Perceptron and It's Learning Algorithm

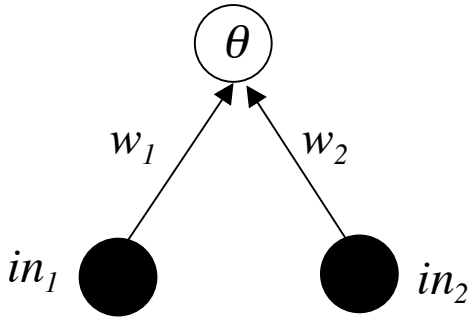
Dr. Mohammad Rashedur Rahman

Perceptron

- Perceptrons had perhaps the most far-reaching impact of any of the early neural networks
- Together with several other researchers, Frank Rosenblatt introduced and developed a large class of artificial networks called perceptrons.
- The pereptron learning rule uses and iterative weight adjustment that is more powerful than the Hebb rule.
- Perceptron learning can be proved to converge to the correct weights if there are weights that will solve the problem at hand

Decision Boundaries in Two Dimensions

For simple logic gate problems, it is easy to visualise what the neural network is doing. It is forming ***decision boundaries*** between classes. Remember, the network output is:



$$out = step\left(w_1 in_1 + w_2 in_2 - \theta\right)$$

The decision boundary (between $out = 0$ and $out = 1$) is at

$$w_1 in_1 + w_2 in_2 - \theta = 0$$

i.e. along the straight line:

$$in_2 = \left(\frac{-w_1}{w_2}\right) in_1 + \left(\frac{\theta}{w_2}\right)$$

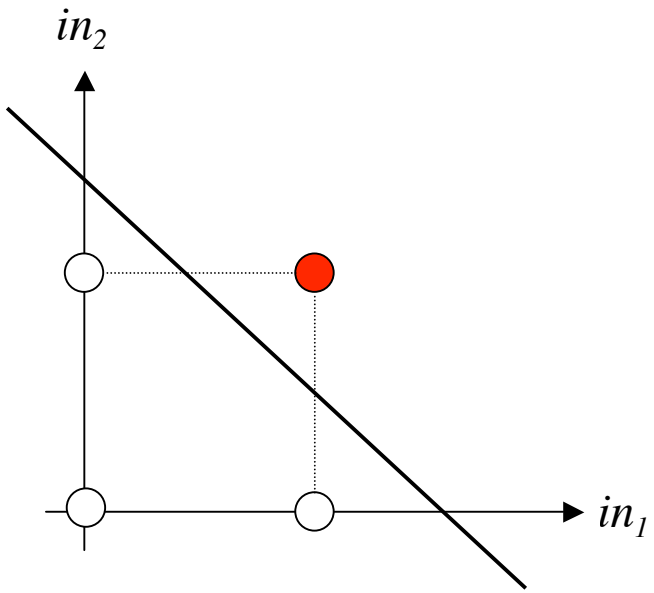
So, in two dimensions the decision boundaries are always straight lines.

Decision Boundaries for AND and OR

We can easily plot the decision boundaries we found by inspection last lecture:

