UNIT 5 SINGLE LAYER PERCEPTRONS

Structure **Page No.**

5.1 INTRODUCTION

We have discussed model of McCulloch-Pitts units. Now the question of how to find the parameters adequate for a given task was left open. If two sets of points have to be separated linearly with a perceptron, adequate weights for the computing unit must be found. The perceptron learning algorithm deals with this problem. These are discussed in Section 5.2.

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. In this connection, we shall discuss XOR problem in Section 5.3.

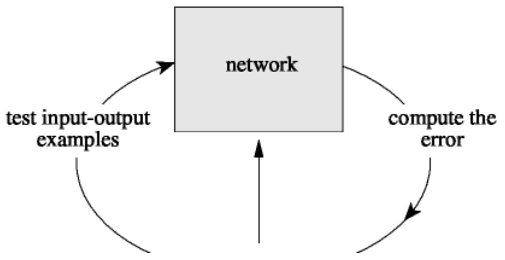
Objectives

After studying the unit you should be able to:

- define single layer perceptron;
- write the perecptron algorithm;
- apply the perceptron algorithm in different situation;
- describe the XOR problem.

5.2 SINGLE LAYER PERCEPTRON ALGORITHM AND ITS CONVERGENCE

Let us first start by defining the perceptron learning algorithms for Neural Networks. A learning algorithm is an adaptive method by which a network of computing units self-organizes to implement the desired behavior. This is done in some learning algorithms by presenting some examples of the desired input-output mapping to the network. A correction step is executed iteratively until the network learns to produce the desired response. The learning algorithm is a closed loop of presentation of examples and of corrections to the network parameters, as shown in Fig. 1. In some simple cases the weights for the computing units can be found through a sequential test of stochastically generated numerical combinations. However, such algorithms which look blindly for a solution do not qualify as “learning”. A learning algorithm must adapt the network parameters according to previous experience until a solution is found, if it exists.



fix network parameters

Fig. 1: Learning Process in a parametric system

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

Unsupervised learning is used when, for a given input, the exact numerical output a network should produce is unknown. Assume, for example, that some points in two-dimensional space are to be classified into three clusters. For this task we can use a classifier network with three output lines, one for each class (Fig. 2). Each of the three computing units at the output must specialize by firing only for inputs corresponding to elements of each cluster. If one unit fires, the others must keep silent. In this case we do not know a priori which unit is going to specialize on which cluster. Generally we do not even know how many well-defined clusters are present. Since no “teacher” is available, the network must organize itself in order to be able to associate clusters with units.

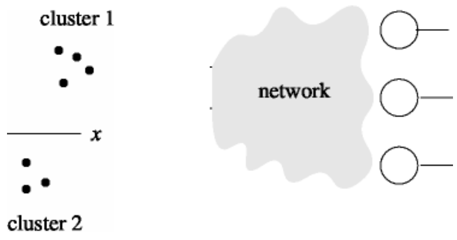


Fig. 2: Three clusters in a classifier network

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

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

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

$$w.x \geq \theta,$$

if w and x are the weight and input vectors, and

$$w.x \geq \theta$$

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

Example 1: We are looking for the weights and threshold needed to implement the AND function with a perceptron, the input vectors and their associated outputs are given in Table 1.

Table 1

Input	Output
0, 0	0
0, 1	0
1, 0	0
1, 1	1

If a perceptron with threshold zero is used, the input vectors must be extended and the desired mappings are given in Table 2.

Table 2

Input	Output
0, 0, 1	0
0, 1, 1	0
1, 0, 1	0
1, 1, 1	1

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

The proof of convergence of the perceptron learning algorithm assumes that each perceptron performs the test $w \cdot x > 0$. So far we have been working with perceptrons which perform the test $w.x \geq 0$. We must just show that both classes of computing units are equivalent when the training set is finite, as is always the case in learning problems. Not let us define the following definitions.

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

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

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

Definition 2: The open (closed) positive half-space associated with the n-dimensional weight vector w is the set of all points $x \in R^n$ for which $w.x > 0$ $(w.x \geq 0)$. The open (closed) negative half-space associated with w is the set of all points $x \in R^n$ for which $w.x < 0$ $(w.x \geq 0)$.

