POLYTECHNIC UNIVERSITY
Department of Computer and Information Science

Perceptron for Pattern Classification

K. Ming Leung

Abstract: A neural network, known as the perceptron, capable of classifying patterns into two or more categories is introduced. The network is trained using the perceptron learning rule.

Directory

Table of Contents

1. Simple Perceptron for Pattern Classification

We consider here a NN, known as the Perceptron, which is capable of performing pattern classification into two or more categories. The perceptron is trained using the perceptron learning rule. We will first consider classification into two categories and then the general multi-class classification later.

For classification into only two categories, all we need is a single output neuron. Here we will use bipolar neurons. The simplest architecture that could do the job consists of a layer of N input neurons, an output layer with a single output neuron, and no hidden layers. This is the same architecture as we saw before for Hebb learning.

However, we will use a different transfer function here for the output neuron:

$$f_\theta(x) = \begin{cases} +1, & \text{if } x > \theta, \\ 0, & \text{if } -\theta \le x \le \theta, \\ -1, & \text{if } x < -\theta. \end{cases}$$

This transfer function has an undecided band of width 2θ. The value

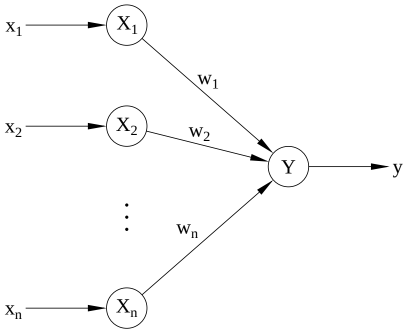

Figure 1: *A feedforward NN having 1 output neuron.*

of θ is held fixed at a relatively small value. The undecided case has an output of 0, exactly half way between 1 and -1.

Thus if we represent the N components of the input vector by $\mathbf{x}$, the N components of the weight vector by $\mathbf{w}$, and the bias by b, the output is then given by

$$y = f_\theta(\mathbf{x} \cdot \mathbf{w} + b).$$

Notice that because of the form of the transfer function, the threshold, θ, can no longer be absorbed by the bias, b. However we can still introduce a fictitious input neuron X_0 whose activation is always given by $x_0 = 1$ and its connecting weight w_0 with the output neuron Y is defined to be b.

The final question is how are we going to find the set of weights and bias to solve a given problem? For this purpose we will be using the Perceptron learning rule. Just like Hebb's rule, the Perceptron learning rule is also a supervised learning rule. Thus we assume that we are given a training set:

$$\{\mathbf{s}^{(q)}, t^{(q)}\}, \qquad q = 1, 2, \ldots, Q.$$

where $\mathbf{s}^{(q)}$ is a training vector, and $t^{(q)}$ is its corresponding targeted output value.

The Perceptron rule is:

1. Set learning rate $\alpha (> 0)$ and threshold θ. Initialize weights and bias to zero (or some random values).
2. Repeat the following steps, while cycling through the training set $q = 1, 2, \ldots, Q$, until all training patterns are classified correctly
   (a) Set activations for input vector $\mathbf{x} = \mathbf{s}^{(q)}$.
   (b) Compute total input for the output neuron:
   $$y_{\text{in}} = \mathbf{x} \cdot \mathbf{w} + b$$
   (c) Compute the output
   $$y = f_\theta(y_{\text{in}}).$$
   (d) Update the weights and bias only if that pattern is misclassified
   $$\mathbf{w}^{\text{new}} = \mathbf{w}^{\text{old}} + \alpha t^{(q)} \mathbf{x},$$
   $$b^{\text{new}} = b^{\text{old}} + \alpha t^{(q)}.$$

Notice that when a training vector is presented to the NN and the output agrees with the targeted value, then there is no change to the weights and bias. There is a common saying: when things are working don't change them. Also notice that by the design of the algorithm, the NN always correctly classify all the training vectors using the weights and bias obtained from the algorithm. We will consider later a theorem that guarantees the convergence of the Perceptron learning algorithm.

