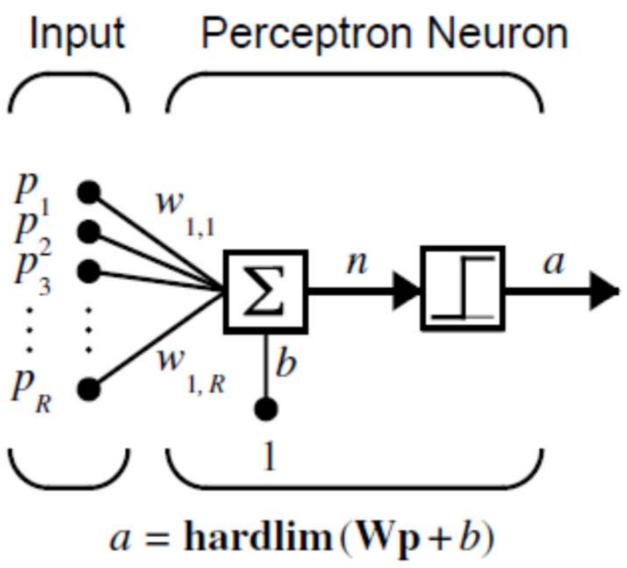
Neuralne mreže

Perceptrons

Perceptron neuron

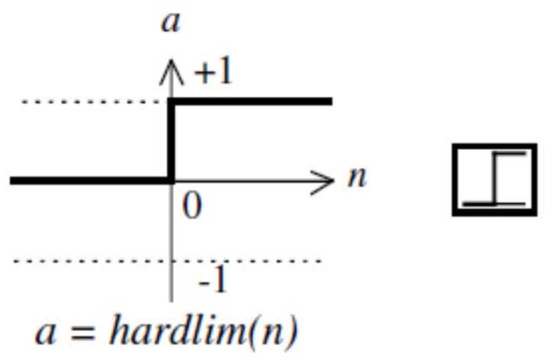
- One of the simplest was a single-layer network whose weights and biases could be trained to produce a correct target vector when presented with the corresponding input vector.
- The training technique used is called the perceptron learning rule.
- The perceptron generated great interest due to its ability to generalize from its training vectors and learn from initially randomly distributed connections.
- Perceptrons are especially suited for simple problems in pattern classification. They are fast and reliable networks for the problems they can solve.
- In addition, an understanding of the operations of the perceptron provides a good basis for understanding more complex networks.

