

### 3 Perceptron

The perceptron was introduced by McCulloch and Pitts in 1943 as an artificial neuron with a hard-limiting activation function,  $\sigma$ . Recently the term *multilayer perceptron* has often been used as a synonym for the term *multilayer feedforward neural network*. In this section we will be referring to the former meaning.

A structure of a perceptron is depicted in Figure 3–1

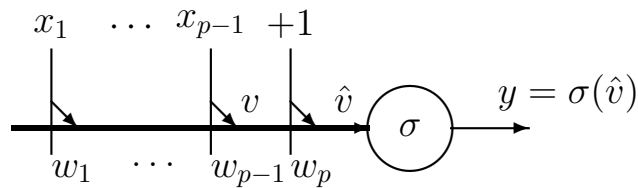


Figure 3–1: A single perceptron with a biasing input

Input signals,  $x_i$ , are assumed to have real values. The activation function,  $\sigma$ , is a unipolar step function (sometimes called the Heaviside function), therefore, the output signal is binary,  $y \in \{0, 1\}$ . One input signal is constant ( $x_p = 1$ ), and the related weight is interpreted as the bias, or threshold,  $\theta$ :

$$w_p = -\theta$$

The **input** signals and **weights** are arranged in the following column and row vectors, respectively:

$$\mathbf{x} = \begin{bmatrix} x_1 & \cdots & x_{p-1} & 1 \end{bmatrix}^T ; \quad \mathbf{w} = \begin{bmatrix} w_1 & \cdots & w_{p-1} & w_p \end{bmatrix}$$

Aggregation of the “proper” input signals results in the **activation potential**,  $v$ , which can be expressed as the **inner product** of “proper” input signals and related weights:

$$v = \sum_{i=1}^{p-1} w_i x_i = \mathbf{w}_{1:p-1} \cdot \mathbf{x}_{1:p-1}$$

The augmented activation potential,  $\hat{v}$ , can now be expressed simply as:

$$\hat{v} = \mathbf{w} \cdot \mathbf{x} = v - \theta$$

For each input signal, the output is determined as

$$y = \sigma(\hat{v}) = \begin{cases} 0 & \text{if } v < \theta \quad (\hat{v} < 0) \\ 1 & \text{if } v \geq \theta \quad (\hat{v} \geq 0) \end{cases}$$

Hence, a perceptron works as a **threshold element**, the output being “active” if the activation potential exceeds the threshold.

### 3.1 A Perceptron as a Pattern Classifier

A single perceptron classifies input patterns,  $\mathbf{x}$ , into **two classes**.

A linear combination of signals and weights for which the augmented activation potential is zero,  $\hat{v} = 0$ , describes a **decision surface** which partitions the input space into two regions. The decision surface is a **hyperplane** specified as:

$$\mathbf{w} \cdot \mathbf{x} = 0, \quad x_p = 1, \quad \text{or} \quad v = \theta \quad (3.1)$$

and we say that an input vector (pattern)

$$\mathbf{x} \text{ belongs to a class } \begin{cases} \#1 \\ \#0 \end{cases} \text{ if } \begin{cases} y = 1, & (v \geq \theta) \\ y = 0, & (v < \theta) \end{cases}$$

The input patterns that can be classified by a single perceptron into two distinct classes are called **linearly separable patterns**.

### Useful remarks

- The inner product of two vectors can be also expressed as:

$$\mathbf{w} \cdot \mathbf{x} = \|\mathbf{w}\| \cdot \|\mathbf{x}\| \cdot \cos \alpha$$

where  $\|\cdot\|$  denotes the norm (magnitude, length) of the vector, and  $\alpha$  is the angle between vectors.

- The weight vector,  $\mathbf{w}$ , is orthogonal to the decision plane in the augmented input space, because

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

that is, the inner product of the weight and augmented input vectors is zero.

- Note that the weight vector,

$$\mathbf{w}_{1:p-1}$$

is also orthogonal to the decision plane

$$\mathbf{w}_{1:p-1} \cdot \mathbf{x}_{1:p-1} + w_p = 0$$

in the “proper”,  $(p-1)$ –dimensional input space. For any pair of input vectors, say,  $(\mathbf{x}_{1:p-1}^a, \mathbf{x}_{1:p-1}^b)$ , for which the decision plane equation is satisfied, we have

$$\mathbf{w}_{1:p-1} \cdot (\mathbf{x}_{1:p-1}^a - \mathbf{x}_{1:p-1}^b) = 0$$

Note that the difference vector  $(\mathbf{x}_{1:p-1}^a - \mathbf{x}_{1:p-1}^b)$  lies in the decision plane.

## Geometric Interpretation

Let us consider a 3-dimensional augmented input space ( $p = 3$ ). Input vectors are 2-dimensional, and  $\mathbf{x} = [x_1 x_2 1]^T$ . The decision plane, is described as follows:

$$w_1x_1 + w_2x_2 + w_3x_3 = 0, \quad x_3 = 1 \quad (3.2)$$

The sketch of the geometrical objects in the 3-D space is given in Figure 3–2

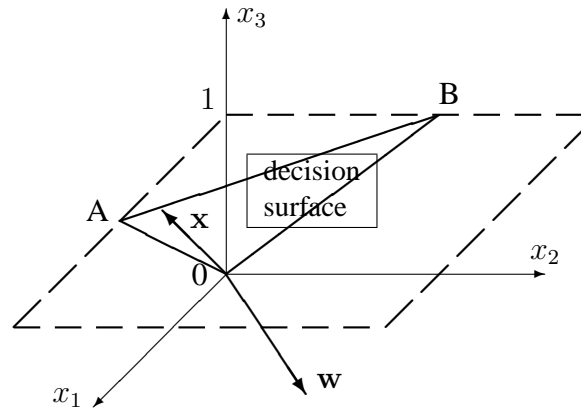


Figure 3–2: Augmented input space with the decision plane

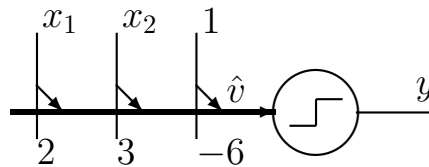
The augmented decision plane, OAB, goes through the origin of the augmented input space (a homogeneous equation).

