ is positive or zero, we want to reach the global minimum where $E(w) = 0$. This will be done by starting with a random weight vector w, and then searching in weight space a better alternative, in an attempt to reduce the error function $E(w)$ at each step.

3.3.4 General decision curves

A perceptron makes a decision based on a linear separation of the input space. This reduces the kinds of problem solvable with a single perceptron. More general separations of input space can help to deal with other kinds of problem unsolvable with a single threshold unit. Assume that a single computing unit can produce the separation shown in Figure 3.8. Such a separation of the input space into two regions would allow the computation of the XOR function with a single unit. Functions used to discriminate between regions of input space are called *decision curves* [329]. Some of the decision curves which have been studied are polynomials and splines.

In statistical pattern recognition problems we assume that the patterns to be recognized are grouped in clusters in input space. Using a combination of decision curves we try to isolate one cluster from the others. One alternative is combining several perceptrons to isolate a convex region of space. Other alternatives which have been studied are, for example, so-called Sigma-Pi units which, for a given input $x_1, x_2, \ldots, x_n$, compute the sum of all or some partial products of the form $x_i x_j$ [384].

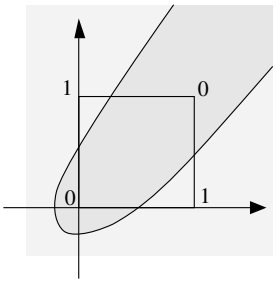


Fig. 3.8. Non-linear separation of input space

In the general case we want to distinguish between regions of space. A neural network must learn to identify these regions and to associate them with the correct response. The main problem is determining whether the free parameters of these decision regions can be found using a learning algorithm. In the next chapter we show that it is always possible to find these free parameters for linear decision curves, if the patterns to be classified are indeed linearly separable. Finding learning algorithms for other kinds of decision curves is an important research topic not dealt with here [45, 4].

3.4 Applications and biological analogy

The appeal of the perceptron model is grounded on its simplicity and the wide range of applications that it has found. As we show in this section, weighted threshold elements can play an important role in image processing and computer vision.

3.4.1 Edge detection with perceptrons

A good example of the pattern recognition capabilities of perceptrons is edge detection (Figure 3.9). Assume that a method of extracting the edges of a figure darker than the background (or the converse) is needed. Each pixel in the figure is compared to its immediate neighbors and in the case where the pixel is black and one of its neighbors white, it will be classified as part of an edge. This can be programmed sequentially in a computer, but since the decision about each point uses only local information, it is straightforward to implement the strategy in parallel.

Assume that the figures to be processed are projected on a screen in which each pixel is connected to a perceptron, which also receives inputs from its immediate neighbors. Figure 3.10 shows the shape of the receptive field (a so-called Moore neighborhood) and the weights of the connections to the perceptron. The central point is weighted with 8 and the rest with -1. In the field of image processing this is called a *convolution operator*, because it is

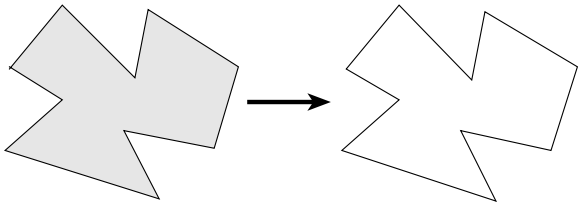


Fig. 3.9. Example of edge detection

used by centering it at each pixel of the image to produce a certain output value for each pixel. The operator shown has a maximum at the center of the receptive field and local minima at the periphery.

-1	-1	-1
-1	8	-1
-1	-1	-1

Fig. 3.10. Edge detection operator

Figure 3.11 shows the kind of interconnection we have in mind. A perceptron is needed for each pixel. The interconnection pattern repeats for each pixel in the projection lattice, taking care to treat the borders of the screen differently. The weights are those given by the edge detection operator.