2. Application: Bipolar Logical Function: AND

The training set is given by the following table:

q	$\mathbf{s}^{(q)}$	$t^{(q)}$
1	[1 1]	1
2	[1 -1]	-1
3	[-1 1]	-1
4	[-1 -1]	-1

We assume that the weights and bias are initially zero, and the threshold is 0.2 We choose a learning rate $\alpha = 1$. We obtain the following results by applying the Perceptron learning rule.

step	$\mathbf{s}^{(q)}$	y_{in}	y	$t^{(q)}$	$\Delta\mathbf{w}$	Δb	$\mathbf{w}$	b
1	[1 1]	0	0	1	[1 1]	1	[1 1]	1
2	[1 -1]	1	1	-1	[-1 1]	-1	[0 2]	0
3	[-1 1]	2	1	-1	[1 -1]	-1	[1 1]	-1
4	[-1 -1]	-3	-1	-1	[0 0]	0	[1 1]	-1
5	[1 1]	1	1	1	[0 0]	0	[1 1]	-1
6	[1 -1]	-1	-1	-1	[0 0]	0	[1 1]	-1
7	[-1 1]	-1	-1	-1	[0 0]	0	[1 1]	-1

Notice that in step 4 of the loop, the weights and bias do not change. However we still have to present the other 3 training vectors to the NN to make sure that they are also classified correctly and no changes in the weights and bias need to be made. After step 7, it is clear that the situation will repeat itself in groups of 4, and thus we can exit the loop. Therefore in general the loop in the Perceptron algorithm should be stopped when the weights and bias do not change for Q consecutive times.

The resulting set of weights, $\mathbf{w} = [1\ 1]$, and bias, $b = -1$, gives the decision boundary determined by

$$x_1 w_1 + x_2 w_2 = x_1 + x_2 = 1.$$

This boundary is a straight line given by

$$x_2 = 1 - x_1,$$

which has a slope of -1 and passes through the point $[1\ 0]$. This is the best decision boundary for this problem in terms of robustness (the training vectors are farthest from the decision boundary). Thus in this example, the Perceptron learning algorithm converges to a set of weights and bias that is the best choice for this NN.

However this is all quite fortuitous. In general we cannot expect the Perceptron learning algorithm to converge to a set of weights and bias that is the best choice for any given NN. It is easy to see that because if we had assumed some none zero values for the initial weights or bias, in all likelihood we would not have gotten the best decision boundary for this problem.

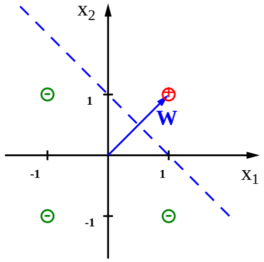

Figure 2: *The Perceptron rule finds the best decision boundary for the case of the* `AND` *function if the initial weights and bias were chosen to be zero.*

3. Perceptron Learning Rule Convergence Theorem

To consider the convergence theorem for the Perceptron Learning Rule, it is convenient to absorb the bias by introducing an extra input neuron, X_0, whose signal is always fixed to be unity.

Convergence Theorem for the Perceptron Learning Rule: For a Perceptron, if there is a correct weight vector $\mathbf{w}^*$ that gives correct response for all training patterns, that is

$$f_\theta(\mathbf{s}^{(q)} \cdot \mathbf{w}^*) = t^{(q)}, \qquad \forall q = 1, \dots, Q,$$

then for any starting weight vector $\mathbf{w}$, the Perceptron learning rule will converge in a finite number of steps to a correct weight vector.

According to the theorem, a table like the one given above for the NN functioning as an `AND` gate, must terminate after a finite number of steps. Note that the vectors appearing in the second column are all chosen from the training set. The same vector may appear more that once in that column. First we can ignore from that column those vectors that are classified correctly at the particular point in the loop, since they lead to no changes in the weights or the bias. Next, We consider those vectors in that column, say $\mathbf{s}^{(q)}$. whose target output is $t^{(q)} = -1$. This means that we want $\mathbf{s}^{(q)} \cdot \mathbf{w} < -\theta$ so that the output $y = -1$. If we multiply this inequality by minus one, we have $-\mathbf{s}^{(q)} \cdot \mathbf{w} > \theta$. We can reverse the signs of all these vectors so that we want their target output to be $+1$.

We denote the list of these vectors by $\mathcal{F}$. The individual vectors are denoted by $\mathbf{x}(k)$, so that

$$\mathcal{F} = \{\mathbf{x}(0), \mathbf{x}(1), \ldots, \mathbf{x}(n), \ldots\}.$$

Out goal is to prove that this list must terminate after a finite number of steps, and therefore contains only a finite number of vectors. By design, all these vectors have target outputs of $+1$, and they are all incorrectly classified at their own step in the loop.

