Neural Networks

Chp.3 : Learning

• Given a layered structure and the input x, the output of the neural network will depend on the weights :

• $o = F(W, x)$

• We arrange the weights so that the neural network performs a given task (possibly with an acceptable performance). Such tasks are e.g. classification, pattern recognition, function approximation, memorizing patterns, etc...

• **Learning** is a systematic way of changing the weights so that the given task is achieved (either after a finitely many steps, or asymptotically)

• Usually we have a training set set $\mathcal{S}_{Tr} = \{x^1, x^2, \ldots x^N\}$

• We start with an initial structure, and hence an initial weight $W(1)$

• Choose a training set example, say $x^1 \rightarrow$ Find $o(1) = F(W(1)x^1)$

• Then, based on some criterion we **change = update** the weights :

• $W(2) \leftarrow W(1) + \Delta W(1)$

• The way we select ΔW is called a **learning rule**

• Then choose a training set example (which is not chosen before), say x^2 $\rightarrow$ Find $o(2) = F(W(2)x^2)$

• Then, based on the same criterion we **change = update** the weights :

• $W(3) \leftarrow W(2) + \Delta W(2)$

• We keep this procedure till all the training set examples are exhausted. This is called an **epoch** in learning. After the end of the epoch, we obtain a set of weights $W(n)$. If this weight is not sufficient for our performance requirements, we start **another** epoch, and continue this way, till our performance measures are satisfied.

• Sometimes, this process may not converge (i.e. performance may degrade). Then we may stop the training, and start the process with different initial weight, or change some parameters of the learning process...

• There are basically 3 types of learning :

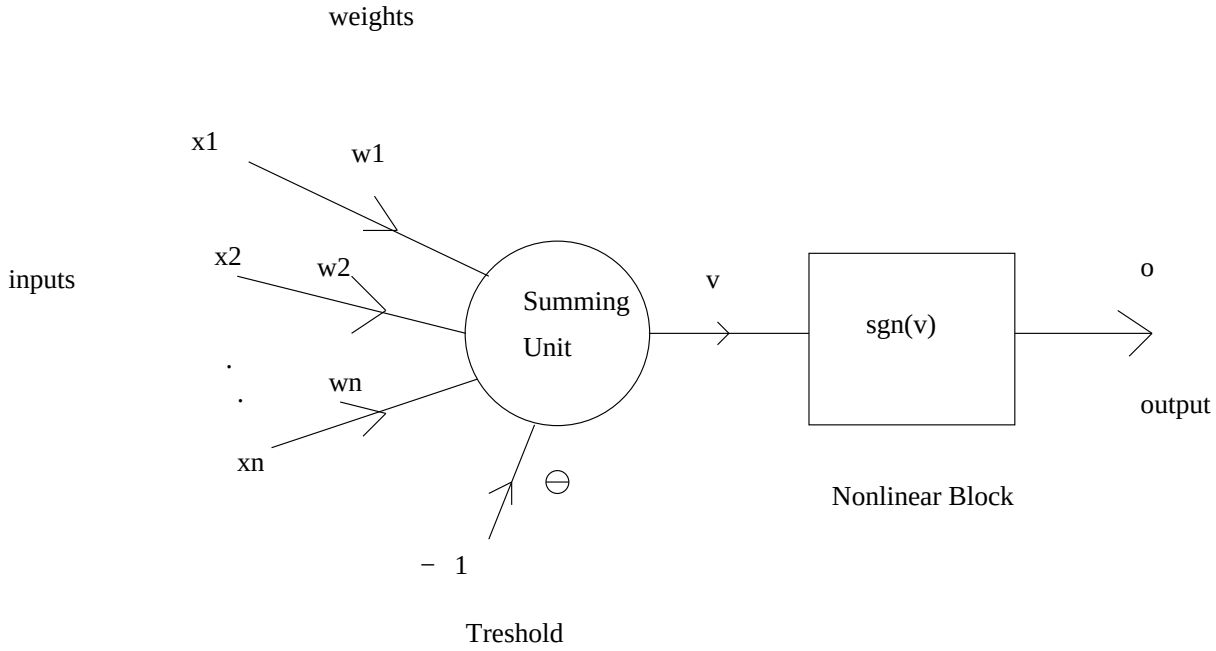
• **1 : Supervised Learning**. Here, at each pattern in the training set, we know what the correct output should be. And we decide on ΔW accordingly. This is called **learning with a teacher**.

• **2 : Unsupervised Learning**. Here, we do not know the correct output for the training patters. The network should organize itself → **Self-Organization**

• **3 : Reinforcement Learning**. Here, we have an **idea** about the **good** outputs, and **bad** outputs, and we decide on ΔW accordingly (reward and punishment) → game learning.