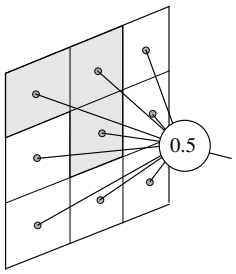


Fig. 3.11. Connection of a perceptron to the projection grid

For each pattern projected onto the screen, the weighted input is compared to the threshold 0.5. When all points in the neighborhood are black or all white, the total excitation is 0. In the situation shown below the total excitation is 5 and the point in the middle belongs to an edge.

There are many other operators for different uses, such as detecting horizontal or vertical lines or blurring or making a picture sharper. The size of

the neighborhood can be adjusted to the specific application. For example, the operator

$$\begin{pmatrix} -1 & 0 & 1 \\ -1 & 0 & 1 \\ -1 & 0 & 1 \end{pmatrix}$$

can be used to detect the vertical edges between a white surface to the left and a dark one to the right.

3.4.2 The structure of the retina

The visual pathway is the part of the human brain which is best understood. The retina can be conceived as a continuation of this organ, since it consists of neural cells organized in layers and capable of providing *in situ* some of the information processing necessary for vision. In frogs and other small vertebrates some neurons have been found directly in the retina which actually fire in the presence of a small blob in the visual field. These are bug detectors which tell these animals when a small insect has been sighted.

Researchers have found that the cells in the retina are interconnected in such a way that each nerve going from the eyes to the brain encodes a summary of the information detected by several photoreceptors in the retina. As in the case of the convolution operators discussed previously, each nerve transmits a signal which depends on the relative luminosity of a point in relation to its immediate neighborhood.

Figure 3.12 shows the interconnection pattern found in the retina [205, 111]. The cones and rods are the photoreceptors which react to photons by depolarizing. Horizontal cells compute the average luminosity in a region by connecting to the cones or rods in this region. Bipolar and ganglion cells fire only when the difference in the luminosity of a point is significantly higher than the average light intensity.

Although not all details of the retinal circuits have been reverse-engineered, there is a recurrent feature: each receptor cell in the retina is part of a roughly circular receptive field. The receptive fields of neighboring cells overlap. Their function is similar to the edge processing operators, because the neighborhood inhibits a neuron whereas a photoreceptor excites it, or conversely. This kind of weighting of the input has a strong resemblance to the so-called Mexican hat function.

David Marr tried to summarize what we know about the visual pathway in humans and proposed his idea of a process in three stages, in which the brain first decomposes an image into features (edges, blobs, etc.), which are then used to build an interpretation of surfaces, depth relations and groupings of tokens (the "$2\frac{1}{2}$ sketch") and which in turn leads to a full interpretation of the objects present in the visual field (the primal sketch) [293]. He tried to explain the structure of the retina from the point of view of the computational machinery needed for vision. He proposed that at a certain stage of

