Linear Classification-
The Perceptron

Why Linear Classifiers?

- 2 class problem: If the number of patterns is less than the number of features, there is always a hyperplane which separates the patterns fully.
- It follows that the linear classifiers are useful:
- In tasks of very high dimensionality
- In tasks of low dimensionality, where we have a relatively small number of training patterns at our disposal.
- Moreover, the number of features can be increased using new components which are non-linear functions (e.g. Polynomials) of the original features.

Example: Document classification

Representation of documents using vectors „weighing” different words :

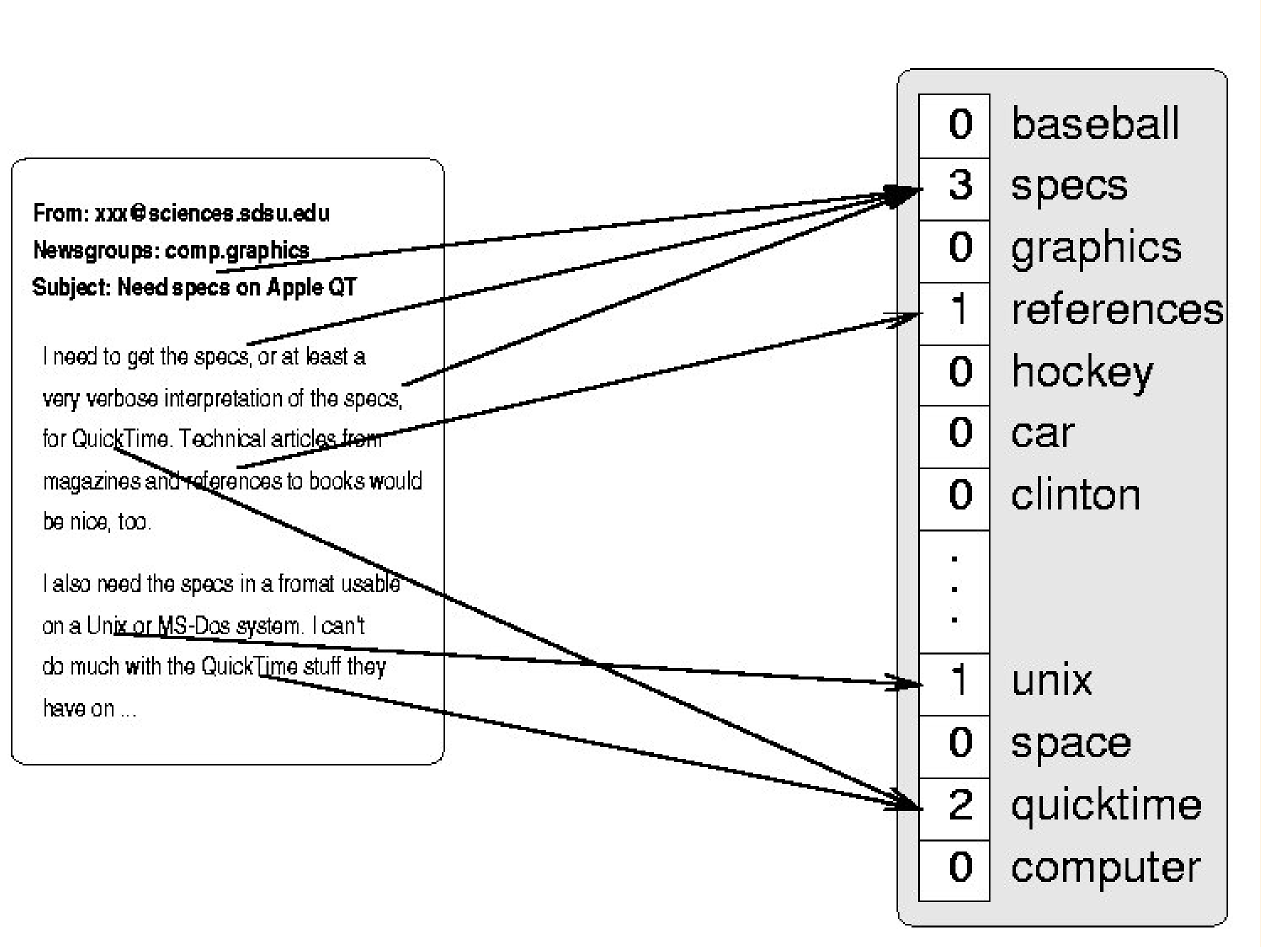
«weight» of word *i* in document *d*:

Term Frequency – Inverse Document Frequency

$$W(i\ ,\ d) = tf\ (i\ ,\ d)\ /\ df\ (i)$$

- $tf\ (i\ ,\ d)$ = times the word *i* appears in text *d*
  (word *i* is important for document *d* if it appears frequently in that document)

- $df\ (i)$ = number of documents in which word *i* appears
  (words appearing in many documents are less important overall, e.g. stopwords)



Typical example: Identifying "spam" messages

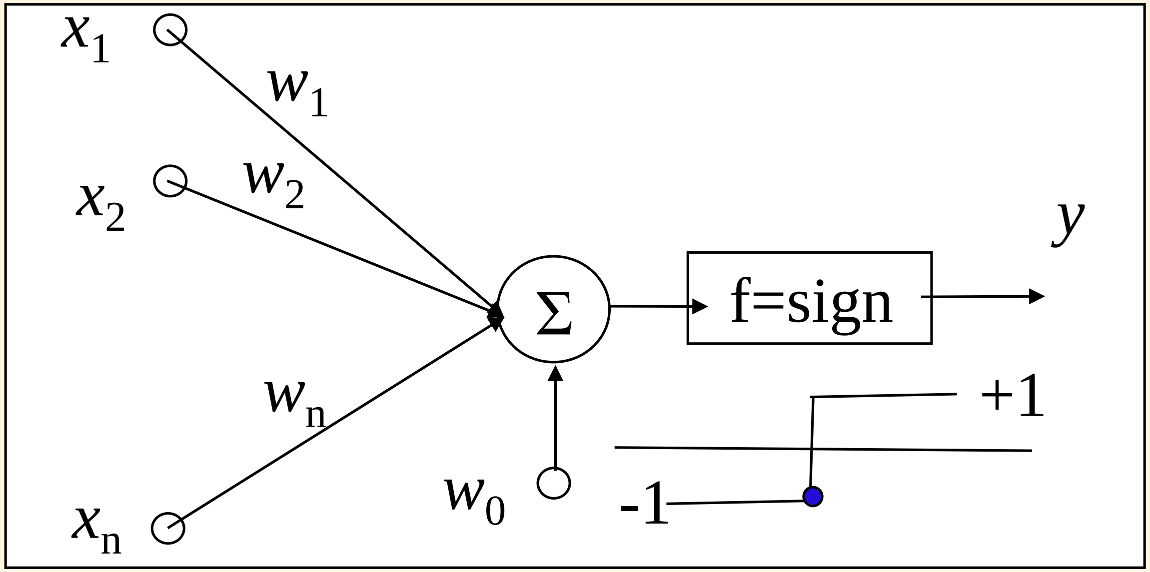
2 classes (spam and non-spam)

Typical size of training set: 10^3 messages

Typical size of patterns: 10^4 -10^5 components

The simple perceptron

- ***Architecture:*** Single layered network with *N inputs and M* neurons arranged in a single layer. Synaptic connections link every neuron with every input.
- ***Neurons:*** McCulloch-Pitts with hard limiter and adaptive activation threshold $$y_i = \mathrm{sign}(\sum_j w_{ij} x_j - w_{0i})$$
- Given that the outputs are mutually independent, we can study them independently, considering each output neuron on its own:



$$y = \mathrm{sign}(\sum_j w_j x_j - w_0)$$
$$= \mathrm{sign}(\mathbf{w} \cdot \mathbf{x} - w_0)$$

The problem: We are given a training set of patterns

$$\{\mathbf{x}^{\mu}, \quad \mu = 1, 2, \ldots, P\} \subseteq \mathbb{R}^{n}$$

split into 2 classes

$$C_1 = \{\mathbf{x}^{\mu}, \quad \mu = 1, 2, \ldots, K\} \quad C_2 = \{\mathbf{x}^{\mu}, \quad \mu = K+1, K+2, \ldots, P\}$$