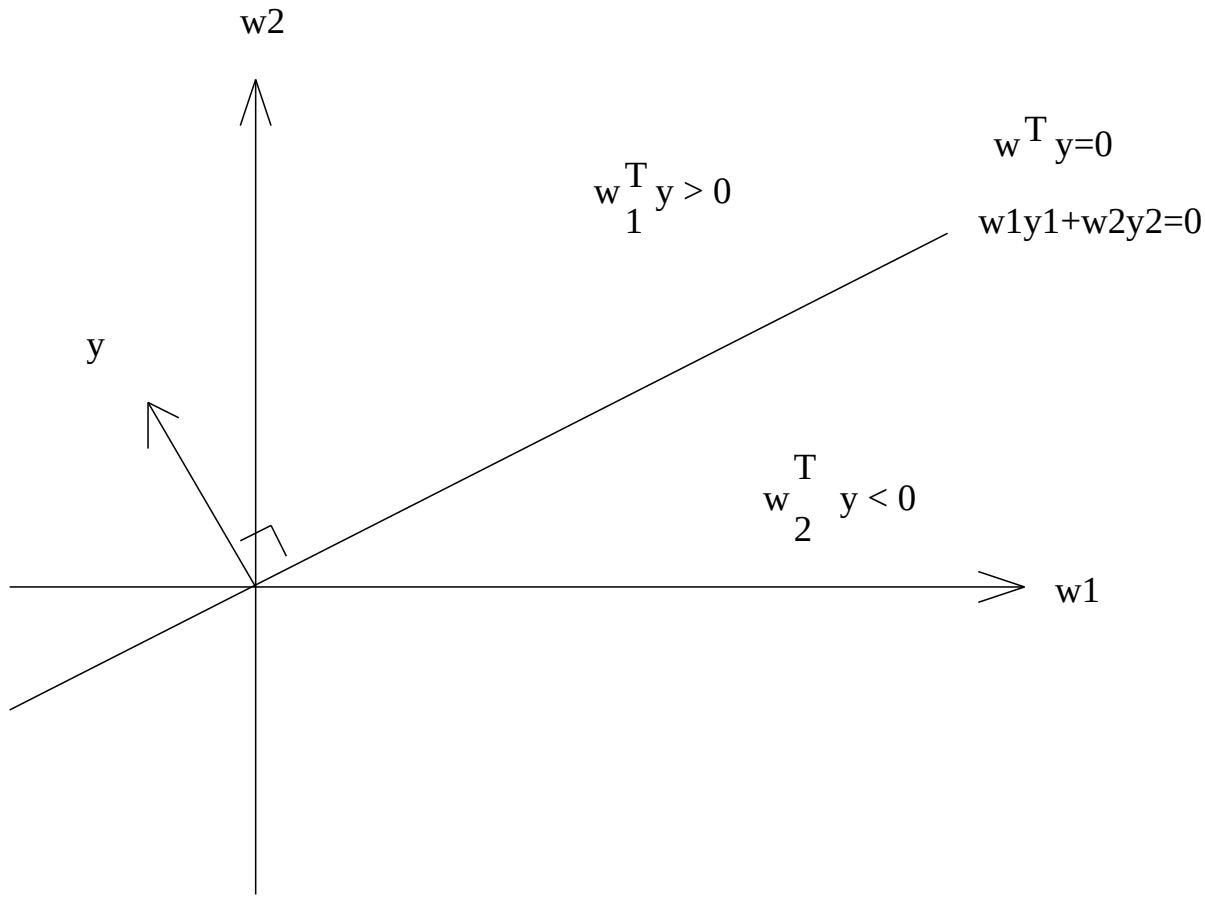
Chp.4 : Perceptron Classifiers

• We will come back to learning rules once again, but due to time and project limitations, now we will consider perceptrons and their usage in classification...



• $v = w_1x_1 + w_2x_2 + \ldots + w_nx_n - \theta = w^Tx - \theta = w_e^Tx_e \rightarrow o = sgn(v)$

• $w = \begin{pmatrix} w_1 \\ w_2 \\ \vdots \\ w_n \end{pmatrix} \quad x = \begin{pmatrix} x_1 \\ x_2 \\ \vdots \\ x_n \end{pmatrix} \quad w_e = \begin{pmatrix} w_1 \\ w_2 \\ \vdots \\ w_n \\ \theta \end{pmatrix} \quad x_e = y = \begin{pmatrix} x_1 \\ x_2 \\ \vdots \\ x_n \\ -1 \end{pmatrix}$



• Note that the plane (space) is in terms of **weights**. In this space, the line $w^T y = 0$ is determined by the vector y, which is **normal** to the line (hyperplane) and always points at the + side : Any weight in y pointing side satisfies $w^T y > 0$, and any weight in the other side satisfies $w^T y < 0$.

• This could easily be seen by Cauchy formula : If w and y are any two vectors, whose angle is γ, we have

• $w^T y = \|w\|\|y\| \cos\gamma$.

• Here $\|\cdot\|$ denotes the **length** of the vector (**a norm in general**)

• In the above side, for any weight vector w, we have $-90 < \gamma < 90$

• $\rightarrow w^T y > 0$

• **Perceptron Classification Problem :**

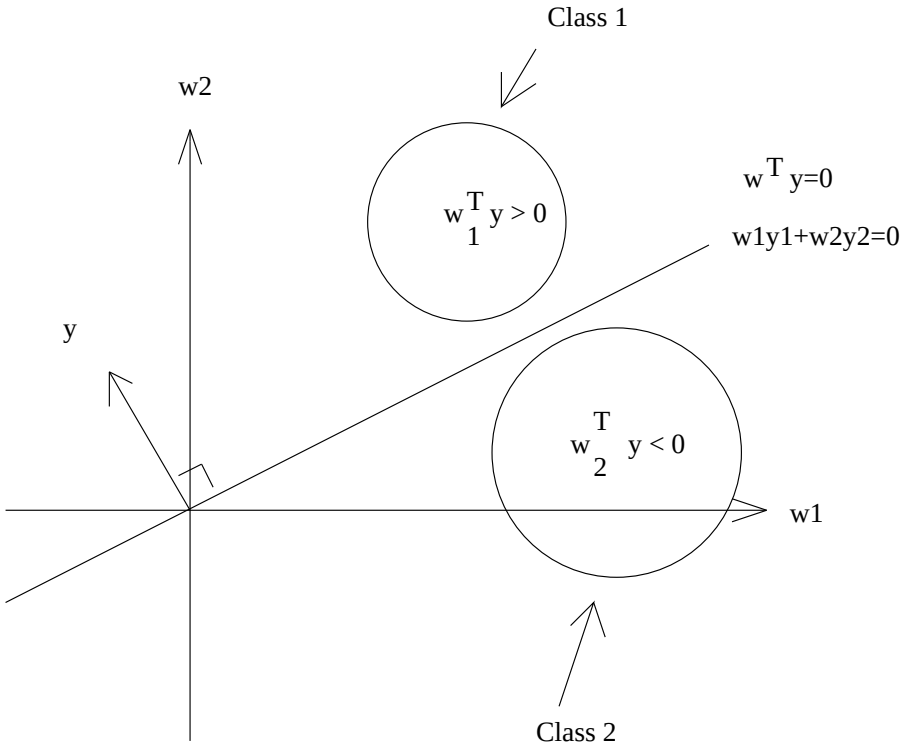
• Given two classes $\mathcal{C}_1 = \{x^1,\ x^2,\ \ldots,\ x^m\},\ \mathcal{C}_2 = \{x^{m+1},\ x^{m+2},\ \ldots,\ x^n\}$