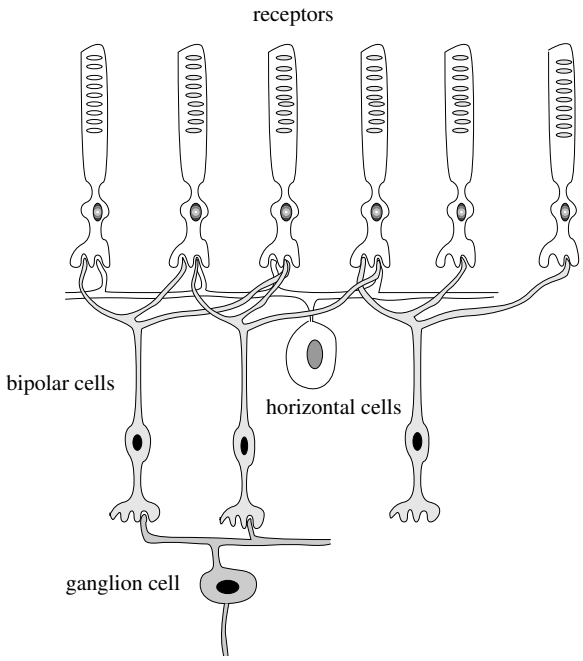


Fig. 3.12. The interconnection pattern in the retina

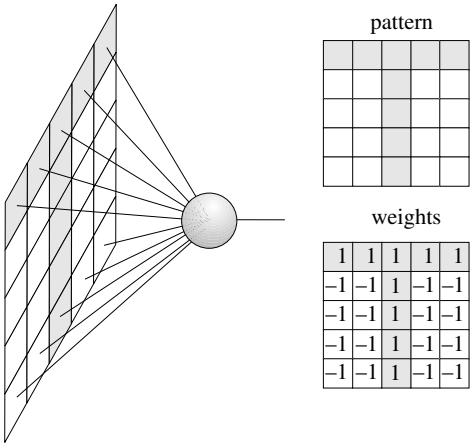


Fig. 3.13. Feature detector for the pattern T

the computation the retina blurs an image and then extracts from it contrast information. Blurring an image can be done by averaging at each pixel the values of this pixel and its neighbors. A Gaussian distribution of weights can be used for this purpose. Information about changes in darkness levels can be

extracted using the sum of the second derivatives of the illumination function, the so-called Laplace operator $\nabla^2 = \partial^2/\partial x^2 + \partial^2/\partial y^2$. The composition of the Laplace operator and a Gaussian blurring corresponds to the Mexican hat function. Processing of the image is done by computing the convolution of the discrete version of the operator with the image. The 3×3 discrete version of this operator is the edge detection operator which we used before. Different levels of blurring, and thus feature extraction at several different resolution levels, can be controlled by adjusting the size of the receptive fields of the computing units. It seems that the human visual pathway also exploits feature detection at several resolution levels, which has led in turn to the idea of using several resolution layers for the computational analysis of images.

3.4.3 Pyramidal networks and the neocognitron

Single perceptrons can be thought of as *feature detectors*. Take the case of Figure 3.13 in which a perceptron is defined with weights adequate for recognizing the letter 'T' in which t pixels are black. If another 'T' is presented, in which one black pixel is missing, the excitation of the perceptron is $t - 1$. The same happens if one white pixel is transformed into a black one due to noise, since the weights of the connections going from points that should be white are -1. If the threshold of the perceptron is set to $t - 1$, then this perceptron will be capable of correctly classifying patterns with one noisy pixel. By adjusting the threshold of the unit, 2, 3 or more noisy pixels can be tolerated. Perceptrons thus compute the similarity of a pattern to the ideal pattern they have been designed to identify, and the threshold is the minimal similarity that we require from the pattern. Note that since the weights of the perceptron are correlated with the pattern it recognizes, a simple way to visualize the connections of a perceptron is to draw the pattern it identifies in its receptive field. This technique will be used below.

The problem with this pattern recognition scheme is that it only works if the patterns have been normalized in some way, that is, if they have been centered in the window to which the perceptron connects and their size does not differ appreciably from the ideal pattern. Also, any kind of translational shift will lead to ideal patterns no longer being recognized. The same happens in the case of rotations.

An alternative way of handling this problem is to try to detect patterns not in a single step, but in several stages. If we are trying, for example, to recognize handwritten digits, then we could attempt to find some small distinctive features such as lines in certain orientations and then combine our knowledge about the presence or absence of these features in a final logical decision. We should try to recognize these small features, regardless of their position on the projection screen.

The *cognitron* and *neocognitron* were designed by Fukushima and his colleagues as an attempt to deal with this problem and in some way to try to mimic the structure of the human vision pathway [144, 145]. The main idea of