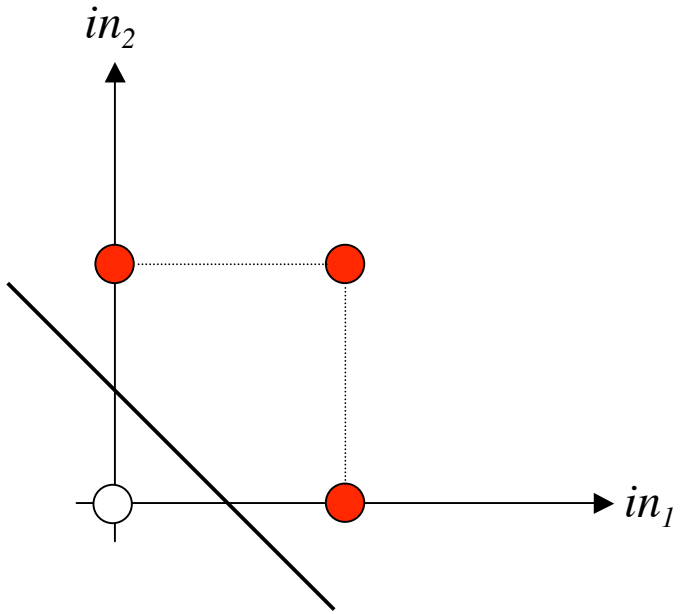
AND

$w_1 = 1,\ w_2 = 1,\ \theta = 1.5$



OR

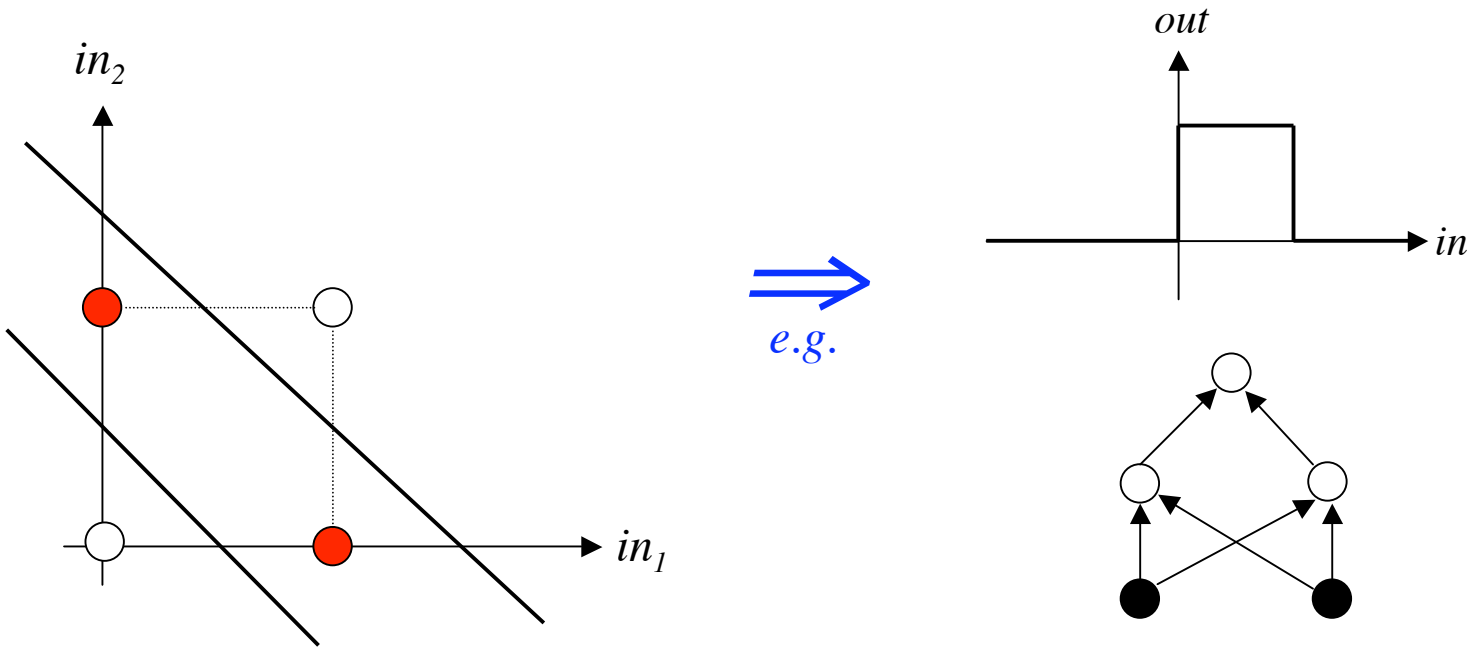
$w_1 = 1,\ w_2 = 1,\ \theta = 0.5$



The extent to which we can change the weights and thresholds without changing the output decisions is now clear.

Decision Boundaries for XOR

The difficulty in dealing with XOR is beginning to look obvious. We need two straight lines to separate the different outputs/decisions:



There are two obvious remedies: either change the transfer function so that it has more than one decision boundary, or use a more complex network that is able to generate more complex decision boundaries.

Decision Hyperplanes and Linear Separability

If we have two inputs, then the weights define a decision boundary that is a one dimensional straight line in the two dimensional ***input space*** of possible input values.

If we have n inputs, the weights define a decision boundary that is an n–1 dimensional ***hyperplane*** in the n dimensional input space:

$$w_1 in_1 + w_2 in_2 + \ldots + w_n in_n - \theta = 0$$

This hyperplane is clearly still linear (i.e., straight or flat or non-curved) and can still only divide the space into two regions. We still need more complex transfer functions, or more complex networks, to deal with XOR type problems.

Problems with input patterns that can be classified using a single hyperplane are said to be ***linearly separable***. Problems (such as XOR) which cannot be classified in this way are said to be ***non-linearly separable***.

Training a Neural Network

Whether the neural network is a simple Perceptron, or a much more complicated multi-layer network with special activation functions, we need to develop a systematic procedure for determining appropriate connection weights.

The general procedure is to have the network ***learn*** the appropriate weights from a representative set of training data.

For all but the simplest cases, however, direct computation of the weights is intractable.

Instead, a good all-purpose process is to start off with ***random initial weights*** and adjust them in small steps until the required outputs are produced.

We shall first look at a brute force derivation of such an ***iterative learning algorithm*** for simple Perceptrons. Then, in later lectures, we shall see how more powerful and general techniques can easily lead to learning algorithms which will work for neural networks of any specification we could possibly dream up.