The initial weight is $\mathbf{w}(0)$, and at step n (not counting those steps where the training vectors presented to the NN are classified correctly) the weight vector is $\mathbf{w}(n)$. Therefore for any n, $\mathbf{x}(n)$ is misclassified by definition and therefore we always have $\mathbf{x}(n) \cdot \mathbf{w}(n) < 0$.

We let $\mathbf{w}^*$ be a correct weight vector assuming that there is such a vector. Then by the fact that $\mathbf{w}^*$ gives a decision boundary that correctly classifies all the training vectors in $\mathcal{F}$, we have

$$\mathbf{x}(n) \cdot \mathbf{w}^* > 0, \qquad \forall \mathbf{x}(n) \in \mathcal{F}.$$

We need to define the following two quantities to be used in the proof:

$$m = \min_{\mathbf{x}(n)\in\mathcal{F}} \mathbf{x}(n) \cdot \mathbf{w}^*,$$

$$M = \max_{\mathbf{x}(n)\in\mathcal{F}} \|\mathbf{x}(n)\|^2.$$

Notice that both quantities are strictly positive.

Now we want to prove that the number of steps in the Perceptron learning rule is always finite provided that $\mathbf{w}^*$ exists.

For this proof, we assume that $\theta = 0$.

The first step in the loop of the Perceptron learning rule gives

$$\mathbf{w}(1) = \mathbf{w}(0) + \alpha\mathbf{x}(0),$$

and the second step gives

$$\mathbf{w}(2) = \mathbf{w}(1) + \alpha\mathbf{x}(1) = \mathbf{w}(0) + \alpha\mathbf{x}(0) + \alpha\mathbf{x}(1).$$

Thus after k steps, we have

$$\mathbf{w}(k) = \mathbf{w}(0) + \alpha\sum_{n=0}^{k-1}\mathbf{x}(n).$$

We take the dot-product of this equation with $\mathbf{w}^*$ to give

$$\mathbf{w}(k)\cdot\mathbf{w}^* = \mathbf{w}(0)\cdot\mathbf{w}^* + \alpha\sum_{n=0}^{k-1}\mathbf{x}(n)\cdot\mathbf{w}^*.$$

But by the definition of m, we have $\mathbf{x}(n)\cdot\mathbf{w}^* \geq m$ for any n. So we have the inequality

$$\mathbf{w}(k)\cdot\mathbf{w}^* > \mathbf{w}(0)\cdot\mathbf{w}^* + \alpha\sum_{n=0}^{k-1}m = \mathbf{w}(0)\cdot\mathbf{w}^* + \alpha km.$$

Notice that in the above relation, equality is not possible because there is at least one vector $\mathbf{x}(p)$ in the sum such that $\mathbf{x}(p)\cdot\mathbf{w}^* > m$.

Next, we make use of the Cauchy-Schwartz inequality:

$$\|\mathbf{a}\|^2\|\mathbf{b}\|^2 \geq (\mathbf{a}\cdot\mathbf{b})^2$$

for any vectors **a** and **b**. Therefore

$$\|\mathbf{a}\|^2 \geq \frac{(\mathbf{a}\cdot\mathbf{b})^2}{\|\mathbf{b}\|^2},$$

for any nonzero vector **b**. Identifying **a** as $\mathbf{w}(k)$ and **b** as $\mathbf{w}^*$ yields

$$\|\mathbf{w}(k)\|^2 \geq \frac{(\mathbf{w}(k)\cdot\mathbf{w}^*)^2}{\|\mathbf{w}^*\|^2} > \frac{(\mathbf{w}(0)\cdot\mathbf{w}^* + \alpha km)^2}{\|\mathbf{w}^*\|^2}.$$

Thus the square of the length of the weight vector at the k-th step, $\mathbf{w}(k)$, grows faster than $(\alpha km)^2$ (quadratic in k) for large k. The right-hand side of the inequality provides a lower bound for $\|\mathbf{w}(k)\|^2$. Note that $\mathbf{w}^*$ clearly cannot be a zero vector.

