EE4305 Fuzzy/Neural Systems for Intelligent Robotics

Part I: Neural Networks

Chapter Two: The Perceptron

Xiang Cheng

Associate Professor

Department of Electrical & Computer Engineering
The National University of Singapore

Phone: 65166210 Office: Block E4-08-07
Email: elexc@nus.edu.sg

What is a Neural Network (NN)?

A neural network is a *massively parallel distributed processor* that has a natural propensity for *storing experiential knowledge* and making it available for use.

It employs a massive inter-connection of "simple" computing units - *neurons.*

It is capable of organizing its structure consists of many neurons, to perform tasks that are many times faster than the fastest digital computers nowadays.

Knowledge is obtained by learning from the data/input signals presented to the network.

The artificial neural network resembles the brain in two respects:

1. **How does a neural network acquire new knowledge?**

Knowledge is acquired by the network through a *learning process*.

2. Where does the neural network store the knowledge?

Inter-neuron connection strengths known as *synaptic weights* are used to store the knowledge.

How do you implement an artificial neural network?

Artificial neural networks are either implemented on a general-purpose computer or are built into a dedicated hardware.

Is artificial neural network really a parallel computing machine when it is implemented by MATLAB on PC?

The power of parallel processing can only be realized by building the ANN into a circuit!

Is there any such hardware available now?

The first Cellular neural networks (CNN) processors were built in 1993 by Leon Chua at Berkeley. Currently, CNN processors can achieve up to 50,000 frames per second for image processing, and for certain applications such as missile tracking, flash detection, and spark-plug diagnostics these microprocessors have outperformed the conventional supercomputers.

Recently, IBM is building the brain chip!

What is the most important key word in neural networks?

Learning

What is learning in neural networks?

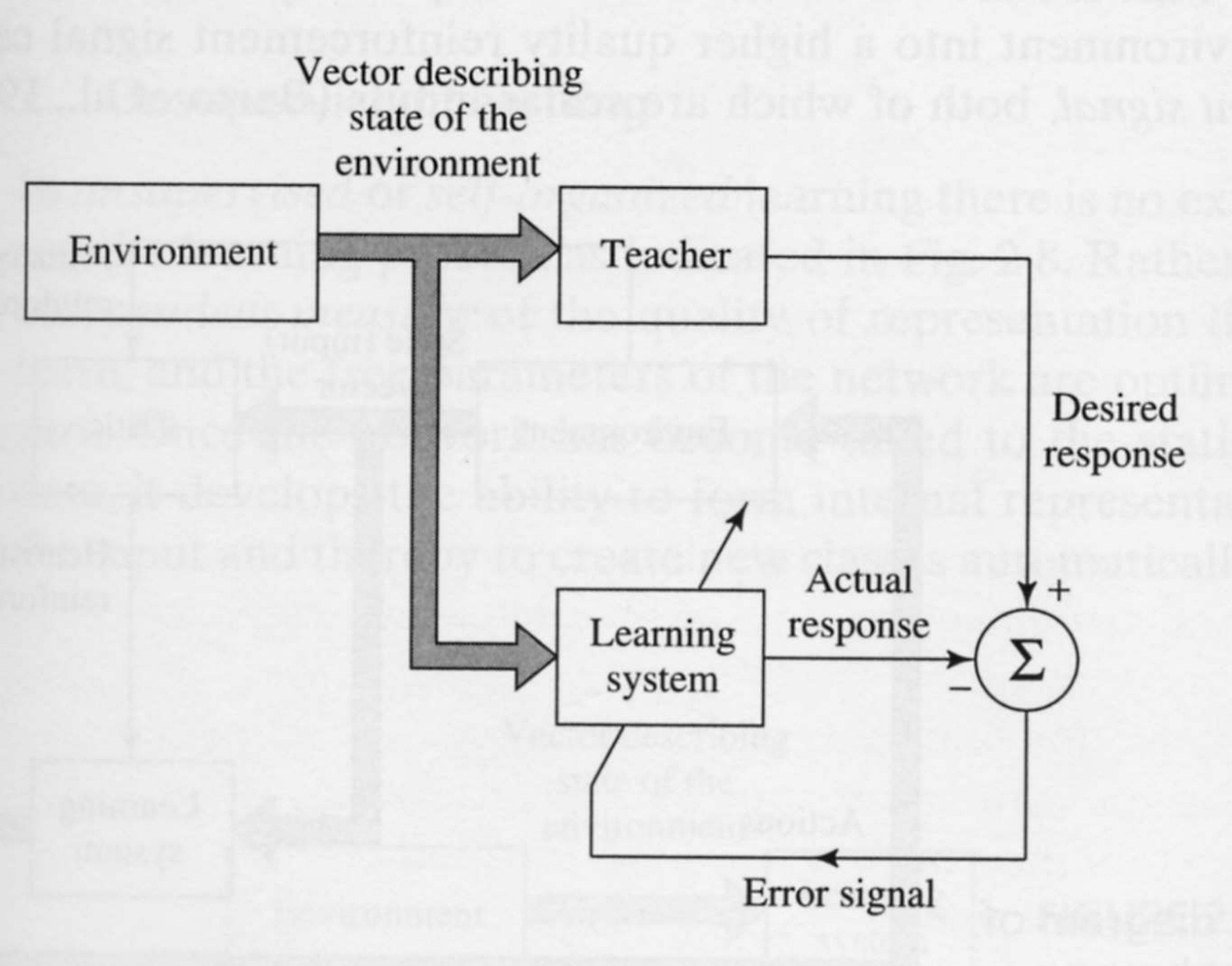
Learning is a process by which the free parameters (synaptic weights) of a neural network are adapted through a process of stimulation by the environment in which the network is embedded.

The type of learning is determined by the manner in which the parameter changes take place.

Process of learning:

1. *The neural network is stimulated by an environment.*
2. *The neural network undergoes changes in its free parameters, i.e. synaptic weights, as a result of this stimulation.*
3. *The neural network responds in a new way to the environment because of the changes that have occurred in its internal structure.*

Learning with a teacher: Supervised Learning



Process of learning:

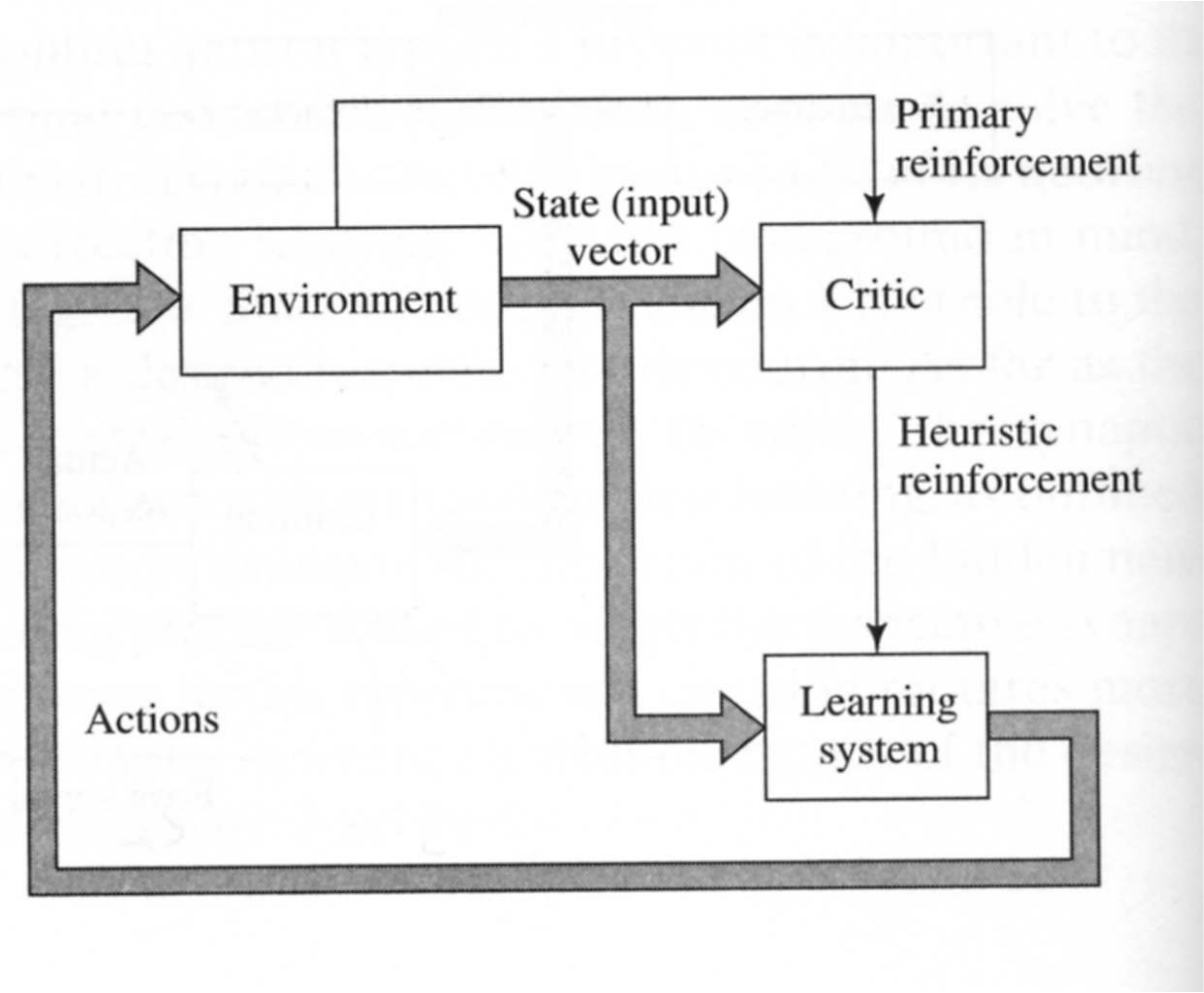
1. The neural network is fed with input, and produce an output.

2. The teacher will tell what the desired output should be, and the error signal is generated.

3. The weights are adjusted by the error signals.

Example: Automatic Flying Helicopter

Learning without a teacher: Reinforcement Learning



Process of learning:

1. *The neural network is interacting with the environment by taking various actions.*

2. *The learning system will be rewarded or penalized by its actions. But no explicit error signal is provided!*

3. *The weights are adjusted by the reinforcement signal.*

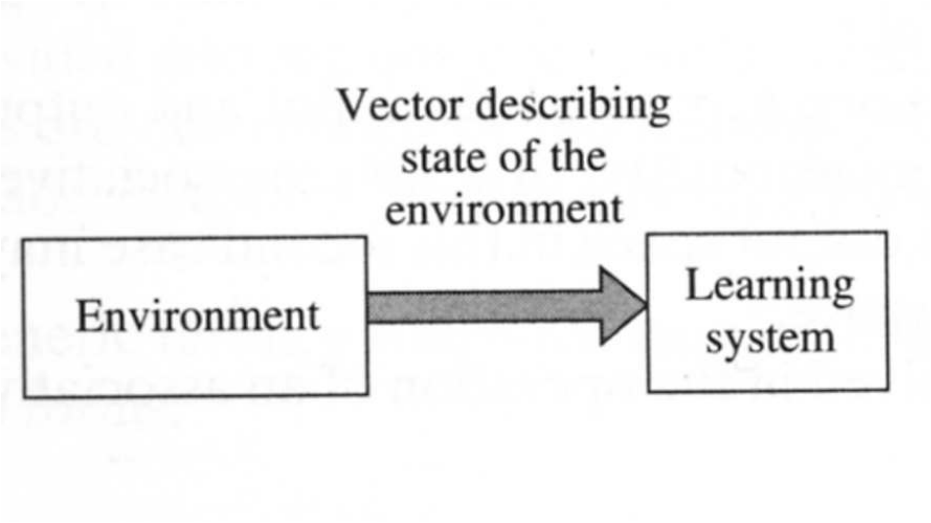
The term is borrowed from psychology

Real life example?