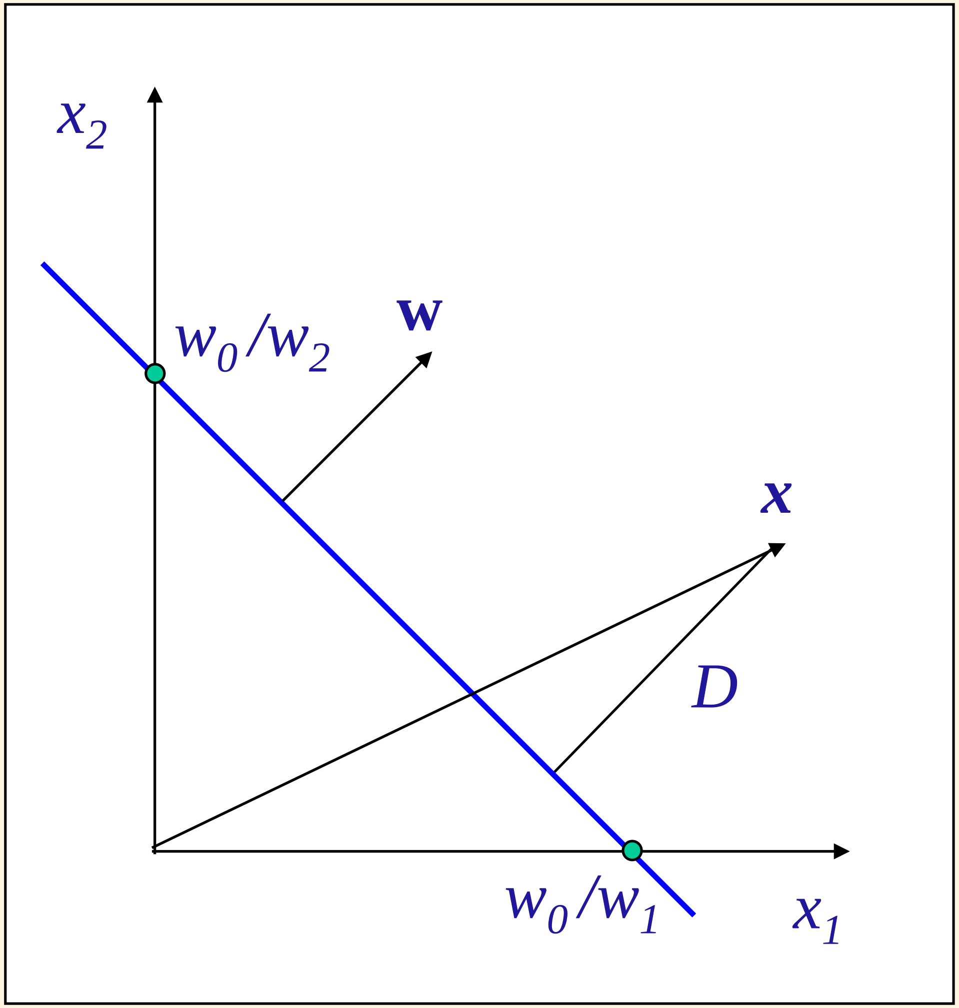
which are linearly separable, meaning that there exists a vector $\hat{\mathbf{w}}$ which satisfies:

$$\hat{\mathbf{w}} \cdot \mathbf{x}^{\mu} - w_0 > 0 \quad \forall \mathbf{x}^{\mu} \in C_1$$

$$\hat{\mathbf{w}} \cdot \mathbf{x}^{\mu} - w_0 < 0 \quad \forall \mathbf{x}^{\mu} \in C_2$$

We seek to find such a vector that linearly separates the two classes following an iterative process.

The geometry of the problem



- Weight vector is perpendicular to the separating hyperplane
- Weight vector is directed towards the positive semi-hyperplane, where

$$\mathbf{w}\cdot\mathbf{x}-w_0>0$$

- Distance of pattern **x** from the separating hyperplane:

$$D=\frac{|\mathbf{w}\cdot\mathbf{x}-w_0|}{\|\mathbf{w}\|}$$

Trick:

- By adding a feature equal to -1 for every pattern, we can incorporate the threshold into the formalism:

$$\mathbf{x} \to (\mathbf{x}, -1) \qquad\qquad y = \mathrm{sign}(\mathbf{w} \cdot \mathbf{x})$$

Coding of outputs:

- For the patterns of the two classes, we adopt „target outputs" :

$t^\mu = +1$ for patterns in class C_1

$t^\mu = -1$ for patterns in class C_2

Training phase: Iterative update of the synaptic weights, so that all outputs become equal to the desired target outputs.