Perceptron Learning

For simple Perceptrons performing classification, we have seen that the decision boundaries are hyperplanes, and we can think of ***learning*** as the process of shifting around the hyperplanes until each training pattern is classified correctly.

Somehow, we need to formalise that process of "shifting around" into a systematic ***algorithm*** that can easily be implemented on a computer.

The "shifting around" can conveniently be split up into a number of small steps.

If the network weights at time t are $w_{ij}(t)$, then the shifting process corresponds to moving them by a small amount $\Delta w_{ij}(t)$ so that at time $t+1$ we have weights

$$w_{ij}(t+1) = w_{ij}(t) + \Delta w_{ij}(t)$$

It is convenient to treat the thresholds as weights, as discussed previously, so we don't need separate equations for them.

Formulating the Weight Changes

Suppose the target output of unit j is $targ_j$ and the actual output is $out_j = \text{step}(\sum in_i w_{ij})$, where in_i are the activations of the previous layer of neurons (i.e. the network inputs for a Perceptron). Then we can just go through all the possibilities to work out an appropriate set of small weight changes, and put them into a common form:

If $out_j = targ_j$ do nothing Note $targ_j - out_j = 0$

so $w_{ij} \rightarrow w_{ij}$

If $out_j = 1$ and $targ_j = 0$ Note $targ_j - out_j = -1$

then $\sum in_i w_{ij}$ is too large

first when $in_i = 1$ decrease w_{ij}

so $w_{ij} \rightarrow w_{ij} - \eta = w_{ij} - \eta\, in_i$

and when $in_i = 0$ w_{ij} doesn't matter

so $w_{ij} \rightarrow w_{ij} - 0 = w_{ij} - \eta\, in_i$

so $w_{ij} \rightarrow w_{ij} - \eta\, in_i$

If $out_j = 0$ and $targ_j = 1$ Note $targ_j - out_j = 1$

then $\sum in_i w_{ij}$ is too small

first when $in_i = 1$ increase w_{ij}

so $w_{ij} \rightarrow w_{ij} + \eta = w_{ij} + \eta\, in_i$

and when $in_i = 0$ w_{ij} doesn't matter

so $w_{ij} \rightarrow w_{ij} - 0 = w_{ij} + \eta\, in_i$

so $w_{ij} \rightarrow w_{ij} + \eta\, in_i$

It has become clear that each case can be written in the form:

$$w_{ij} \rightarrow w_{ij} + \eta\, (targ_j - out_j)\, in_i$$

$$\Delta w_{ij} = \eta\, (targ_j - out_j)\, in_i$$

This weight update equation is called the ***Perceptron Learning Rule***. The positive parameter η is called the ***learning rate*** or ***step size*** – it determines how smoothly we shift the decision boundaries.

Convergence of Perceptron Learning

The weight changes Δw_{ij} need to be applied repeatedly – for each weight w_{ij} in the network, and for each training pattern in the training set. One pass through all the weights for the whole training set is called one ***epoch*** of training.

Eventually, usually after many epochs, when all the network outputs match the targets for all the training patterns, all the Δw_{ij} will be zero and the process of training will cease. We then say that the training process has ***converged*** to a solution.

It is possible to prove that if there does exist a possible set of weights for a Perceptron which solves the given problem correctly, then the Perceptron Learning Rule will find them in a finite number of iterations.

Moreover, it can be shown that if a problem is linearly separable, then the Perceptron Learning Rule will find a set of weights in a finite number of iterations that solves the problem correctly.

4 Multiple-Neuron Perceptron

Each neuron will have its own decision boundary.

$$_i\mathbf{w}^T\mathbf{p} + b_i = 0$$

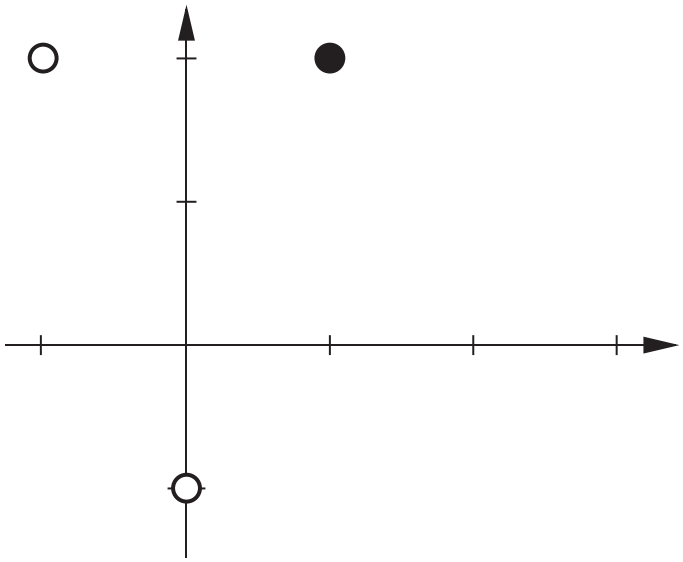
A single neuron can classify input vectors into two categories.