Learning Robot

Learning without a teacher:
Unsupervised or self-organized learning

Process of learning:



1. The neural network is fed with input.

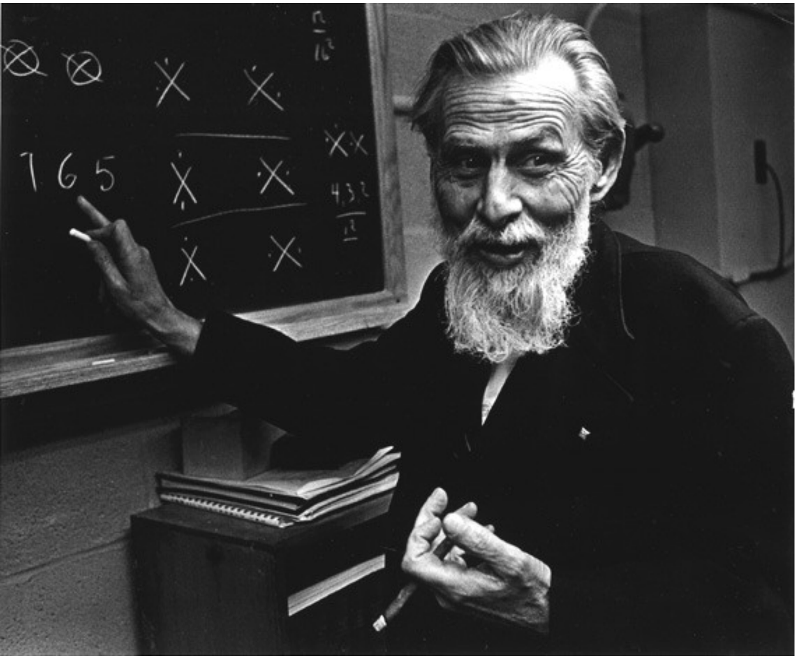
2. The weights are adjusted based upon the input signals only!

How can the system adjust the weights without any error or reward signal?

It depends upon the purpose of the learning system. It can perform different tasks such as principal components analysis and clustering.

Let’s start with the learning of the simplest neural network: perceptron

The beginning of the artificial neural networks: *McCulloch and Pitts*, 1943



Warren Sturgis McCulloch (1898-1969)

Walter Pitts (1923-1969)

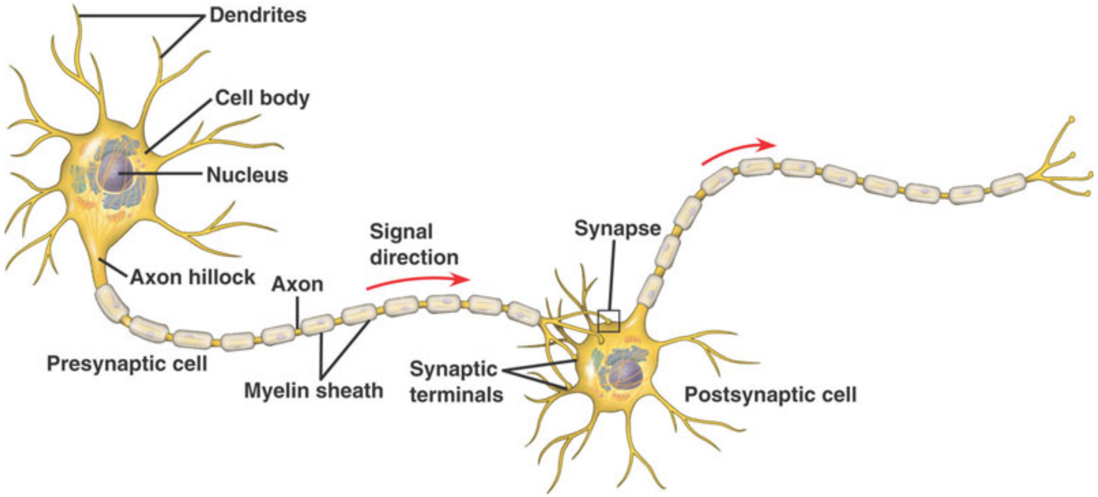
McCulloch was an American neurophysiologist. He studied philosophy and psychology at Yale (B.A. 1921) and Columbia (M.A. 1923). Receiving his MD in 1927, he was an intern at a Hospital before returning to academia in 1934.

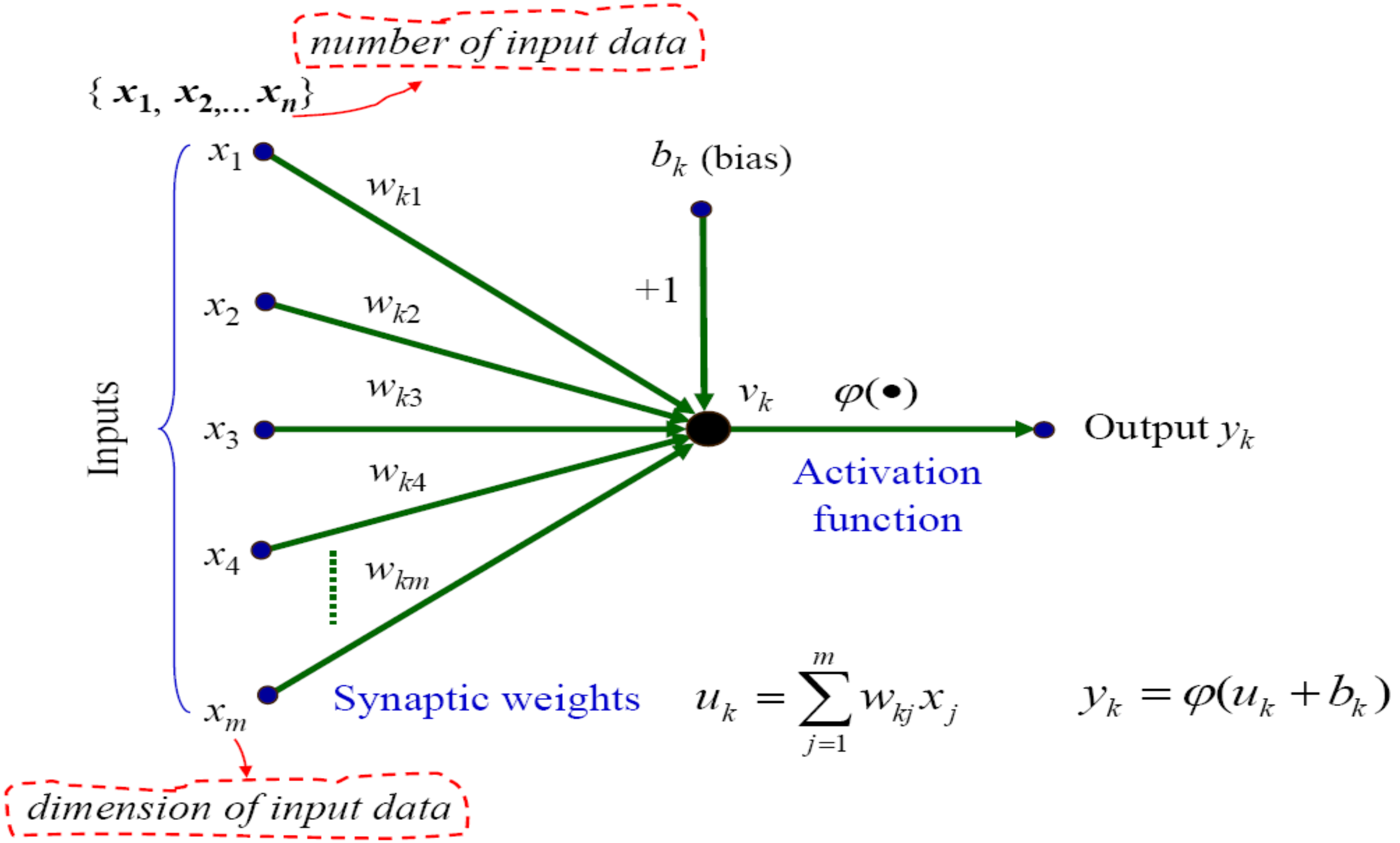
Pitts was a logician whose life was more colorful.

In early 1942, McCulloch invited Pitts, who was homeless, to live with his family. In the evenings McCulloch and Pitts collaborated. This led to their famous paper in 1943.

In 1951, Wiener set up a neuro-group at MIT with Pitts and McCulloch. Pitts wrote a thesis on the properties of neural nets connected in three dimensions. Pitts was described as an eccentric, refusing to allow his name to be made publicly available. He refused all offers of advanced degrees or official positions at MIT as he would have to sign his name.

Wiener suddenly turned against McCulloch because his wife Margaret Wiener hated McCulloch. He broke off relations with anyone connected to him including Pitts. This sent Walter Pitts into 'cognitive suicide'. He burnt the manuscript on three dimensional networks and took little further interest in work.

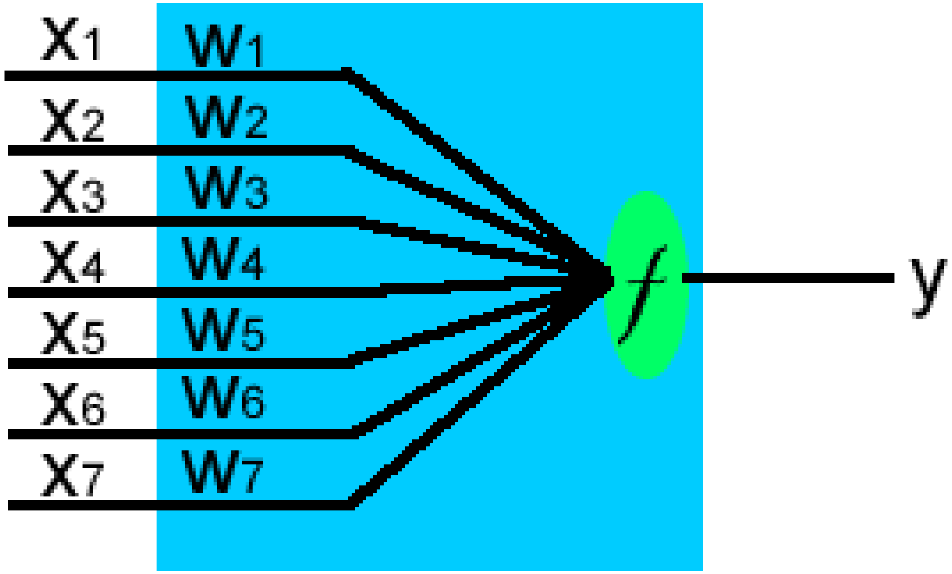




$$u_k = \sum_{j=1}^{m} w_{kj} x_j \qquad y_k = \varphi(u_k + b_k)$$

Perceptron—single layer neural networks

Frank Rosenblatt (1928-1969), at Cornell University in 1958.



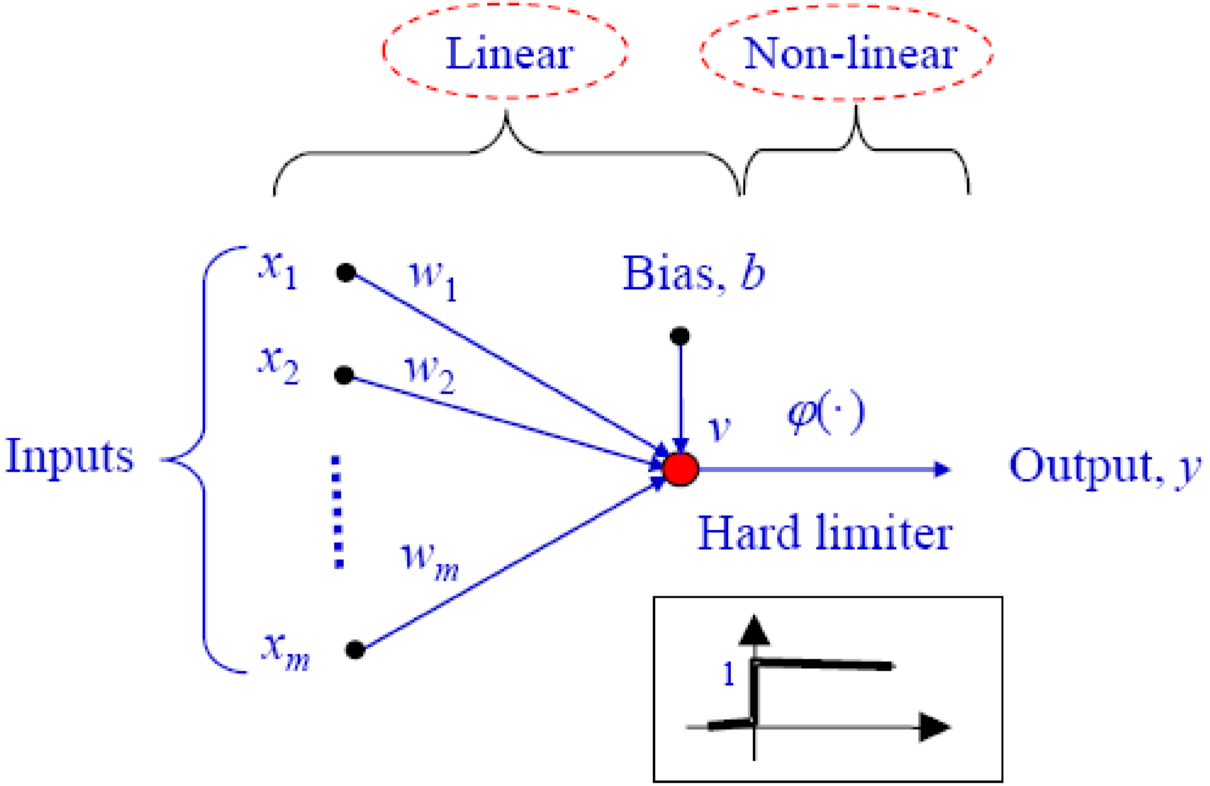
Perceptron was the first computer that could learn new skills by trial and error.