What we want next is to find an upper bound for $\|\mathbf{w}(k)\|^2$. Since $\mathbf{w}(k)$ is given by

$$\mathbf{w}(k) = \mathbf{w}(k-1) + \alpha\mathbf{x}(k-1),$$

taking the dot-product with itself gives

$$\|\mathbf{w}(k)\|^2 = \|\mathbf{w}(k-1)\|^2 + \alpha^2\|\mathbf{x}(k-1)\|^2 + 2\alpha\mathbf{x}(k-1)\cdot\mathbf{w}(k-1).$$

Since $\mathbf{x}(k-1)\cdot\mathbf{w}(k-1)$ must be negative because $\mathbf{x}(k-1)$ is classified incorrectly, the last term on the right-hand side of the above equation can be omitted to obtain the inequality

$$\|\mathbf{w}(k)\|^2 < \|\mathbf{w}(k-1)\|^2 + \alpha^2\|\mathbf{x}(k-1)\|^2.$$

Replacing k by $k-1$ gives

$$\|\mathbf{w}(k-1)\|^2 < \|\mathbf{w}(k-2)\|^2 + \alpha^2\|\mathbf{x}(k-2)\|^2.$$

Using this above inequality to eliminate $\|\mathbf{w}(k-1)\|^2$ yields

$$\|\mathbf{w}(k)\|^2 < \|\mathbf{w}(k-2)\|^2 + \alpha^2\|\mathbf{x}(k-2)\|^2 + \alpha^2\|\mathbf{x}(k-1)\|^2.$$

This procedure can be iterated to give

$$\|\mathbf{w}(k)\|^2 < \|\mathbf{w}(0)\|^2 + \alpha^2\|\mathbf{x}(0)\|^2 + \ldots + \alpha^2\|\mathbf{x}(k-2)\|^2 + \alpha^2\|\mathbf{x}(k-1)\|^2.$$

Using the definition of M, we have the inequality

$$\|\mathbf{w}(k)\|^2 < \|\mathbf{w}(0)\|^2 + \alpha^2 kM.$$

This inequality gives an upper bound for the square of $\mathbf{w}(k)$, and shows that it cannot grow faster than $\alpha^2 kM$ (linear in k).

Combining this upper bound with the lower bound for $\|\mathbf{w}(k)\|^2$, we have

$$\frac{(\mathbf{w}(0)\cdot\mathbf{w}^* + \alpha k m)^2}{\|\mathbf{w}^*\|^2} \le \|\mathbf{w}(k)\|^2 \le \|\mathbf{w}(0)\|^2 + \alpha^2 kM.$$

This bound is clearly valid initially when $k = 0$, because in that case the bound is

$$(\mathbf{w}(0)\cdot\hat{\mathbf{w}}^*)^2 \le \|\mathbf{w}(0)\|^2 \le \|\mathbf{w}(0)\|^2.$$

Notice that the square of the projection of $\mathbf{w}(0)$ in any direction cannot be larger than $\|\mathbf{w}(0)\|^2$.

Next consider what happens as k increases. Since the lower bound grows faster (quadratically with k) than the upper bound, which grows only linearly with k, there must be a value of k^* such that this condition is violated for all $k \ge k^*$. This means the iteration cannot continue forever and must therefore terminate after k^* steps.

To determine k^*, let κ be a solution of the following quadratic equation

$$\frac{(\mathbf{w}(0)\cdot\mathbf{w}^* + \alpha\kappa m)^2}{\|\mathbf{w}^*\|^2} = \|\mathbf{w}(0)\|^2 + \alpha^2\kappa M.$$

We find that κ is given by

$$\kappa = \frac{R}{2} - D \pm \sqrt{\frac{R^2}{4} - RD + L^2},$$

where we have defined

$$R = \frac{\max_{\mathbf{x}(n)\in\mathcal{F}}\{\mathbf{x}(n)\cdot\mathbf{x}(n)\}}{\left(\min_{\mathbf{x}(n)\in\mathcal{F}}\{\mathbf{x}(n)\cdot\hat{\mathbf{w}}^*\}\right)^2},$$

$$D = \frac{\mathbf{w}(0)\cdot\hat{\mathbf{w}}^*}{\alpha\min_{\mathbf{x}(n)\in\mathcal{F}}\{\mathbf{x}(n)\cdot\hat{\mathbf{w}}^*\}},$$

$$L^2 = \frac{\|\mathbf{w}(0)\|^2}{\alpha^2\left(\min_{\mathbf{x}(n)\in\mathcal{F}}\{\mathbf{x}(n)\cdot\hat{\mathbf{w}}^*\}\right)^2}.$$

In these expressions, $\hat{\mathbf{w}}^*$ is a unit vector (having unit magnitude) pointing in the direction of $\mathbf{w}^*$.