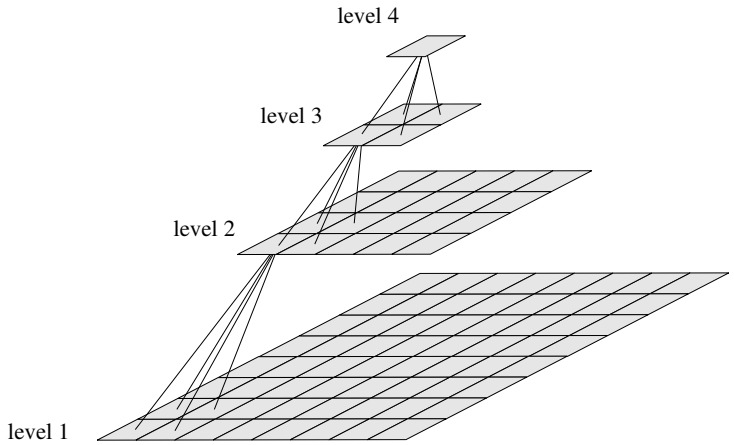


Fig. 3.14. Pyramidal architecture for image processing

the neocognitron is to transform the contents of the screen into other screens in which some features have been enhanced, and then again into other screens, and so on, until a final decision is made. The resolution of the screen can be changed from transformation to transformation or more screens can be introduced, but the objective is to reduce the representation to make a final classification in the last stage based on just a few points of input.

The general structure of the neural system proposed by Fukushima is a kind of variant of what is known in the image processing community as a pyramidal architecture, in which the resolution of the image is reduced by a certain factor from plane to plane [56]. Figure 3.14 shows an example of a quad-pyramid, that is, a system in which the resolution is reduced by a factor of four from plane to plane. Each pixel in one of the upper planes connects to four pixels in the plane immediately below and deals with them as elements of its receptive field. The computation to determine the value of the upper pixel can be arbitrary, but in our case we are interested in threshold computations. Note that in this case receptive fields do not overlap. Such architectures have been studied intensively to determine their capabilities as data structures for parallel algorithms [80, 57].

The neocognitron has a more complex architecture [280]. The image is transformed from the original plane to other planes to look for specific features.

Figure 3.15 shows the general strategy adopted in the neocognitron. The projection screen is transformed, deciding for each pixel if it should be kept white or black. This can be done by identifying the patterns shown for each of the three transformations by looking at each pixel and its eight neighbors. In the case of the first transformed screen only horizontal lines are kept; in the second screen only vertical lines and in the third screen only diagonal lines. The convolution operators needed have the same distribution of positive

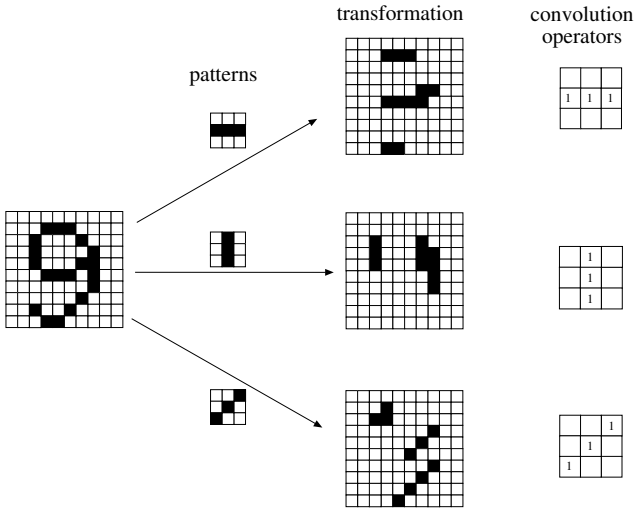


Fig. 3.15. Feature extraction in the neocognitron

weights, as shown for each screen. The rest of the weights is 0. Note that these special weights work better if the pattern has been previously 'skeletonized'.

Strictly speaking Fukushima's neocognitron uses linear computing units and not perceptrons. The units compute their total weighted input and this is interpreted as a kind of correlation index with the patterns that each unit can identify. This means that black and white patterns are transformed into patterns with shadings of gray, according to the output of each mapping unit. Figure 3.16 shows the general structure of the neocognitron network. The input layer of the network is called UC^0. This input layer is processed and converted into twelve different images numbered US_0^1 to US_{11}^1 with the same resolution. The superindex in front of a name is the layer number and the subindex the number of the plane in this layer. The operators used to transform from UC^0 to each of the US_i^1 planes have a receptive field of 3×3 pixels and one operator is associated with each pixel in the US_i^1 planes. In each plane only one kind of feature is recognized. The first plane US_1^1, for example, could contain all the vertical edges found in UC^0, the second plane US_2^1 only diagonal edges, and so forth. The next level of processing is represented by the UC_j^1 planes. Each pixel in one of these planes connects to a receptive field in one or two of the underlying US_i^1 planes. The weights are purely excitatory and the effect of this layer is to overlap the activations of the selected US_i^1 images, blurring them at the same time, that is, making the patterns wider. This is achieved by transforming each pixel's value in the weighted average of its own and its neighbor's values.