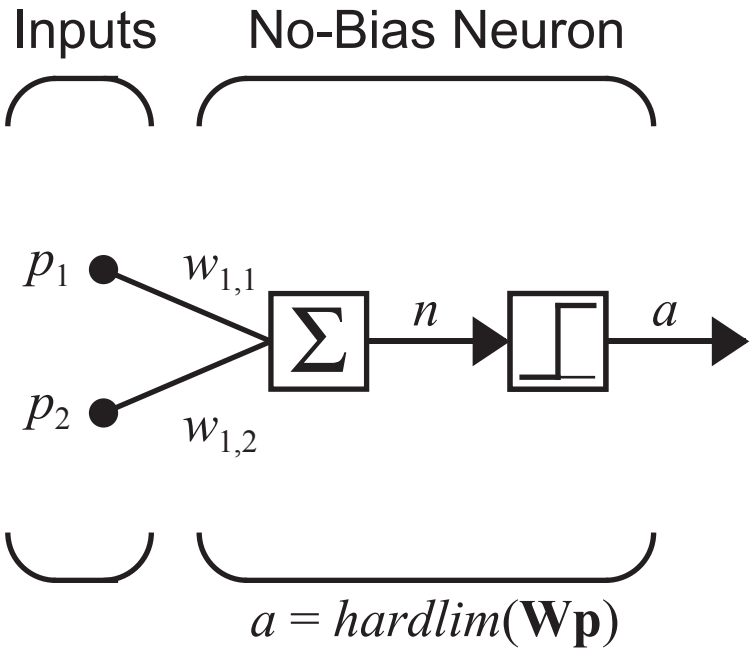
A multi-neuron perceptron can classify input vectors into 2^S categories.

Learning Rule Test Problem

$$\{\mathbf{p}_1, \mathbf{t}_1\}, \{\mathbf{p}_2, \mathbf{t}_2\}, \ldots, \{\mathbf{p}_Q, \mathbf{t}_Q\}$$

$$\left\{\mathbf{p}_1 = \begin{bmatrix} 1 \\ 2 \end{bmatrix}, t_1 = 1\right\} \quad \left\{\mathbf{p}_2 = \begin{bmatrix} -1 \\ 2 \end{bmatrix}, t_2 = 0\right\} \quad \left\{\mathbf{p}_3 = \begin{bmatrix} 0 \\ -1 \end{bmatrix}, t_3 = 0\right\}$$

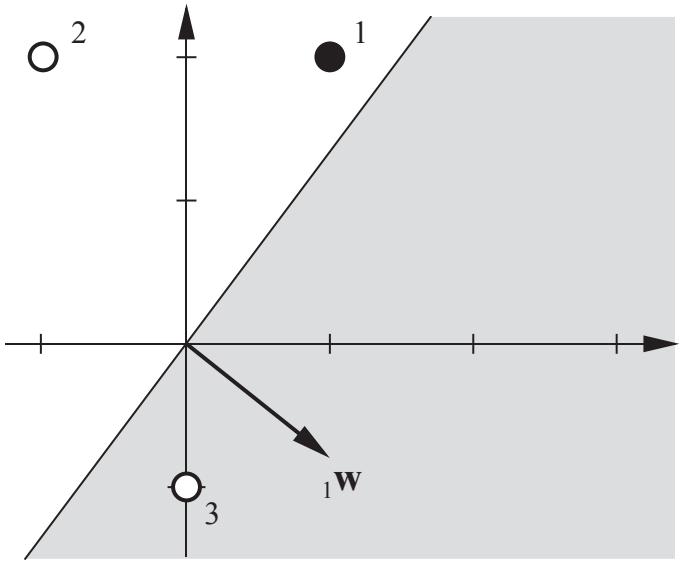




Starting Point

Random initial weight:

$$ {}_1\mathbf{w} = \begin{bmatrix} 1.0 \\ -0.8 \end{bmatrix} $$



Present $\mathbf{p}_1$ to the network:

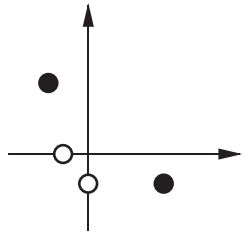
$$ a = hardlim({}_1\mathbf{w}^T\mathbf{p}_1) = hardlim\left(\begin{bmatrix} 1.0 & -0.8 \end{bmatrix} \begin{bmatrix} 1 \\ 2 \end{bmatrix} \right) $$

$$ a = hardlim(-0.6) = 0 $$

Incorrect Classification.