Perceptron neuron



Where

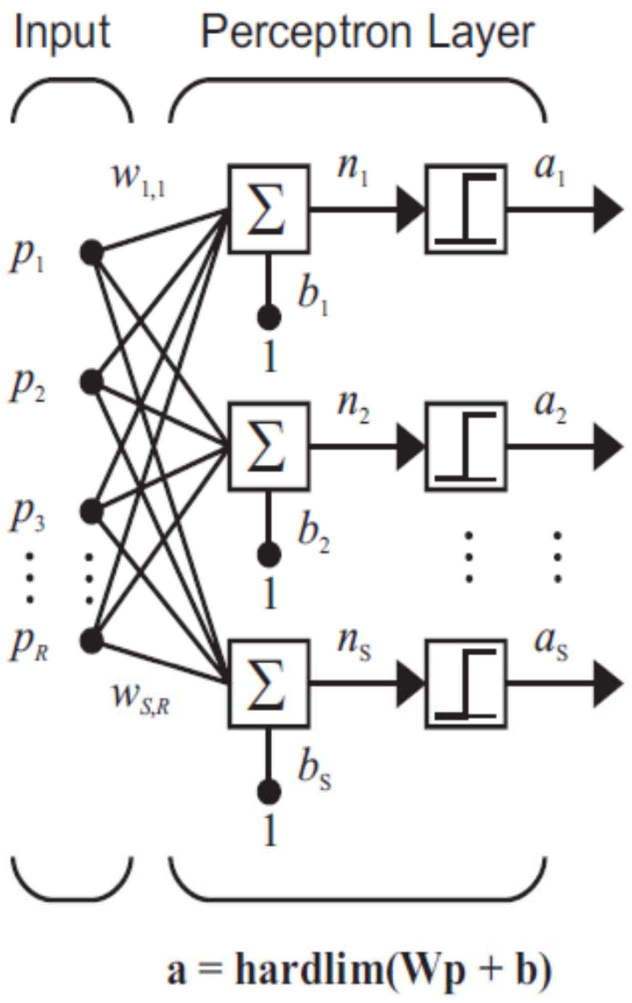
R = number of elements in input vector

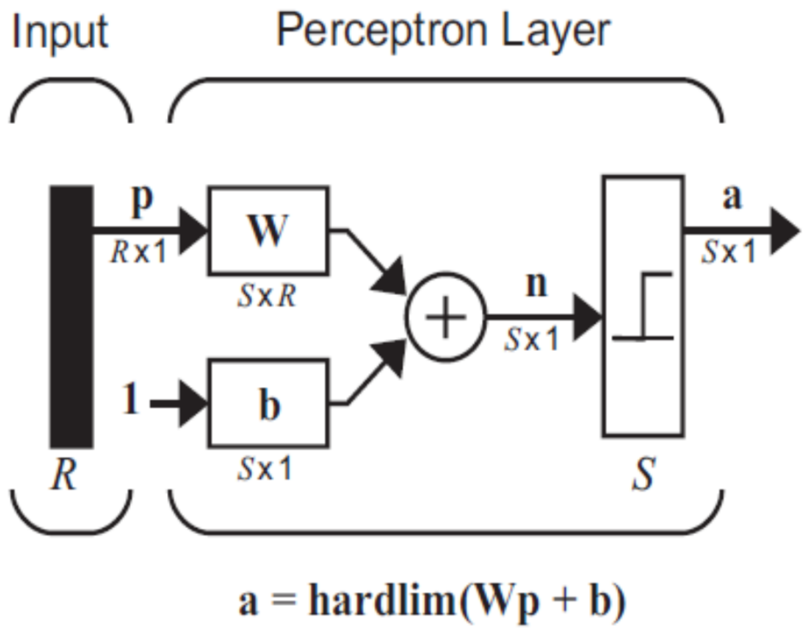


Hard-Limit Transfer Function

- A perceptron neuron, which uses the hard-limit transfer function **`hardlim`**

Perceptron Architecture



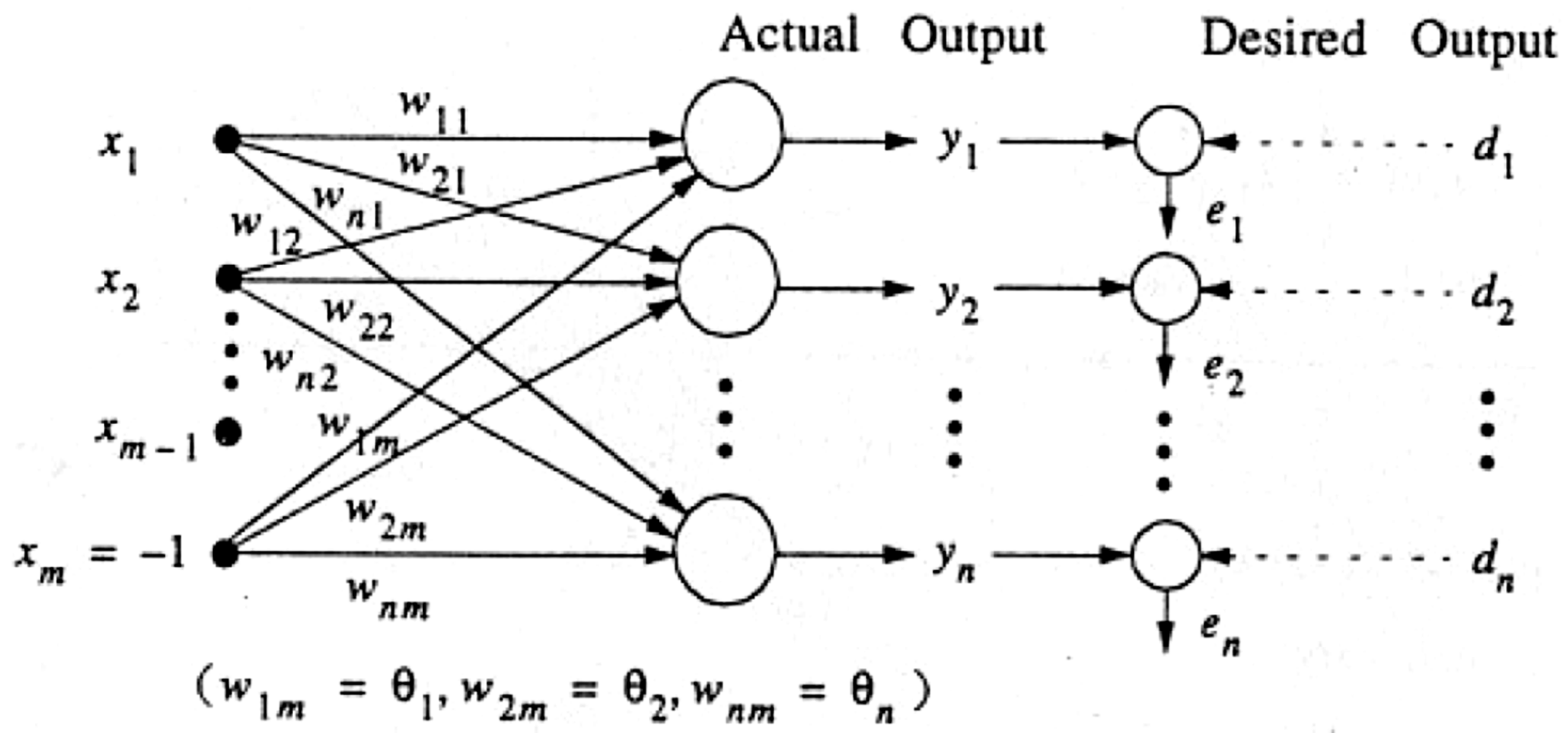


Where

R = number of elements in input

S = number of neurons in layer

Perceptron architecture



$$y_i^{(k)} = a\left(\boldsymbol{w}_i^T \boldsymbol{x}^{(k)}\right) = a\left(\sum_{j=1}^{m} w_{ij} x_j^{(k)}\right) = d_i^{(k)} \quad i = 1,2,\ldots,n; \quad k = 1,2,\ldots,p$$

Perceptron learning rules

- Simple perceptrons with linear threshold units (LTUs) -> *corresponding perceptron learning rule*
- Simple perceptrons with linear graded units (LGUs) -> corresponding *Widrow-Hoff learning rule.*

Perceptron Learning Rule

$$y_i^{(k)} = \operatorname{sgn}\left(\boldsymbol{w}_i^T \boldsymbol{x}^{(k)}\right) = \operatorname{sgn}\left(\sum_{j=1}^{m} w_{ij} x_j^{(k)}\right) = d_i^{(k)} \quad i = 1,2,\dots,n; \quad k = 1,2,\dots,p$$