• $\rightarrow \mathcal{C}_1 = \{y^1,\ y^2,\ \ldots,\ y^m\},\ \mathcal{C}_2 = \{y^{m+1},\ y^{m+2},\ \ldots,\ y^n\}$

• Find an **extended** weight vector W_*, if possible, such that :

• $$\begin{cases} W_*^T y > 0 & o = 1 & y \in \mathcal{C}_1 \\ W_*^T y < 0 & o = -1 & y \in \mathcal{C}_2 \end{cases}$$

• Obviously, a necessary condition is **linear separability**. Perceptron classification algorithm proofs that this condition is **sufficient** as well :



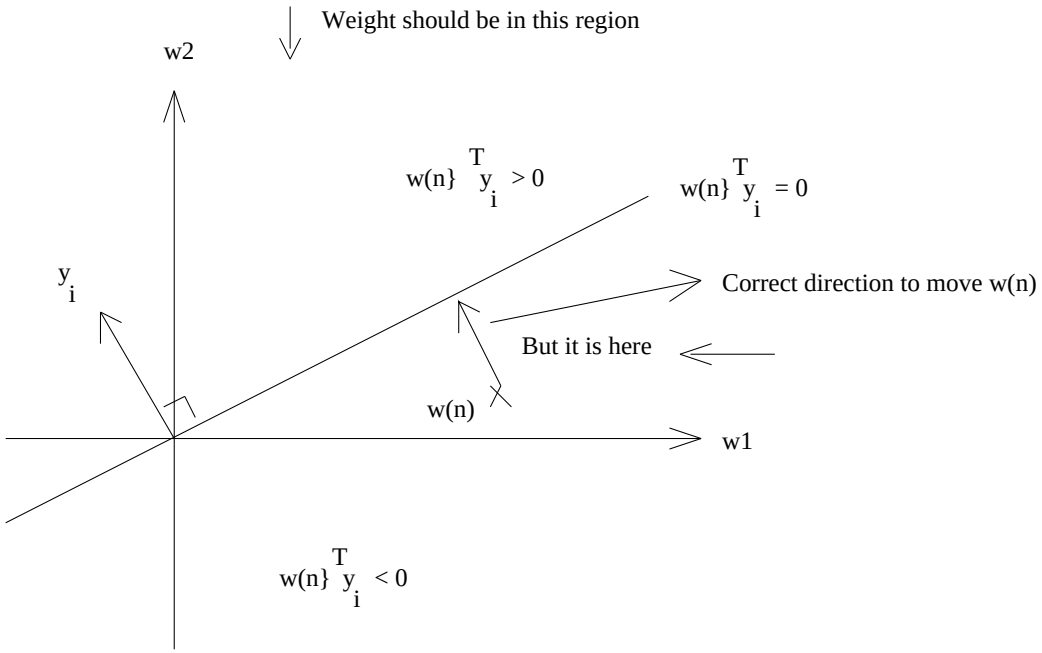
• Suppose that at nth step, we have the current weight as $W(n)$, and we picked a pattern y_i from the training set.

• There are 3 possibilities.

• **Case 1 :** y_i is correctly detected. That is, the sign of $W(n)^T y_i$ is correct for the class of y_i.

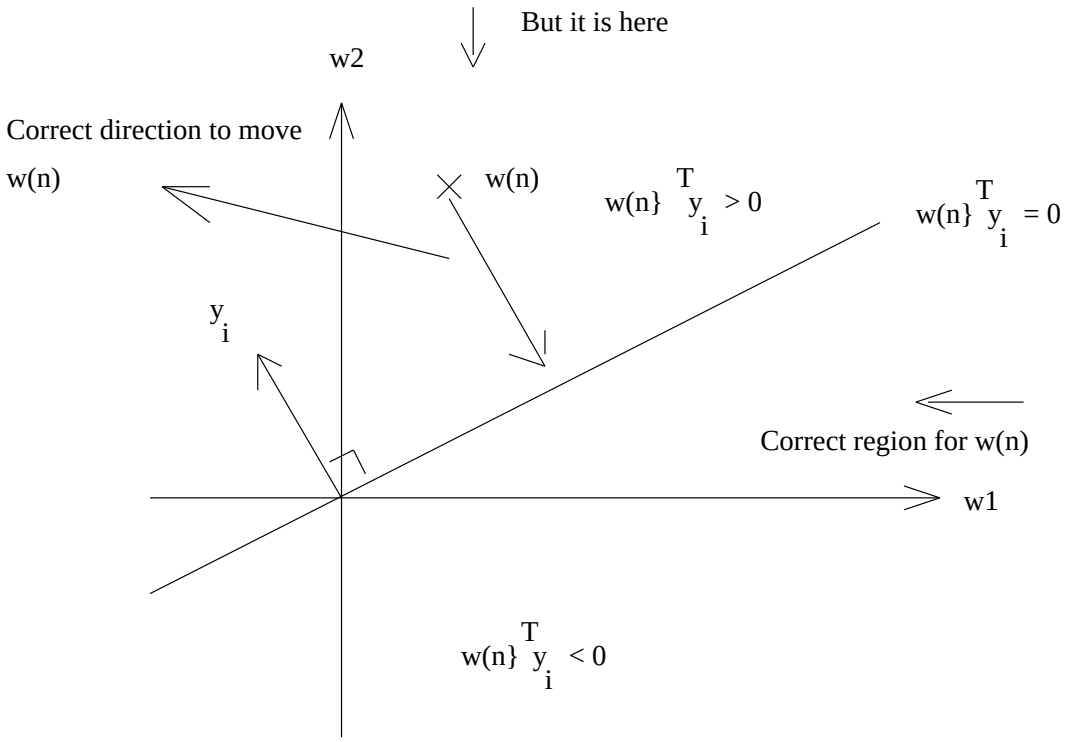
• **No update is necessary** $\rightarrow W(n+1) = W(n)$.