Tentative Learning Rule

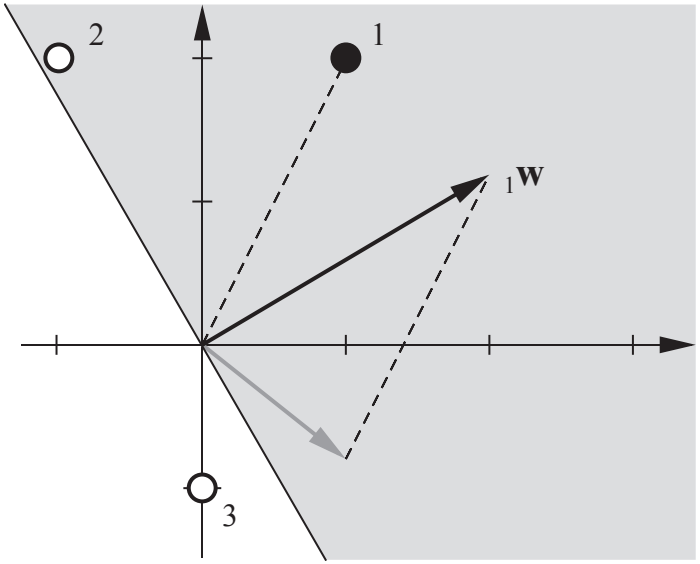
- Set ${}_1\mathbf{w}$ to $\mathbf{p}_1$
  - Not stable ×



- Add $\mathbf{p}_1$ to ${}_1\mathbf{w}$ ✓

Tentative Rule: If $t = 1$ and $a = 0$, then ${}_1\mathbf{w}^{new} = {}_1\mathbf{w}^{old} + \mathbf{p}$

$${}_1\mathbf{w}^{new} = {}_1\mathbf{w}^{old} + \mathbf{p}_1 = \begin{bmatrix} 1.0 \\ -0.8 \end{bmatrix} + \begin{bmatrix} 1 \\ 2 \end{bmatrix} = \begin{bmatrix} 2.0 \\ 1.2 \end{bmatrix}$$



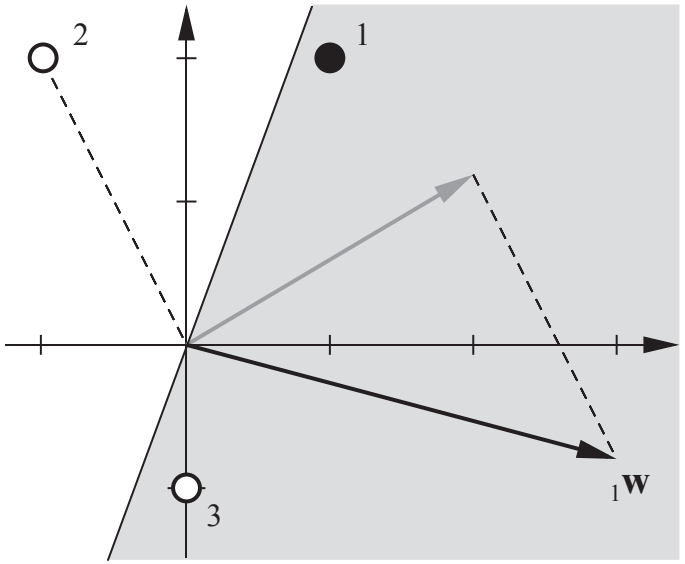
Second Input Vector

$$a = hardlim({}_1\mathbf{w}^{\mathrm{T}}\mathbf{p}_2) = hardlim\left(\begin{bmatrix}2.0 & 1.2\end{bmatrix}\begin{bmatrix}-1\\2\end{bmatrix}\right)$$

$$a = hardlim(0.4) = 1 \quad \text{(Incorrect Classification)}$$

Modification to Rule: If $t = 0$ and $a = 1$, then ${}_1\mathbf{w}^{new} = {}_1\mathbf{w}^{old} - \mathbf{p}$

$${}_1\mathbf{w}^{new} = {}_1\mathbf{w}^{old} - \mathbf{p}_2 = \begin{bmatrix}2.0\\1.2\end{bmatrix} - \begin{bmatrix}-1\\2\end{bmatrix} = \begin{bmatrix}3.0\\-0.8\end{bmatrix}$$