Each augmented input vector,  $\mathbf{x}$ , lies in this plane. The weight vector,  $\mathbf{w}$ , is orthogonal to the OAB plane hence to each augmented input vector.

Intersection of the augmented decision plane, OAB, and the horizontal plane,  $x_3 = 1$ , gives a 2-D “plane” AB (a straight line, in this case) described by eqn (3.2).

### 3.2 Example — a three-synapse perceptron

Consider a three-input perceptron with weights  $\mathbf{w} = [2 \ 3 \ -6]$



The total (augmented) activation potential is determine as:

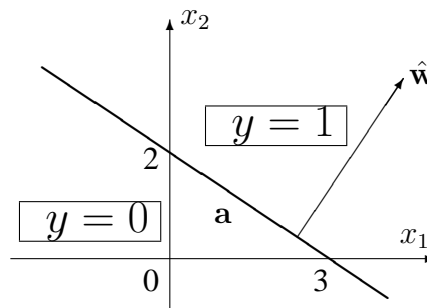
$$\hat{v} = \mathbf{w} \cdot \mathbf{x} = [2 \ 3 \ -6] \begin{bmatrix} x_1 \\ x_2 \\ 1 \end{bmatrix} = 2x_1 + 3x_2 - 6$$

The input space,  $[x_1 \ x_2]$ , is 2-dimensional and the decision plane is reduced to a straight line:

$$2x_1 + 3x_2 - 6 = 0$$

The 2-D weight vector  $\hat{\mathbf{w}} = [2 \ 3]$  is perpendicular to the decision line, that is, to the vector  $\mathbf{a} = [-3 \ 2]^T$  which is parallel to (lies in) the decision line, because the inner product  $\hat{\mathbf{w}} \cdot \mathbf{a} = 0$ , that is

$$[2 \ 3] \begin{bmatrix} -3 \\ 2 \end{bmatrix} = 0$$



Note that the weight vector  $\mathbf{w}$  points in the “positive” direction, that is, to the side of the plane where  $y = 1$ .

□

### 3.3 Selection of weights for the perceptron

In order for the perceptron to perform a desired task, its weights must be properly selected.

In general two basic methods can be employed to select a suitable weight vector:

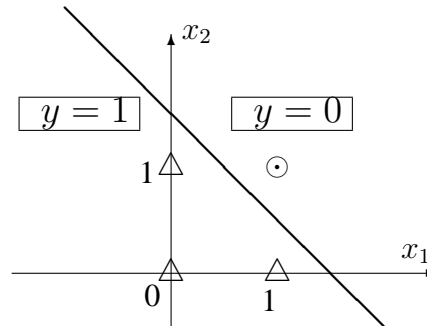
- By off-line calculation of weights. If the problem is relatively simple it is often possible to calculate the weight vector from the specification of the problem.
- By learning procedure. The weight vector is determined from a given (training) set of input-output vectors (exemplars) in such a way to achieve the best classification of the training vectors.

Once the weights are selected (the **encoding process**), the perceptron performs its desired task (e.g., pattern classification) (the **decoding process**).

### 3.3.1 Selection of weights by off-line calculations — Example

Consider the problem of building the **NAND** gate using the perceptron. In this case the desired input-output relationship is specified by the following truth table and related plot:

|       |   |   |   |   |
|-------|---|---|---|---|
| $x_1$ | 0 | 0 | 1 | 1 |
| $x_2$ | 0 | 1 | 0 | 1 |
| $y$   | 1 | 1 | 1 | 0 |



The input patterns (vectors) belong to two classes and are marked in the input space (the plane  $(x_1, x_2)$ ) with  $\triangle$  and  $\odot$ , respectively. The decision plane is the straight line described by the following equation

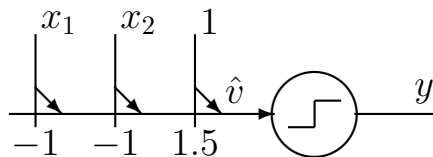
$$x_1 + x_2 = 1.5 \quad \text{or} \quad -x_1 - x_2 + 1.5 = 0$$

In order for the weight vector to point in the direction of  $y = 1$  patterns we select it as:

$$\mathbf{w} = [-1 \quad -1 \quad 1.5]$$

For each input vector the output signal is now calculated as

$$\hat{v} = [-1 \quad -1 \quad 1.5] \begin{bmatrix} x_1 \\ x_2 \\ 1 \end{bmatrix} ; \quad y = \begin{cases} 0 & \text{if } \hat{v} < 0 \\ 1 & \text{if } \hat{v} \geq 0 \end{cases}$$



Verify that such a perceptron does implement a two-input NAND gate.

### 3.4 The Perceptron learning law

**Learning** is a recursive procedure of modifying weights from a given set of input-output patterns.

For a single perceptron, the objective of the learning (encoding) procedure is to find the decision plane, (that is, the related weight vector), which separates two classes of given **input-output training vectors**.

Once the learning is finalised, every input vector will be classified into an appropriate class. A single perceptron can classify only the **linearly separable** patterns.

The perceptron learning procedure is an example of a **supervised error-correcting learning** law.

More specifically, the supervised learning procedure can be described as follows

given the set of  $N$  **training patterns** consisting of input signals  $\mathbf{x}(n)$  and the related **desired** output signals,  $d(n)$ :

$$\begin{bmatrix} \mathbf{x}(n) \\ d(n) \end{bmatrix} = \begin{bmatrix} x_1(n) \\ \vdots \\ x_{p-1}(n) \\ 1 \\ d(n) \end{bmatrix}, \quad \text{for } n = 1, 2, \dots, N$$

obtain the correct decision plane specified by the weight vector  $\mathbf{w}$ .