Learning signal (general weight learning rule)

$$r \triangleq d_i - y_i$$

y_i - actual output
d_i - desired output

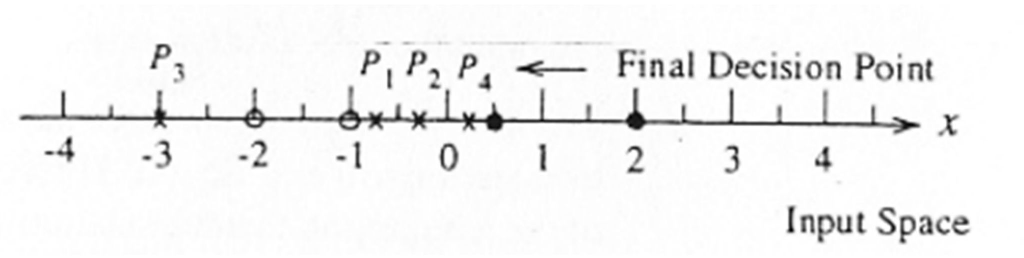
Since the desired output d_i takes the values ± 1 we have

$$\Delta w_{ij} = \eta\left[d_i - \operatorname{sgn}\left(\boldsymbol{w}_i^T \boldsymbol{x}^{(k)}\right)\right] x_j = \begin{cases} 2\eta d_i x_j & \text{if } y_i \neq d_i \\ 0, & \text{otherwise} \end{cases} \quad \text{for } \ j = 1,2,\dots,m$$

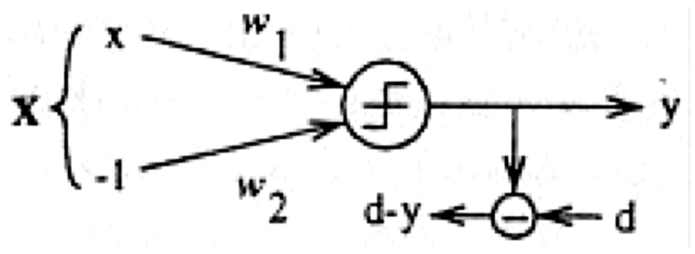
Example

Class 1: $x^{(1)} = 0.5; x^{(3)} = 2; \quad d^{(1)} = d^{(3)} = +1$

Class 2: $x^{(2)} = -1; x^{(4)} = -2; \quad d^{(2)} = d^{(4)} = -1$



$$x^{(1)} = \begin{pmatrix} x^{(1)} \\ -1 \end{pmatrix} = \begin{pmatrix} 0.5 \\ -1 \end{pmatrix}; \quad x^{(2)} = \begin{pmatrix} x^{(2)} \\ -1 \end{pmatrix} = \begin{pmatrix} -1 \\ -1 \end{pmatrix}; \quad x^{(3)} = \begin{pmatrix} 2 \\ -1 \end{pmatrix}; \quad x^{(4)} = \begin{pmatrix} -2 \\ -1 \end{pmatrix}$$



$$\boldsymbol{w}^{(1)} = \begin{pmatrix} -2 \\ 1.5 \end{pmatrix}, \qquad \eta = 0.5,$$

Solution

Step 1

$$y^{(1)} = \operatorname{sgn}\left(\begin{bmatrix}-2 & 1.5\end{bmatrix}\begin{bmatrix}0.5\\-1\end{bmatrix}\right) = -1 \neq d^{(1)}$$

$$\boldsymbol{w}^{(2)} = \boldsymbol{w}^{(1)} + \boldsymbol{x}^{(1)} = \begin{pmatrix}-1.5\\0.5\end{pmatrix}$$

Step 2

$$y^{(2)} = \operatorname{sgn}\left(\begin{bmatrix}-1.5 & 0.5\end{bmatrix}\begin{bmatrix}-1\\-1\end{bmatrix}\right) = 1 \neq d^{(2)}$$

$$\boldsymbol{w}^{(3)} = \boldsymbol{w}^{(2)} - \boldsymbol{x}^{(2)} = \begin{pmatrix}-0.5\\1.5\end{pmatrix}$$

Step 3

$$y^{(3)} = \operatorname{sgn}\left(\begin{bmatrix}-0.5 & 1.5\end{bmatrix}\begin{bmatrix}2\\-1\end{bmatrix}\right) = -1 \neq d^{(3)}$$

$$\boldsymbol{w}^{(4)} = \boldsymbol{w}^{(3)} + \boldsymbol{x}^{(3)} = \begin{pmatrix}1.5\\0.5\end{pmatrix}$$

Step 4

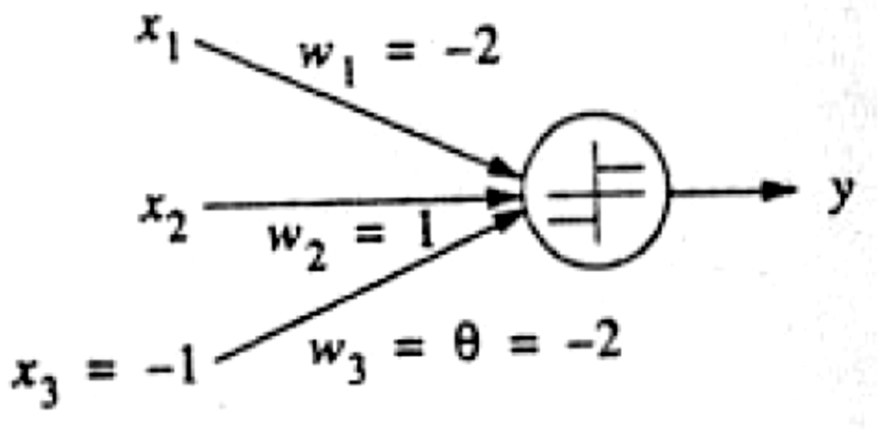
$$y^{(4)} = \operatorname{sgn}\left(\begin{bmatrix}1.5 & 0.5\end{bmatrix}\begin{bmatrix}-2\\-1\end{bmatrix}\right) = -1 = d^{(4)}$$