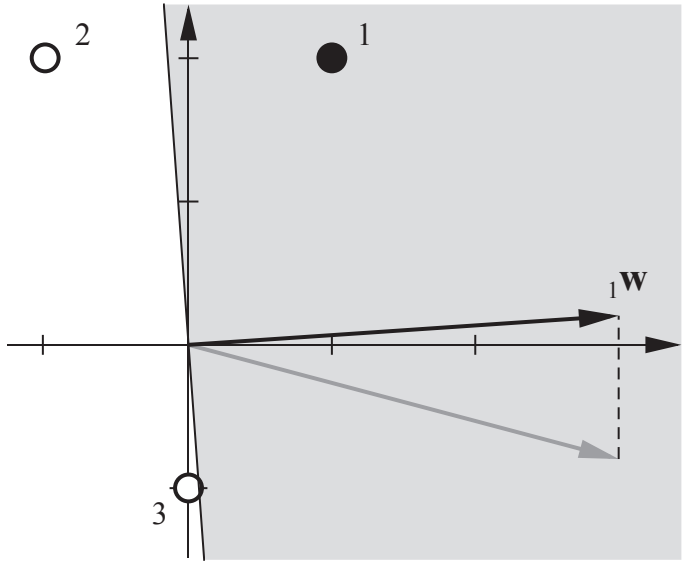


Third Input Vector

$$a = hardlim({}_1\mathbf{w}^T \mathbf{p}_3) = hardlim\left(\begin{bmatrix}3.0 & -0.8\end{bmatrix}\begin{bmatrix}0\\-1\end{bmatrix}\right)$$

$$a = hardlim(0.8) = 1 \qquad \text{(Incorrect Classification)}$$

$${}_1\mathbf{w}^{new} = {}_1\mathbf{w}^{old} - \mathbf{p}_3 = \begin{bmatrix}3.0\\-0.8\end{bmatrix} - \begin{bmatrix}0\\-1\end{bmatrix} = \begin{bmatrix}3.0\\0.2\end{bmatrix}$$



Patterns are now correctly classified.

If $t = a$, then ${}_1\mathbf{w}^{new} = {}_1\mathbf{w}^{old}$.

Unified Learning Rule

If $t = 1$ and $a = 0$, then ${}_1\mathbf{w}^{new} = {}_1\mathbf{w}^{old} + \mathbf{p}$

If $t = 0$ and $a = 1$, then ${}_1\mathbf{w}^{new} = {}_1\mathbf{w}^{old} - \mathbf{p}$

If $t = a$, then ${}_1\mathbf{w}^{new} = {}_1\mathbf{w}^{old}$

$$e = t - a$$

If $e = 1$, then ${}_1\mathbf{w}^{new} = {}_1\mathbf{w}^{old} + \mathbf{p}$

If $e = -1$, then ${}_1\mathbf{w}^{new} = {}_1\mathbf{w}^{old} - \mathbf{p}$

If $e = 0$, then ${}_1\mathbf{w}^{new} = {}_1\mathbf{w}^{old}$

$$ {}_1\mathbf{w}^{new} = {}_1\mathbf{w}^{old} + e\mathbf{p} = {}_1\mathbf{w}^{old} + (t-a)\mathbf{p}$$

$$b^{new} = b^{old} + e$$

A bias is a weight with an input of 1.

Multiple-Neuron Perceptrons

To update the ith row of the weight matrix:

$$ {}_i\mathbf{w}^{new} = {}_i\mathbf{w}^{old} + e_i\mathbf{p} $$

$$ b_i^{new} = b_i^{old} + e_i $$

Matrix form:

$$ \mathbf{W}^{new} = \mathbf{W}^{old} + \mathbf{e}\mathbf{p}^T $$

$$ \mathbf{b}^{new} = \mathbf{b}^{old} + \mathbf{e} $$

Now we find the bias values for each perceptron by picking a point on the decision boundary and satisfying Eq. (4.15).

$$
\begin{aligned}
{}_1\mathbf{w}^T\mathbf{p} + b &= 0 \\
b &= -{}_1\mathbf{w}^T\mathbf{p}
\end{aligned}
$$

This gives us the following three biases:

$$(a)\ b = -\begin{bmatrix}-2 & 1\end{bmatrix}\begin{bmatrix}0\\0\end{bmatrix} = 0, (b)\ b = -\begin{bmatrix}0 & -2\end{bmatrix}\begin{bmatrix}0\\-1\end{bmatrix} = -2, (c)\ b = -\begin{bmatrix}2 & -2\end{bmatrix}\begin{bmatrix}-2\\1\end{bmatrix} = 6$$

We can now check our solution against the original points. Here we test the first network on the input vector $\mathbf{p} = \begin{bmatrix}-2 & 2\end{bmatrix}^T$.

$$
\begin{aligned}
a &= hardlim({}_1\mathbf{w}^T\mathbf{p} + b) \\
&= hardlim\left(\begin{bmatrix}-2 & 1\end{bmatrix}\begin{bmatrix}-2\\2\end{bmatrix} + 0\right) \\
&= hardlim(6) \\
&= 1
\end{aligned}
$$

We can use MATLAB to automate the testing process and to try new points. Here the first network is used to classify a point that was not in the original problem.

```
w=[-2 1]; b = 0;
a = hardlim(w*[1;1]+b)
a =
       0
```

P4.2 Convert the classification problem defined below into an equivalent problem definition consisting of inequalities constraining weight and bias values.