By the study of neural networks such as the Perceptron, Rosenblatt hoped that "the fundamental laws of organization which are common to all information handling systems, machines and men included, may eventually be understood."

Rosenblatt was a colorful character at Cornell in the early 1960s. A handsome bachelor, he drove a classic MGA sports car and was often seen with his cat named Tobermory. He enjoyed mixing with undergraduates, and for several years taught an interdisciplinary undergraduate honors course entitled "Theory of Brain Mechanisms" that drew students equally from Cornell's Engineering and Liberal Arts colleges.

Perceptron is built around the McCulloch-Pitts model.

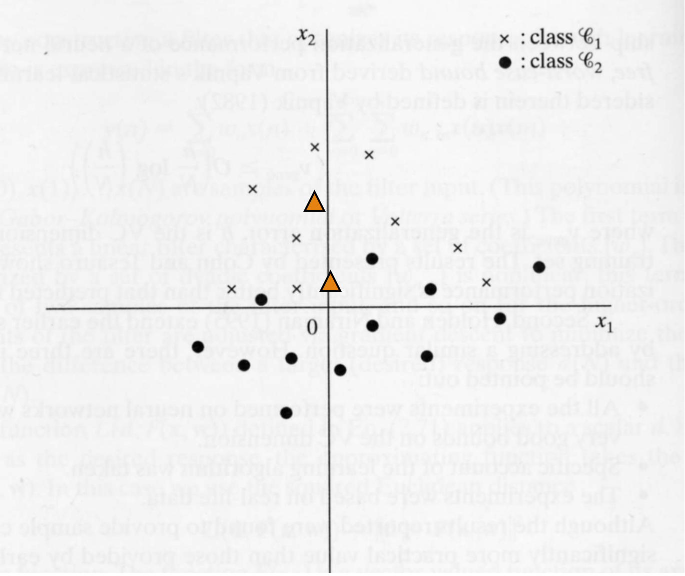
A Simple Perceptron



$$v = \sum_{i=1}^{m} w_i x_i + b$$

Perceptron is designed to solve the pattern recognition problem.

Goal: To correctly classify the set of externally applied stimuli $\boldsymbol{x}_1, \boldsymbol{x}_2, \ldots, \boldsymbol{x}_n$ into one of two classes, C_1 and C_2.



To simplify the mathematical notation, define:

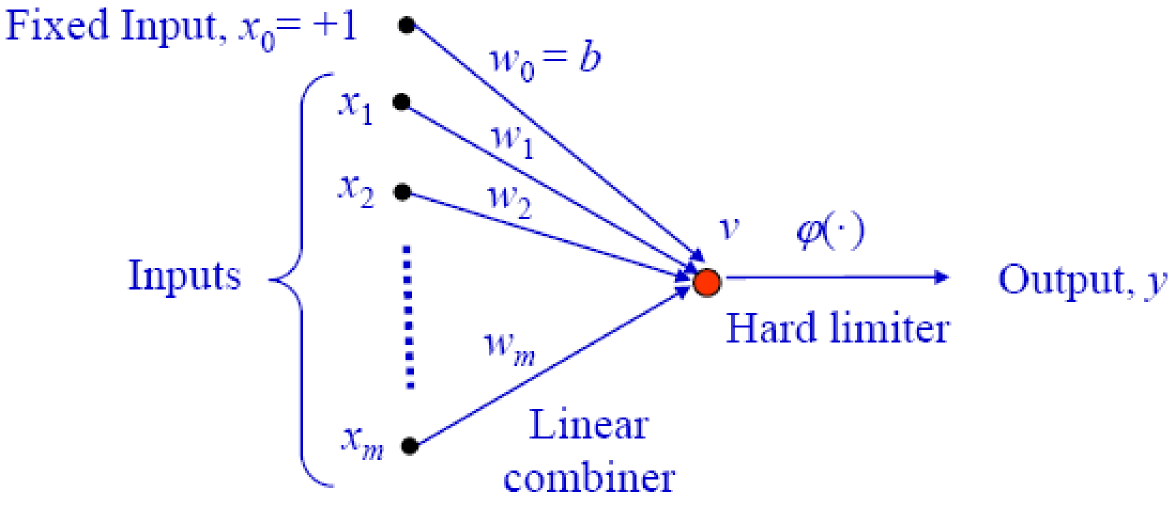
- the input vector:

$$x(n) = [+1, x_1(n), x_2(n), \cdots, x_m(n)]^T$$

- the weight vector:

$$w(n) = [b(n), w_1(n), w_2(n), \cdots, w_m(n)]^T$$

  - where n denotes the iteration step.



<u>At step n, the input vector x(n) is presented to the perceptron, how does the perceptron assign the label to the input pattern (pattern classification)?</u>

The process is very simple.

<u>What is the induced local field v(n)?</u>

$$v(n) = \sum_{i=1}^{m} w_i(n)x_i(n) + b(n) = \sum_{i=0}^{m} w_i(n)x_i(n) = w^T(n)x(n)$$

<u>What is the output of the neuron y(n)?</u>

$v(n) = w^T(n)x(n) > 0$ ⟹ **y(n)=1** ⟹ **class 1**

$v(n) = w^T(n)x(n) < 0$ ⟹ **y(n)=0** ⟹ **class 2**

<u>What if v(n) is ZERO?</u>

The induced local field v(n):

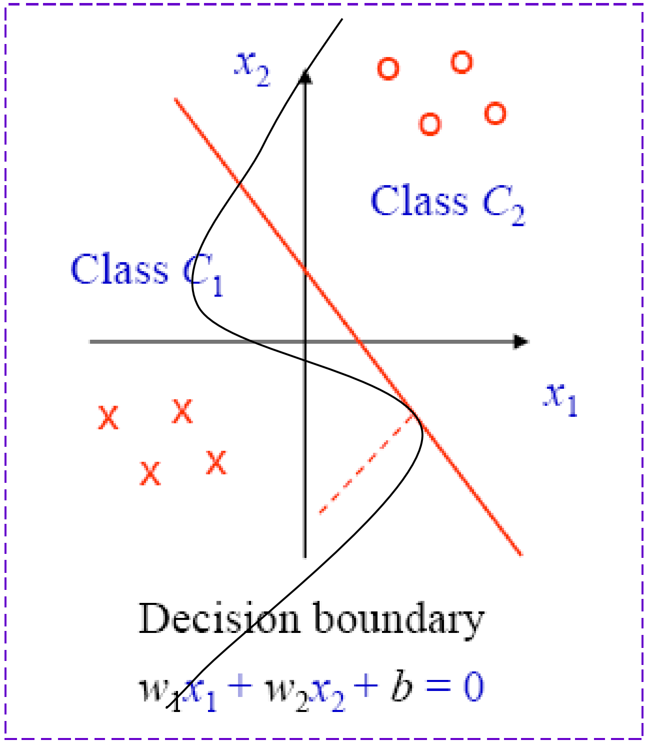
$$v(n) = \sum_{i=1}^{m} w_i(n)x_i(n) + b(n) = \sum_{i=0}^{m} w_i(n)x_i(n) = w^T(n)x(n)$$

What if v(n) is ZERO?

That corresponds to the decision boundary!

$$v(n) = w^T(n)x(n) = 0$$

$$w_1x_1 + w_2x_2 + \cdots w_m x_m + b = 0$$



What is the geometrical shape of the decision boundary?

If m=1 $w_1x_1 + b = 0$ $\Longrightarrow$ A point on a line

If m=2 $w_1x_1 + w_2x_2 + b = 0$ $\Longrightarrow$ A line in a 2-dimensional plane

If m=3 $w_1x_1 + w_2x_2 + w_3x_3 + b = 0$ $\Longrightarrow$ A plane in the 3-dimensional space

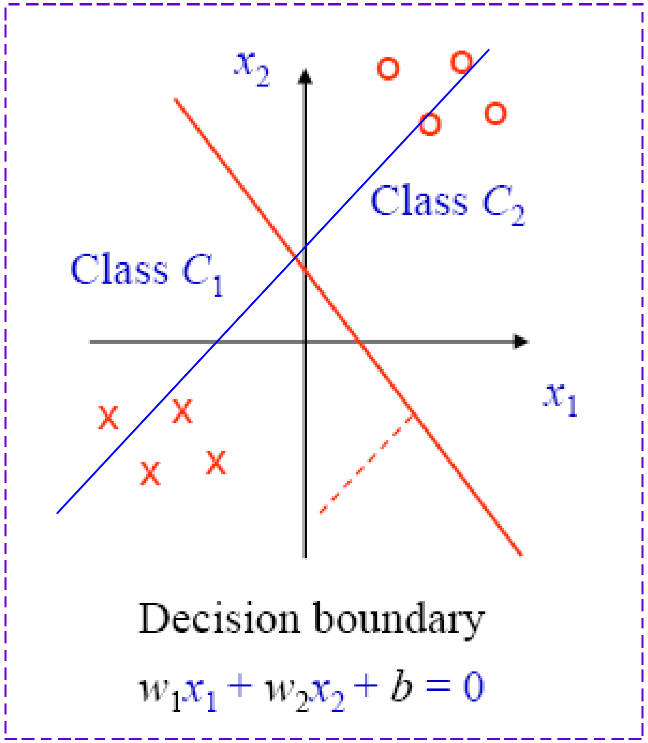
How about $w_1x_1 + w_2x_2 + \cdots w_m x_m + b = 0$? $\Longrightarrow$ A hyper-plane in the m-dimensional space (input vector space)

Is it possible for the perceptron to produce a decision boundary that is a nonlinear curve?

No.

Given the synaptic weights and bias, how do you find out the decision boundary produced by the perceptron?

$$w^T x = w_1 x_1 + w_2 x_2 + \cdots w_m x_m + b = 0$$



Decision boundary

$w_1 x_1 + w_2 x_2 + b = 0$

In order for the perceptron to perform a desired task, its synaptic weights (including the bias) must be properly selected. Otherwise, it may not do the job correctly.

The Key Question: How to choose the proper weights?

In general, two basic methods can be employed to select a suitable weight vector:

- By off-line calculation of weights (without learning). If the problem is relatively simple, it is often possible to calculate the weight vector from the specification of the problem.
- By learning procedure. The weight vector is determined from a given (training) set of input-output vectors (exemplars) in such a way to achieve the best classification of the training vectors.

Let's start with the simple method: Selection of weights by off-line calculations.

Selection of weights by off-line calculations - Example

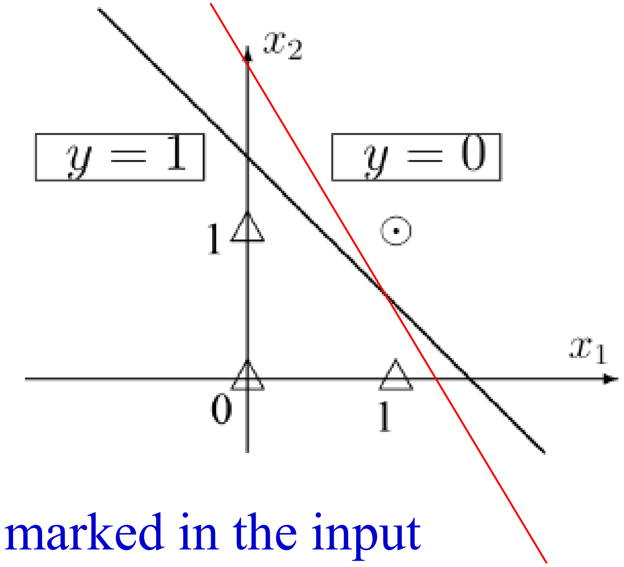
- Consider the problem of building the NAND gate using a perceptron.

Truth table of NAND

x_1	0	0	1	1
x_2	0	1	0	1
y	1	1	1	0

Can you formulate it as pattern classification problem?

Three points (0,0), (0,1) and (1,0) belong to one class. And (1,1) belongs to another class.



- The input patterns (vectors) belong to two classes and are marked in the input space {the plane (x_1, x_2)} with △ and ⊙, respectively. The decision boundary is the straight line described by the following equation

$$x_2 = -x_1 + 1.5 \qquad \text{or} \qquad -x_1 - x_2 + 1.5 = 0$$

Is the decision line unique for this problem?