The training patterns are arranged in a **training set** which consists of a  $p \times N$  input matrix,  $X$ , and an  $N$ -element output vector,  $D$ :

$$\begin{bmatrix} X \\ D \end{bmatrix} = \begin{bmatrix} x_1(1) & x_1(2) & \cdots & x_1(N) \\ x_2(1) & x_2(2) & \cdots & x_2(N) \\ \vdots & \vdots & & \vdots \\ x_{p-1}(1) & x_{p-1}(2) & \cdots & x_{p-1}(N) \\ 1 & 1 & 1 & 1 \\ \hline d(1) & d(2) & \cdots & d(N) \end{bmatrix}$$

We can assume that an **untrained perceptron** can generate an incorrect output  $y(n) \neq d(n)$  due to incorrect values of weights. We can think about the **actual output**  $y(n)$  as an “**estimate**” of the **correct, desired** output  $d(n)$ .

The **perceptron learning law** proposed by Rosenblatt in 1959 also known as the **perceptron convergence procedure**, can be described as follows

1. weights are initially set to “small” random values

$$\mathbf{w}(0) = \text{rand}$$

2. for  $n = 1, 2, \dots, N$  training input vectors,  $\mathbf{x}(n)$ , are presented to the perceptron and the **actual output**  $y(n)$ , is compared with the **desired output**,  $d(n)$ , and the **error**  $\varepsilon(n)$  is calculated as follows

$$y(n) = \sigma(\mathbf{w}(n) \cdot \mathbf{x}(n)) , \quad \varepsilon(n) = d(n) - y(n)$$

Note that because

$$d(n), y(n) \in \{0, 1\}, \quad \text{then} \quad \varepsilon(n) \in \{-1, 0, +1\}$$

3. at each step, the weights are adapted as follows

$$\mathbf{w}(n+1) = \mathbf{w}(n) + \eta \varepsilon(n) \mathbf{x}(n) \quad (3.3)$$

where  $0 < \eta < 1$  is a learning parameter (gain), which controls the adaptation rate. The gain must be adjusted to ensure the convergence of the algorithm.

Note that the weight update

$$\Delta \mathbf{w} = \eta \varepsilon(n) \mathbf{x}(n) \quad (3.4)$$

is zero if the error  $\varepsilon(n) = 0$ .

For convenience the weight update equation (3.4) is presented in the following table:

| $d(n)$ | $y(n)$ | $\varepsilon(n)$ | $\Delta \mathbf{w}(n)$ |
|--------|--------|------------------|------------------------|
| 0      | 0      | 0                | 0                      |
| 1      | 1      | 0                | 0                      |
| 0      | 1      | -1               | $-\eta \mathbf{x}(n)$  |
| 1      | 0      | +1               | $+\eta \mathbf{x}(n)$  |

The sketch of the geometric relationships involved in the above algorithm is presented in Figure 3–3.

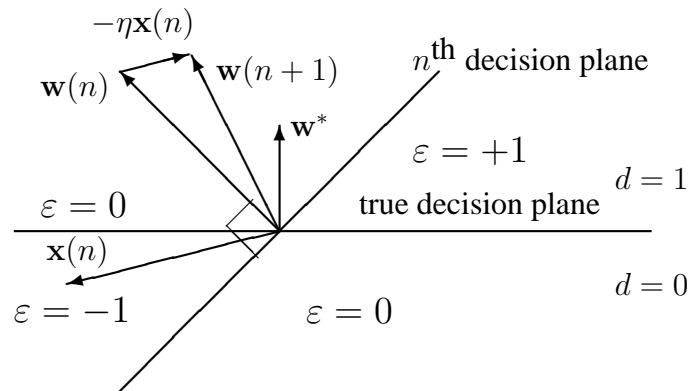


Figure 3–3: Geometric interpretation of the perceptron learning law

In Figure 3–3 we can identify a current weight vector,  $\mathbf{w}(n)$ , the next weight vector,  $\mathbf{w}(n+1)$ , and the correct weight vector,  $\mathbf{w}^*$ . Related decision planes are orthogonal to these vectors and are depicted as straight lines.

During the learning process the current weight vector  $\mathbf{w}(n)$  is modified in the direction of the current input vector  $\mathbf{x}(n)$ , if the input pattern is misclassified, that is, if the error is non-zero.

Presenting the perceptron with enough training vectors, the weight vector  $\mathbf{w}(n)$  will tend to the correct value  $\mathbf{w}^*$ .

Rosenblatt proved that if input patterns are linearly separable, then the perceptron learning law converges, and the hyperplane separating two classes of input patterns can be determined.

The structure of the perceptron and its learning law are presented in the following block-diagram

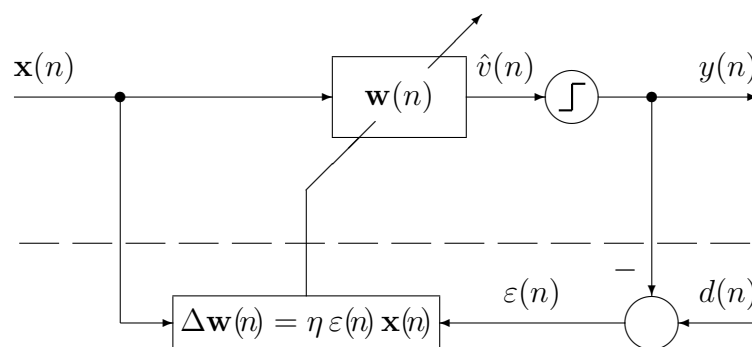


Figure 3–4: The structure of a perceptron with its learning part.

Note the **decoding part** of the perceptron is a static, feedforward system, whereas the **encoding part** is a discrete-time dynamic system described by the difference equation (3.3).

### 3.5 Implementation of the perceptron learning law in MATLAB — Example

The following numerical example demonstrates the perceptron learning law. In the example we will demonstrate three main steps typical to most of the neural network applications, namely:

**Preparation of the training patterns.** Normally, the training patterns are specific to the particular application. In our example we generate a set of vectors and partition this set with a plane into two linearly separable classes. The set of vectors is split into two halves, one used as a **training** set, the other as a **validation** set.