In the next level of processing each pixel in a US_i^2 plane connects to a receptive field at the same position in all of the UC_j^1 images. At this level the resolution of the US_i^2 planes can be reduced, as in standard pyramidal

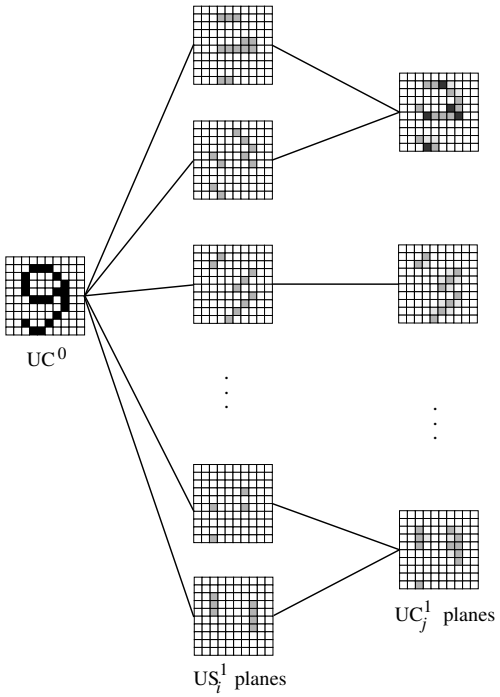


Fig. 3.16. The architecture of the neocognitron

architectures. Fig 3.16 shows the sizes of the planes used by Fukushima for handwritten digit recognition. Several layers of alternating US and UC planes are arranged in this way until at the plane UC^4 a classification of the handwritten digit in one of the classes $0, \ldots, 9$ is made. Finding the appropriate weights for the classification task is something we discuss in the next chapter. Fukushima has proposed several improvements of the original model [147] over the years.

The main advantage of the neocognitron as a pattern recognition device should be its tolerance to shifts and distortions. Since the UC layers blur the image and the US layers look for specific features, a certain amount of displacement or rotation of lines should be tolerated. This can happen, but the system is highly sensitive to the training method used and does not outperform other simpler neural networks [280]. Other authors have examined variations of the neocognitron which are more similar to pyramidal networks [463]. The neocognitron is just an example of a class of network which relies extensively on convolution operators and pattern recognition in small receptive fields. For an extensive discussion of the neocognitron consult [133].

3.4.4 The silicon retina

Carver Mead's group at Caltech has been active for several years in the field of *neuromorphic engineering*, that is, the production of chips capable of emulating the sensory response of some human organs. Their *silicon retina*, in particular, is able to simulate some of the features of the human retina.

Mead and his collaborators modeled the first three layers of the retina: the photoreceptors, the horizontal, and the bipolar cells [303, 283]. The horizontal cells are simulated in the silicon retina as a grid of resistors. Each photoreceptor (the dark points in Figure 3.17) is connected to each of its six neighbors and produces a potential difference proportional to the logarithm of the luminosity measured. The grid of resistors reaches electric equilibrium when an average potential difference has settled in. The individual neurons of the silicon retina fire only when the difference between the average and their own potential reaches a certain threshold.

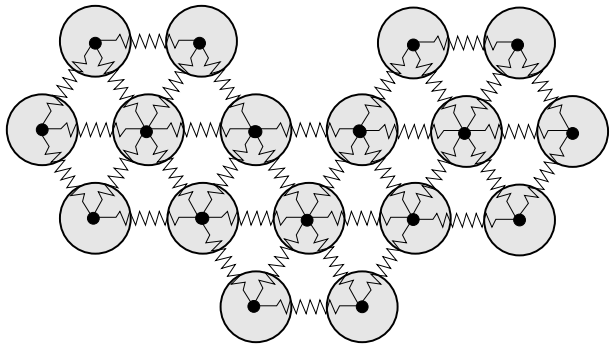


Fig. 3.17. Diagram of a portion of the silicon retina

The average potential S of n potentials S_i can be computed by letting each potential S_i be proportional to the logarithm of the measured light intensity H_i, that is,