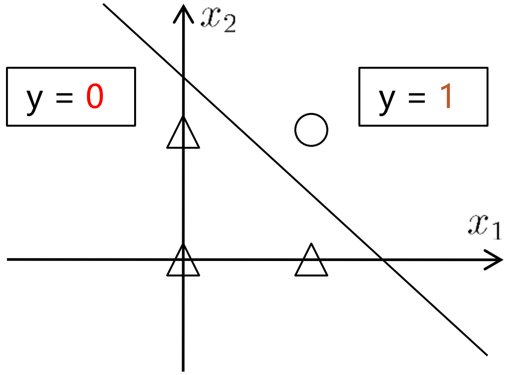
- Let's select w = (1.5, −1, − 1).
  For each input vector, the output signal is now calculated as

$$v = \begin{bmatrix} 1.5 & -1 & -1 \end{bmatrix} \begin{bmatrix} 1 \\ x_1 \\ x_2 \end{bmatrix} \quad ; \quad y = \begin{cases} 0 & if\ v < 0 \\ 1 & if\ v \geq 0 \end{cases}$$

Example: What if we have the following truth table (AND gate)?

x_1	0	0	1	1
x_2	0	1	0	1
y	1	1	1	0

x_1	0	0	1	1
x_2	0	1	0	1
y	0	0	0	1



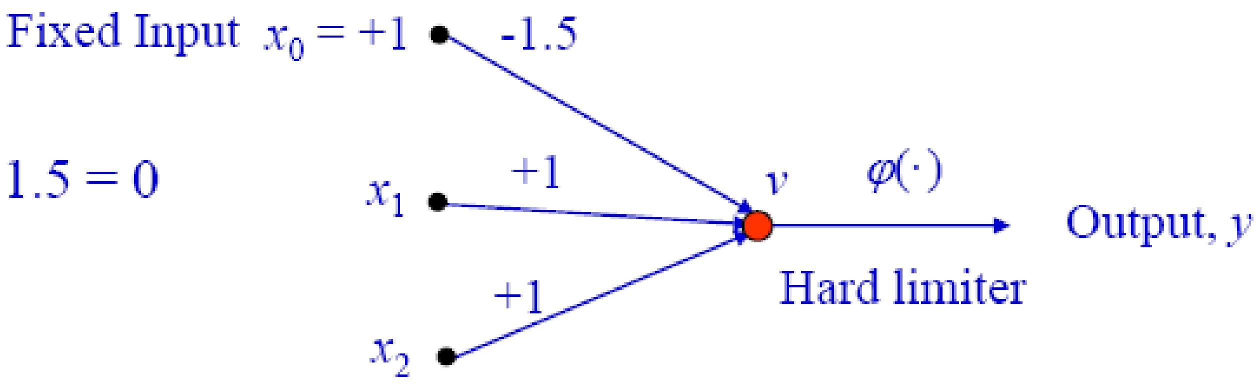
w = (1.5, −1, − 1).

Is the decision boundary the same as that for NAND gate? Yes

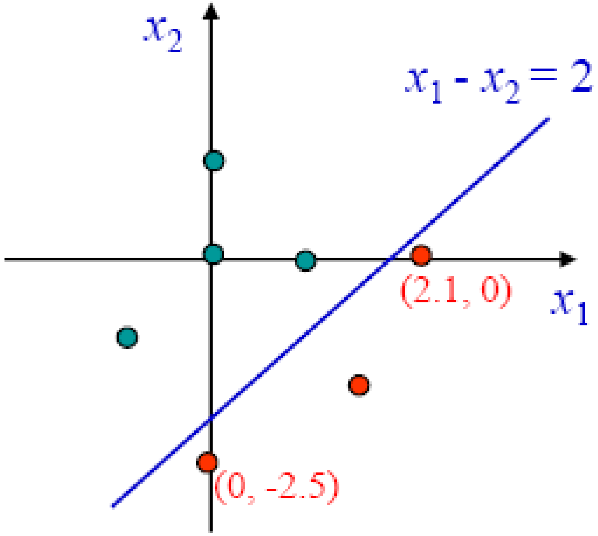
Can we use the same weights? No. The output values of the logic gate is important!

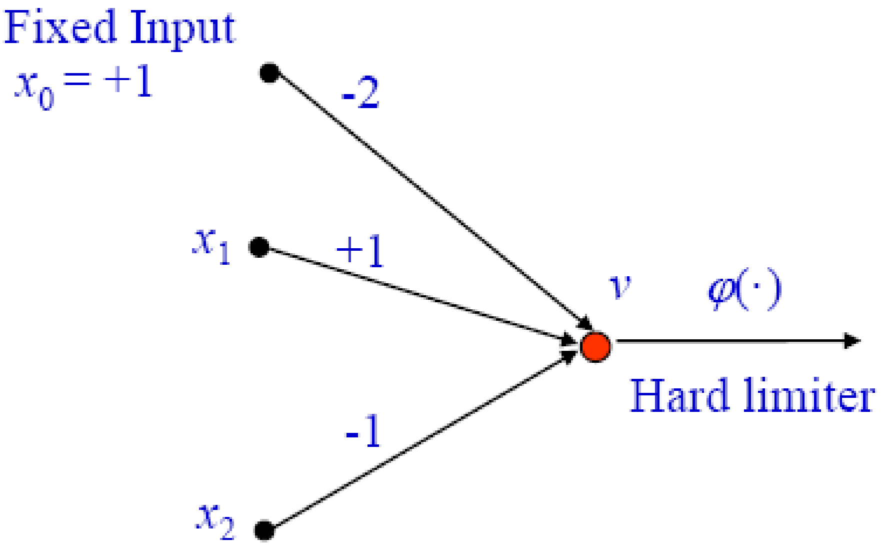
What are the proper weights? w’= -w = (-1.5, +1, +1).

$$x_1 + x_2 - 1.5 = 0$$



Example: Consider 2-dimensional samples (0, 0), (0, 1), (1, 0), (-1, -1) that belong to one class, and samples (2.1, 0), (0, -2.5), (1.6, -1.6) that belong to another class. A two-class partition of the input space and a perceptron that separates samples of opposite classes are as follows:





Output = 1 if $(x_1 - x_2 - 2) \geq 0$ and 0 otherwise.

<u>Can we use (2,-1,+1) as the synaptic weights?</u>

For pattern recognition problem, the labels are user-defined and flexible.

For logic gate, the outputs are fixed!

Definition: Linearly Separable

If two classes can be separated by one line (plane or hyper-plane in higher dimensional space).

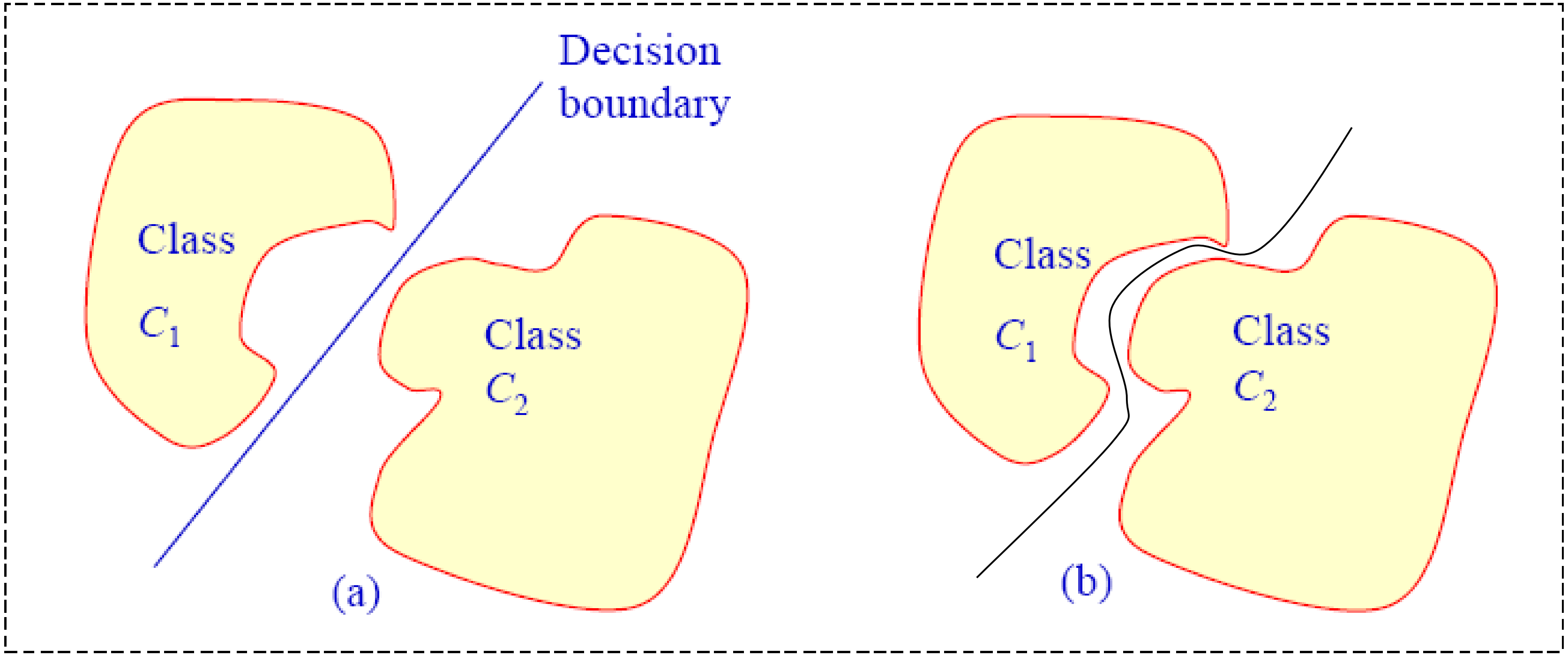


Figure: (a) A pair of linearly separable patterns; (b) A pair of non-linearly separable patterns.

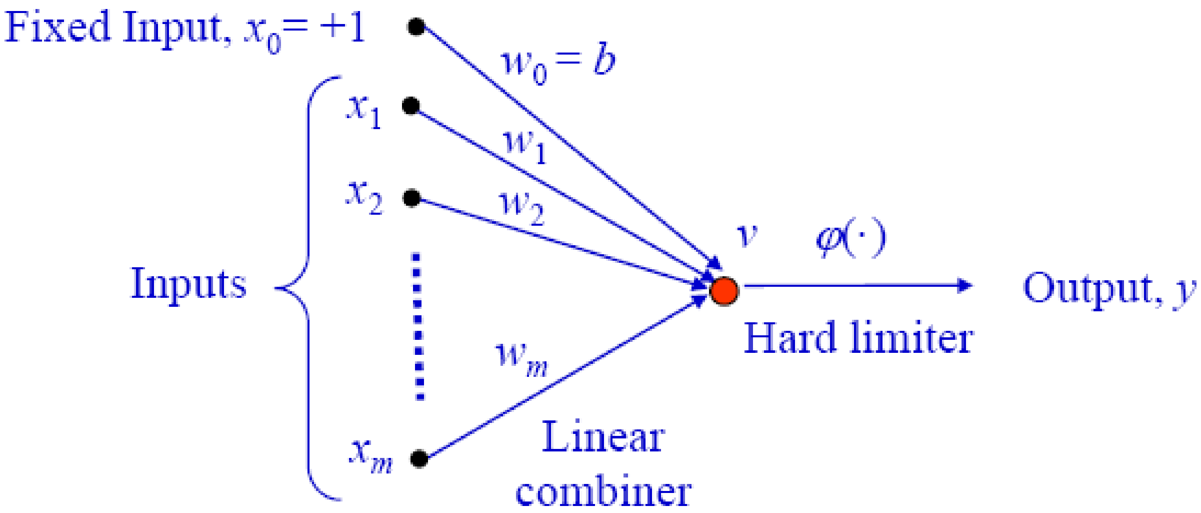
Two classes are *linearly separable* if and only if there exists a weight vector *w* based on which the perceptron can correctly perform the classification.

If the decision line (hyper-plane) is provided, can you always find a perceptron to perform the task?

1. Try to find out the equation for the hyper-plane and put it in the following form

$$w_1x_1 + w_2x_2 + \cdots w_mx_m + b = 0$$

2. Then the corresponding perceptron can be easily constructed.



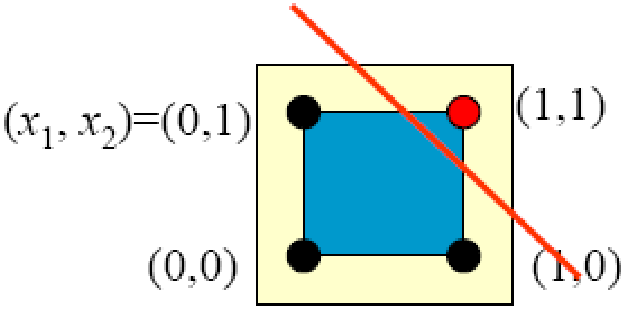
But what if the samples are *NOT linearly separable* (i.e. no straight line can possibly separate samples belonging to two classes)? Can you find out a perceptron to perform the task correctly?