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

Example 2: Consider three weights w_1, w_2, and $w_3 = -\theta$ are needed to implement the desired separation with a generic perceptron. The first step is to extend the input vectors with a third coordinate $x_3 = 1$ and to write down the four inequalities that must be fulfilled:

$$(0, 0, 1).(w_1, w_2, w_3) < 0 \qquad (1)$$

$$(1, 0,1).(w_1, w_2, w_3) < 0 \qquad (2)$$

$$(0, 1, 1).(w_1, w_2, w_3) < 0 \qquad (3)$$

$$(1, 1, 1).(w_1, w_2, w_3) > 0 \qquad (4)$$

This can be written as

$$Aw > 0$$

where A is the 4×3 matrix and w the weight vector (written as a column vector) given as $A \begin{bmatrix} 0 & 0 & -1 \\ -1 & 0 & -1 \\ 0 & -1 & -1 \\ 1 & 1 & 1 \end{bmatrix}$ and $w = \begin{bmatrix} w_1 \\ w_2 \\ w_3 \end{bmatrix}$. The learning problem is to find the appropriate weight vector w.

Now, let us introduce the perceptron learning algorithm. For this, consider two training sets, P and N, in n-dimensional extended input space, and a weight vector w. These are selected such that all vectors in P belong to the open positive half-space and all vectors in N to the open negative half-space of the linear separation.

Algorithm Perceptron

Step 1: Initialize the weight vector w_0 randomly and set $x = 0$.

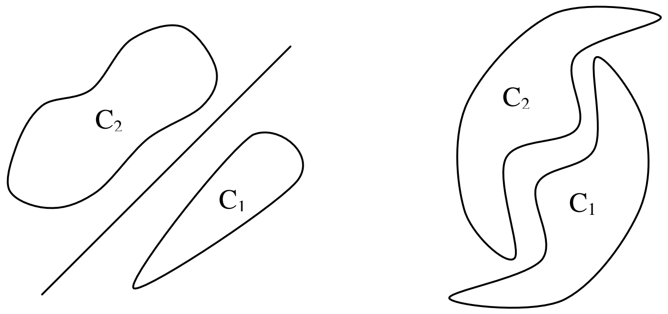
Step 2: Select a vector $x \in P \cup N$ randomly.

Step 3: If $x \in P$ and $w_t.x > 0$, or if $x \in N$ and $w_t.x < 0$, then go to step 2. Otherwise if $x \in N$ and $w_t.x \geq 0$, then set $w_{t+1} = w_t + \alpha x$ and $t = t + 1$ and go to step 2 otherwise if $x \in P$ and $w_t.x \leq 0$, then set $w_{t+1} = w_t - \alpha x$ and $t = t + 1$, and go to step 2.

Here α is the learning rate parameter. w(uc) ith the help of this algorithm, the weight vectors can be corrected in case of incorrect classification of the selected vectors P and N. Here it is clear that if P and N are separable, the vector w is updated by the finite number of iterations.

The iterations are stopped by getting the all vectors correctly. It is clear that the weight w_{t+1} is the new weight, which has been adjusted by the old weight w_t with the input vector x. If the value of x is fixed, then the learning algorithm is termed as fixed increment algorithm. This perceptron is one of the important achievement due to Rosemblatt. Further Minsky and Papert contributed the linearly separable solution space and an illustration as XOR problem.

Perceptrons can handle only linearly separable tasks. In two dimensional space linearly separability them by a straight line.



(a) Linearly separable patterns **(b) Non-linearly separable patterns**
Fig. 3

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

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

Example 3: (The dynamics of perceptron learning using the error surface for the OR function). The input $(1, 1)$ must produce the output 1 (for simplicity we fix the threshold of the perceptron to 1). The two weights w_1 and w_2 must fulfill the inequality $w_1 + w_2 \geq 1$. Any other combination produces an error. The contribution to the total error is shown in Fig. 4 as a step in the error surface. If the initial weight vector lies in the triangular region with error 1, it must be brought up to the verge of the region with error 0. This can be done by adding the vector $(1, 1)$ to w. However, if the input vector is, for example, $(0.1, 0.1)$, it should be added a few times before the weight combination (w_1, w_2) falls to the region of error 0. In this case we would like to make the correction in a single iteration.

These considerations provide an improvement for the perceptron learning algorithm: if at iteration t the input vector $x \in P$ is classified erroneously, then we have $w_t :\geq 0$.