$$\left\{\mathbf{p}_1 = \begin{bmatrix}0\\2\end{bmatrix}, t_1 = 1\right\}\left\{\mathbf{p}_2 = \begin{bmatrix}1\\0\end{bmatrix}, t_2 = 1\right\}\left\{\mathbf{p}_3 = \begin{bmatrix}0\\-2\end{bmatrix}, t_3 = 0\right\}\left\{\mathbf{p}_4 = \begin{bmatrix}2\\0\end{bmatrix}, t_4 = 0\right\}$$

Each target t_i indicates whether or not the net input in response to $\mathbf{p}_i$ must be less than 0, or greater than or equal to 0. For example, since t_1 is 1, we know that the net input corresponding to $\mathbf{p}_1$ must be greater than or equal to 0. Thus we get the following inequality:

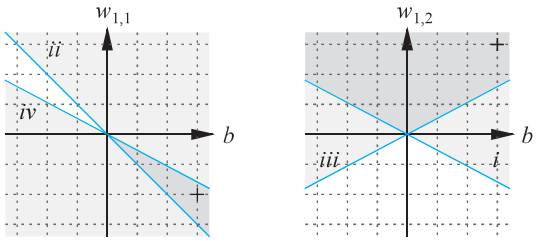
$$\begin{aligned} \mathbf{W}\mathbf{p}_1 + b &\geq 0 \\ 0w_{1,1} + 2w_{1,2} + b &\geq 0 \\ 2w_{1,2} + b &\geq 0. \end{aligned}$$

Applying the same procedure to the input/target pairs for $\{\mathbf{p}_2, t_2\}$, $\{\mathbf{p}_3, t_3\}$ and $\{\mathbf{p}_4, t_4\}$ results in the following set of inequalities.

$$\begin{aligned} 2w_{1,2} + b &\geq 0 \quad (i) \\ w_{1,1} + b &\geq 0 \quad (ii) \\ -2w_{1,2} + b &< 0 \quad (iii) \\ 2w_{1,1} + b &< 0 \quad (iv) \end{aligned}$$

Solving a set of inequalities is more difficult than solving a set of equalities. One added complexity is that there are often an infinite number of solutions (just as there are often an infinite number of linear decision boundaries that can solve a linearly separable classification problem).

However, because of the simplicity of this problem, we can solve it by graphing the solution spaces defined by the inequalities. Note that $w_{1,1}$ only appears in inequalities (ii) and (iv), and $w_{1,2}$ only appears in inequalities (i) and (iii). We can plot each pair of inequalities with two graphs.



Any weight and bias values that fall in both dark gray regions will solve the classification problem.

Here is one such solution:

$$\mathbf{W} = \begin{bmatrix} -2 & 3 \end{bmatrix} \qquad b = 3.$$

P4.3 We have a classification problem with four classes of input vector. The four classes are

$$\text{class 1:}\left\{\mathbf{p}_1 = \begin{bmatrix}1\\1\end{bmatrix}, \mathbf{p}_2 = \begin{bmatrix}1\\2\end{bmatrix}\right\}, \text{class 2: } \left\{\mathbf{p}_3 = \begin{bmatrix}2\\-1\end{bmatrix}, \mathbf{p}_4 = \begin{bmatrix}2\\0\end{bmatrix}\right\},$$

$$\text{class 3:}\left\{\mathbf{p}_5 = \begin{bmatrix}-1\\2\end{bmatrix}, \mathbf{p}_6 = \begin{bmatrix}-2\\1\end{bmatrix}\right\}, \text{class 4: } \left\{\mathbf{p}_7 = \begin{bmatrix}-1\\-1\end{bmatrix}, \mathbf{p}_8 = \begin{bmatrix}-2\\-2\end{bmatrix}\right\}.$$

Design a perceptron network to solve this problem.

To solve a problem with four classes of input vector we will need a perceptron with at least two neurons, since an S-neuron perceptron can categorize 2^S classes. The two-neuron perceptron is shown in Figure P4.2.

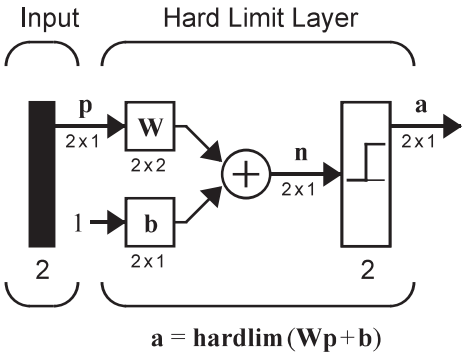


Figure P4.2 Two-Neuron Perceptron

Let's begin by displaying the input vectors, as in Figure P4.3. The light circles ○ indicate class 1 vectors, the light squares □ indicate class 2 vectors, the dark circles ● indicate class 3 vectors, and the dark squares ■ indicate class 4 vectors.