• **Case 2 :** $y_i \in \mathcal{C}_1$ (i.e. we should have $W(n)^T y_i > 0$), but NN misclassifies y_i (i.e. we have $W(n)^T y_i < 0$)



• **Update is necessary** $\rightarrow W(n+1) = W(n) + cy_i \qquad c > 0.$

• **Case 3 :** $y_i \in \mathcal{C}_2$ (i.e. we should have $W(n)^T y_i < 0$), but NN misclassifies y_i (i.e. we have $W(n)^T y_i > 0$)



• **Update is necessary** $\rightarrow W(n+1) = W(n) - cy_i \qquad c > 0.$

• **What is the idea?** Consider the case 2, i.e. we should have $W(n)^T y_i > 0$), but NN misclassifies y_i, hence we have $W(n)^T y_i < 0$.

• Suppose we apply the update rule $W(n+1) = W(n) + cy_i$, and choose y_i again.

• $\rightarrow W(n+1)^T y_i = W(n) y_i + c y_i^T y_i > W(n) y_i \qquad (c y_i^T y_i > 0)$

• Consider the case 3, i.e. we should have $W(n)^T y_i < 0$), but NN misclassifies y_i, hence we have $W(n)^T y_i > 0$.

• Suppose we apply the update rule $W(n+1) = W(n) - cy_i$, and choose y_i again.

• $\rightarrow W(n+1)^T y_i = W(n)y_i - cy_i^T y_i < W(n)y_i \qquad (cy_i^T y_i > 0)$

• **Conclusion 1 :** In both cases $W(n)$ is moved towards the correct region.

• **Conclusion 2 :** If $c > 0$ is sufficiently big, $W(n+1)$ will be in the correct region.

• We can summarize these algorithm as follows :

• **Perceptron Training Algorithm :**

• Given two classes $\mathcal{C}_1 = \{y^1,\ y^2,\ \ldots,\ y^m\},\ \mathcal{C}_2 = \{y^{m+1},\ y^{m+2},\ \ldots,\ y^n\}$

• Here we use extended notation. Let for input y_i, $d(i)$ denotes the **desired=correct** output.

• $\begin{cases} d(i) = 1 & y_i \in \mathcal{C}_1 \\ d(i) = -1 & y_i \in \mathcal{C}_2 \end{cases}$

• **Algorithm :**

• **Step 1 :** Choose an **arbitrary** initial weight vector $W(1)$ (extended), and a **learning constant** $c > 0$.

• Suppose that the algorithm evolves up to step i, we have the current weight $W(i)$...

• **Step 2 :** for $i = 1, 2, \ldots, n$, choose a pattern y_i from the training set. Calculate $v_i = W(i)^T y_i$ and the output $o_i = sgn(v(i))$.

• **Step 3 :** Update the weights as follows :

• $$W(i+1) = W(i) + \Delta W(i)$$

• $$\Delta W(i) = \begin{cases} cy_i & y_i \in \mathcal{C}_1, o(i) = -1 \\ -cy_i & y_i \in \mathcal{C}_2,\ o(i) = 1 \\ 0 & \text{otherwise} \end{cases}$$

• This can equivalently be written as :

• $$\Delta W(i) = \begin{cases} cy_i & d(i) = 1,\ o(i) = -1 \\ -cy_i & d(i) = -1,\ o(i) = 1 \\ 0 & d(i) = o(i) \end{cases}$$

• Hence we can equivalently write a single formula :

• $\Delta W(i) = 0.5c(d(i) - o(i))y_i$

• **Step 4 :** After all training set patterns are used (an **epoch**), if the final weight does not classify correctly all patterns, start the procedure again (start a new **epoch**).

• Does this process ever stop ?

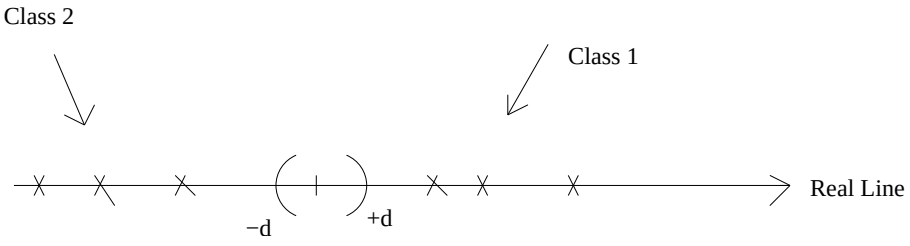
• **Perceptron Convergence Theorem**. Let two classes be given as $\mathcal{C}_1 = \{y^1,\ y^2,\ \ldots,\ y^m\},\ \mathcal{C}_2 = \{y^{m+1},\ y^{m+2},\ \ldots,\ y^n\}$ and without loss of generality, let us assume that $y \in \mathcal{C}_1$ if $W^T y > 0$ $(o = +1)$, and $y \in \mathcal{C}_2$ if $W^T y < 0$ $(o = -1)$. Assume that the classes are **linearly separable**. Then the perceptron algorithm given above always converge to a correct weight vector in a finite step.

• **Proof :** Since the classes are linearly separable, there exists a line which separates them $\rightarrow$ for some W_* we have $W_*^T y > 0$ if $y \in \mathcal{C}_1$, and $W_*^T y < 0$ if $y \in \mathcal{C}_2$.