$$\boldsymbol{w}^{(5)} = \boldsymbol{w}^{(4)}$$

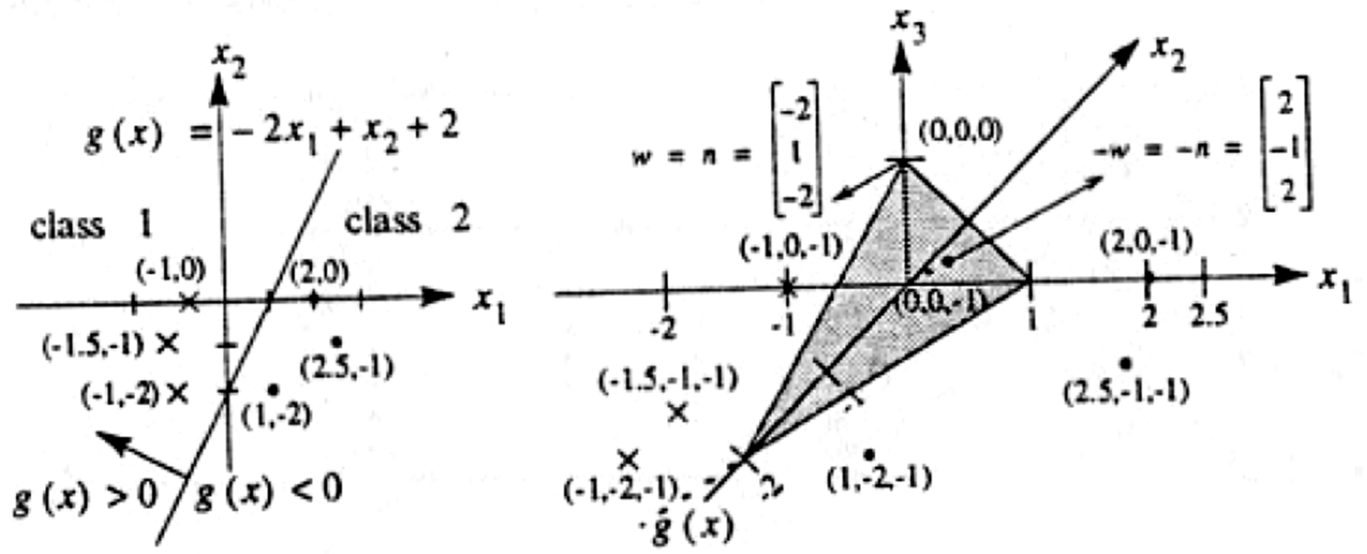
Example

Class 1: $x^{(1)} = \begin{bmatrix} -1 \\ 0 \end{bmatrix}; x^{(2)} = \begin{bmatrix} -1.5 \\ -1 \end{bmatrix}; x^{(3)} = \begin{bmatrix} -1 \\ -2 \end{bmatrix}$ $\quad d^{(1),(2),(3)} = +1$

Class 2: $x^{(4)} = \begin{bmatrix} 2 \\ 0 \end{bmatrix}; x^{(5)} = \begin{bmatrix} 2.5 \\ -1 \end{bmatrix}; x^{(6)} = \begin{bmatrix} 1 \\ -2 \end{bmatrix}$ $\quad d^{(4),(5),(6)} = -1$



Solution



$$g(\boldsymbol{x}) = -2x_1 + x_2 + 2 = 0$$

$$g(\boldsymbol{x}) > 0 \text{ class } 1$$

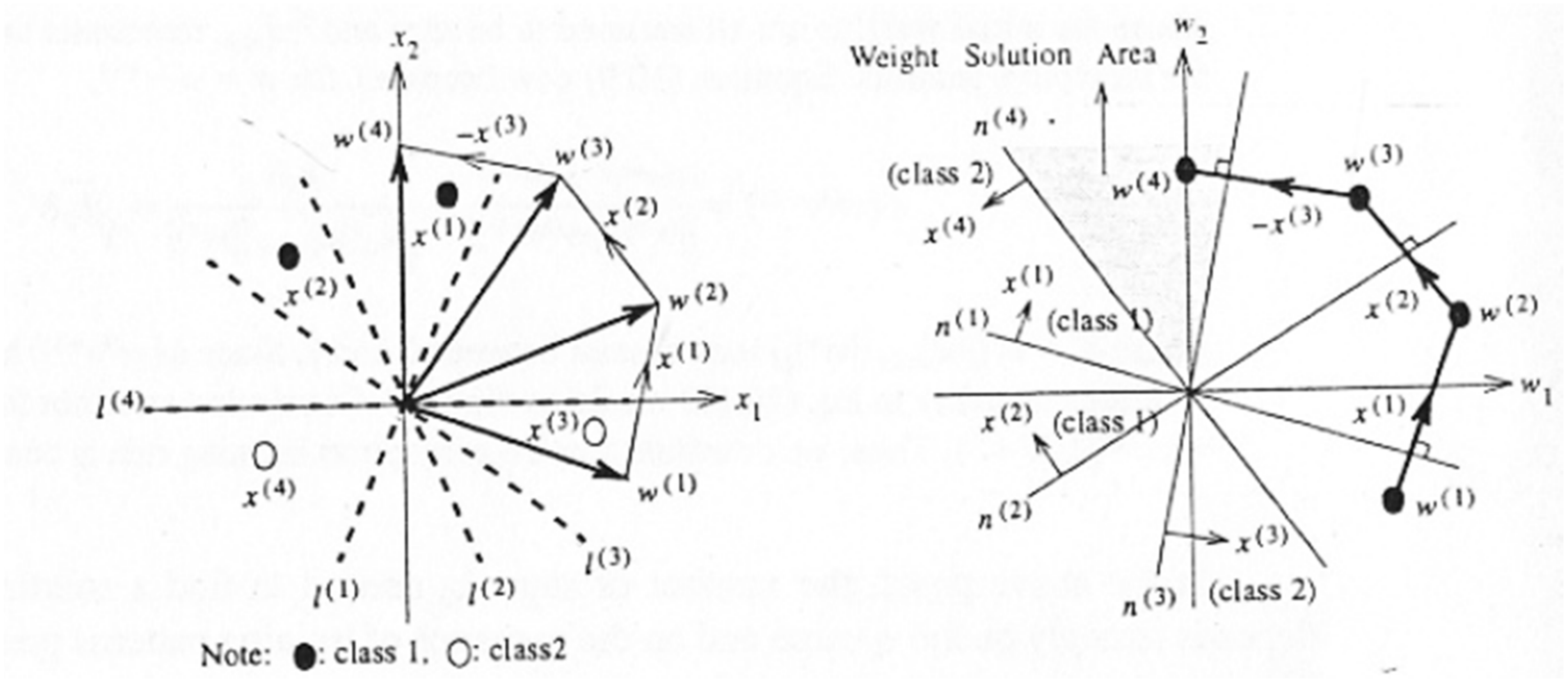
$$g(\boldsymbol{x}) < 0 \text{ class } 2$$