We want to show that only the upper sign leads to acceptable solution. First we assume that $\frac{R}{2} - D$ is positive. Clearly the upper sign will give a positive κ as required. The lower sign will lead to an unacceptable negative κ. We can prove that by showing that

$$\frac{R}{2} - D < \sqrt{\frac{R^2}{4} - RD + L^2}.$$

Since both sides of this inequality are positive, we can square both sides to give

$$\left(\frac{R}{2} - D\right)^2 = \frac{R^2}{4} - RD + D^2 < \frac{R^2}{4} - RD + L^2.$$

This relation becomes $D^2 < L^2$ or equivalently $\left(\mathbf{w}(0) \cdot \hat{\mathbf{w}}^*\right)^2 < \|\mathbf{w}(0)\|^2$. This final relation is obviously true because the square of the projection of any vector in any direction cannot be larger than the magnitude square of the vector itself.

Next we assume that $\frac{R}{2} - D$ is negative. Then the lower sign will always give a negative κ. On the other hand, the upper sign will always give an acceptable positive κ because

$$\sqrt{\frac{R^2}{4} - RD + L^2} > -\left(\frac{R}{2} - D\right),$$

as can be seen by squaring both sides.

Our conclusion is that k^* is given by $\lceil \kappa \rceil$ where

$$\kappa = \frac{R}{2} - D + \sqrt{\frac{R^2}{4} - RD + L^2}.$$

If the weight vector is initialized to zero, then $D = L = 0$ and therefore $k^* = \lceil \kappa \rceil = \lceil R \rceil$. From the definition of R, we see that the number of iteration is determined by the training vector that lie closest to the decision boundary. The smaller that distance is the higher the number of iterations has to be for convergence.

4. Perceptron for Multi-Category Classification

We will now extend to the case where a NN has to be able to classify input vectors, each having N components, into more than 2 categories.

We will continue to use only bipolar neurons whose transfer functions are given by

$$f_\theta(x) = \begin{cases} +1, & \text{if } x > \theta, \\ 0, & \text{if } -\theta \le x \le \theta, \\ -1, & \text{if } x < -\theta. \end{cases}$$

Clearly the NN now needs to have multiple neurons, Y_1, Y_2, ..., Y_M, in the output layer. The number of output neurons is specified by M. Each input neuron is connected to each of these output neuron with a weight factor. The weight associated with the connection between input neuron X_i with output neuron Y_j is denoted by w_{ij}. The figure shows a diagram of this NN.

What was the weight vector now becomes the weight matrix having two separate indices: i goes from 1 to N, and j goes from 1 to M. The weight matrix, $\mathbf{W}$, is in general a rectangular one, unless it happens that $N=M$. We follow common mathematical convention to use boldface capital letters to denote matrices. The total number of these weight components is given by NM.

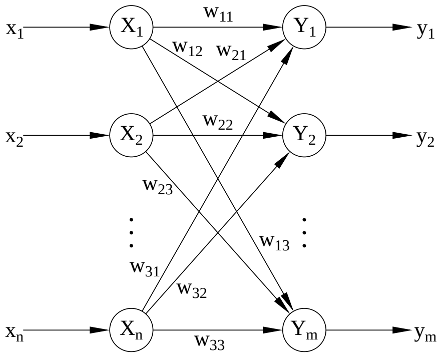

Figure 3: *A single-layer feedforward NN having multiple output neurons.*

Each output neuron also has its own bias, denoted by b_j, where $j = 1, \ldots, M$. Thus the total input into neuron Y_j is

$$y_{\text{in},j} = b_j + \sum_{i=1}^{N} x_i w_{ij}.$$

As before we can absorb a bias by introducing an extra input neuron, X_0, so that the weight connecting it to Y_j is $w_{0j} = b_j$, provided that the signal in X_0 is always fixed to be $x_0 = 1$. Then the total input into neuron Y_j can be rewritten as

$$y_{\text{in},j} = \sum_{i=0}^{N} x_i w_{ij}.$$

The output of neuron Y_j is then given by

$$y_j = f_\theta(y_{\text{in},j})$$

We need to be able to train this NN using a more general form of the Perceptron learning rule. To train the NN, we assume that we are given a training set:

$$\{\mathbf{s}^{(q)}, \mathbf{t}^{(q)}\}, \qquad q = 1, 2, \ldots, Q.$$

where $\mathbf{s}^{(q)}$ is a training vector, each having N components, and $\mathbf{t}^{(q)}$ is an M-component vector, representing the corresponding targeted output values. Notice that the target is no longer a scalar quantity. Since each component of the output can take on two different values, a NN with M output neuron is capable of classifying input vectors into 2^M categories (under practical situation where $N \gg M$)[2].