**Learning.** Using the training set, we apply the perceptron learning law as described in sec. 3.4

**Validation.** Using the validation set we verify the quality of the learning process.

Calculations are performed in MATLAB.

**Description of a demo script perc.m**

`p = 5 ; % dimensionality of the augmented input space`

`N = 50 ; % number of training patterns — size of the training epoch`

**PART 1: Generation of the training and validation sets**

`X = 2*rand(p-1, 2*N)-1 ; % a  $(p-1) \times 2N$  matrix of  
uniformly distributed random numbers from the interval  $[-1, +1]$`

`nn = round((2*N-1)*rand(N,1))+1 ; % generation of  $N$   
random integer numbers from the range  $[1..2N]$ . Repetitions are  
possible.`

`X(:,nn) = sin(X(:,nn)) ; % Columns of the matrix  $X$   
pointed to by  $nn$  are "coloured" with the function 'sin', in order  
to make the training patterns more interesting.`

`X = [X; ones(1, 2*N)] ; % Each input vector is appended with  
a constant 1 to implement biasing.`

Comment: to generate  $N$  integer indices from  $[1..2N]$  without  
repetition use:

`[srt,nn] = sort(rand(1, 2*N)) ; nn=nn(1:N) ;`

The resulting matrix of input patterns,  $X$ , may be of the following  
form:

|     |   |       |      |       |     |       |       |
|-----|---|-------|------|-------|-----|-------|-------|
| $X$ | = | -0.02 | 0.83 | -0.72 | ... | -0.68 | -0.04 |
|     |   | 0.33  | 0.73 | -0.10 | ... | 0.45  | -0.09 |
|     |   | 0.36  | 0.78 | 0.98  | ... | -0.41 | 0.04  |
|     |   | -0.57 | 0.09 | -0.57 | ... | -0.90 | -0.11 |
|     |   | 1.00  | 1.00 | 1.00  | ... | 1.00  | 1.00  |

where only the three initial and two final columns of the matrix  $X$  are  
shown.

Specification of an arbitrary separation plane in the augmented input space:

```
wht = 3*rand(1,p)-1; wht = wht/norm(wht); % This
      is a unity length vector orthogonal to the augmented separation
      plane. It is also the target weight vector. The result may be of the
      form:
```

```
wht = 0.38 0.66 -0.14 0.61 0.14
```

```
D = (wht*X >= 0); % Classification of every point from the
      input space with respect to the class number, 0 or 1. The three
      initial and two terminal values of the output vector may be of the
      following form:
```

```
D = 0 1 1 ... 0 0
```

```
Xv = X(:, N+1:2*N) ; % The validation set: Xv is  $p \times N$ 
```

```
Dv = D(:, N+1:2*N) ; % Dv is  $1 \times N$ 
```

```
X = X(:, 1:N) ; % The training set: X is  $p \times N$ 
```

```
D = D(:, 1:N) ; % D is  $1 \times N$ 
```

**Visualisation** of the input-output patterns.

The input space is  $p$ -dimensional, hence difficult to visualise.

We will plot projections of input patterns on a 2-D plane, say  $(x_1 - x_3)$  plane. The projection is performed by extracting rows `pr` from the  $X$  and  $D$  matrices.

```
pr = [1, 3]; Xp = X(pr, :);
```

```
wp = wht([pr p]); % projection of the weight vector
```

```
c0 = find(D==0); c1 = find(D==1); % c0 and c1 are
      vectors of pointers to input patterns  $X$  belonging to the class 0 or
      1, respectively.
```

```
figure(1), clf reset
plot(Xp(1,c0),Xp(2,c0),'o',Xp(1,c1),Xp(2,c1),'x')
    % The input patterns are plotted on the selected projection plane.
    % Patterns belonging to the class 0, or 1 are marked with 'o', or 'x',
    % respectively

axis(axis), hold on % The axes and the contents of the
    current plot are frozen
```

Superimposition of the projection of the separation plane on the plot.  
The projection is a straight line:

$$x_1 \cdot w_1 + x_3 \cdot w_3 + w_5 = 0$$

To plot this line we need at least two points. However, in order to account for all possible situations we will use four co-linear points as presented in sec. 3.7.

```
L = [-1 1] ;
S = -diag([1 1]./wp(1:2))*(wp([2,1])'*L + wp(3)) ;
plot([S(1,:) L], [L S(2,:)]), grid, drawnow
```

The relevant plot is in Figure 3–5.

**PART 2: Learning**

The training input-output patterns are stored in matrices  $X$  ( $p \times N$ ) and  $D$  ( $1 \times N$ ). We will start with a randomly selected weight vector (thus related separation/decision plane) and in the training procedure the weight vector should converge to the weight vector specifying the correct separation plane.

```
eta = 0.5; % The training gain.
```

```
wh = 2*rand(1,p)-1; % Random initialisation of the weight
    vector with values from the range  $[-1, +1]$ . An example of an
    initial weight vector follows
```

```
wh = 0.36 -0.34 -0.89 0.97 0.08
```

Projection of the initial decision plane which is orthogonal to `wh` is plotted as previously:

```
wp = wh([pr p]); % projection of the weight vector
S = -diag([1 1]./wp(1:2))*(wp([2,1])'*L +wp(3)) ;
plot([S(1,:) L], [L S(2,:)]), grid on, drawnow
```