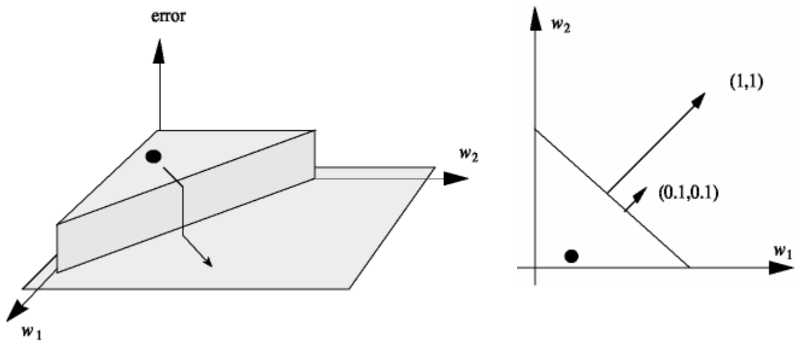


Fig. 4: A step on the error surface

The number ε guarantees that the new weight vector just barely skips over the border of the region with a higher error. The constant ε should be made small enough to avoid skipping to another region whose error is higher than the current one. When $x \in N$ the correction step is made similarly, but using the factor $\delta - \varepsilon$ instead of $\delta + \varepsilon$.

The accelerated algorithm is an example of corrective learning: We do not just "reinforce" the weight vector, but completely correct the error that has been made. A variant of this rule is correction of the weight vector using a proportionality constant γ as the learning factor, in such a way that at each update the vector $\gamma(\delta + \varepsilon)x$ is added to w. The learning constant falls to zero as learning progresses.

The perceptron learning algorithm selects a search direction in weight space according to the incorrect classification of the last tested vector and does not make use of global information about the shape of the error function. It is a greedy, local algorithm. This can lead to an exponential number of updates of the weight vector. Fig. 5 shows the different error regions in a worst case scenario. The region with error 0 is bounded by two lines which meet at a small angle. Starting the learning algorithm at point $w0$, the weight updates will lead to a search path similar to the one shown in the figure. In each new iteration a new weight vector is computed, in such a way that one of two vectors is classified correctly. However, each of these corrections leads to the other vector being incorrectly classified. The iteration jumps from one region with error 1 to the other one. The algorithm converges only after a certain number of iterations, which can be made arbitrarily large by adjusting the angle at which the lines meet.

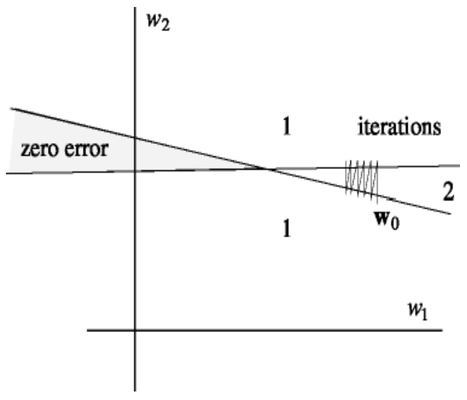


Fig. 5: Worst case for perceptron learning (weight space)

Fig. 5 corresponds to the case in which two almost antiparallel vectors are to be classified in the same half-space (Fig. 6). An algorithm which rotates the separation line in one of the two directions (like perceptron learning) will require more and more time when the angle between the two vectors approaches 180 degrees. This example is a good illustration of the advantages of visualizing learning algorithms in both the input space and its dual, weight space. Fig. 6 shows the concrete problem and Fig. 5 illustrates why?

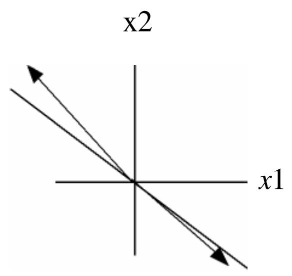


Fig. 6: Worst case for perceptron learning (input space)

Now let us discuss the single layer perceptron in the following.

A neural network with a single layer is also capable of processing for some important applications, such as integrated circuit implementations for assembly line control. The most common capability of the different models of neural networks is pattern recognition. But one network, called the Brain-State-in-a-Box, which is a single-layer neural network, can do pattern completion. Adaline is a network with A and B fields of neurons, but aggregation or processing of input signals is done only by the field B neurons.

The Hopfield network is a single-layer neural network. The Hopfield network makes an association between different pattern (heteroassociation) or associates a pattern with itself (autoassociation). We may characterize this as being able to recognize a given pattern. The idea of viewing it as a case of pattern recognition becomes more relevant if a pattern is presented with some noise, meaning that there is some slight deformation in the pattern, and if the network is able to relate it to the correct pattern.