• Hence there exists a $d > 0$ such that :

• $W_*^T y > d$ if $y \in \mathcal{C}_1$, and $W_*^T y < -d$ if $y \in \mathcal{C}_1$.

• Note that this is always possible since we have **finitely many** training patterns.



• There are two possible update cases

• **Case 1 :** $y_i \in \mathcal{C}_1$ (i.e. we should have $W(k)^T y_i > 0$), but NN misclassifies y_i (i.e. we have $W(k)^T y_i < 0$)

• **1 :** $W(k+1) = W(k) + cy_i$

• $\longrightarrow W_*^T W(k+1) = W_*^T W(k) + cW_*^T y_i > W_*^T W(k) + cd$

• **2 :** $\|W(k+1)\|^2 = W(k+1)^T W(k+1) = (W(k) + cy_i)^T (W(k) + cy_i)$
$= \|W(k)\|^2 + 2cW(k)^T y_i + c^2 \|y_i\|^2$

• $\longrightarrow \|W(k+1)\|^2 < \|W(k)\|^2 + c^2 \|y_i\|^2$

• **Case 2 :** $y_i \in \mathcal{C}_2$ (i.e. we should have $W(k)^T y_i < 0$), but NN misclassifies y_i (i.e. we have $W(k)^T y_i > 0$)

• **1 :** $W(k+1) = W(k) - cy_i$ (Note that $W_*^T y_i < -d \Rightarrow -W_*^T y_i > d$)

• $\longrightarrow W_*^T W(k+1) = W_*^T W(k) - cW_*^T y_i > W_*^T W(k) + cd$

• **2 :** $\|W(k+1)\|^2 = W(k+1)^T W(k+1) = (W(k) - cy_i)^T (W(k) - cy_i)$
$= \|W(k)\|^2 - 2cW(k)^T y_i + c^2 \|y_i\|^2$

• $\longrightarrow \|W(k+1)\|^2 < \|W(k)\|^2 + c^2 \|y_i\|^2$

• Now let $M = \max \|y_i\|$.

• **1 :** $W_*^T W(k+1) > W_*^T W(k) + cd$

• **2 :** $\|W(k+1)\|^2 < \|W(k)\|^2 + c^2 \|y_i\|^2 \leq \|W(k)\|^2 + c^2 M^2$

- Now iterate these inequalities :

- $W_*^T W(2) > W_*^T W(1) + cd$
- $W_*^T W(3) > W_*^T W(2) + cd > W_*^T W(1) + 2cd$
- $\vdots$
- $W_*^T W(k+1) > W_*^T W(1) + kcd$

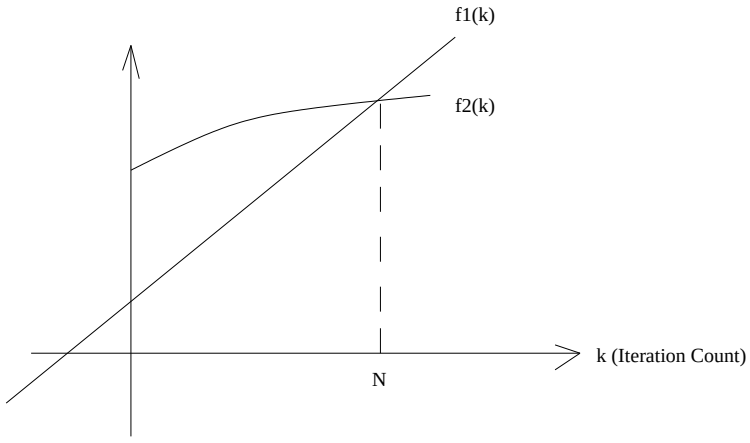
- $\|W(2)\|^2 \leq \|W(1)\|^2 + c^2 M^2$
- $\|W(3)\|^2 \leq \|W(2)\|^2 + c^2 M^2 \|W(1)\|^2 + 2c^2 M^2 \leq$
- $\vdots$
- $\|W(k+1)\|^2 \leq \|W(1)\|^2 + kc^2 M^2$

- Cauchy-Schwartz inequality $\rightarrow | \ x^T y \ | \leq \|x\| \|y\|$

- $\|W_*\| \|W(k+1)\| > W_*^T W(1) + kcd$

- $\|W(k+1)\| \leq \sqrt{W(1)\|^2 + kc^2 M^2}$

- $f_1(k) = \frac{W_*^T W(1) + kcd}{\|W_*\|} \leq \|W(k+1)\| \leq \sqrt{W(1)\|^2 + kc^2 M^2} = f_2(k)$



• Hence there exist an integer N such that for any iteration $k > N$, the above inequality is not satisfied $\rightarrow$ **contradiction**.

• Therefore, the algorithm stops at most after N steps.

• For an estimation, and for simplicity assume that $\|W_*\| = 1$, $W(1) = 0$.

• $Ncd > \sqrt{Nc^2M^2} = cM\sqrt{N} \rightarrow N \approx M^2/d^2$

• **Example**

$$\mathcal{C}_1 = \{y_1 = \begin{pmatrix} 10 \\ 2 \\ -1 \end{pmatrix}, y_2 = \begin{pmatrix} 9 \\ 3 \\ -1 \end{pmatrix}\} \ , \ \mathcal{C}_2 = \{y_3 = \begin{pmatrix} 0 \\ 2 \\ -1 \end{pmatrix}, y_4 = \begin{pmatrix} -9 \\ -2 \\ -1 \end{pmatrix}\}$$

• Choose $c = 1$, $W(1) = \begin{pmatrix} -1 \\ -1 \\ -1 \end{pmatrix}$

• **Epoch 1 :**

$$v(1) = W(1)^T y_1 = -11 < 0 \rightarrow o(1) = -1 \rightarrow W(2) = W(1) + y_1 = \begin{pmatrix} 9 \\ 1 \\ -2 \end{pmatrix}$$

$$v(2) = W(2)^T y_2 = 86 > 0 \rightarrow o(1) = 1 \rightarrow W(3) = W(2)$$

$$v(3) = W(3)^T y_3 = 4 > 0 \rightarrow o(1) = 1 \rightarrow W(4) = W(3) - y_3 = \begin{pmatrix} 9 \\ -1 \\ -1 \end{pmatrix}$$