In what follows, the internal loop controlled by the variable  $n$  goes through  $N$  training exemplars (one epoch). The loop is repeated until the performance index (error)  $E$  is small but not more than  $C$  times ( $C$  epochs). The projection of the current decision surface is plotted after the previous projection has been erased.

```
C = 50; % Maximum number of training epochs
```

```
E = [C+1, zeros(1,C)]; % Initialization of the vector of the
    total sums of squared errors over an epoch.
```

```
WW = zeros(C*N,p); % The matrix WW will store all weight
    vectors wh, one weight vector per row of the matrix WW
```

```

c = 1; % c is an epoch counter

cw = 0 ; % cw total counter of weight updates

while (E(c)>1) | (c==1)
    c = c+1 ;

    plot([S(1,:) L], [L S(2,:)], 'w') % At the
        beginning of each internal loop the former projection of the
        decision plane is erased (option 'w')

    for n = 1:N % The internal loop goes once through all training
        exemplars.

        eps = D(n) - ((wh*X(:,n)) >= 0); %
             $\varepsilon(n) = d(n) - y(n)$ 

        wh = wh + eta * eps * X(:,n)'; % The Perceptron Learning Law

        cw = cw + 1;

        WW(cw,:) = wh / norm(wh); % The updated and normalised weight
            vector is stored in WW for feature plotting

        E(c) = E(c) + abs(eps); %  $|\varepsilon| = \varepsilon^2$ 

    end;

    wp = wh([pr pr]); % projection of the weight vector
    S = -diag([1 1] ./ wp(1:2)) * (wp([2,1])' * L + wp(3));
    plot([S(1,:) L], [L S(2,:)], 'g'), drawnow
end;

```

After every pass through the set of training patterns, the projection of the current decision plane, which is determined by the current weight vector, is plotted after the previous projection has been erased.

```

WW = WW(1:cw, pr);
E = E(2:c+1)

```

An example of the performance index  $E$  and the final value of the weight vector is as follows:

$E = 10 \ 6 \ 4 \ 3 \ 1 \ 0$   
 $wt = 0.29 \ 0.73 \ -0.18 \ 0.61 \ 0.14$

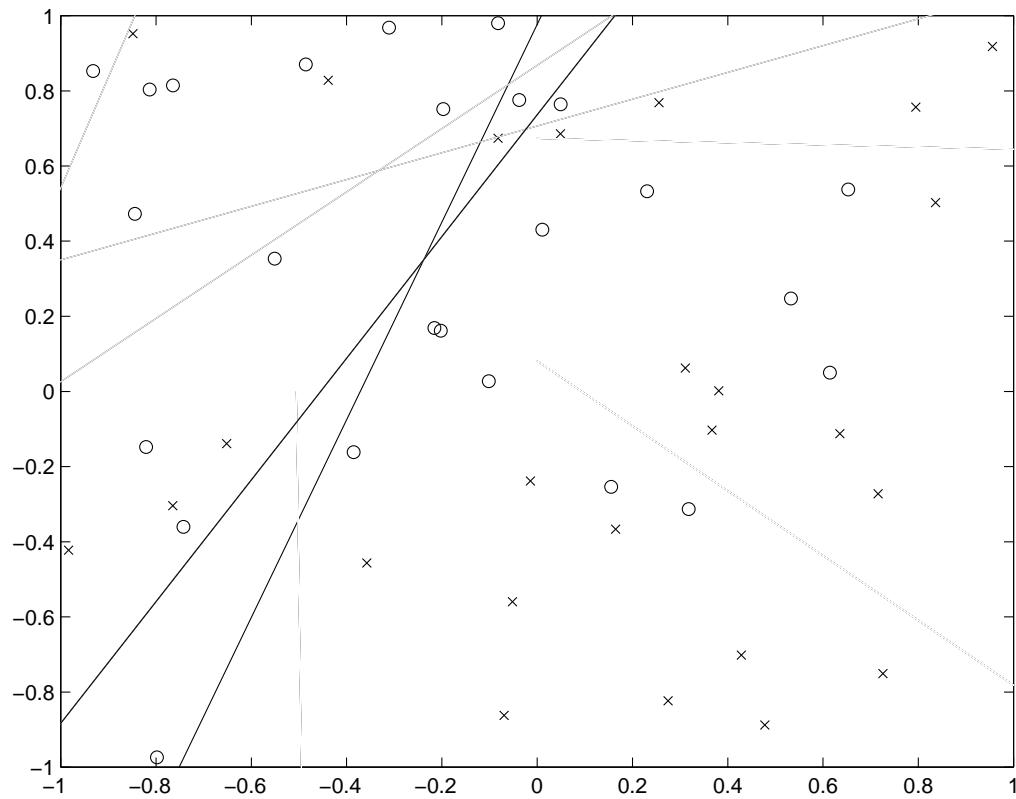


Figure 3–5: Projection of the input space onto a 2–D plane.

## Validation

The final decision (separation) planes differs slightly from the correct one. Therefore, if we use input signals from the validation set which were not used in training, it may be expected that some patterns will be misclassified and the following total sum of squared errors will be non-zero.

$$E_v = \text{sum}(\text{abs}(D_v - (\text{wh} * X_v \geq 0)))'$$

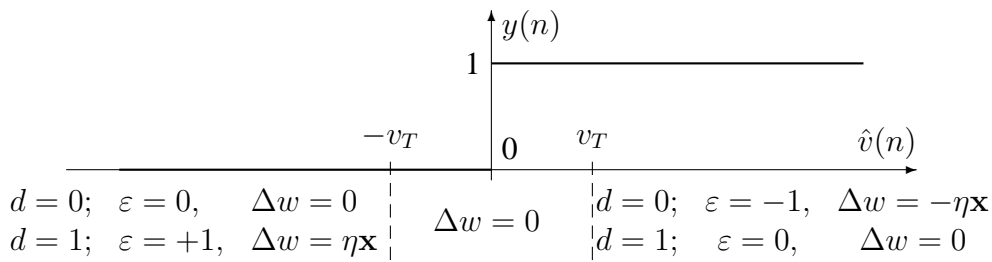
The values of  $E_v$  over many training sessions were between 0 and 5 (out of  $N = 50$ ).