The Perceptron technically has two layers, but has only one group of weights. We therefore still refer to it as a single-layer network. The second layer consists solely of the output neuron, and the first layer consists of the neurons that receive input(s). Also, the neurons in the same layer, the input layer in this case, are not interconnected, that is, no connections are made between two neurons in that same layer. On the other hand, in the Hopfield network, there is no separate output layer, and hence, it is strictly a single-layer network. In addition, the neurons are all fully connected with one another.

The ADALINE network is abbreviated for **Adaptive Linear Neural Element Network**. This network was framed by Bernard Widrow of Stanford University, makes use of supervised learning. A simple ADALINE network is shown in Fig. 7. Here, there is only one output neuron and the output values are bipolar $(-1 \text{ or } +1)$, and the inputs x_i may be binary, bipolar or real valued. The bias weight is w_0 with an input link of $x_0 = +1$. If $\sum_{i=0}^{n} x_i w_i \geq 0$, then the output is 1 otherwise it is -1.

The supervised learning algorithm adopted by the network is similar to the perceptron learning algorithm, devised by Widrow-Hoff (1960). The learning algorithm is also known as the Least Mean Square (LMS) or Delta rule. The rule is given by

$$w_{n+1} = w_n + \alpha(t - y)x_i$$

where α is the learning coefficient, t is the target, y is the computed output, and x_i is the input.

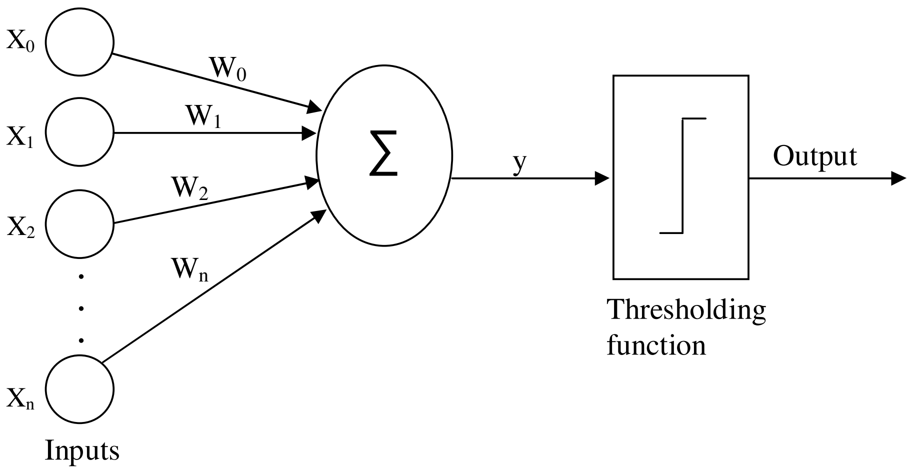


Fig. 7: A simple ADALINE network

ADALINE network is used virtually in all high speed modems and telephone switching systems to cancel the echo in long distance communication circuits.

Try the following exercises.

E1) The input to a single-input neuron is 2.0, its weight is 2.3 and its bias is – 3.

i) What is the net input to the transfer function?

ii) What is the neuron output for the following transfer function
   a) Hard limit,
   b) Linear,
   c) Log-sigmoid.

E2) Given a two-input neuron with the following parameters: b = 1.2, W = [3 2] and $x = [-5 \quad 6]^T$, calculate the neuron output for the following transfer functions:

i) A symmetrical hard limit transfer function.

ii) A linear transfer function.

iii) A hyperbolic tangent sigmoid (tansig) transfer function.

E3) A single-layer neural network is to have six inputs and two outputs. The outputs are to be limited to and continuous over the range 0 to 1. What can you tell about the network architecture? Specifically:

i) How many neurons are required?

ii) What are the dimensions of the weight matrix?

iii) What kind of transfer functions could be used?

iv) Is a bias required?

In the following section, we shall discuss the XOR problem.

5.3 XOR PROBLEM

The perceptron cannot find weights for classification type of problems that are not linearly separable. An example is the XOR (exclusive OR) problem. The ability of a Perceptron in evaluating functions was brought into question when Minsky and Papert proved that a simple function like XOR (the logical function exclusive or) could not be correctly evaluated by a Perceptron. XOR is a logical operation which can be described by its truth table. The truth table is presented in Table 3.