The general Perceptron rule is:

1. Set $\alpha(> 0)$ and θ. Initialize $\mathbf{W}$ and $\mathbf{b}$.
2. Repeat steps, while cycling through training set, until all training patterns are classified correctly
   (a) Set input vector $x_i = s_i^{(q)}, i = 1, \ldots, N$.
   (b) Compute total input for each output neuron:

$$y_{\text{in},j} = \sum_{i=1}^{N} x_i w_{ij} + b_j, \qquad j = 1, \ldots, M.$$

   (c) Compute the output

$$y_j = f_\theta(y_{\text{in},j}), \qquad j = 1, \ldots, M.$$

   (d) Update $\mathbf{W}$ and $\mathbf{b}$ only if pattern is misclassified

$$w_{ij}^{\text{new}} = w_{ij}^{\text{old}} + \alpha x_i t_j^{(q)},$$

$$b_j^{\text{new}} = b_j^{\text{old}} + \alpha t_j^{(q)}.$$

For $\theta = 0$, there is a total of M decision hyperplanes determined by

$$\sum_{i=1}^{N} x_i w_{ij} + b_j = \sum_{i=0}^{N} x_i w_{ij} = 0, \qquad j = 1, \dots, M$$

and for $N \geq M$ these hyperplanes divide the N dimensional input space into 2^M distinct regions. Thus each column of the weight matrix, $\mathbf{W}$, gives the weight vector that determines the normal direction of each of the M hyperplanes. If we do not absorb the biases by introducing a zeroth neuron, then we can define weight vectors $\mathbf{w}_j = [w_{1j} w_{2j} \dots w_{Nj}]$ so that the equation for the j-th hyperplane can be written as

$$\mathbf{x} \cdot \hat{\mathbf{w}}_j = \frac{-b_j}{\|\mathbf{w}_j\|}.$$

However, if we absorb the biases by introducing a zeroth neuron, then we can define weight vectors $\mathbf{w}_j = [w_{0j} w_{1j} w_{2j} \dots w_{Nj}]$ so that the equation for the j-th hyperplane can be written as

$$\mathbf{x} \cdot \hat{\mathbf{w}}_j = 0.$$

For finite value of θ, these hyperplanes become slabs of width $2d$, where the value of d will be determined below. Each slab divides the input vector space into three regions:

$$\sum_{i=1}^{N} x_i w_{ij} + b_j > \theta \qquad \Rightarrow y = +1,$$

$$-\theta \leq \sum_{i=1}^{N} x_i w_{ij} + b_j \leq \theta \qquad \Rightarrow y = 0,$$

$$\sum_{i=1}^{N} x_i w_{ij} + b_j < -\theta \qquad \Rightarrow y = -1.$$

The slab is bounded on each side by hyperplanes determined by

$$\mathbf{x} \cdot \hat{\mathbf{w}}_j = \frac{-b_j + \theta}{\|\mathbf{w}_j\|},$$

$$\mathbf{x} \cdot \hat{\mathbf{w}}_j = \frac{-b_j - \theta}{\|\mathbf{w}_j\|},$$

where the weight vectors are defined by $\mathbf{w}_j = [w_{1j}w_{2j} \dots w_{Nj}]$. Therefore the slab has a thickness $2d_j$, where

$$d_j = \frac{\theta}{\|\mathbf{w}_j\|}.$$

However, if we absorb the biases by introducing a zeroth neuron, then the slab is bounded on each side by hyperplanes determined by

$$\mathbf{x} \cdot \hat{\mathbf{w}}_j = \frac{\theta}{\|\mathbf{w}_j\|},$$

$$\mathbf{x} \cdot \hat{\mathbf{w}}_j = \frac{-\theta}{\|\mathbf{w}_j\|},$$

where the weight vectors are now defined by $\mathbf{w}_j = [w_{0j}w_{1j}w_{2j} \dots w_{Nj}]$. Therefore the slab has a thickness $2d_j$, where

$$d_j = \frac{\theta}{\|\mathbf{w}_j\|}.$$

Although this expression for d_j looks identical to the one above, the weight vectors are defined slightly differently, and the slab is located

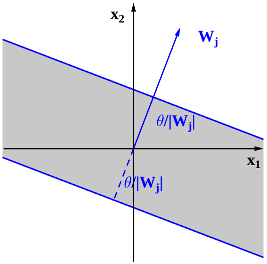

Figure 4: *A hyperplane with a safety margin.*

in an input vector space one higher than the one before. The figure shows one of the hyperplanes with a safety margin.