Testing phase: Every new pattern **x** shown to the network is classified in one of the two classes depending on the output of the network:

$$y = \mathrm{sign}(\mathbf{w} \cdot \mathbf{x}) = 1 \Rightarrow \mathbf{x} \in C_1$$

$$y = \mathrm{sign}(\mathbf{w} \cdot \mathbf{x}) = -1 \Rightarrow \mathbf{x} \in C_2$$

The perceptron algorithm

- Initialization: Zero initial synaptic weights.
- We present the training patterns to the network, one by one. For each pattern, we ask if the output is equal to the desired output.
- If it is equal, we do nothing.
- If it is not equal, we add to each synaptic weight a quantity proportional to the product of the input by the desired output.

$$\mathbf{w} \rightarrow \mathbf{w} + \Delta\mathbf{w}$$

$$y^{\mu} \neq t^{\mu} \Rightarrow \Delta\mathbf{w} = \varepsilon t^{\mu}\mathbf{x}^{\mu}$$

$$y^{\mu} = t^{\mu} \Rightarrow \Delta\mathbf{w} = 0$$

- When all training patterns have been presented to the network, we repeat the process by presenting the patterns again, one by one. Termination: When, after a round of presentation of the training set, it is found out that all patterns are correctly classified.

REMARK # 1:

The condition $y^\mu \neq t^\mu$, under which updating of the synaptic weights takes place, is equivalent to the following:

$$(\mathbf{w} \cdot \mathbf{x}^\mu) t^\mu \leq 0$$

REMARK # 2:

Updating takes place after presenting each wrongly classified pattern (incremental mode).

In a variant of the algorithm (batch mode), updating takes place only after presentation of all training pattens. Updates follow the rule:

$$\mathbf{w} \rightarrow \mathbf{w} + \Delta\mathbf{w}$$

$$\Delta\mathbf{w} = \varepsilon \sum_{\mathbf{x}^\mu \in Y} t^\mu \mathbf{x}^\mu$$

where Y is the set of the wrongly classified training patterns.

REMARK # 3:

Evidently, the weight vector at each iteration of the algorithm is a linear combination of the training patterns:

$$\mathbf{w} = \sum_{\mu} \lambda_{\mu} \mathbf{x}^{\mu} t^{\mu}$$

with positive coefficients λ_{μ}. *We may consider the λ_{μ} as the parameters that we seek to find, instead of the synaptic weights w_i. The λ_{μ} are called dual variables with respect to the w_i and vice versa.*

$$\mathbf{w} \leftarrow \text{Dual variables} \rightarrow \boldsymbol{\lambda}$$

Dimension: Equal to the dimension of each pattern

Dimension: Equal to the total number of patterns

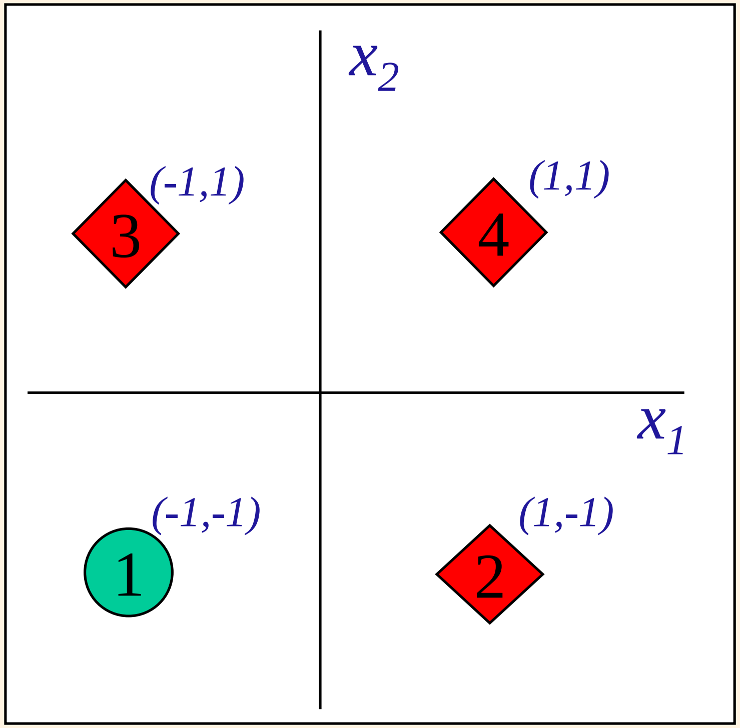
REMARK # 4:

If we use the dual variables $\boldsymbol{\lambda}$, the network output becomes:

$$y = sign\left[\sum_{\mu} \lambda_{\mu} t^{\mu} (\mathbf{x}^{\mu} \cdot \mathbf{x})\right]$$

The input data appear in this expression in the form of an ***inner product.***

Example (modified OR problem)



a/a	x_1	x_2	x_0	tx_1	tx_2	tx_0
1	-1	-1	-1	1	1	1
2	1	-1	-1	1	-1	-1
3	-1	1	-1	-1	1	-1
4	1	1	-1	1	1	-1

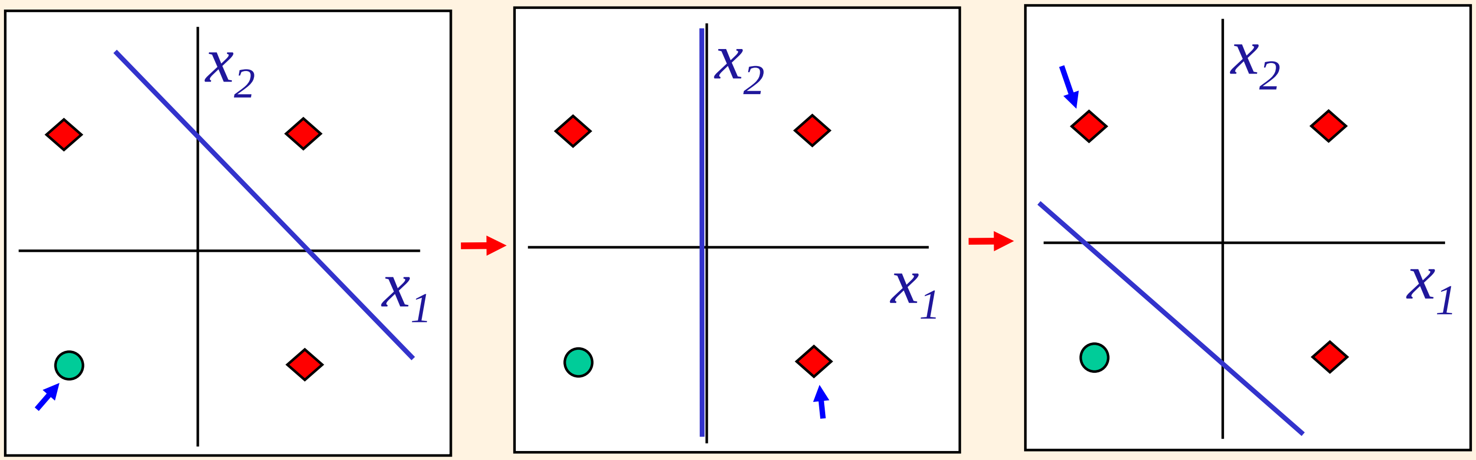
Patterns 2,3,4 → $t = 1$

Pattern 1 → $t = -1$

We „manufacture” the modified patterns $\mathbf{x}^{\mu}$ by adding to all of them a component equal to -1 in order to account for the influence of the threshold.

Progress of the perceptron algorithm (incremental mode) with ε=1:

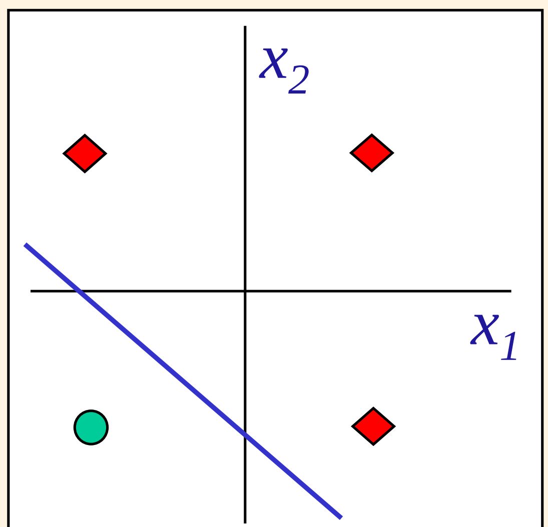
a/a	w_1	w_2	w_0	tx_1	tx_2	tx_0	$t\mathbf{w.x}$	Update	Δw_1	Δw_2	Δw_0
1	0	0	0	1	1	1	0	YES	1	1	1
2	1	1	1	1	-1	-1	-1	YES	1	-1	-1
3	2	0	0	-1	1	-1	-2	YES	-1	1	-1
4	1	1	-1	1	1	-1	3	NO	0	0	0
1	1	1	-1	1	1	1	1	NO	0	0	0
2	1	1	-1	1	-1	-1	1	NO	0	0	0
3	1	1	-1	-1	1	-1	1	NO	0	0	0
4	1	1	-1	1	1	-1	3	NO	0	0	0