A two-neuron perceptron creates two decision boundaries. Therefore, to divide the input space into the four categories, we need to have one decision boundary divide the four classes into two sets of two. The remaining boundary must then isolate each class. Two such boundaries are illustrated in Figure P4.4. We now know that our patterns are linearly separable.

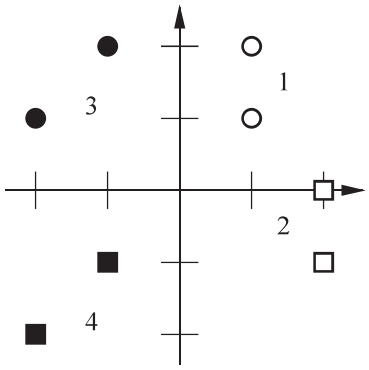


Figure P4.3 Input Vectors for Problem P4.3

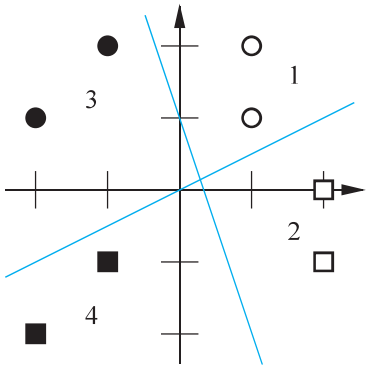


Figure P4.4 Tentative Decision Boundaries for Problem P4.3

The weight vectors should be orthogonal to the decision boundaries and should point toward the regions where the neuron outputs are 1. The next step is to decide which side of each boundary should produce a 1. One choice is illustrated in Figure P4.5, where the shaded areas represent outputs of 1. The darkest shading indicates that both neuron outputs are 1. Note that this solution corresponds to target values of

$$\text{class 1:}\left\{\mathbf{t}_1 = \begin{bmatrix}0\\0\end{bmatrix}, \mathbf{t}_2 = \begin{bmatrix}0\\0\end{bmatrix}\right\}, \text{class 2: } \left\{\mathbf{t}_3 = \begin{bmatrix}0\\1\end{bmatrix}, \mathbf{t}_4 = \begin{bmatrix}0\\1\end{bmatrix}\right\},$$

$$\text{class 3:}\left\{\mathbf{t}_5 = \begin{bmatrix}1\\0\end{bmatrix}, \mathbf{t}_6 = \begin{bmatrix}1\\0\end{bmatrix}\right\}, \text{class 4: } \left\{\mathbf{t}_7 = \begin{bmatrix}1\\1\end{bmatrix}, \mathbf{t}_8 = \begin{bmatrix}1\\1\end{bmatrix}\right\}.$$

We can now select the weight vectors:

$$ {}_1\mathbf{w} = \begin{bmatrix} -3 \\ -1 \end{bmatrix} \text{ and } {}_2\mathbf{w} = \begin{bmatrix} 1 \\ -2 \end{bmatrix}. $$

Note that the lengths of the weight vectors is not important, only their directions. They must be orthogonal to the decision boundaries. Now we can calculate the bias by picking a point on a boundary and satisfying Eq. (4.15):

$$ b_1 = -{}_1\mathbf{w}^T\mathbf{p} = -\begin{bmatrix} -3 & -1 \end{bmatrix}\begin{bmatrix} 0 \\ 1 \end{bmatrix} = 1, $$

$$ b_2 = -{}_2\mathbf{w}^T\mathbf{p} = -\begin{bmatrix} 1 & -2 \end{bmatrix}\begin{bmatrix} 0 \\ 0 \end{bmatrix} = 0. $$

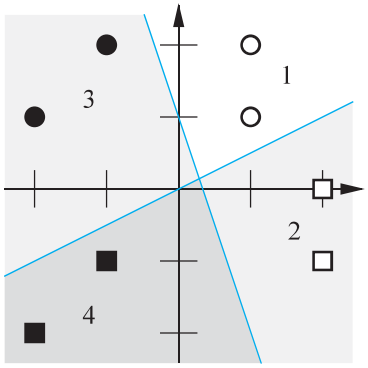


Figure P4.5 Decision Regions for Problem P4.3

In matrix form we have

$$ \mathbf{W} = \begin{bmatrix} {}_1\mathbf{w}^T \\ {}_2\mathbf{w}^T \end{bmatrix} = \begin{bmatrix} -3 & -1 \\ 1 & -2 \end{bmatrix} \text{ and } \mathbf{b} = \begin{bmatrix} 1 \\ 0 \end{bmatrix}, $$

which completes our design.

P4.4 Solve the following classification problem with the perceptron rule. Apply each input vector in order, for as many repetitions as it takes to ensure that the problem is solved. Draw a graph of the problem only after you have found a solution.