5. Example of a Perceptron with Multiple Output Neurons

We consider the following Perceptron with multiple output units. We want to design a NN, using bipolar neurons, and train it using the

Perceptron learning rule with the following training set:

(class 1)

$$\mathbf{s}^{(1)} = \begin{bmatrix} 1 & 1 \end{bmatrix}, \ \mathbf{s}^{(2)} = \begin{bmatrix} 1 & 2 \end{bmatrix} \quad \text{with} \quad \mathbf{t}^{(1)} = \mathbf{t}^{(2)} = \begin{bmatrix} -1 & -1 \end{bmatrix}$$

(class 2)

$$\mathbf{s}^{(3)} = \begin{bmatrix} 2 & -1 \end{bmatrix}, \ \mathbf{s}^{(4)} = \begin{bmatrix} 2 & 0 \end{bmatrix} \quad \text{with} \quad \mathbf{t}^{(3)} = \mathbf{t}^{(4)} = \begin{bmatrix} -1 & 1 \end{bmatrix}$$

(class 3)

$$\mathbf{s}^{(5)} = \begin{bmatrix} -1 & 2 \end{bmatrix}, \ \mathbf{s}^{(6)} = \begin{bmatrix} -2 & 1 \end{bmatrix} \quad \text{with} \quad \mathbf{t}^{(5)} = \mathbf{t}^{(6)} = \begin{bmatrix} 1 & -1 \end{bmatrix}$$

(class 4)

$$\mathbf{s}^{(7)} = \begin{bmatrix} -1 & -1 \end{bmatrix}, \ \mathbf{s}^{(8)} = \begin{bmatrix} -2 & -2 \end{bmatrix} \quad \text{with} \quad \mathbf{t}^{(7)} = \mathbf{t}^{(8)} = \begin{bmatrix} 1 & 1 \end{bmatrix}$$

It is clear from the training set that $N = 2$, $Q = 8$, and the number of classes is 4. The number of output neuron is chosen to be $M = 2$ so that each target vector has 2 components, each of which taking on 2 possible values, and so $2^M = 4$ classes can be represented. (Note that $M \not> N$ here.)

Setting $\alpha = 1$ and $\theta = 0$, and assuming zero initial weights and biases, we find that the Perceptron algorithm converges after 11 iterations (a little more than 1 epoch through the training set) to give

$$\mathbf{W} = \begin{bmatrix} -3 & 1 \\ 0 & -2 \end{bmatrix}, \qquad b = \begin{bmatrix} -2 & 0 \end{bmatrix}.$$

The 2 decision boundaries are given by the equations

$$-3x_1 - 2 = 0, \qquad x_1 - 2x_2 = 0,$$

which correspond to the following 2 straight lines through the origin

$$x_1 = -\frac{2}{3}, \qquad x_2 = \frac{1}{2}x_1.$$

These 2 lines separate the input vector space into 4 regions and it is easy to see that the NN correctly classifies the 8 training vectors into 4 classes. However these boundaries are not the most robust against random fluctuations in the components of the input vectors.

Here we have chosen a threshold $\theta = 0$. One expects that for a larger value of θ, a more robust set of decision slabs can be obtained. Experimentation by keeping $\alpha = 1$ but using larger values of θ shows

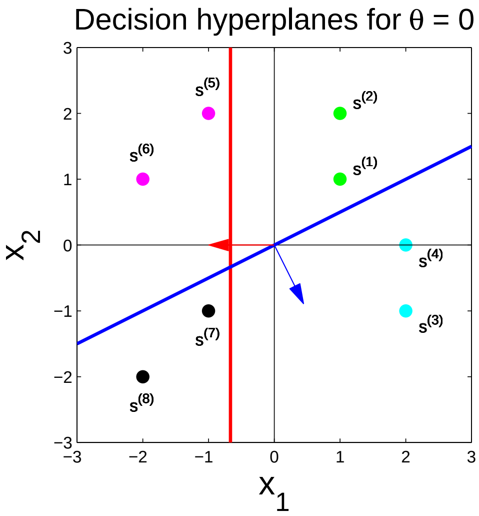

Figure 5: *A Perceptron capable of classifying patterns into 4 classes.*

that exactly the same number of iterations, the same sequence of weight matrices and bias vectors are obtained as long as $\theta < 1$. This is due to the fact that no training vector falls within the the decision slabs during training if $\theta < 1$.