There CANNOT be any simple perceptron that achieves the classification task.

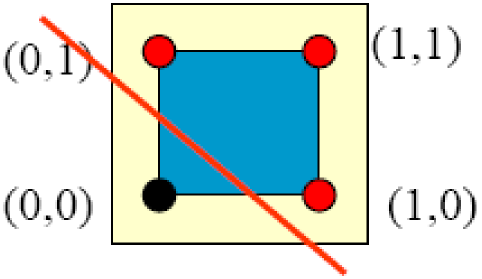
 Fundamental limitation of simple perceptrons (Minsky 1969).

Example: Linearly separable?

- 2-input "AND" logic function

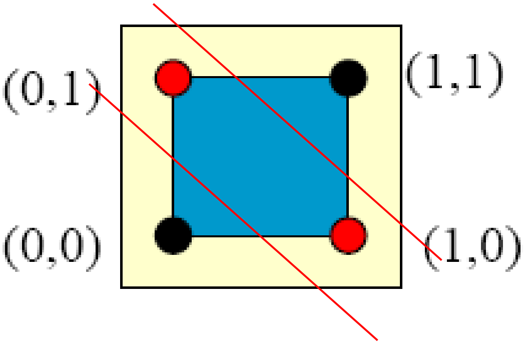


- 2-input "OR" logic function



$x_1 + x_2 - 0.5 = 0$

- XOR function



Can they be separated by one line? No.

How about two lines? Yes.

Can a single perceptron produce a decision boundary with two lines? No.

How about the linearly separable case where the dimension of the input is high?

Hyper-plane in *m* dimensions is defined by the equation

- $w_0 + w_1x_1 + w_2x_2 + \ldots + w_mx_m = 0$
- Dividing *m*-dimensional space into two regions

$w_0 + w_1x_1 + w_2x_2 + \ldots + w_mx_m > 0$ $\Longrightarrow$ **class 1**

$w_0 + w_1x_1 + w_2x_2 + \ldots + w_mx_m < 0$ $\Longrightarrow$ **class 2**

For 2-dimensional input spaces, it is easy to determine by *geometric construction* whether two classes are linearly separable and to find a two-class classifier.

Can you do the similar thing for 3-dimensional input space? Still possible!

Is it easy to visualize 4 or higher dimensional space and draw the hyper-plane to separate the classes?

It is not so straightforward for 4- dimensional input spaces and beyond.

In real world applications, the dimension of the input pattern is very high. For instance the dimension of a 100x100 sized image is 10000.

How to find out the decision boundary in 10000-dimensional space?

In 1958, Rosenblatt demonstrated that Perceptron can learn to do the job correctly!

Perceptron was the first computer that could learn new skills simply by trial and error.

Perceptron Learning

Now, suppose that the input variables of the perceptron originate from 2 *linearly separable* classes C_1 and C_2.

We know if C_1 and C_2 are linearly separable, there must exist at least one weight vector $\boldsymbol{w_0}$ such that

$$w_0^T x > 0 \qquad \text{for } \forall\, \boldsymbol{x} \in C_1 \qquad \textbf{(1)}$$

$$w_0^T x < 0 \qquad \text{for } \forall\, \boldsymbol{x} \in C_2 \qquad \textbf{(2)}$$

In other words, we know there is a solution. But where is it?

Training problem: Find a weight vector $\boldsymbol{w}$ such that the perceptron can correctly classify the training set X.

But how?

It turns out that it can be found out simply by trial and error!

Let's try to figure out the learning algorithm step by step.

Feed a pattern x to the perceptron with weight vector w, it will produce a binary output y (1 or 0, **i.e. Fire or Not Fire**).

First consider the case, $v = w^T x < 0$ $\Longrightarrow$ $y = 0$

If the correct label (all the labels of the training samples are known, i.e. there is a teacher!) is d=0; should we update the weights?

If it does not break, do not fix it.

If the desired output is d=1, should we update the weights?
Of course. Assume the new weight vector is w', then we have

$$w' = w + \Delta w$$

But how to choose Δw ?

Let's calculate the induced local field of the updated one: $v' = w'^T x$

Should we increase the induced local field or decrease?

$$v' - v = w'^T x - w^T x = \Delta w^T x > 0$$

Given a vector x, what is the simplest way to choose Δw such that $\Delta w^T x > 0$?

$$\Delta w = x$$

To avoid big jump of the weights, let's put a small learning rate

$\Delta w = \eta x, \eta > 0$ $\Longrightarrow$ $\Delta w^T x = \eta x^T x > 0$

If the true label is d=1, and the perceptron makes a mistake, its synaptic weights are adjusted by

$$w' = w + \eta x$$

Now consider the case,

$$v = w^T x > 0 \quad \Longrightarrow \quad y = 1$$

If the correct label is d=1; should we update the weights?

We will only adjust the weights when the perceptron makes mistakes (d=0).
Assume the new weight vector is w', then we have

$$w' = w + \Delta w$$

But how to choose Δw ?

Let's calculate the induced local field v'=w'^{T}x. Should we increase the induced local field or decrease?

$$v' - v = w'^T x - w^T x = \Delta w^T x < 0$$

What is the simplest way to choose Δw such that $\Delta w^T x < 0$?

$$\Delta w = -\eta x \quad \Longrightarrow \quad \Delta w^T x = -\eta x^T x < 0$$

If the true label is d=0, and the perceptron makes a mistake, its synaptic weights are adjusted by

$$w' = w - \eta x$$

Let’s summarize what we have now

If the true label is d=1, and the perceptron makes a mistake, its synaptic weights are adjusted by

$$w' = w + \eta x$$

If the true label is d=0, and the perceptron makes a mistake, its synaptic weights are adjusted by

$$w' = w - \eta x$$

It seems that these two learning algorithms take different forms.

Let’s try to unify these two algorithms into one algorithm.

Let’s consider the error signal: e=d-y.

What is the error signal when d=1?

e=d-y=1-0=1! ⇒ $w' = w + \eta x$

What is the error signal when d=0?

e=d-y=0-1=-1! ⇒ $w' = w - \eta x$

How to unify these two algorithms with the help of the error signal, e?

$$w' = w + \eta e x$$

Perceptron Learning Algorithm

```
Start with a randomly chosen weight vector *w(1)*;

while there exist input vectors that are misclassified by *w(n)*
Do  Let *x(n)* be a misclassified input vector;
    Update the weight vector to
        $w(n+1) = w(n) + \eta e(n) x(n)$
        $e(n) = d(n) - y(n)$
    where $\eta > 0$ and $d = \begin{cases} 1 \text{ if } x \text{ belongs to class } C_1 \\ 0 \text{ if } x \text{ belongs to class } C_2 \end{cases}$
    Increment $n$
end-while
```

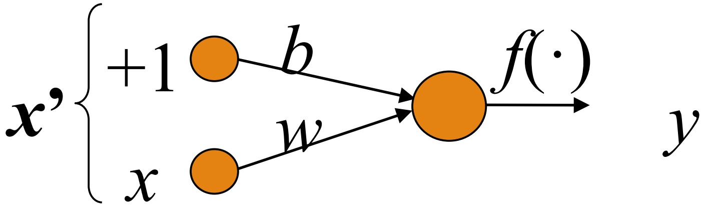
Imagine that you were a student at 1958, would you be able to discover this simple algorithm as Rosenblatt did?

Creating a proper problem is as important as solving a problem, and sometimes even more important!

Example: Let us consider a simple classification problem where the input space is one-dimensional space, i.e., a real line:

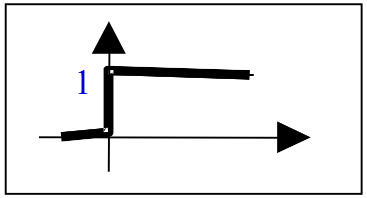
Class 1 (d = 1) : x = 0.5, 2

Class 2 (d = 0) : x = -1, -2



The activation function f is the threshold function, i.e.

$$f(v) = \begin{cases} 1 & \text{if } v \ge 0 \\ 0 & \text{otherwise} \end{cases}$$



Solution:

Let's choose η = 1 and $\boldsymbol{w}'_0 = [-1.5, -2]^T$

Order of presentation: $\boldsymbol{x}' = (1, 0.5)^T, (1, -1)^T, (1, 2)^T, (1, -2)^T$

$$y = f\left(\begin{bmatrix} -1.5 & -2 \end{bmatrix} \begin{pmatrix} 1 \\ 0.5 \end{pmatrix} \right) = 0 \neq (d = 1)$$

Iteration 1: When $\boldsymbol{x}' = (1, 0.5)^T$,

$$\boxed{\boldsymbol{w}'_1 = \boldsymbol{w}'_0 + \eta(1-0)\boldsymbol{x}'} \qquad \boldsymbol{w}'_1 = \boldsymbol{w}'_0 + \begin{pmatrix} 1 \\ 0.5 \end{pmatrix} = \begin{pmatrix} -0.5 \\ -1.5 \end{pmatrix}$$

Iteration 2: When $\boldsymbol{x}' = (1, -1)^T$,

$$y = f\left(\begin{bmatrix}-0.5 & -1.5\end{bmatrix}\begin{pmatrix}1\\-1\end{pmatrix}\right) = 1 \neq (d = 0)$$

$$\boxed{\boldsymbol{w}'_2 = \boldsymbol{w}'_1 + \eta(0 - 1)\boldsymbol{x}'}$$

$$\boldsymbol{w}'_2 = \boldsymbol{w}'_1 - \begin{pmatrix}1\\-1\end{pmatrix} = \begin{pmatrix}-1.5\\-0.5\end{pmatrix}$$

Iteration 3: When $\boldsymbol{x}' = (1, 2)^T$,

$$y = f\left(\begin{bmatrix}-1.5 & -0.5\end{bmatrix}\begin{pmatrix}1\\2\end{pmatrix}\right) = 0 \neq (d = 1)$$

$$\boxed{\boldsymbol{w}'_3 = \boldsymbol{w}'_2 + \eta(1 - 0)\boldsymbol{x}'}$$

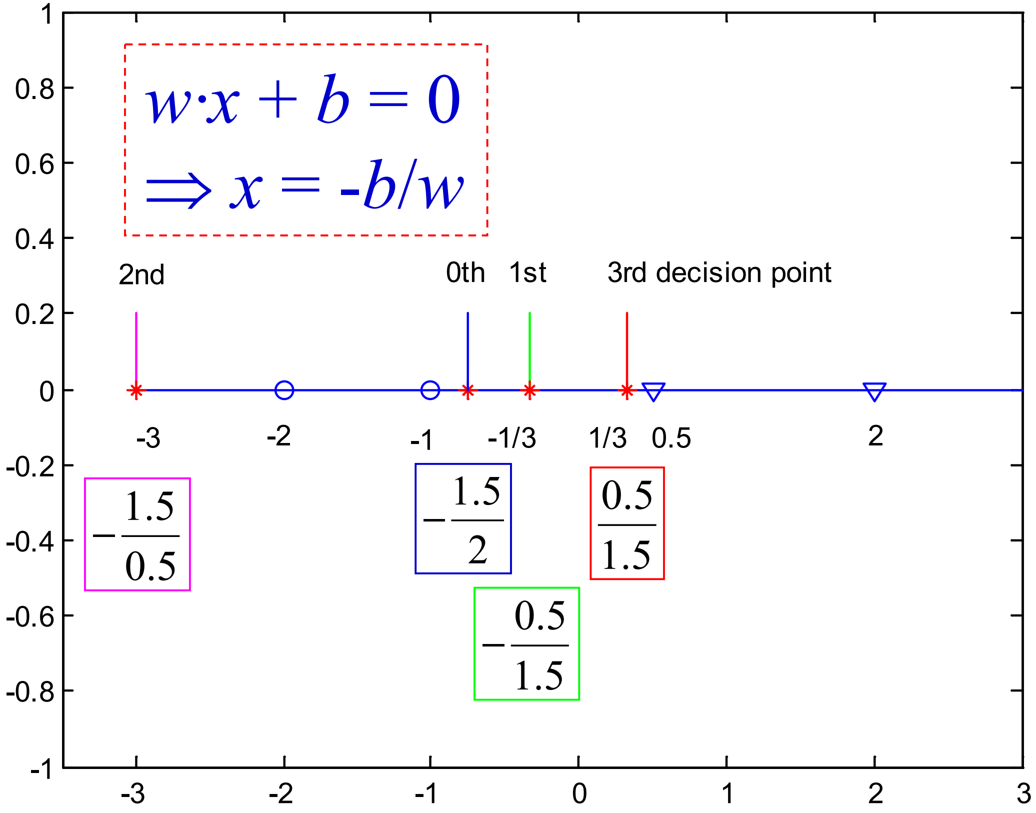
$$\boldsymbol{w}'_3 = \boldsymbol{w}'_2 + \begin{pmatrix}1\\2\end{pmatrix} = \begin{pmatrix}-0.5\\1.5\end{pmatrix}$$