Table 3

Inputs (x_1)	**Inputs (x_2)**	**Output** **f (x_1, x_2) = XOR (x_1, x_2)**
0	0	0
0	1	1
1	0	1
1	1	0

Here, it is clear that the behaviour of the XOR, if both inputs are the same value, the output is 0, otherwise the output is 1.

The problem for the artificial neural network is to classify the inputs as odd parity or even parity. Here odd parity is the odd number of 1 bits in the inputs and even parity is the even number of 1 bit in the inputs. This is not possible, since as is evident by the perceptron, we are unable to find a little separating even parity input patterns from the odd parity input patterns.

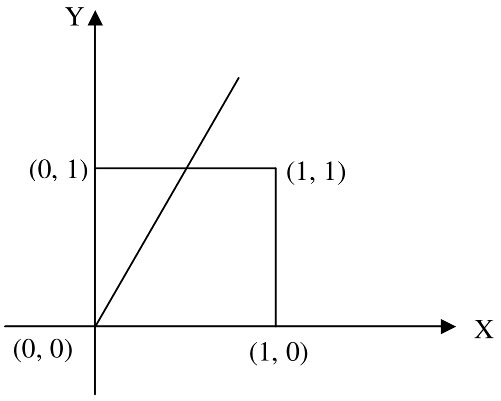


Fig. 8: Patterns of the XOR problem (non-linear separable)

Now, the question arises, that why is these perceptron unable to find weights for non-linearly separable classification problems? This can be explained by means of a simple instance. For this, consider a perceptron network with two inputs x_1 and x_2 and bias $x_0 = 1$ (refer Fig. 9).

$$\text{net output} = w_0 = \sum_{i=1}^{2} w_i x_i$$

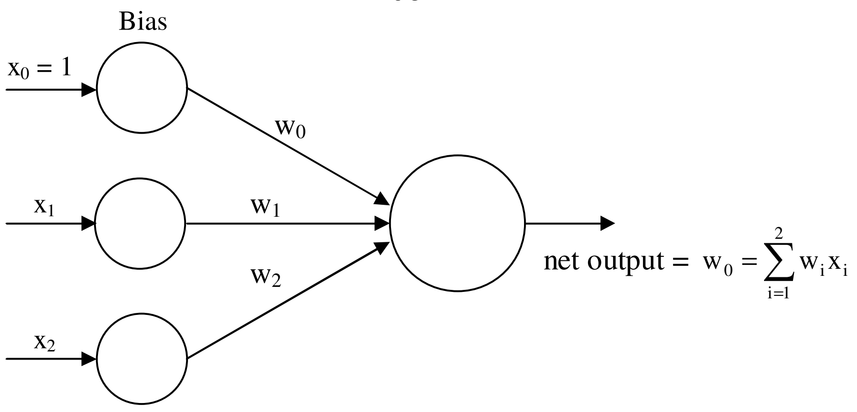


Fig. 9: A perceptron model

which represents the equation of a straight line, which acts as a decision boundary separating the points into classes C_1 and C_2, above and below the line respectively. This is shown in Fig. 10.

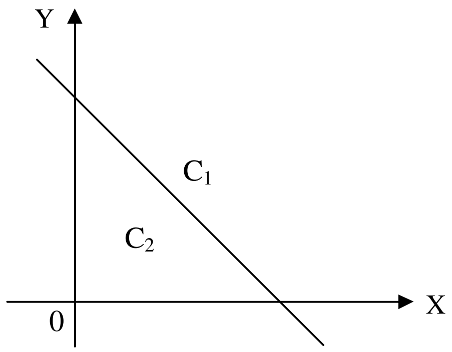


Fig. 10: A 2-classsification problem

This gives the answer that is the perceptron aims to do for a problem when it is able to obtain their weights.

Minsky and Papert showed that it is impossible to come up with the proper set of weights for the neurons in the single layer of a simple Perceptron to evaluate the XOR function. The reason for this is that such a Perceptron, one with a single layer of neurons, requires the function to be evaluated, to be linearly separable by means of the function values.

Since there are two arguments for the XOR function, there would be two neurons in the input layer, and since the function's value is one number, there would be one output neuron. Therefore, we need two weights w_1 and w_2, and a threshold value θ. Let us now look at the conditions to be satisfied by the w's and the θ so that the outputs corresponding to given inputs would be as for the XOR function.