For $\alpha = 1$ and $1 \geq \theta < 2$, the Perceptron algorithm converges in 15 iterations to give

$$\mathbf{W} = \begin{bmatrix} -5 & 1 \\ 1 & -5 \end{bmatrix}, \qquad b = \begin{bmatrix} 0 & 0 \end{bmatrix}.$$

For $\alpha = 1$ and $2 \geq \theta < 3$, the Perceptron algorithm converges in 17 iterations to give

$$\mathbf{W} = \begin{bmatrix} -8 & 2 \\ 0 & -6 \end{bmatrix}, \qquad b = \begin{bmatrix} -2 & 0 \end{bmatrix}.$$

With increasing values of θ, the number of iterations required for convergence becomes higher, and the lengths of the weight vectors become longer (not too surprising from the proof of the Perceptron convergence theorem, although it was strictly for $\theta = 0$). The resulting decision boundaries become more and more robust.

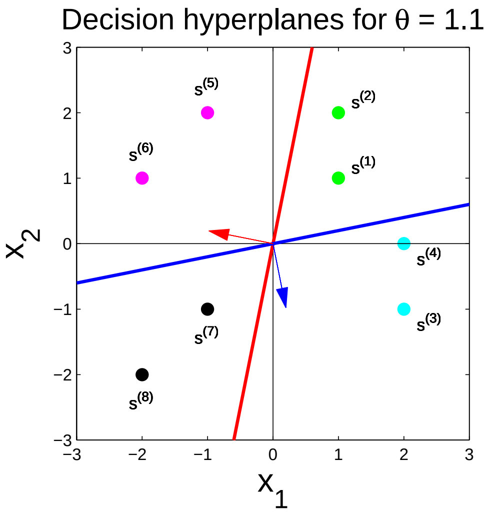

Figure 6: *A Perceptron capable of classifying patterns into 4 classes.*

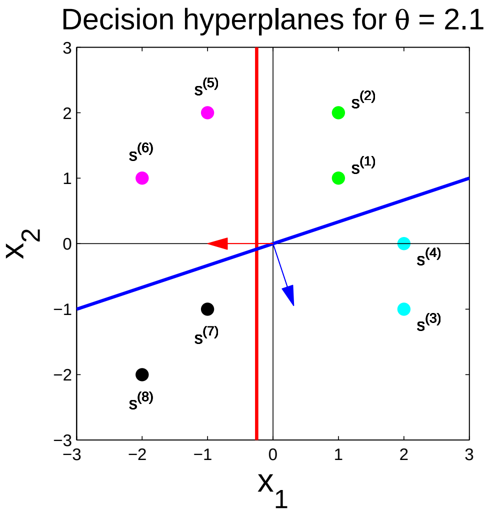

Figure 7: *A Perceptron capable of classifying patterns into 4 classes.*

References

[1] See Chapter 2 in Laurene Fausett, ”Fundamentals of Neural Networks - Architectures, Algorithms, and Applications”, Prentice Hall, 1994.

[2] In the case of pattern classification, the number of categories, M, is not a very large number under practical circumstances, but the number of of input vectors components, N, is usually large, and therefore we expect $N \gg M$. It is obvious that the number of training vectors, Q, must be larger than M, otherwise we cannot expect the NN to be able to learn anything. (We clearly need at least one training vector for each class.)

In many examples and homework problems, N, M and Q, are usually not very large, especially when we want to go through algorithms and analyze results analytically by hand. We have to be careful and make sure we don't draw wrong conclusions because of the smallness of N, M and Q. 25

For example, in a NN where $N < M$, the number of possible output categories is not 2^M for binary or bipolar output neurons,

but is substantially smaller. To illustrate, let us take $N = 2$ and $M = 3$. Thus we have 3 straight lines representing the 3 decision boundaries in $N = 2$ dimension space. A single line divides the two-dimensional plane into 2 regions. A second line not parallel with the first line will cut the first line and divide the space into 4 regions. A third line not parallel with the first 2 lines will cut each of them at one place thus creating 3 more regions. Thus the total number of separated regions is 7 not $2^3 = 8$. However if N is equal to or larger than M, then the M decision boundaries, each of which is a hyperplane, indeed divide the N dimensional space into 2^M regions. For example, the three coordinate planes divide the three dimensional space into 8 quadrants.