Iteration 4: When $\boldsymbol{x}' = (1, -2)^T$,

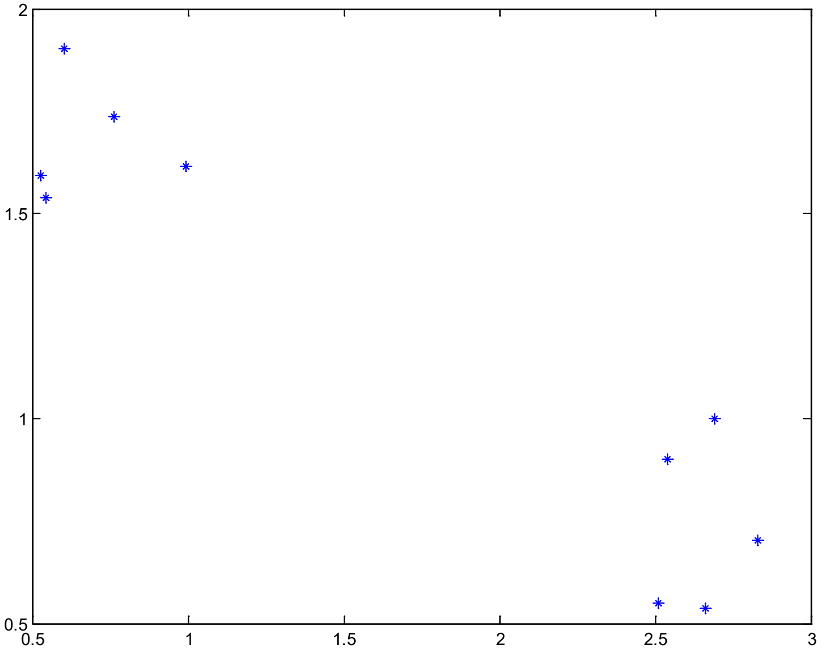
$$y = f\left(\begin{bmatrix}-0.5 & 1.5\end{bmatrix}\begin{pmatrix}1\\-2\end{pmatrix}\right) = 0 = d$$

$$\boldsymbol{w}'_4 = \boldsymbol{w}'_3$$

- We now completed the first *epoch*. We then reuse the training data and repeat the above training procedure, and it shows that the algorithm actually converges after the first epoch (i.e., $\boldsymbol{w}'_4$ is the solution) for this problem.



- **Example:** There are two classes of patterns surrounding (1, 2) and (3, 1) as shown in the following figure. For each class there is 5 data points:



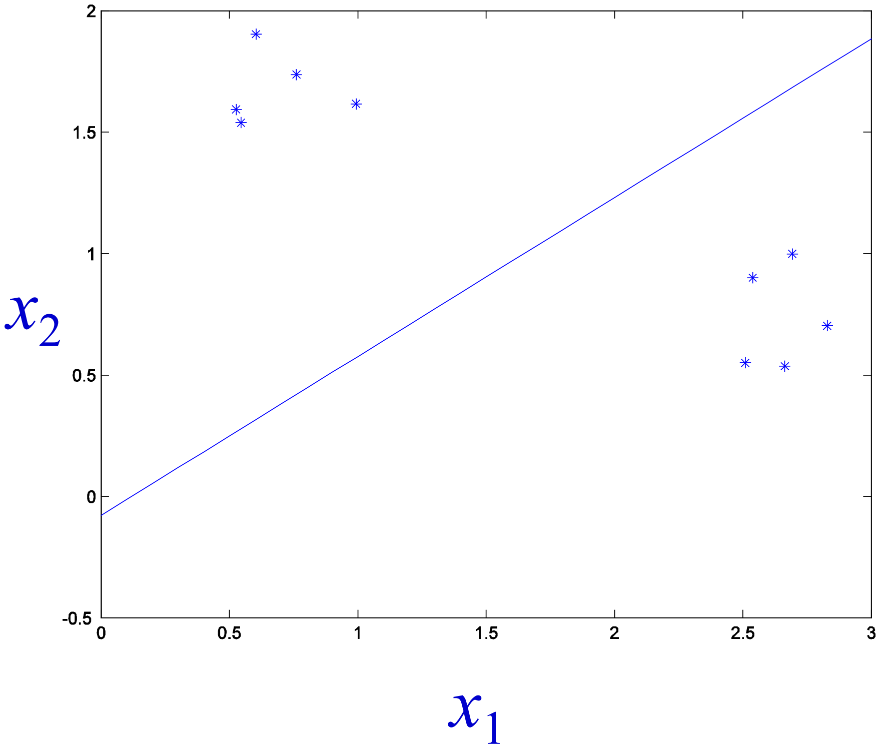
- Using the perceptron learning algorithm, *after 100 epochs*, the weight vector is

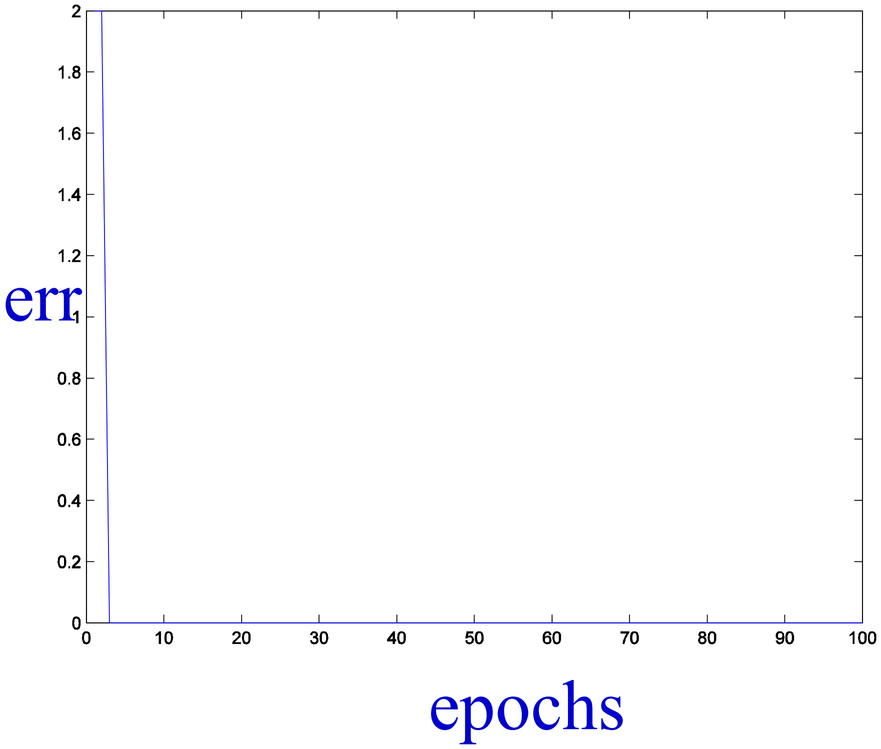
$$w' = \begin{bmatrix} 0.0041 \\ -0.0349 \\ 0.0534 \end{bmatrix}$$

- Thus the decision line is computed as

$$0.0041 - 0.0349x_1 + 0.0534x_2 = 0$$

- The classification result is shown below,



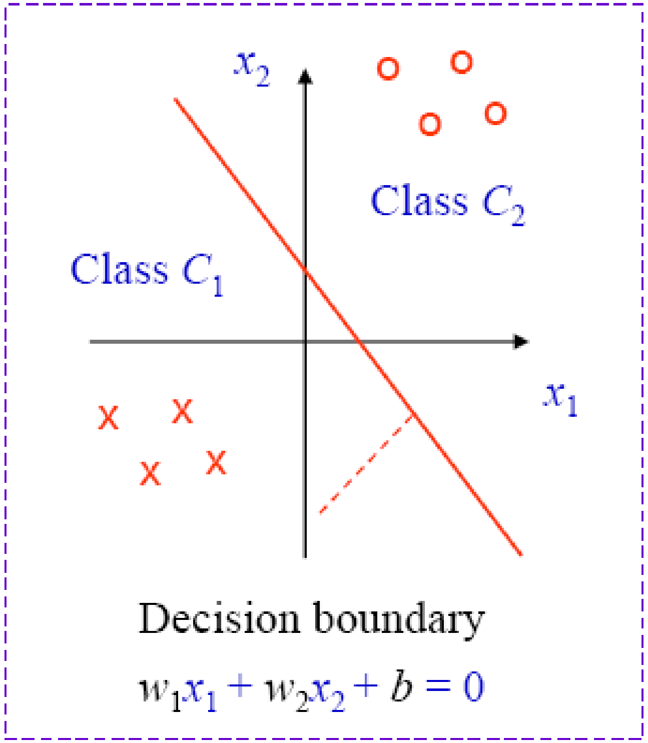


Does it always converge?

Frank Rosenblatt took another big step in mathematically proving the convergence in 1962, 4 years after the algorithm was proposed.

Perceptron Convergence Theorem: (Rosenblatt, 1962)

If C_1 and C_2 are linearly separable, then the perceptron training algorithm "converges" in the sense that after a *finite number* of steps, ***the synaptic weights*** remain unchanged and the perceptron correctly classifies all elements of the training set.



The Proof is too beautiful to skip!

We will only prove the case when w(1)=0, and η =1.

$$w(n+1) = w(n) + e(n)x(n)$$

What do we know?

There exists w_0, such that

For samples in Class C_1, $w_0^T x(k) > 0$

For samples in Class C_2, $w_0^T x(k) < 0$

What do we need to prove?

After a finite number of steps, the weights stop changing.

If the weights stop changing, does it automatically imply that the perceptron can classify all the training patterns correctly?

Yes.

Proof: $w(n+1) = w(n) + e(n)x(n)$ w(1)=0

$$w(2) = w(1) + e(1)x(1) = e(1)x(1)$$

$$w(3) = w(2) + e(2)x(2) = e(1)x(1) + e(2)x(2)$$

$$w(n+1) = w(n) + e(n)x(n) = e(1)x(1) + e(2)x(2) + \cdots + e(n)x(n)$$

Question: If n →infinity, will |w(n)| blow up to infinity?

Let's figure out the answer from two ways:

Let's first project the weights along the direction of w_0 .

Multiply w(n+1) by w_0^T

$$w_0^T w(n+1) = e(1)w_0^T x(1) + e(2)w_0^T x(2) + \cdots + e(n)w_0^T x(n)$$

Let's check out all the terms on the right side

$$e(k)w_0^T x(k) = (d(k) - y(k))w_0^T x(k)$$

If $w_0^T x(k) > 0 \Longrightarrow d(k) = 1, y(k) = 0 \Longrightarrow e(k)w_0^T x(k) = |w_0^T x(k)| > 0$

If $w_0^T x(k) < 0 \Longrightarrow d(k) = 0, y(k) = 1 \Longrightarrow e(k)w_0^T x(k) = |w_0^T x(k)| > 0$

Is it possible that e(k)=0?

Yes, but nothing would happen if e(k)=0. Therefore we just skip these steps.

Now let's find out the lower bound of the magnitudes of all possible $w_0^T x(i)$

$\alpha = \min\{|w_0^T x(i)|\}$ for all input patterns x(i) in the training set

Is it possible that $\alpha = 0$? No, we assume that no samples lie on the boundary.

Perceptron Convergence Theorem: (Rosenblatt, 1962)

$$w_0^T w(n+1) = e(1)w_0^T x(1) + e(2)w_0^T x(2) + \cdots + e(n)w_0^T x(n) = \sum_{i=1}^{n} | w_0^T x(i) |$$

$$\alpha = \min\{| w_0^T x(i) |\} \quad \Longrightarrow \quad w_0^T w(n+1) \geq n\alpha$$

Using the Cauchy-Schwarz inequality

$$\| x \| \| y \| \geq | x^T y |$$ For any vectors x and y

$$\| w_0 \| \| w(n+1) \| \geq \| w_0^T w(n+1) \| \geq n\alpha \quad \Longrightarrow \quad \| w(n+1) \| \geq \frac{n\alpha}{\| w_0 \|}$$

We can see if the synaptic weights keep on changing forever,

$$n \to \infty \quad \Longrightarrow \quad \| w(n+1) \| \to \infty$$

The crucial step in reaching this stage is multiplying w(n+1) by w_0^T

That step is not trivial at all. That's why it took another 4 years for Frank to figure out the convergence proof!

Of course, the proof is not over yet. We need to show that the weights will certainly not grow to infinity.