### 3.6 A modified perceptron learning rule

In the basic version of the perceptron learning rule, the weight update occurs for each input vector,  $\mathbf{x}(n)$ , which generates output,  $y(n)$ , different from desired  $d(n)$ , that is, when the error  $\varepsilon(n)$  is non-zero. In the modified version of the perceptron learning rule, in addition to misclassification, the input vector  $\mathbf{x}(n)$  must be located far enough from the decision surface, or, alternatively, the net activation potential  $\hat{v}(n)$  must exceed a preset margin,  $v_T$ . The reason for adding such a condition is to avoid an erroneous classification decision in a situation when  $\hat{v}(n)$  is very small, and due to presence of noise, the sign of  $\hat{v}(n)$  cannot be reliably established.

The modified perceptron learning rule can be illustrated in the following tabular and graphical forms:

| $d(n)$ | $y(n)$ | $\varepsilon(n)$ | $\hat{v}(n)$ | update if  | $\Delta w(n)$         |
|--------|--------|------------------|--------------|------------|-----------------------|
| 0      | 0      | 0                | $v < 0$      | —          | 0                     |
| 1      | 1      | 0                | $v \geq 0$   | —          | 0                     |
| 0      | 1      | -1               | $v \geq 0$   | $v > v_T$  | $-\eta \mathbf{x}(n)$ |
| 1      | 0      | +1               | $v \geq 0$   | $v < -v_T$ | $+\eta \mathbf{x}(n)$ |



Finally, the additional update condition

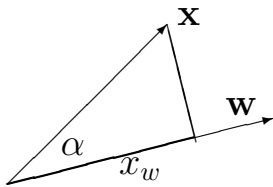
$$|\hat{v}(n)| > v_T \quad (3.5)$$

will be converted into an equivalent condition for the input vector and illustrated in the input space. Eqn (3.5) can be written as

$$|\mathbf{w}(n) \cdot \mathbf{x}(n)| > v_T$$

But the absolute value of the inner product can be expressed as

$$|\mathbf{w} \cdot \mathbf{x}| = \|\mathbf{w}\| \cdot \|\mathbf{x}\| \cdot \cos \alpha = \|\mathbf{w}\| \cdot |\hat{x}_w|$$

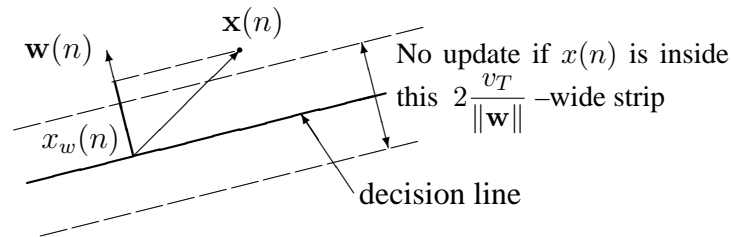


where  $\|\cdot\|$  denotes the length (norm) of the vector, and  $x_w = \|\mathbf{x}\| \cdot \cos \alpha$  is the projection of the input vector,  $\mathbf{x}$  on the weight vector,  $\mathbf{w}$ .

The additional update condition (3.5) can be now written as

$$\|\mathbf{w}\| \cdot |x_w| > v_T \quad \text{or as} \quad |x_w| > \frac{v_T}{\|\mathbf{w}\|}$$

The condition says that in order the update to occur the current input vector,  $\hat{x}$  must lie outside the  $2v_T/\|\hat{\mathbf{w}}(n)\|$  strip surrounding the decision plane, as illustrated for the 2-D case in the following figure:



### 3.7 Intersection of a cube by a plane. A 2-D case.

Consider a straight line described by the following equation

$$\mathbf{w} \cdot \mathbf{x} = 0 \quad (3.6)$$

where

$$\mathbf{w} = [w_1 \ w_2 \ w_3] ; \quad \mathbf{x} = [x_1 \ x_2 \ 1]^T$$

Given a square limited by the following four edge lines:

$$\begin{aligned} x_1 &= l_{11} ; & x_1 &= l_{12} \\ x_2 &= l_{21} ; & x_2 &= l_{22} \end{aligned}$$

we would like to calculate all four intersection points of the square edges by the straight line, as in Figure 3–6

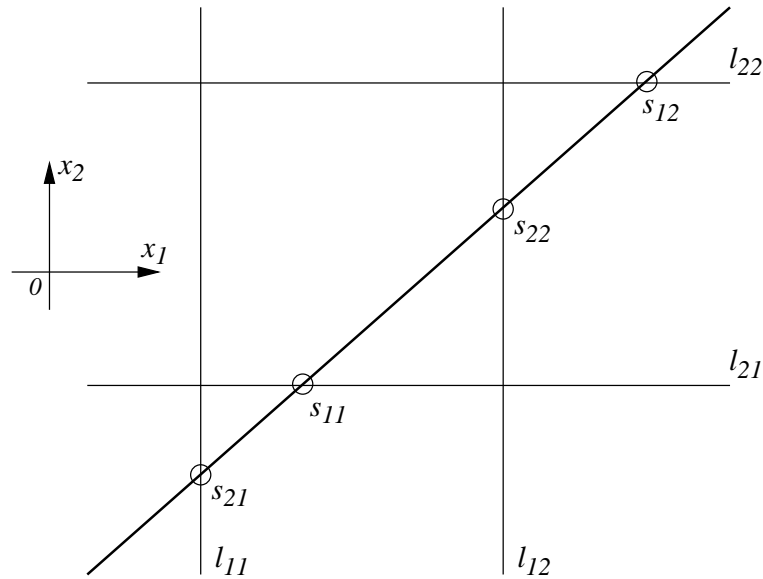


Figure 3–6: A straight line intersecting a square

The coordinates of the four intersection points can be arranged as follows:

$$\begin{array}{c|cc|cc} x_1 & l_{11} & l_{12} & s_{11} & s_{12} \\ x_2 & s_{21} & s_{22} & l_{21} & l_{22} \end{array} \quad (3.7)$$