Successive changes of the separating straight line are shown. The blue arrow shows which pattern is responsible for the change.

Progress of the perceptron algorithm (batch mode) with ε=1:

a/a	w_1	w_2	w_0	tx_1	tx_2	tx_0	t**w.x**	Update	Δw_1	Δw_2	Δw_0
1	0	0	0	1	1	1	0	YES	2	2	-2
2				1	-1	-1	0	YES			
3				-1	1	-1	0	YES			
4				1	1	-1	0	YES			
1	2	2	-2	1	1	1	2	NO	0	0	0
2				1	-1	-1	2	NO			
3				-1	1	-1	2	NO			
4				1	1	-1	6	NO			



Convergence (incremental mode)

- Let us assume that the set of patterns is linearly separable, meaning that there exists a vector $\hat{\mathbf{w}}$, for which the inequality $\hat{\mathbf{w}} \cdot \mathbf{x}^{\mu} t^{\mu} > 0$ holds for all patterns. In this case, the perceptron algorithm converges to a solution that linearly separates the patterns in a finite number of iterations.

- Taking account of the weight update rule, we observe that on termination of the algorithm, all patterns are correctly classified.
- It follows that, in order to demonstrate linear separability on termination of the algorithm, it suffices to show that the number of iterative steps is finite.

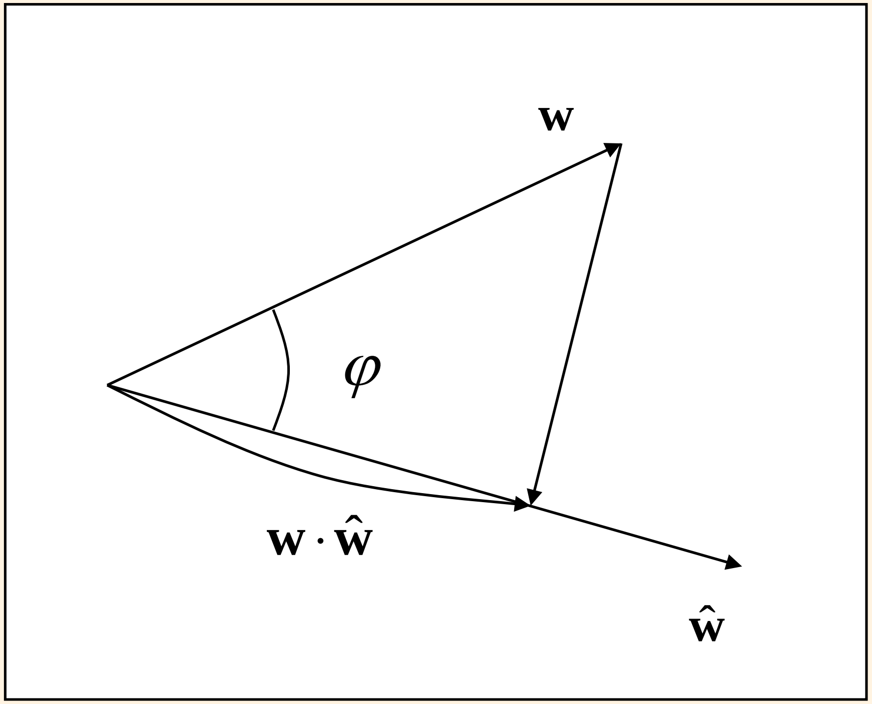
- In each iteration, a specific training pattern is presented to the network.
- According to the perceptron rule, presentation of a specific pattern μ may or may not trigger a weight update, depending on whether it is correctly or wrongly classified.
- Suppose that we have already performed N iterations.
- Let N_μ be the number of iterations in which presentation of pattern μ has triggered weight updates.