$$y^{(k)} = \operatorname{sgn}(w_1 x_1^{(k)} + w_2 x_2^{(k)} - \theta)$$

Solution

- $w_1 = -1, w_2 = 1, \theta = -2$
- $x^{(1)} = \begin{pmatrix} -1 \\ 0 \\ -1 \end{pmatrix}, x^{(2)} = \begin{pmatrix} -1.5 \\ -1 \\ -1 \end{pmatrix}, x^{(3)} = \begin{pmatrix} -1 \\ -2 \\ -1 \end{pmatrix},$
- $x^{(4)} = \begin{pmatrix} 2 \\ 0 \\ -1 \end{pmatrix}, x^{(5)} = \begin{pmatrix} 2.5 \\ -1 \\ -1 \end{pmatrix}, x^{(6)} = \begin{pmatrix} 1 \\ -2 \\ -1 \end{pmatrix},$

Solution


x_2
$w^{(4)}$
$-x^{(3)}$
$w^{(3)}$
$x^{(1)}$
$x^{(2)}$
$w^{(2)}$
$x^{(2)}$
$x^{(1)}$
$l^{(4)}$
x_1
$x^{(3)}$
$w^{(1)}$
$x^{(4)}$
$l^{(1)}$
$l^{(2)}$
$l^{(3)}$
Note: ●: class 1. ○: class2
Weight Solution Area
w_2
$n^{(4)}$
(class 2)
$x^{(4)}$
$w^{(4)}$
$w^{(3)}$
$-x^{(3)}$
$x^{(1)}$
$n^{(1)}$
(class 1)
$x^{(2)}$
$w^{(2)}$
w_1
$x^{(2)}$
(class 1)
$x^{(1)}$
$n^{(2)}$
$w^{(1)}$
$x^{(3)}$
$n^{(3)}$
(class 2)

Adaline

- Simple perceptrons with linear threshold units
- A network with a single linear unit is called an *Adaline (Adaptive* Linear Element) [Widrow, 1962]

$$y^{(k)} = \boldsymbol{w}^T \boldsymbol{x}^{(k)} = \sum_{j=1}^{m} w_j x_j^{(k)} = d^{(k)}$$

- Cost function which measures the system's performance error by

$$\mathrm{E}(\boldsymbol{w}) = \frac{1}{2}\sum_{k=1}^{p}\left(d^{(k)} - y^{(k)}\right)^2 = \frac{1}{2}\sum_{k=1}^{p}\left(d^{(k)} - \boldsymbol{w}^T \boldsymbol{x}^{(k)}\right)^2 =$$

$$= \frac{1}{2}\sum_{\mathrm{k}=1}^{\mathrm{p}}\left(\mathrm{d}^{(\mathrm{k})} - \sum_{j=1}^{m} w_j x_j^{(k)}\right)^2$$

Adaline

- The usual *gradient-descent algorithm* suggests adjusting each weight w_i by an amount Δw_i proportional to the negative of the gradient of $\mathrm{E}(\boldsymbol{w})$ at the current location:

$$\Delta\boldsymbol{w} = \eta \nabla_{\boldsymbol{w}} E(\boldsymbol{w}) = \eta \frac{\partial E}{\partial w_j} = \eta \sum_{k=1}^{p} (d^{(k)} - \boldsymbol{w}^T \boldsymbol{x}^{(k)}) x_j$$

- If these changes are made individually for each input pattern $x^{(k)}$ in turn, then the change in response to pattern $x^{(k)}$ is simply

$$\Delta w_j = \eta (d^{(k)} - \boldsymbol{w}^T \boldsymbol{x}^{(k)}) x_j^{(k)}$$

- *Adaline learning rule* or the *Widrow-Hoff leaming rule* [Widrow and Hoff, 1960].
- It is also referred to as the *least mean square* (LMS) rule.

Creating a Perceptron (newp)

- A perceptron can be created with the function **`newp`**

  ```
  net = newp(P,T, TF, LF)
  ```

  - P is an R-by-Q matrix of Q input vectors of R elements each.
  - T is an S-by-Q matrix of Q target vectors of S elements each.
  - TF - Transfer function, default = 'hardlim'.
    - The transfer function TF can be HARDLIM or HARDLIMS.
  - LF - Learning function, default = 'learnp‘.
    - The learning function LF can be LEARNP or LEARNPN.

Creating a Perceptron (newp)

- Commonly the hardlim function is used in perceptrons, so it is the default
- Perceptron network with a single two-element input vector and one neuron.

-
  ```
      net = newp([-2 2;-2 2],[0 1]);
  ```

Simulation (sim)

- Perceptron with a single two-element input vector,

```
net = newp([-2 2;-2 +2],[0 1])
```

- This gives zero weights and biases

```
net.IW{1,1}= [-1 1];
net.b{1} = [1];
```

Simulation (sim)