For each column of table (3.7) we can write eqn (3.6) as

$$\begin{aligned} [w_1 \ w_2] \cdot \begin{bmatrix} s_{11} \\ l_{21} \end{bmatrix} &= -w_3, & [w_1 \ w_2] \cdot \begin{bmatrix} s_{12} \\ l_{22} \end{bmatrix} &= -w_3 \\ [w_1 \ w_2] \cdot \begin{bmatrix} l_{11} \\ s_{21} \end{bmatrix} &= -w_3, & [w_1 \ w_2] \cdot \begin{bmatrix} l_{12} \\ s_{22} \end{bmatrix} &= -w_3 \end{aligned}$$

Grouping the unknown  $s_{ij}$  on the left-hand side we have

$$\begin{aligned} w_1[s_{11} \ s_{12}] &= -w_2[l_{21} \ l_{22}] - w_3[1 \ 1] \\ w_2[s_{21} \ s_{22}] &= -w_1[l_{11} \ l_{12}] - w_3[1 \ 1] \end{aligned}$$

or in a matrix form

$$S = - \begin{bmatrix} 1/w_1 & 0 \\ 0 & 1/w_2 \end{bmatrix} \left( \begin{bmatrix} w_2 & 0 \\ 0 & w_1 \end{bmatrix} L + w_3 \begin{bmatrix} 1 & 1 \\ 1 & 1 \end{bmatrix} \right) \quad (3.8)$$

where

$$S = \begin{bmatrix} s_{11} & s_{12} \\ s_{21} & s_{22} \end{bmatrix} ; \quad L = \begin{bmatrix} l_{21} & l_{22} \\ l_{11} & l_{12} \end{bmatrix} ;$$

### 3.8 Design Constraints for a Multi-Layer Perceptron

In this section we consider a design procedure for implementation of combinational circuits using a multi-layer perceptron. In such a context, the perceptron is also referred to as a Linear Threshold Gate (LTG) [?].

Consider a single-hidden-layer perceptron as in Figure 3–7 described by the following equations

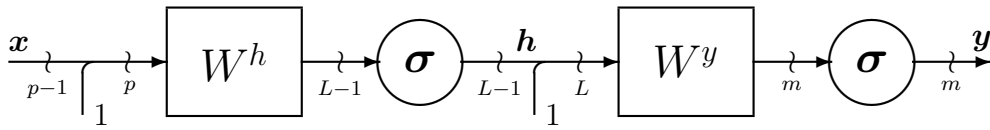


Figure 3–7: A multi-layer perceptron

$$\mathbf{h} = \sigma \left( W^h \cdot \begin{bmatrix} \mathbf{x} \\ 1 \end{bmatrix} \right), \quad \mathbf{y} = \sigma \left( W^y \cdot \begin{bmatrix} \mathbf{h} \\ 1 \end{bmatrix} \right) \quad (3.9)$$

where

$$\sigma(x) = \begin{cases} 0 & \text{for } x \leq 0 \\ 1 & \text{for } x > 0 \end{cases} \quad \text{is a step function (a hard limiter).}$$

Note that the step function has been shifted so that for a binary signal

$$x \in \{0, 1\}, \quad \sigma(x) = x$$

$\mathbf{x} = [x_1 \dots x_{p-1}]^T$  is a  $(p-1)$ –element input vector,

$\mathbf{h} = [h_1 \dots h_{L-1}]^T$  is an  $(L-1)$ –element hidden vector,

$\mathbf{y} = [y_1 \dots y_m]^T$  is an  $m$ –element output vector,

$W^h$  is an  $(L-1) \times p$  hidden weight matrix, and

$W^y$  is an  $m \times L$  output weight matrix.

The input and hidden signals are augmented with constants:

$$x_p = 1, \quad \text{and } h_L = 1$$

The total number of adjustable weights is

$$n_w = (L - 1)p + m \cdot L = L(p + m) - p \quad (3.10)$$

### Design Procedure

Assume that we are given the set of  $N$  points

$$\begin{bmatrix} X \\ Y \end{bmatrix} = \begin{bmatrix} \mathbf{x}(1) & \dots & \mathbf{x}(N) \\ \mathbf{y}(1) & \dots & \mathbf{y}(N) \end{bmatrix} \quad (3.11)$$

to be mapped by the perceptron so that

$$\mathbf{y}(n) = f(\mathbf{x}(n)) \quad \forall n = 1, \dots, N. \quad (3.12)$$

Combining eqns (3.9), (3.11), and (3.12) we obtain a set of **design constraints** (equations) which must be satisfied by the perceptron in order for the required mapping to be achieved:

$$\sigma \left( [W_{:,1:L-1}^y \cdot \sigma ([W_{:,1:p-1}^h \cdot X] \oplus_c W_{:,p}^h)] \oplus_c W_{:,L}^h \right) = Y \quad (3.13)$$

where

$$M \oplus_c \mathbf{c} \stackrel{\text{def}}{=} M + (\mathbf{1} \otimes \mathbf{c})$$

describe addition of a column vector  $\mathbf{c}$  to every column of the matrix  $M$  ( $\otimes$  denotes the Kronecker product, and  $\mathbf{1}$  is a row vector of ones). Eqn (3.13) can be re-written as a set of  $N$  individual constraints, one for each output,  $\mathbf{y}(n)$  where  $n = 1, \dots, N$ .

For a binary case, the eqn (3.11) represent a truth table of  $m$  Boolean functions of  $(p - 1)$  variables. The size of this table is

$$N = 2^{p-1}$$

The constraints (3.13) can now be simplify to

$$[W_{:,1:L-1}^y \cdot \sigma ([W_{:,1:p-1}^h \cdot X] \oplus_c W_{:,p}^h)] \oplus_c W_{:,L}^h = Y \quad (3.14)$$

**Conjecture 1** *Under appropriate assumptions the necessary condition for a solution to the design problem as in eqn (3.13) or (3.14) to exist is that the number of weights must be greater that the of points to be mapped, that is,*

$$L(p + m) - p > N$$

*Hence, the number of hidden signals*

$$L > \frac{N + p}{p + m} \quad (3.15)$$

For a single output ( $m = 1$ ) binary mapping when  $N = 2^{p-1}$  eqn (3.15) becomes

$$L > \frac{2^{p-1} + p}{p + 1} \quad (3.16)$$

Once the structural parameters  $p, L, m$  are known the weight matrices  $W_h, W_y$  that satisfy the constraint equation (3.13), or (3.14) can be selected.