First the output should be 0 if inputs are 0 and 0. The activation works out as 0. To get an output of 0, you need $0 < \theta$. This is your first condition. Table 4 shows this and two other conditions we need, and why.

Table 4: Conditions on Weights

Input	Activation	Output	Needed Condition
0, 0	0	0	$0 < \theta$
1, 0	w_1	1	$w_1 > \theta$
0, 1	w_2	1	$w_2 > \theta$
1, 1	$w_1 + w_2$	0	$w_1 + w_2 < \theta$

From the first three conditions, we can deduce that the sum of the two weights has to be greater than θ, which has to be positive itself. Line 4 is inconsistent with lines 1, 2, and 3, since line 4 requires the sum of the two weights to be less than θ. This affirms the contention that it is not possible to compute the XOR function with a simple perceptron.

Geometrically, the reason for this failure is that the inputs (0, 1) and (1, 0) with which we want output 1, are situated diagonally opposite each other, when plotted as points in the plane.

We can't separate the T's and the F's with a straight line. This means that we cannot draw a line in the plane in such a way that neither (1, 1) nor (0, 0) is on the same side of the line as (0, 1) and (1, 0).

Implementation of Logical Functions

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 3: 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 θ.

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.

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 shown in Fig. 11 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.

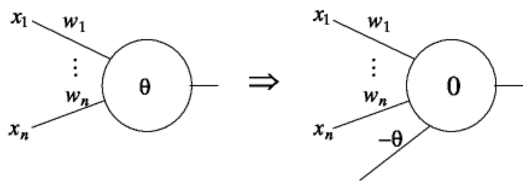


Fig. 11: A perceptron with a bias

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 $w_{n+1} = \theta$.

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?

Example 4: Consider a function of two variables. Table 5 shows all 16 possible Boolean functions of two variables f_0 to f_{15}. Each column f_1 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 5

x_1	x_2	f_0	f_1	f_2	f_3	f_4	f_5	f_6	f_7
0	0	0	1	0	1	0	1	0	1
0	1	0	0	1	1	0	0	1	1
1	0	0	0	0	0	1	1	1	1
1	1	0	0	0	0	0	0	0	0

x_1	x_2	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	0	1	1	0	0	1	1
1	0	0	0	0	0	1	1	1	1
1	1	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. Fig. 12 shows two possible separations to compute the OR and the AND functions.

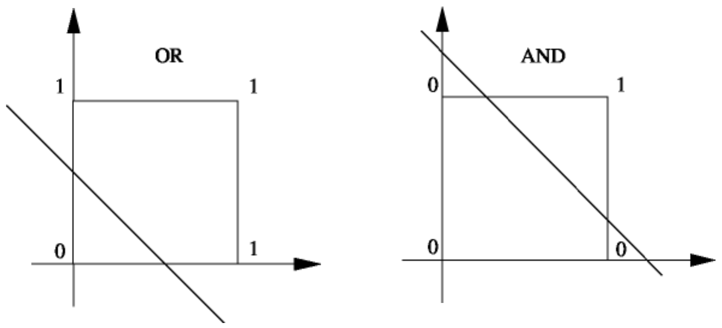


Fig. 12: Separations of input space corresponding to OR and AND

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:

$$x_1 = 0 \; x_2 = 0 \; w_1x_1 + w_2x_2 = 0 \Rightarrow 0 < \theta$$
$$x_1 = 1 \; x_2 = 0 \; w_1x_1 + w_2x_2 = w_1 \Rightarrow w_1 \geq \theta$$
$$x_1 = 0 \; x_2 = 1 \; w_1x_1 + w_2x_2 = w_2 \Rightarrow w_2 \geq 0$$
$$x_1 = 1, \; x_2 = 1, \; w_1 + w_2 \Rightarrow w_1 + w_2 < \theta$$

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.

Try the following exercises.

E4) Consider the ADALINE filter with three neurons in the input layer having weights $w_{11} = 2$, $w_{12} = -1$ and $w_{13} = 3$ and the input sequences is $\{\cdots,0,0,0,5,-4,0,0,0,\cdots\}$

i) What is the filter output just prior to 0 ?

ii) What is the filter output from 0 to 5 ?

E5) The following is a training set for a 2-classification problem. Iterate the perceptron through the training set and obtain the weights.