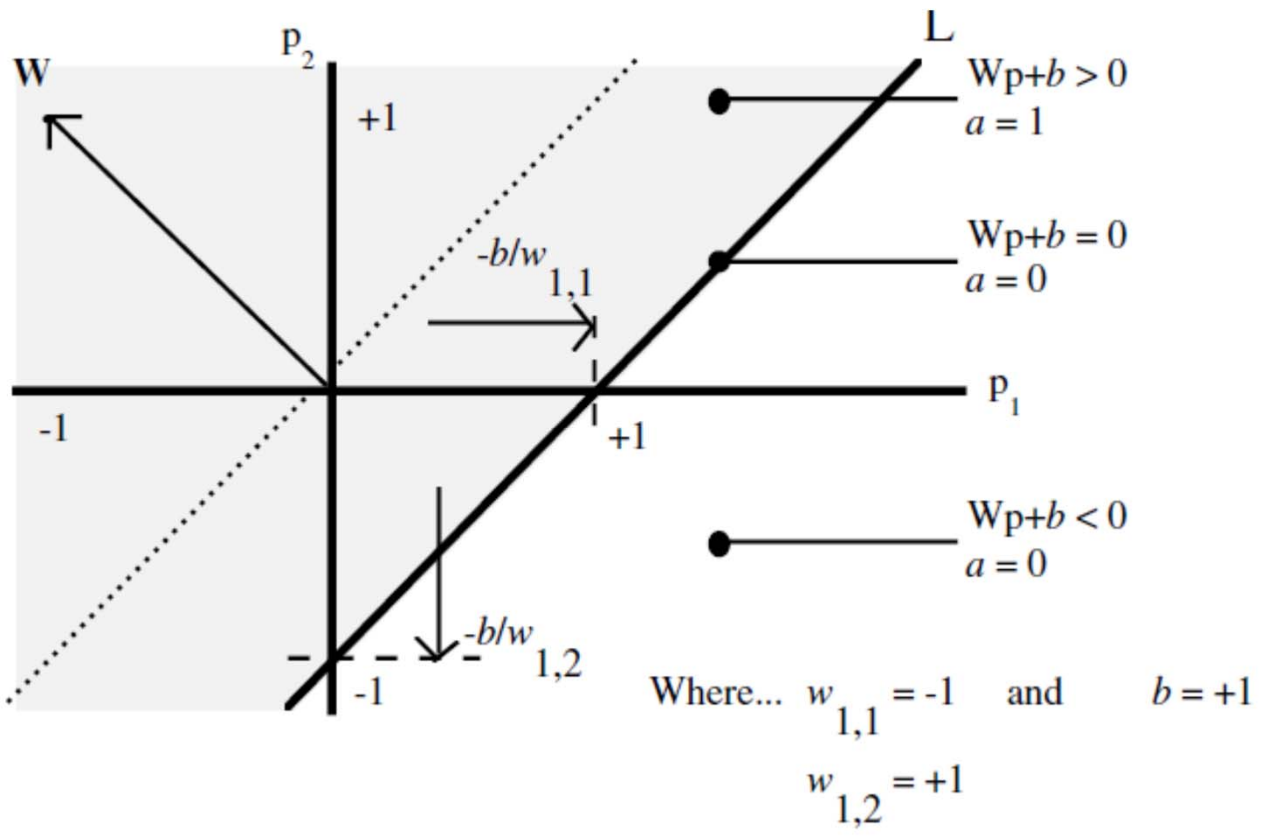
```
p1 = [1;1];
a1 = sim(net,p1)
a1 =
1
```

```
p2 = [1;-1];
a2 = sim(net,p2)
a2 =
0
```

- Two inputs in a sequence and get the outputs in a sequence as well

```
p3 = {[1;1] [1;-1]};
a3 = sim(net,p3)
a3 =
[1] [0]
```

Perceptron clasification



L
$\mathbf{W}$
p_2
+1
$\mathbf{W}\mathbf{p}+b > 0$
$a = 1$
$\mathbf{W}\mathbf{p}+b = 0$
$a = 0$
$-b/w_{1,1}$
p_1
-1
+1
$\mathbf{W}\mathbf{p}+b < 0$
$a = 0$
$-b/w_{1,2}$
-1
Where... $w_{1,1} = -1$ and $b = +1$
$w_{1,2} = +1$

Classification with a 2-input Perceptron

% Each of the five column vectors in P defines a 2-element input vectors and a row vector T defines the vector's target categories. We can plot these vectors with PLOTPV.