### Example

Consider the XOR function for which

$$\begin{bmatrix} X \\ Y \end{bmatrix} = \begin{bmatrix} 0 & 1 & 0 & 1 \\ 0 & 0 & 1 & 1 \\ 1 & 1 & 1 & 0 \end{bmatrix}, \quad p = 3, \quad N = 2^{p-1} = 4 \quad (3.17)$$

From eqn (3.16) we can estimate that

$$L > \frac{N + p}{p + 1}, \quad \text{Let} \quad L = 3$$

The dendritic structure of the resulting two-layer perceptron is depicted in Figure 3–8.

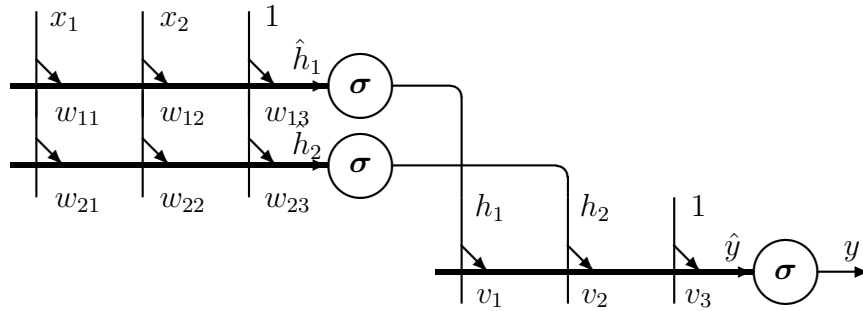


Figure 3–8: A dendritic diagram of a two-layer perceptron which implements the XOR function

A known combination of weights which implements the XOR function specified in eqn (3.17) can be obtained from the following plots in the input and hidden spaces presented in Figure 3–9. Equations of the straight lines illustrated in Figure 3–9 and related equations for activation potentials (“hat” signals) for all individual perceptrons are as follows

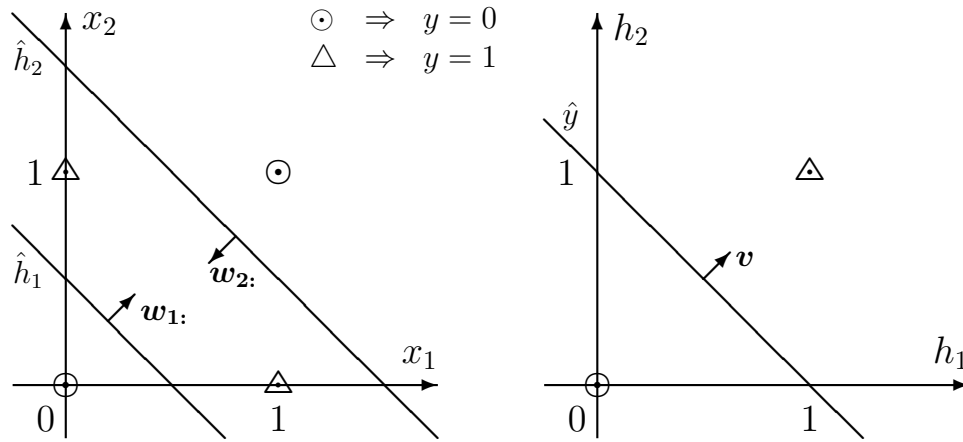


Figure 3-9: A possible configuration of the input and hidden spaces for the XOR perceptron

equations of lines:

activation signals:

$$x_1 + x_2 = 0.5 \implies \hat{h}_1 = x_1 + x_2 - 0.5$$

$$x_1 + x_2 = 1.5 \implies \hat{h}_2 = -x_1 - x_2 + 1.5$$

$$h_1 + h_2 = 1 \implies \hat{y} = h_1 + h_2 - 1$$

From the above equations the weight matrices can be assembled as follows

$$W^h = \begin{bmatrix} 1 & 1 & -0.5 \\ -1 & -1 & 1.5 \end{bmatrix}, \quad W^y = \mathbf{v} = [1 \quad 1 \quad -1] \quad (3.18)$$

### Constraint Equations for the XOR perceptron

In order to calculate other combinations of weights for the XOR perceptron we can re-write the constraint equation (3.14) using the structure as in Figure 3–8. The result is as follows

$$\begin{bmatrix} v_1 & v_2 \end{bmatrix} \cdot \sigma \left( \begin{bmatrix} w_{11} & w_{12} \\ w_{21} & w_{22} \end{bmatrix} \cdot \begin{bmatrix} 0 & 1 & 0 & 1 \\ 0 & 0 & 1 & 1 \end{bmatrix} \oplus_c \begin{bmatrix} w_{13} \\ w_{23} \end{bmatrix} \right) + v_3 = \begin{bmatrix} 0 & 1 & 1 & 0 \end{bmatrix} \quad (3.19)$$

Eqn (3.19) is equivalent to the following four scalar equations, the total number of weights to be selected being 9.

$$\begin{aligned} v_1 \cdot \sigma(w_{13}) + v_2 \cdot \sigma(w_{23}) + v_3 &= 0 \\ v_1 \cdot \sigma(w_{11} + w_{13}) + v_2 \cdot \sigma(w_{21} + w_{23}) + v_3 &= 1 \\ v_1 \cdot \sigma(w_{12} + w_{13}) + v_2 \cdot \sigma(w_{22} + w_{23}) + v_3 &= 1 \\ v_1 \cdot \sigma(w_{11} + w_{12} + w_{13}) + v_2 \cdot \sigma(w_{21} + w_{22} + w_{23}) + v_3 &= 0 \end{aligned}$$

It can easily be verified that the weights specified in eqn (3.18) satisfy the above constraints.