$v(4) = W(4)^T y_4 = -78 < 0 \rightarrow o(1) = -1 \rightarrow W(5) = W(4)$

• **end of Epoch 1 :**

• **Epoch 2 :**

$v(5) = W(5)^T y_1 = 89 > 0 \rightarrow o(1) = 1 \rightarrow W(6) = W(5)$

$v(6) = W(6)^T y_2 = 79 > 0 \rightarrow o(1) = 1 \rightarrow W(7) = W(6)$

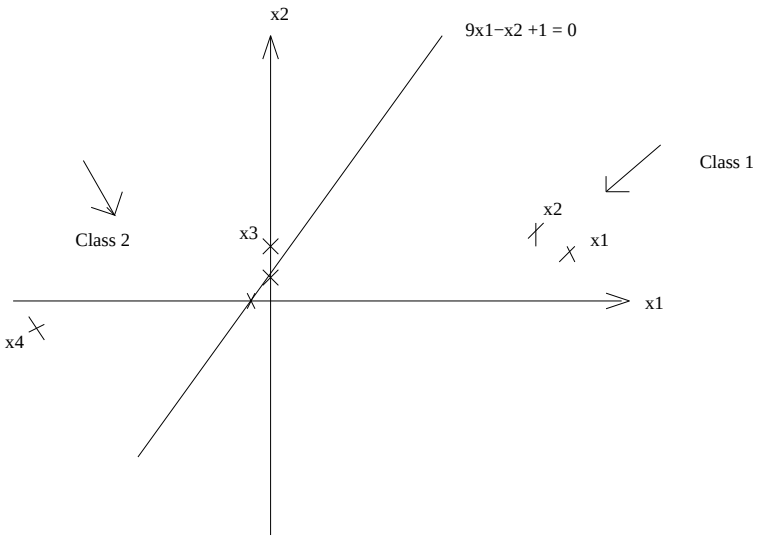
$v(7) = W(7)^T y_3 = -1 < 0 \rightarrow o(1) = -1 \rightarrow W(8) = W(7)$

$v(8) = W(8)^T y_4 = -78 < 0 \rightarrow o(1) = -1 \rightarrow W(9) = W(8)$

• **end of Epoch 2 :**

• Since no update takes place in epoch 2, algorithm stops, and the last updated weight is **a correct weight**.

• Separating line : $9x_1 - x_2 + 1 = 0$



Chp.4 : Perceptron Classifiers : Multicategory Case

• Here we are given R classes $\mathcal{C}_1 = \{x^1,\ x^2,\ \ldots,\ x^m\} \ldots\ \mathcal{C}_R = \{x^{q+1},\ x^{q+2},\ \ldots,\ x^N\}$

• Use extended notation $\mathcal{C}_1 = \{y^1,\ y^2,\ \ldots,\ y^m\} \ldots\ \mathcal{C}_R = \{y^{q+1},\ y^{q+2},\ \ldots,\ y^N\}$

• We have R perceptron neurons $\rightarrow$ R set of weights. Given a (extended) pattern y :

• $v_1 = W_1^T y,\ v_2 = W_2^T y,\ \ldots,\ v_R = W_R^T y$

• $\rightarrow o_i = sgn(v_i) \qquad i = 1, 2, \ldots, R$

$$W_j = \begin{pmatrix} w_{j1} \\ w_{j2} \\ \vdots \\ w_{jn} \end{pmatrix},\ x^i = \begin{pmatrix} x_1^i \\ x_2^i \\ \vdots \\ x_n^i \end{pmatrix},\ W_{je} = \begin{pmatrix} w_{j1} \\ w_{j2} \\ \vdots \\ w_{jn} \\ \theta_j \end{pmatrix},\ x_e^i = y^i = \begin{pmatrix} x_1^i \\ x_2^i \\ \vdots \\ x_n^i \\ -1 \end{pmatrix}$$

• **Perceptron Classification Problem : Multicategory Case**

• Given R classes $\mathcal{C}_1, \ldots, \mathcal{C}_R$, find R extended weights $W_{1*}, \ldots, W_{R*}$ (extended) such that :

• Given any pattern y^i, compute $v_1 = W_{1*}^T y^i, \ldots, v_R = W_{R*}^T y^i$.

• $y^i \in \mathcal{C}_m \Longleftrightarrow v_m > v_j, \qquad j = 1, 2, \ldots, R, \quad j \neq m$

• **Perceptron Training Algorithm : Multicategory case**

• Given R classes $\mathcal{C}_1, \ldots, \mathcal{C}_R$, choose initial weights (extended) $W_1(1), \ldots, W_R(1)$ and the learning constant $c > 0$ **arbitrarily**.

• Suppose that the algorithm evolves n steps, and the **last** updated weights are $W_1(n), \ldots, W_R(n)$

• Pick a pattern y^i from the training set. Assume that $y^i \in \mathcal{C}_m$

• Compute $v_1(n) = W_1^T(n)y^i, \ldots, v_R(n) = W_R^T(n)y^i$

• If $v_j(n) \geq v_m(n)$ for some $j = 1, 2 \ldots, R, \quad j \neq m$, then update the weights as :

- $W_m(n+1) = W_m(n) + cy^i$
- $W_j(n+1) = W_j(n) - cy^i$

• Otherwise, do not update the corresponding weights.

• Choose another pattern from the training set (not the same pattern twice) and repeat the same steps till all the training set patterns are used. (end of an **epoch**).

• If the last updated weights achieves correct classification, stop. Otherwise start a new epoch by using the last updated weights as initial weights of the next epoch.