$$S = \frac{1}{n}\sum_{i=1}^{n} S_i = \frac{1}{n}\sum_{i=1}^{n} \log H_i.$$

This expression can be transformed into

$$S = \frac{1}{n}\log(H_1 H_2 \cdots H_n) = \log(H_1 H_2 \cdots H_n)^{1/n}.$$

The equation tells us that the average potential is the logarithm of the geometric mean of the light intensities. A unit only fires when the measured intensity S_i minus the average intensity lies above a certain threshold γ, that is,

$$\log(H_i) - \log(H_1 H_2 \cdots H_n)^{1/n} \geq \gamma,$$

and this is valid only when

$$\log \frac{H_i}{(H_1 H_2 \cdots H_n)^{1/n}} \geq \gamma.$$

The units in the silicon retina fire when the relative luminosity of a point with respect to the background is significantly higher, such as in a human retina. We know from optical measurements that when outside on a sunny day, the black letters in a book reflect more photons on our eyes than white paper does in a room. Our eyes adjust automatically to compensate for the luminosity of the background so that we can recognize patterns and read books inside and outside.

3.5 Historical and bibliographical remarks

The perceptron was the first neural network to be produced commercially, although the first prototypes were used mainly in research laboratories. Frank Rosenblatt used the perceptron to solve some image recognition problems [185]. Some researchers consider the perceptron as the first serious abstract model of nervous cells [60].

It was not a coincidence that Rosenblatt conceived his model at the end of the 1950s. It was precisely in this period that researchers like Hubel and Wiesel were able to "decode" the structure of the retina and examine the structure of the receptive fields of neurons. At the beginning of the 1970s, researchers had a fair global picture of the architecture of the human eye [205]. David Marr's influential book *Vision* offered the first integrated picture of the visual system in a way that fused biology and engineering, by looking at the way the visual pathway actually computed partial results to be integrated in the raw visual sketch.

The book *Perceptrons* by Minsky and Papert was very influential among the AI community and is said to have affected the strategic funding decisions of research agencies. This book is one of the best ever written on its subject and set higher standards for neural network research, although it has been criticized for stressing the incomputability, not the computability results. The Dreyfus brothers [114] consider the reaction to *Perceptrons* as one of the milestones in the permanent conflict between the *symbolic* and the *connectionist* schools of thought in AI. According to them, reaction to the book opened the way for a long period of dominance of the symbolic approach. Minsky, for his part, now propagates an alternative massively parallel paradigm of a society of agents of consciousness which he calls a *society of mind* [313].

Convolution operators for image processing have been used for many years and are standard methods in the fields of image processing and computer vision. Chips integrating this kind of processing, like the silicon retina, have been produced in several variants and will be used in future robots. Some researchers dream of using similar chips to restore limited vision to blind

persons with intact visual nerves, although this is, of course, still an extremely ambitious objective [123].

Exercises

1. Write a computer program that counts the number of linearly separable Boolean functions of 2, 3, and 4 arguments. *Hint*: Generate the perceptron weights randomly.
2. Consider a simple perceptron with n bipolar inputs and threshold $\theta = 0$. Restrict each of the weights to have the value -1 or 1. Give the smallest upper bound you can find for the number of functions from $\{-1, 1\}^n$ to $\{-1, 1\}$ which are computable by this perceptron [219]. Prove that the upper bound is sharp, i.e., that all functions are different.
3. Show that two finite linearly separable sets A and B can be separated by a perceptron with rational weights. Note that in Def. 2 the weights are real numbers.
4. Prove that the parity function of $n > 2$ binary inputs $x_1, x_2, \ldots, x_n$ cannot be computed by a perceptron.
5. Implement edge detection with a computer program capable of processing a computer image.
6. Write a computer program capable of simulating the silicon retina. Show the output produced by different pictures on the computer's screen.