Perceptron Convergence Theorem: (Rosenblatt, 1962)

The second way to check whether the weights blow up or not:

We need to figure out how the magnitudes of the weights change with time directly:

$$\| w(n+1) \|^2 = w^T(n+1)w(n+1) = (w(n)+e(n)x(n))^T(w(n)+e(n)x(n))$$

$$= w^T(n)w(n) + 2e(n)w^T(n)x(n) + e^2(n)x^T(n)x(n)$$

Let's check out the middle term

$$e(n)w(n)^T x(n) = (d(n)-y(n))w^T(n)x(n)$$

If $w^T(n)x(n) > 0 \Longrightarrow y(n)=1, d(n)=0 \Longrightarrow e(n)w^T(n)x(n) < 0$

If $w^T(n)x(n) < 0 \Longrightarrow y(n)=0, d(n)=1 \Longrightarrow e(n)w^T(n)x(n) < 0$

$$\| w(n+1) \|^2 - \| w(n) \|^2 = 2e(n)w^T(n)x(n) + \| x(n) \|^2 \leq \| x(n) \|^2$$

$$\| w(2) \|^2 - \| w(1) \|^2 \leq \| x(1) \|^2$$

$$\| w(3) \|^2 - \| w(2) \|^2 \leq \| x(2) \|^2$$

$$\| w(n+1) \|^2 - \| w(n) \|^2 \leq \| x(n) \|^2$$

Summing up all the inequalities $\Longrightarrow \| w(n+1) \|^2 - \| w(1) \|^2 \leq \sum_{i=1}^{n} | x(i) \|^2$

Let $\beta = \max\{\| x(i) \|^2\}$ for all input patterns x(i) in the training set

$$\| w(n+1) \|^2 \leq n\beta$$

Perceptron Convergence Theorem: (Rosenblatt, 1962)

On one hand, we have

$$\| w(n+1) \|^2 \le n\beta$$

On the other hand, we have

$$\| w(n+1) \| \ge \frac{n\alpha}{\| w_0 \|}$$

Overall, $\Longrightarrow$ $\dfrac{n^2\alpha^2}{\| w_0 \|^2} \le \| w(n+1) \|^2 \le n\beta$

Note that α, β, w_0 are all constants.

If the synaptic weights keep on changing forever, $n \to \infty$

Is it possible for above inequality to hold?

Impossible!

So the synaptic weights will stop changing in finite time!

Reductio ad absurdum (proof by contradiction)

G.H. Hardy (1877-1947)

English mathematician

"Reductio ad absurdum, which Euclid loved so much, is one of a mathematician's finest weapons. It is a far finer gambit than any chess gambit: a chess player may offer the sacrifice of a pawn or even a piece, but a mathematician offers *the game*."

Quoted from "A Mathematician's Apology", G.H. Hardy, 1940.

Euclid

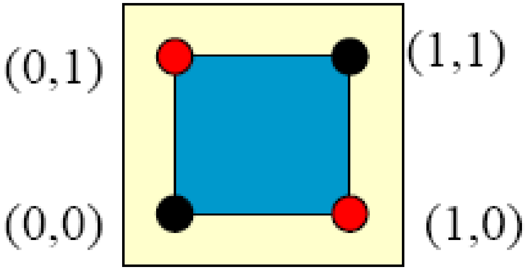
Euclid used this method to prove the existence of an infinity of prime numbers.

Pythagoras used this weapon to show that $\sqrt{2}$ is an irrational number.

If you have difficulty proving something by frontal assault, do not forget this finest weapon!

What would happen if the patterns are not linearly separable?

- XOR function



$$w(n+1) = w(n) + \eta e(n)x(n)$$
$$e(n) = d(n) - y(n)$$

Will the perceptron stop updating its weights?

No!

In real world problem, the dimension of the pattern vectors is usually very high, how to determine whether the problem is linearly separable or not?

It is simple! Just let the perceptron learn the training samples. If the weights converge, then it must be linearly separable. And if the perceptron does not stop learning, then it is not.

How about the choice of the initial weights w(1) ? Would the perceptron converge if the initial weights are chosen randomly?

Yes!

How about the choice of the learning rate η ? Would the perceptron converge if other positive values are chosen?

Although η can be chosen to be any positive value, choosing it properly will dictate how fast the learning algorithm converge to a right solution.

$$w(n+1) = w(n) + \eta e(n)x(n)$$

$$e(n) = d(n) - y(n)$$

How to choose the learning rate η ?

What would happen if the learning rate is large?

Consider the case when the correct label is d(n)=1, and the perceptron output is y(n)=0.

$$w(n+1) = w(n) + \eta x(n) \quad \Longrightarrow \quad w^T(n+1)x(n) = w^T(n)x(n) + \eta x^T(n)x(n)$$

What is the desired value of $w^T(n+1)x(n)$? Positive or negative?

Positive!

Can we make it positive by properly choosing the learning rate η?

If it is chosen very large and applied to the example x(n), then correct answer can be obtained in one step!

Can we conclude that choosing a large learning rate would speed up the convergence?

If it is chosen very large and applied to the example x(n), then learning is excellent as far as the present example is concerned, but at the cost of spoiling the learning that has taken place earlier with respect to other examples. Thus, a large value of η is not necessarily good.

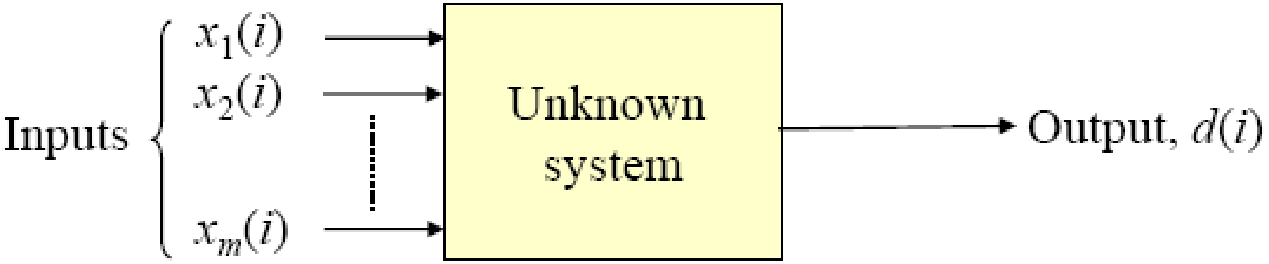
If an extremely small value is chosen for η, that also leads to slow learning. Some intermediate value is the best. Usually the choice is problem dependent.

Perceptron can solve many real world pattern classification problems, which made Rosenblatt very famous in 1960’s.

Can perceptron also deal with another important application of neural networks?

Regression Problem

Consider a multiple input single output system whose mathematical model is unknown:



Inputs $x_1(i)$ $x_2(i)$ $x_m(i)$

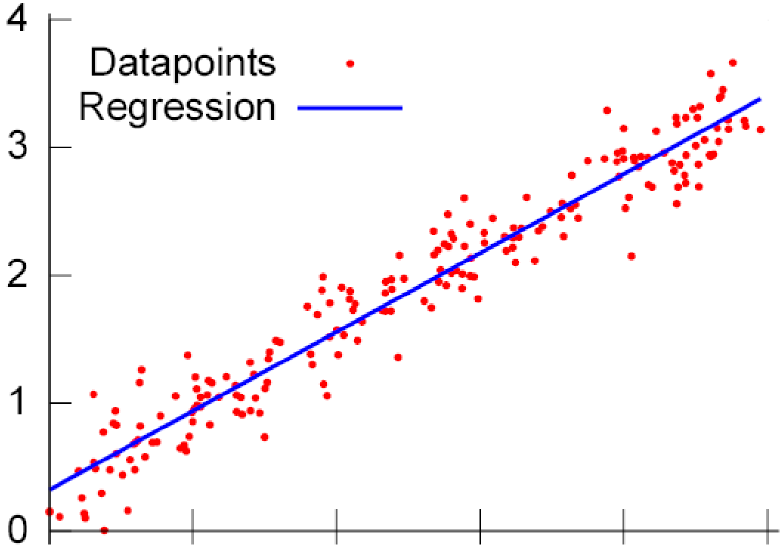
Unknown system

Output, $d(i)$

Given a set of observations of input-output data:

$$T\text{: } \{\boldsymbol{x}(i), d(i); i = 1, 2, \ldots, n\}$$

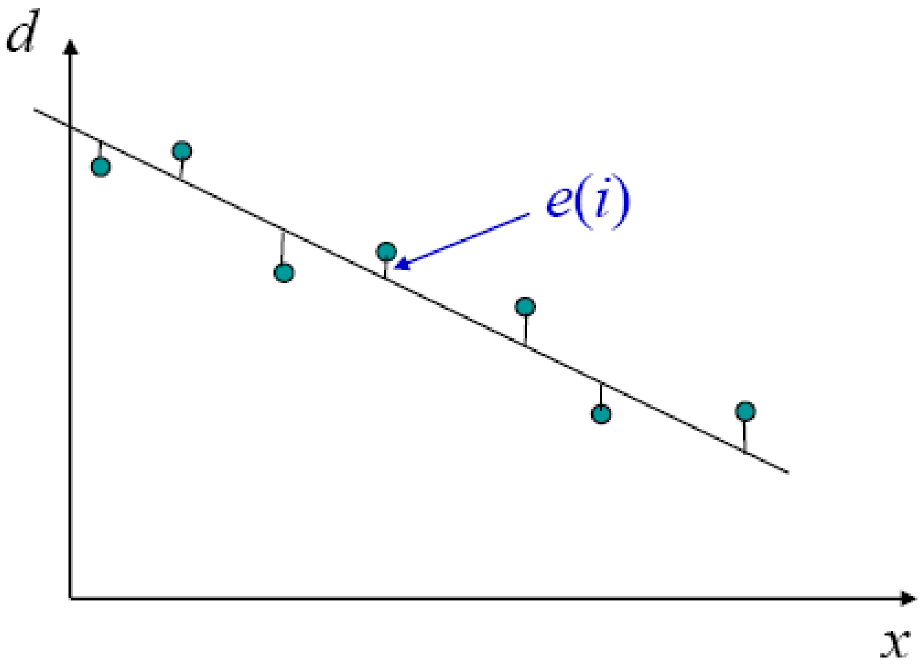
$$\text{where } \boldsymbol{x}(i) = [x_1(i), x_2(i), \ldots, x_m(i)]^T$$



m = dimensionality of the input space; i = time index.

How to design a multiple input single output model for the unknown system?

For example, 1D input



Error signal at time *i*: $e(i) = d(i) - y(i)$

How the error signal *e*(*i*) is used to adjust the synaptic weights in the model for the unknown system is determined mainly by the *cost function* used.

- It can be formulated as an *optimization problem*

What is the most common cost function to evaluate how good the model is?

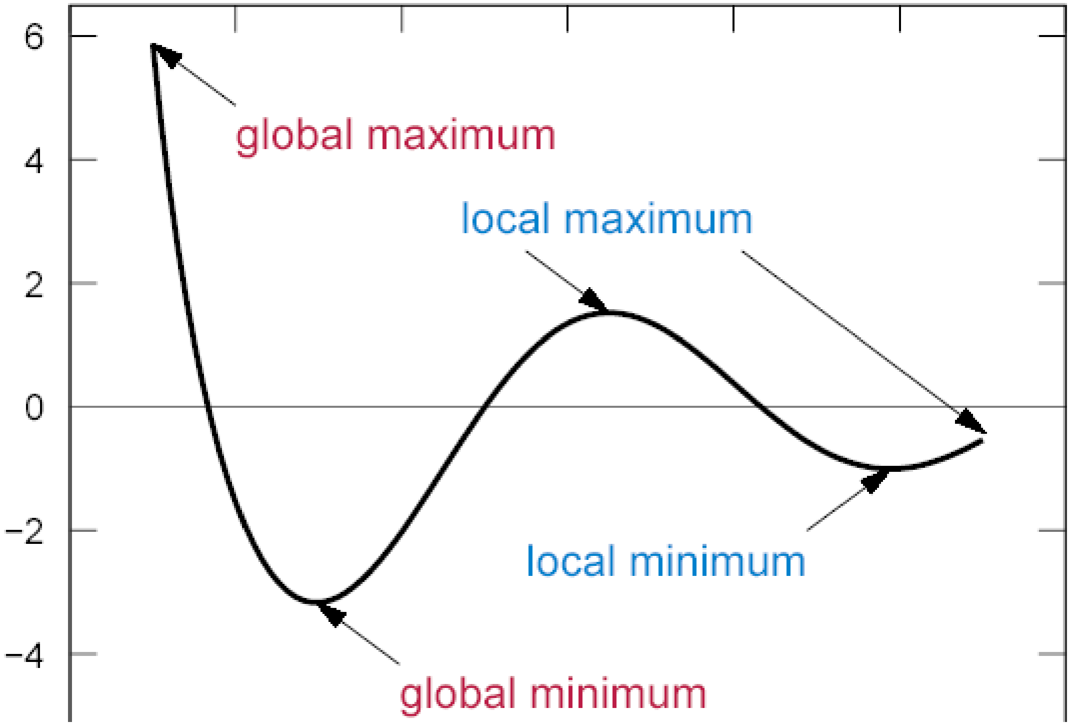
Summation of squares of errors! $E(w) = \sum_{i=1}^{n} e(i)^2 = \sum_{i=1}^{n} (d(i) - y(i))^2$

Why not use $E(w) = \sum_{i=1}^{n} | e(i) |$? The absolute value function is not smooth!