• **Why does it work ?** The reason is similar to two class case.

• Suppose that for pattern $y^i \in \mathcal{C}_m$, the current weights does the misclassification :

• That is we have $v_j(n) \geq v_m(n)$ for some $j = 1, 2 \ldots, R, \quad j \neq m$, then update the weights as :

- $W_m(n+1) = W_m(n) + cy^i$
- $W_j(n+1) = W_j(n) - cy^i$

• Suppose that we apply y^i again :

- $v_j(n+1) = W_j(n+1)^T y^i = [W_j(n) - cy^i]^T y^i$
- $= W_j(n)^T y^i - cy^{iT} y^i < v_j(n)$
- Hence v_j **decreases**

- $v_m(n+1) = W_m(n+1)^T y^i = [W_m(n) + cy^i]^T y^i$
- $= W_m(n)^T y^i + cy^{iT} y^i > v_m(n)$
- Hence v_m **increases**

• Moreover, if $c > 0$ is sufficiently big, or if we apply the same procedure sufficiently many times we obtain

- $v_j(n) < v_m(n)$ for **all** $j = 1, 2 \ldots, R, \quad j \neq m$
- This method uses the **maximum selection** based on v_j :

- **Step 1** Given pattern y, calculate $v_1, v_2, \ldots, v_R$
- **Step 2** Find the index j such that $v_j = \max\{v_1, v_2, \ldots, v_R\}$
- **Step 3** $y \in \mathcal{C}_j$

• Instead of **maximum selection** based on v_j, we could choose **maximum selection** based on $o_j = sgn(v_j)$

• But $o_j = \pm 1$! (due to signum function)

• For correct modification, we define the **desired = correct** output vector d^i for each vector y^i in the training set as follows :

• $d^i = \begin{pmatrix} d_1^i \\ d_2^i \\ \vdots \\ d_R^i \end{pmatrix} \quad \Rightarrow \quad d_m^i = \begin{cases} +1 & y^i \in \mathcal{C}_m \\ -1 & \text{otherwise} \end{cases} \qquad m = 1, 2 \ldots, R$

• Hence, if $y^i \in \mathcal{C}_m \rightarrow d_m = +1$, and $d_j = -1, j = 1, 2 \ldots, R, j \neq m$

• $\Rightarrow y^i \in \mathcal{C}_m \rightarrow d_m > d_j, j = 1, 2 \ldots, R, j \neq m$

• This would mean that $v_m > 0$, and $v_j < 0,\ j = 1, 2 \ldots, R, j \neq m$

• Hence the training algorithm given above can equivalently be converted into the following :

• $W_j(n+1) = W_j(n) + 0.5c\ \ (d_j^i - o_j(n))y^i\ ,\ j = 1, 2 \ldots, R$

• Define the output vector o^i as :

• $o(n) = \begin{pmatrix} o_1(n) \\ o_2(n) \\ \vdots \\ o_R(n) \end{pmatrix}$

• **Perceptron Training Algorithm : Multicategory case**

• Given R classes $\mathcal{C}_1, \ldots, \mathcal{C}_R$, choose initial weights (extended) $W_1(1), \ldots, W_R(1)$ and the learning constant $c > 0$ **arbitrarily** Assign the desired output vectors d^i to each pattern as described before.

• Suppose that the algorithm evolves n steps, and the **last** updated weights are $W_1(n), \ldots, W_R(n)$

• Pick a pattern y^i from the training set.

• Compute $v_1(n) = W_1(n)y^i, \ldots, v_R(n) = W_R(n)y^i$

• Compute the outputs $o_1(n) = sgn(v_1(n)), \ldots, o_R(n) = sgn(v_R(n))$

• Update the weights as follows :

• $$W_j(n+1) = W_j(n) + 0.5c \ \ (d_j^i - o_j(n))y^i \ , \ \ j = 1, 2 \ldots, R$$

• Choose another pattern from the training set (not the same pattern twice) and repeat the same steps till all the training set patterns are used. (end of an **epoch**).

• If the last updated weights achieves correct classification, stop. Otherwise start a new epoch by using the last updated weights as initial weights of the next epoch.

Example : Given 3 classes $\mathcal{C}_1,\ \mathcal{C}_2,\ \mathcal{C}_3$ as follows :

- $\mathcal{C}_1 = \left\{ x^1 = \begin{pmatrix} 10 \\ 2 \end{pmatrix} \right\}\ ,\ \mathcal{C}_2 = \left\{ x^2 = \begin{pmatrix} 2 \\ -5 \end{pmatrix} \right\}\ ,\ \mathcal{C}_3 = \left\{ x^3 = \begin{pmatrix} -5 \\ 1 \end{pmatrix} \right\}$

- $\mathcal{C}_1 = \left\{ y^1 = \begin{pmatrix} 10 \\ 2 \\ -1 \end{pmatrix} \right\}\ ,\ \mathcal{C}_2 = \left\{ y^2 = \begin{pmatrix} 2 \\ -5 \\ -1 \end{pmatrix} \right\}\ ,\ \mathcal{C}_3 = \left\{ y^3 = \begin{pmatrix} -5 \\ 1 \\ -1 \end{pmatrix} \right\}$

- **Desired outputs :**

- $d^1 = \begin{pmatrix} 1 \\ -1 \\ -1 \end{pmatrix}\ ,\ d^2 = \begin{pmatrix} -1 \\ 1 \\ -1 \end{pmatrix}\ ,\ d^3 = \begin{pmatrix} -1 \\ -1 \\ 1 \end{pmatrix}$

- **Initial Weights :**

- $W_1(1) = \begin{pmatrix} 1 \\ -2 \\ 0 \end{pmatrix}\ ,\ W_2(1) = \begin{pmatrix} -2 \\ 4 \\ 3 \end{pmatrix}\ ,\ W_3(1) = \begin{pmatrix} 1 \\ 3 \\ -1 \end{pmatrix}$

- Choose the learning coefficient $c = 1$

- Start of Epoch 1 :

- Apply y^1 :
- $v_1(1) = W_1(1)y^1 = 6 > 0 \Rightarrow o_1(1) = 1 \;\; (= d^1(1))$
- $v_2(1) = W_2(1)y^1 = -15 < 0 \Rightarrow o_2(1) = -1 \;\; (= d^1(2))$
- $v_3(1) = W_3(1)y^1 = 17 > 0 \Rightarrow o_3(1) = 1 \;\; (\neq d^1(3)) \Rightarrow$ Needs correction !