$$N = \sum_\mu N_\mu$$

- Let us assume, for the sake of simplicity, that the initial weights are all set to zero. The perceptron rule yields:

$$\mathbf{w} = \varepsilon \sum_\mu N_\mu \mathbf{x}^\mu t^\mu$$

- Since we know that there exists a weight vector $\hat{\mathbf{w}}$ that leads to total separation of the two classes, it is reasonable to investigate how close the vector that we seek to find is to this already known vector.
- Let us examine how the cosine of the angle formed by the two vectors changes as the algorithm proceeds:



$$\cos\varphi = \frac{\mathbf{w}\cdot\hat{\mathbf{w}}}{|\mathbf{w}||\hat{\mathbf{w}}|}$$

$$0 \le \frac{\mathbf{w}\cdot\hat{\mathbf{w}}}{|\mathbf{w}||\hat{\mathbf{w}}|} \le 1$$

> 0 because $\hat{\mathbf{w}}$ corresponds to the separating hyperplane

- **Numerator**

$$\mathbf{w}\cdot\hat{\mathbf{w}} = \varepsilon\sum_{\mu} N^{\mu} t^{\mu}\mathbf{x}^{\mu}\cdot\hat{\mathbf{w}} \geq \varepsilon\sum_{\mu} N^{\mu}\min(t^{\mu}\mathbf{x}^{\mu}\cdot\hat{\mathbf{w}})$$

$$= \varepsilon\min(t^{\mu}\mathbf{x}^{\mu}\cdot\hat{\mathbf{w}})\sum_{\mu} N^{\mu}$$

$$= \varepsilon N\min(t^{\mu}\mathbf{x}^{\mu}\cdot\hat{\mathbf{w}}) = \varepsilon N\left|\hat{\mathbf{w}}\right| D(\hat{\mathbf{w}})$$

"Distance" of closest pattern from the separating hyperplane

The numerator increases at least linearly with the number of iterations.

- **Denominator**

We need to determine: how fast is the change of the weight vector norm?

- Denominator:

  Consider the change in the norm of the weight vector, following the update using a specific pattern:

$$\Delta|\mathbf{w}|^2 = |\mathbf{w} + \varepsilon t^a \mathbf{x}^\alpha|^2 - |\mathbf{w}|^2 = \left(\varepsilon^2 |\mathbf{x}^\alpha|^2 + |\mathbf{w}|^2 + 2\varepsilon t^a \mathbf{w}\cdot\mathbf{x}^\alpha\right) - |\mathbf{w}|^2$$

$$= \varepsilon^2 |\mathbf{x}^\alpha|^2 + 2\varepsilon t^a \mathbf{w}\cdot\mathbf{x}^\alpha$$

Since the weights were updated, it is evident that $t^a \mathbf{w}\cdot\mathbf{x}^\alpha \le 0$ and therefore:

$$\Delta|\mathbf{w}|^2 \le \varepsilon^2 |\mathbf{x}^\alpha|^2$$

(update using a specific pattern)

We started from an initial weight vector equal to the zero vector. Therefore, the final norm of the weight vector is given by:

$$N_\mu \le N$$

$$|\mathbf{w}|^2 = \sum \Delta|\mathbf{w}|^2 \le \varepsilon^2 \sum_\mu N_\mu |\mathbf{x}^\mu|^2$$

$$\le \varepsilon^2 N \sum_\mu |\mathbf{x}^\mu|^2 = \varepsilon^2 N \beta, \quad \beta = \sum_\mu |\mathbf{x}^\mu|^2$$

(total update after many iterations)

β is a known, constant quantity, depending only on the training patterns.

Lets return to the angle φ and take advantage of the bounds that we already found:

$$\cos\varphi = \frac{\mathbf{w}\cdot\hat{\mathbf{w}}}{|\mathbf{w}||\hat{\mathbf{w}}|} \geq \frac{\varepsilon N |\hat{\mathbf{w}}| D}{\varepsilon\sqrt{N\beta}\,|\hat{\mathbf{w}}|} = D\sqrt{\frac{N}{\beta}}$$

However, $\cos\varphi \leq 1$ and therefore:

$$D\sqrt{N/\beta} \leq 1 \Rightarrow N \leq \beta / D^2$$

It follows that the number of steps are finite. Therefore, the algorithm separates linearly the patterns after a finite number of iterations.