Inputs		Classification
X1	**X2**	**0/1**
0.25	0.353	0
0.25	0.471	1
0.5	0.353	0
0.5	0.647	1
0.75	0.705	0
0.75	0.882	1
1	0.705	0
1	1	1

E6) Attempt solving the XOR problem using the above implementation. Record the weights. What are your observations?

Now, let us summarize the unit.

5.4 SUMMARY

In this unit we have covered the following

1. Learning algorithm in the perceptron is performed for a finite number of iterations and their steps.
2. Single layer perceptrons are the cases where the computation layer consists of a single neuron.
3. The classical architecture of neural network is the Rosenblatt's perceptron. The major application of this is ADALINE.
4. The computing units are too similar to conventional logic gates and the network must be completely specified before it can be used.

5.5 SOLUTIONS/ANSWERS

E1) i) The net input

$$= wx + b = (2.3)\,(2) + (-3) = 1.6$$

ii) For the hard limit transfer function;

$$\text{Output} = \text{hardlim}\,(1.6) = 1.0$$

For the linear transfer function;

$$\text{Output} = \text{purelin}\,(1.6) = 1.6$$

For the log-sigmoid transfer function;

$$\text{Output} = \text{logsig}\,(1.6) = \frac{1}{1+e^{-1.6}} = 0.8320$$

E2) First calculate the net input n:

$$n = Wx + b = \begin{bmatrix}3 & 2\end{bmatrix}\begin{bmatrix}-5\\ 6\end{bmatrix} + (1.2) = -1.8.$$

Now find the outputs for each of the transfer functions.

i) $a = \text{hardlims}\,(-1.8) = -1$

ii) a = lin (–1.8) = 0

iii) a = tansig (–1.8) = – 0.9468

E3) The problem specifications allow you to say the following about the network.

i) Two neurons, one for each output, are required.

ii) The weight matrix has two rows corresponding to the two neurons and six columns corresponding to the six inputs. (The product $\mathbf{W_p}$ is a two-element vector.)

iii) Of the transfer functions we have discussed, the logsig transfer function would be most appropriate.

iv) Not enough information is given to determine if a bias is required.

E4) i) Just prior to 0 three zeros have entered the filter, and the output is zero.

ii) At 0 the digit "5" has entered the filter, and it will be multiplied by w_{11}, which has the value 2, so that output is 10. This can be viewed as the matrix operation:

$$\text{Output} = Wx(0) = \begin{bmatrix} 2 & -1 & 3 \end{bmatrix} \begin{bmatrix} 5 \\ 0 \\ 0 \end{bmatrix} = 10.$$

Similarly, we can calculate the next outputs as

$$Wx(1) = \begin{bmatrix} 2 & -1 & 3 \end{bmatrix} \begin{bmatrix} -4 \\ 5 \\ 0 \end{bmatrix} = -13$$

$$Wx(2) = \begin{bmatrix} 2 & -1 & 3 \end{bmatrix} \begin{bmatrix} 0 \\ -4 \\ 5 \end{bmatrix} = 19$$

$$Wx(3) = \begin{bmatrix} 2 & -1 & 3 \end{bmatrix} \begin{bmatrix} 0 \\ 0 \\ -4 \end{bmatrix} = -12$$

and

$$Wx(4) = \begin{bmatrix} 2 & -1 & 3 \end{bmatrix} \begin{bmatrix} 0 \\ 0 \\ 0 \end{bmatrix} = 0.$$

All remaining outputs will be zero.

5.6 PRACTICAL EXERCISES

Session 6

1) Implement the perceptron learning algorithm in the computer. Find the weights for an edge detection operator using this program. The input-output examples can be taken from a digitized picture of an object and another one in which only the edges of the object have been kept.

2) Implement using C, the fixed increment perceptron learning algorithm.
[**Hint:** for fixed increment perceptron learning algorithm α is fixed.]

5.7 REFERENCES

1. Simon S Haykin and Simon Haykin, (1998), *Neural Networks: A Comprehensive Foundation*, Pearson Education.
2. Raul Rojas, (1996), *Neural Networks: A Systematic Introduction*.
3. S. Rajasekaran and G.A. Vijayalakshmi Pai, (2010), *Neural Netwoks: Fuzzy Logic, and Genetic Algorithms*.
4. D.K. Pratihar, (2008), *Soft Computing*.
5. Valluru B. Rao and Hayagriva V. Rao, *C + + Neural Networks & Fuzzy Logic*.