- Update the weights

- $W_1(2) = W_1(1)$, $W_2(2) = W_2(1)$, $W_3(2) = W_3(1) - y^1 = \begin{pmatrix} -9 \\ 1 \\ 0 \end{pmatrix}$

- Apply y^2 :
- $v_1(2) = W_1(2)y^2 = 12 > 0 \Rightarrow o_1(2) = 1 \;\; (\neq d^2(1)) \Rightarrow$ Needs correction !
- $v_2(2) = W_2(2)y^2 = -27 < 0 \Rightarrow o_2(2) = -1 \;\; (\neq d^2(2) \Rightarrow)$ Needs correction !
- $v_3(2) = W_3(2)y^2 = -23 < 0 \Rightarrow o_3(2) = -1 \;\; (= d^2(3))$

- Update the weights

- $W_1(3) = W_1(2) - y^2 = \begin{pmatrix} -1 \\ 3 \\ 1 \end{pmatrix}$, $W_2(3) = W_2(2) + y^2 = \begin{pmatrix} 0 \\ -1 \\ 2 \end{pmatrix}$
- $W_3(3) = W_3(2)$

- Apply y^3 :
- $v_1(3) = W_1(3)y^3 = 7 > 0 \Rightarrow o_1(3) = 1 \ \ (\neq d^3(1)) \Rightarrow$ Needs correction !
- $v_2(3) = W_2(3)y^3 = -3 < 0 \Rightarrow o_2(3) = -1 \ \ (= d^3(2))$
- $v_3(3) = W_3(3)y^3 = 46 > 0 \Rightarrow o_3(3) = 1 \ \ (= d^3(3))$

- Update the weights

- $W_1(4) = W_1(3) - y^3 = \begin{pmatrix} 4 \\ 2 \\ 2 \end{pmatrix}$, $W_2(4) = W_2(3)$, $W_3(4) = W_3(3)$

- End of Epoch 1.

- Start of Epoch 2.

- Apply y^1 :
- $v_1(4) = W_1(4)y^1 = 42 > 0 \Rightarrow o_1(4) = 1 \ \ (= d^1(1))$
- $v_2(4) = W_2(4)y^1 = -4 < 0 \Rightarrow o_2(4) = -1 \ \ (= d^1(2))$
- $v_3(4) = W_3(4)y^1 = -88 < 0 \Rightarrow o_3(4) = -1 \ \ (= d^1(3))$

- Update the weights

- $W_1(5) = W_1(4)$, $W_2(5) = W_2(4)$, $W_3(5) = W_3(4) \Rightarrow$ No update !

- Apply y^2 :
- $v_1(5) = W_1(5)y^2 = -4 < 0 \Rightarrow o_1(5) = -1 \ \ (= d^2(1))$
- $v_2(5) = W_2(5)y^2 = 8 > 0 \Rightarrow o_2(5) = 1 \ \ (= d^1(2))$
- $v_3(5) = W_3(5)y^2 = -23 < 0 \Rightarrow o_3(5) = -1 \ \ (= d^1(3))$

• Update the weights

• $W_1(6) = W_1(5)\ ,\ W_2(6) = W_2(5)\ ,\ W_3(6) = W_3(5) \Rightarrow$ No update !

• Apply y^3 :

• $v_1(6) = W_1(6)y^3 = -20 < 0 \Rightarrow o_1(6) = -1 \ \ (= d^3(1))$

• $v_2(6) = W_2(6)y^3 = -3 < 0 \Rightarrow o_2(6) = -1 \ \ (= d^3(2))$

• $v_3(6) = W_3(6)y^3 = 46 > 0 \Rightarrow o_3(5) = 1 \ \ (= d^1(3))$

• Update the weights

• $W_1(7) = W_1(6)\ ,\ W_2(7) = W_2(6)\ ,\ W_3(7) = W_3(6) \Rightarrow$ No update !

• End of Epoch 2.

• Since in Epoch 2, no weight update took place, algorithm stops and the last updated weights are the correct weights.

• $W_{1*} = \begin{pmatrix} 4 \\ 2 \\ 2 \end{pmatrix}\ ,\ W_{2*} = \begin{pmatrix} 0 \\ -1 \\ 2 \end{pmatrix}, W_{3*} = \begin{pmatrix} -9 \\ 1 \\ 0 \end{pmatrix}$

• $o_1 = sgn(4x_1 + 2x_2 - 2)$

• $o_2 = sgn(-x_2 - 2)$

• $o_3 = sgn(-9x_1 + x_2)$

• Given R classes (in extended notation) $\mathcal{C}_1 = \{y^1,\ y^2,\ \ldots,\ y^m\} \ldots\ \mathcal{C}_R = \{y^{q+1},\ y^{q+2},\ \ldots,\ y^N\}$

• These classes are called **linearly separable** if there exist R extended weights $W_{1*}, \ldots, W_{R*}$ such that the following holds :

• Given any pattern y^i, compute $v_1 = W_{1*}^T y^i, \ldots, v_R = W_{R*}^T y^i$.

• $y^i \in \mathcal{C}_m \Longleftrightarrow v_m > v_j, \qquad j = 1, 2, \ldots, R, \quad j \neq m$

• **Perceptron Convergence Theorem : Multicategory case** . Given R classes which are linearly separable, the algorithm given above always converge in a finite number of steps.