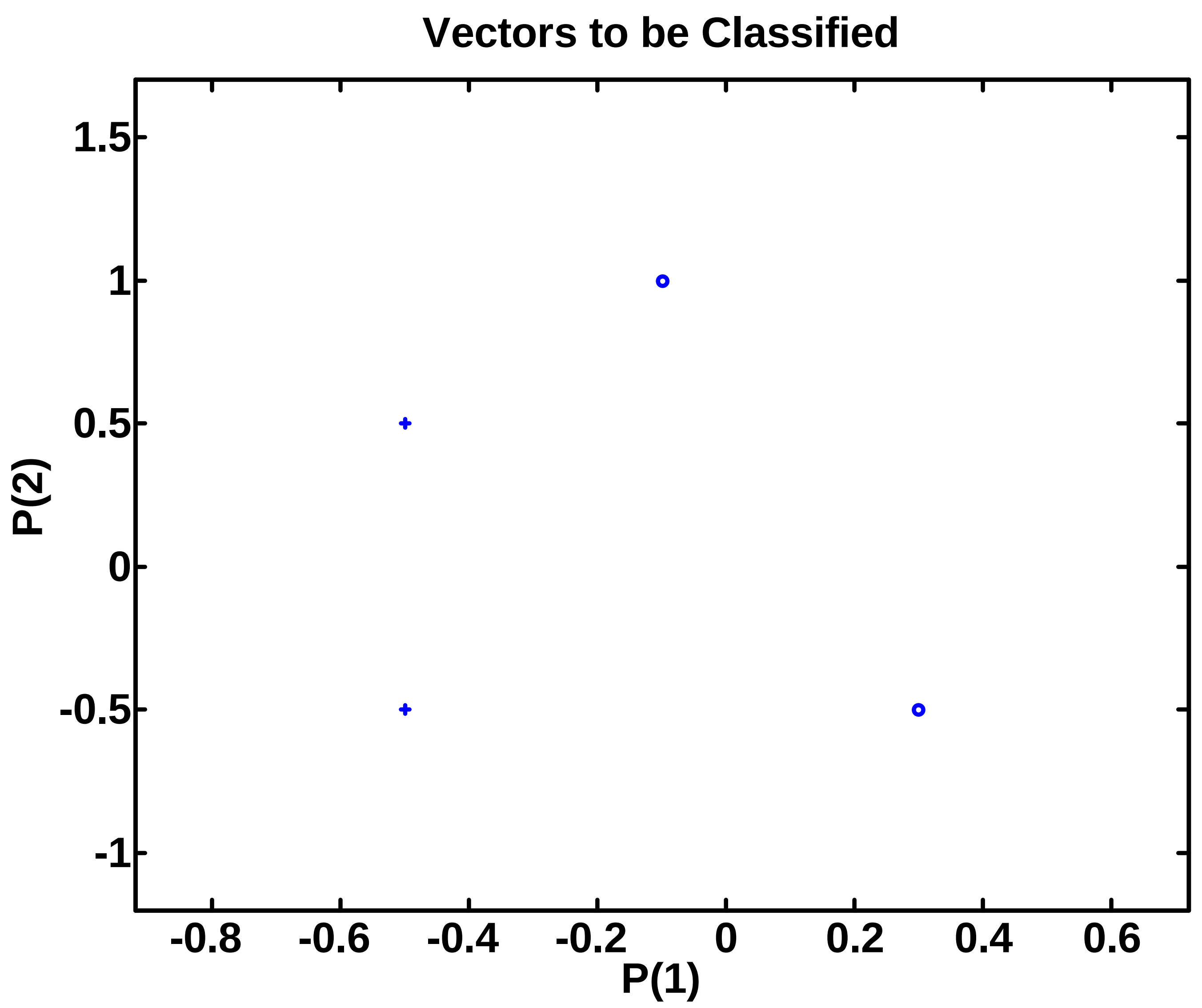
```
P = [ -0.5 -0.5 +0.3 -0.1;  -0.5 +0.5 -0.5 +1.0];
T = [1 1 0 0];
plotpv(P,T);
```

% The perceptron must properly classify the 5 input vectors in P into the two categories defined by T. Perceptrons have HARDLIM neurons. These neurons are capable of separating an input space with a straight line into two categories (0 and 1).

% NEWP creates a network object and configures it as a perceptron. The first argument specifies the expected ranges of two inputs. The second determines that there is only one neuron in the layer.

```
net = newp(P,T);
```

○


Vectors to be Classified
P(1)
P(2)
1.5
1
0.5
0
-0.5
-1
-0.8
-0.6
-0.4
-0.2
0
0.2
0.4
0.6

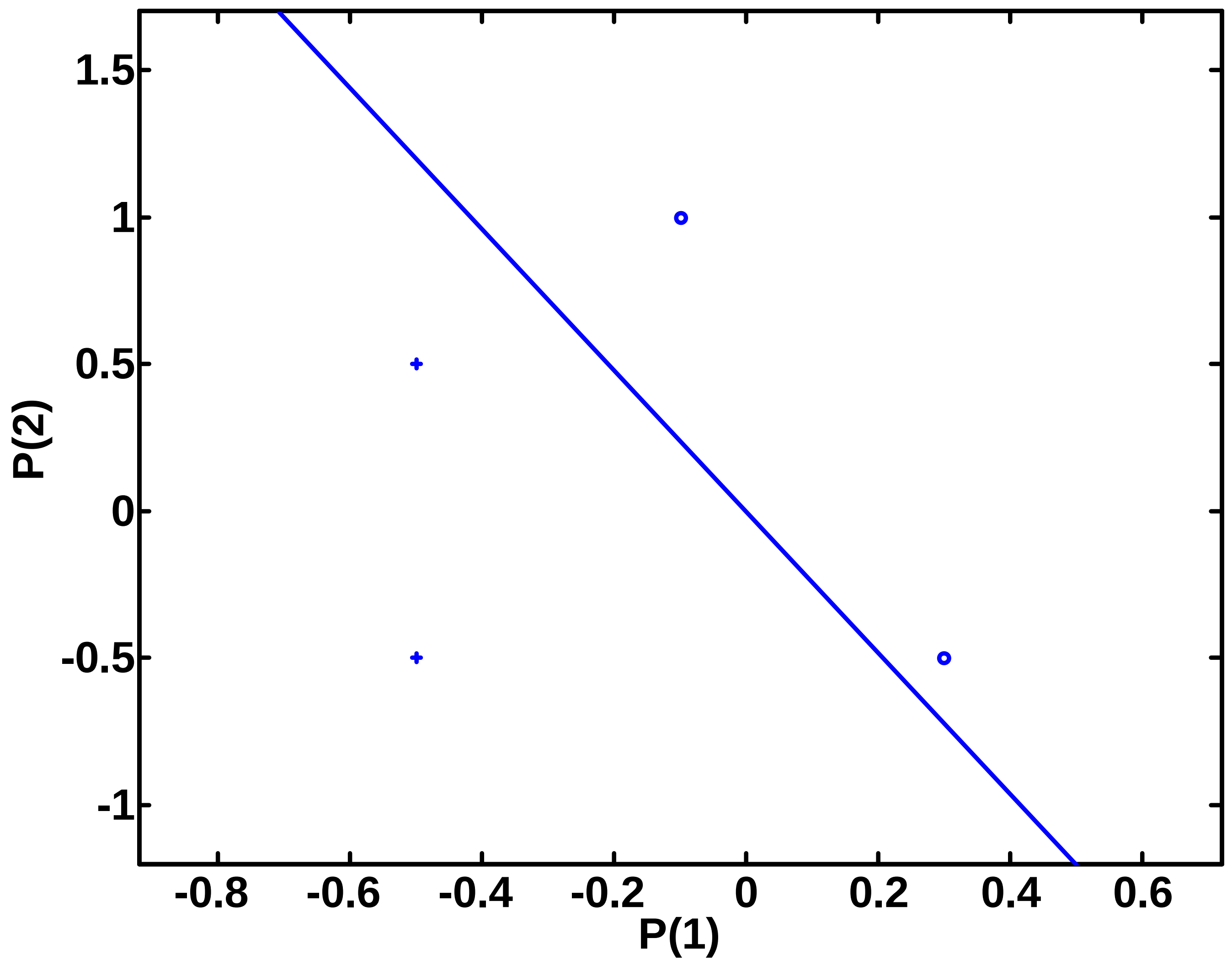
% The input vectors are replotted with the neuron's initial attempt at classification. The initial weights are set to zero, so any input gives the same output and the classification line does not even appear on the plot. Fear not... we are going to train it!

```
plotpv(P,T);
plotpc(net.IW{1},net.b{1});
```

% ADAPT returns a new network object that performs as a better classifier, the network output, and the error.

```
net.adaptParam.passes = 3;
net = adapt(net,P,T);   ili   net = train(net,P,T);
plotpc(net.IW{1},net.b{1});
```

Vectors to be Classified



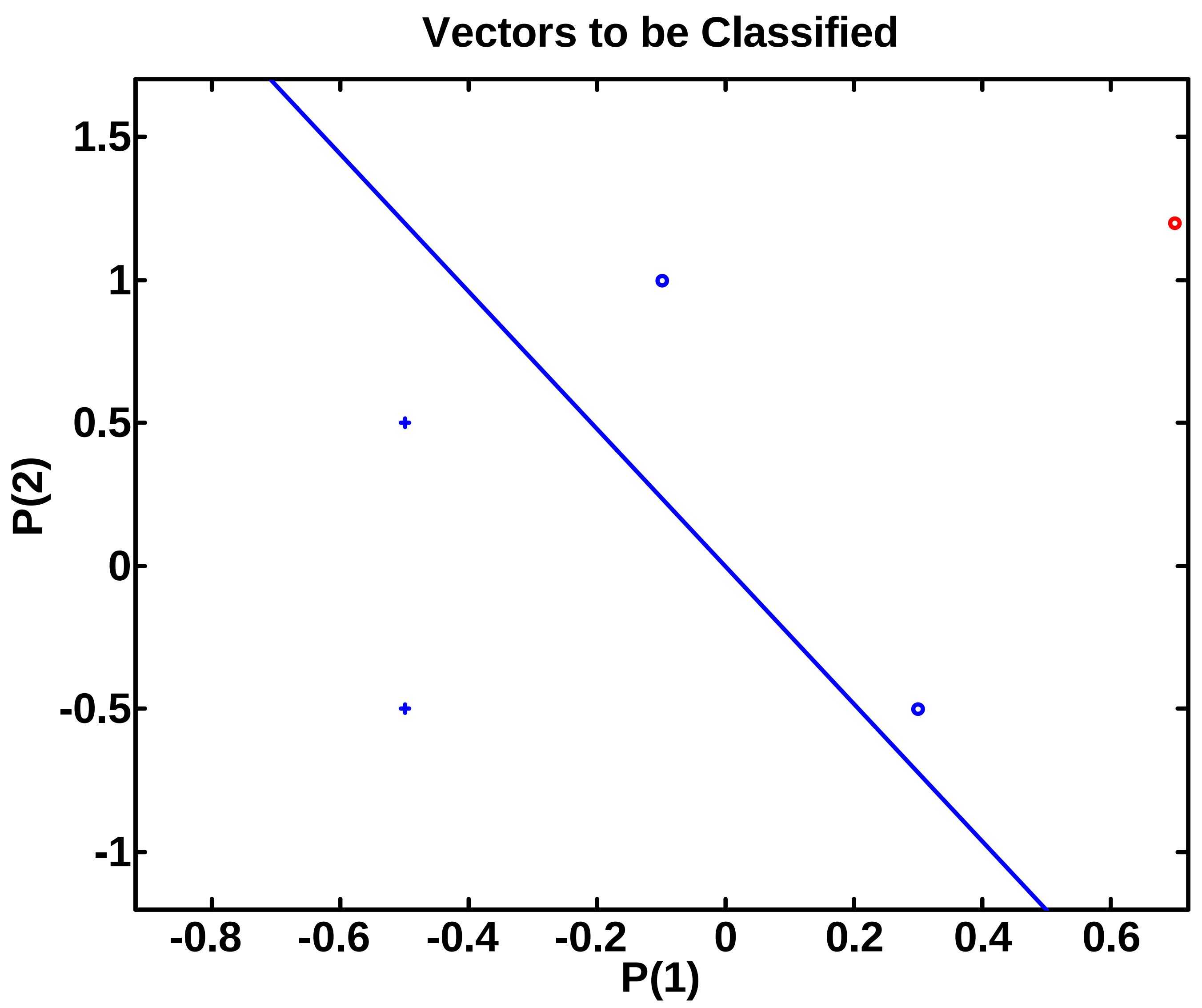
% Now SIM is used to classify any other input vector, like [0.7; 1.2]. A plot of this new point with the original training set shows how the network performs. To distinguish it from the training set, color it red.

```
p = [0.7; 1.2];
a = sim(net,p);
plotpv(p,a);
point = findobj(gca,'type','line');
set(point,'Color','red');
```

% Turn on "hold" so the previous plot is not erased and plot the training set and the classification line.

% The perceptron correctly classified our new point (in red) as category "zero“ (represented by a circle) and not a "one" (represented by a plus).

```
hold on;
plotpv(P,T);
plotpc(net.IW{1},net.b{1});
hold off;
```


Vectors to be Classified
P(2)
P(1)
1.5
1
0.5
0
-0.5
-1
-0.8
-0.6
-0.4
-0.2
0
0.2
0.4
0.6