Consider a cost function $E(\boldsymbol{w})$ that is continuously differentiable function of some unknown weight vector, $\boldsymbol{w}$.

Aim: To minimize the cost function $E(\boldsymbol{w})$ with respect to $\boldsymbol{w}$.

Consider a simple example first, a scalar function f(x):



<u>Where to find the minimal or maximal points?</u>

$$\frac{df(x)}{dx} = 0$$

and the boundary!

Necessary condition for optimality: $\nabla\left(E(\boldsymbol{w}^*)\right) = 0$

where ∇ is the gradient operator: $\nabla = \left[\frac{\partial}{\partial w_0}, \frac{\partial}{\partial w_1}, \dots, \frac{\partial}{\partial w_m}\right]^T$

Usually it is not easy to solve $\nabla\left(E(\boldsymbol{w}^*)\right) = 0$ directly.

Iterative descent algorithm**:** Starting with an initial guess denoted by $\boldsymbol{w}(0)$, generate a sequence of weight vectors $\boldsymbol{w}(1)$, $\boldsymbol{w}(2)$,…, such that the cost function $E(\boldsymbol{w})$ is reduced at each iteration ,

$$E(\boldsymbol{w}(n+1)) < E(\boldsymbol{w}(n))$$

where $\boldsymbol{w}(n)$ and $\boldsymbol{w}(n+1)$ are the old and updated values of the weight vector, respectively.

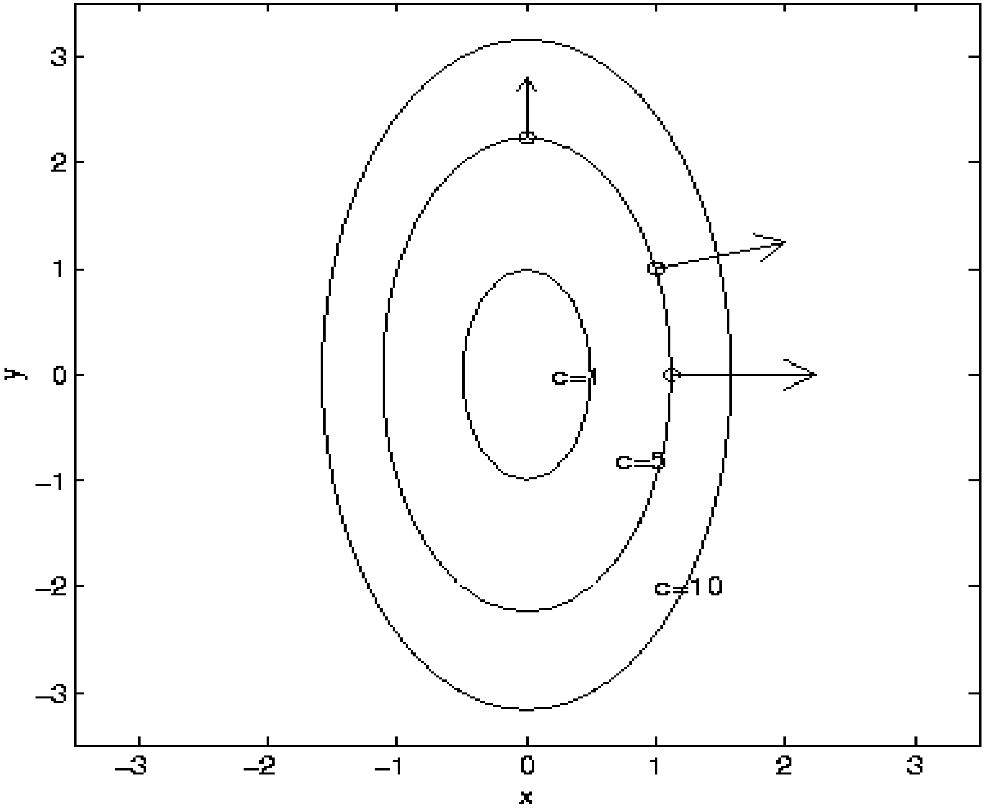
$$w(n+1) = w(n) + \Delta w(n)$$

$$w(n+1) = w(n) + \Delta w(n)$$

How to choose $\Delta w(n)$ such that E(w(n+1)) < E(w(n)) ?

What is the meaning of the direction represented by the gradient vector ∇E(w)?

Two-dimensional Example: f(x,y)



Gradient is the direction along which the function value rises most quickly.

If you want the cost E(w) to decrease, should you let w move along the direction of the gradient or opposite the direction of the gradient?

Method of Steepest Descent (Gradient Descent)

$$w(n+1) = w(n) + \Delta w(n)$$

Successive adjustment applied to the weight vector ***w*** are in the direction of steepest descent (a direction opposite to the gradient vector $\nabla E(\boldsymbol{w})$).

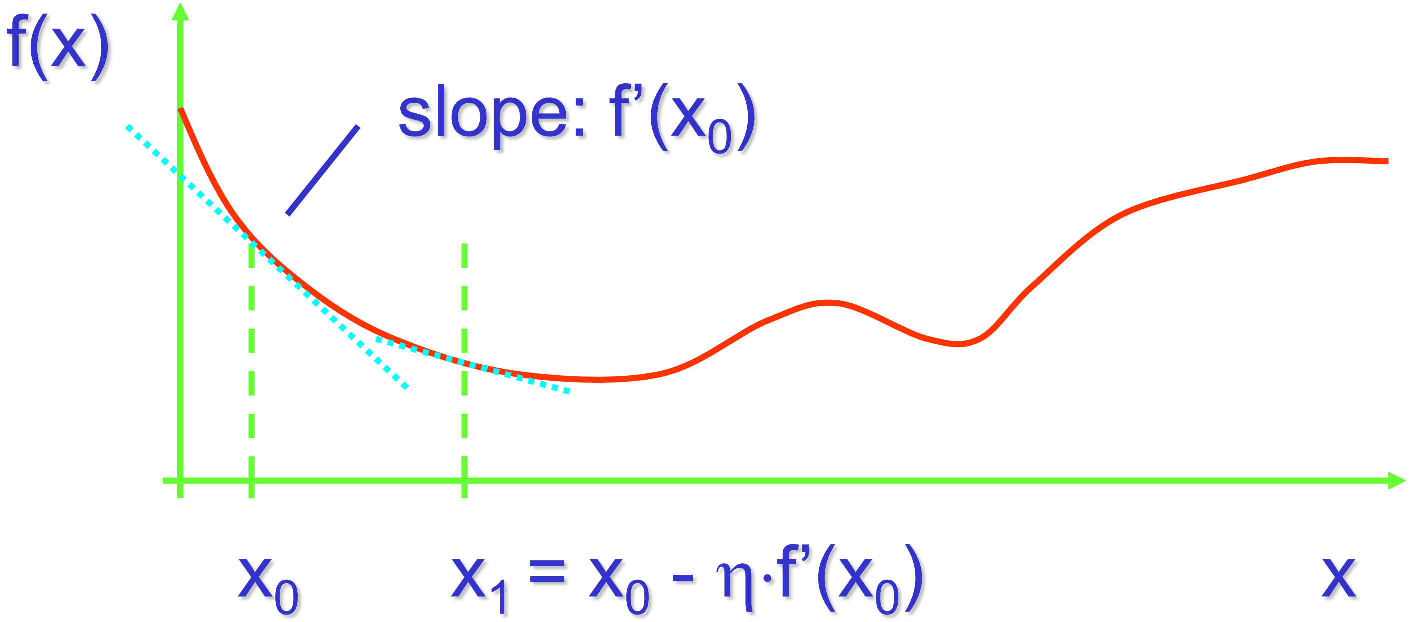
Let ***g(n)*** $=\nabla E(\boldsymbol{w}(n))$, steepest descent algorithm is formally described by

$$\Delta w(n) = -\eta g(n)$$

$$\boldsymbol{w}(n+1) = \boldsymbol{w}(n) - \eta \cdot \boldsymbol{g}(n)$$

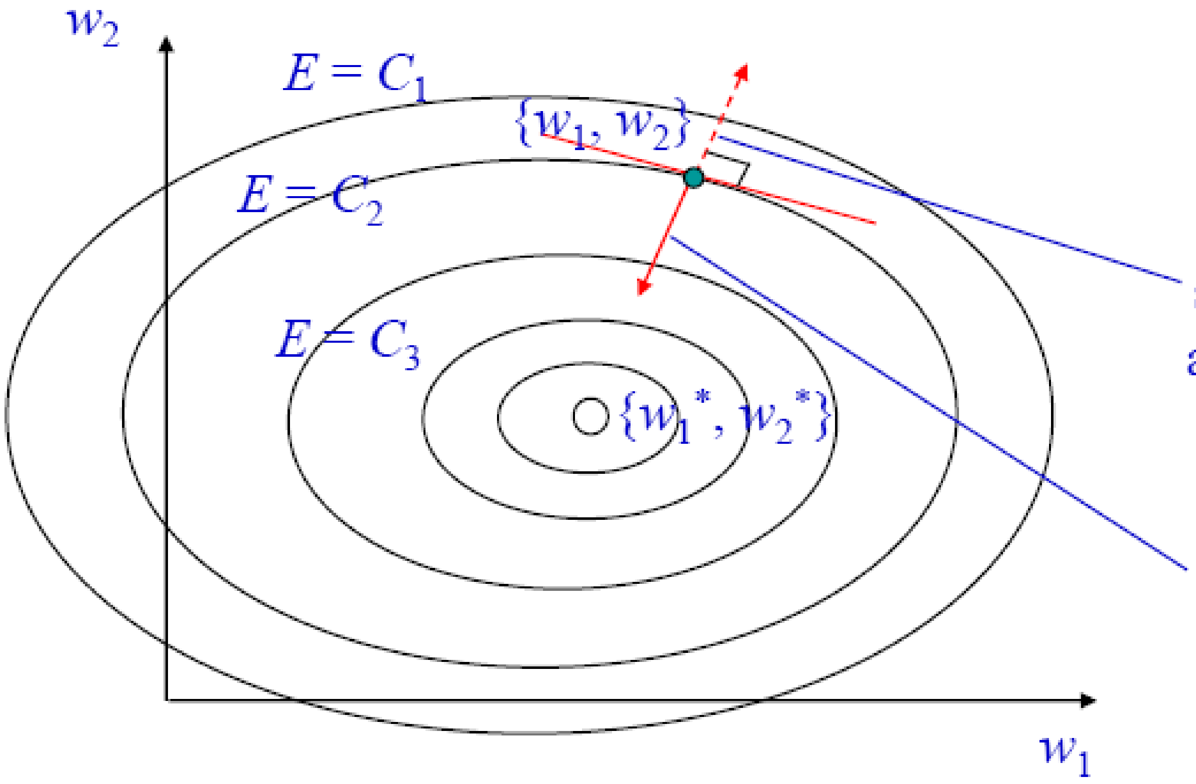
where η is a positive constant called the stepsize or learning-rate parameter, and $\boldsymbol{g}(n)$ is the gradient vector evaluated at the point $\boldsymbol{w}(n)$.

Steepest -descent example: Finding the absolute minimum of a one-dimensional error function f(x):



Repeat this iteratively until for some x_i, $f'(x_i)$ is sufficiently close to 0.

Two-dimensional example



Direction of steepest ascent at $\boldsymbol{w} = \{w_1, w_2\}$

Direction of steepest descent at $\boldsymbol{w} = \{w_1, w_2\}$

$C_1 > C_2 > C_3$

The updating of the weights is given as

$$\mathbf{w}(n+1) = \mathbf{w}(n) - \eta \cdot \mathbf{g}(n) \quad \Longrightarrow \quad \begin{aligned} \Delta\mathbf{w}(n) &= \mathbf{w}(n+1) - \mathbf{w}(n) \\ &= -\eta \cdot \mathbf{g}(n) \end{aligned}$$

Does it satisfy the condition of iterative descent? E(w(n+1)) < E(w(n))

We need to compare E(w(n+1) and E(w(n)).

How to expand E(w(n+1)) around w(n)? Can you write

$$E(w(n+1)) = E(w(n) + \Delta w(n)) \approx E(w(n)) + ???$$

You learned that in Calculus! Taylor series

$$f(x) = f(a) + f'(a)(x-a) + \frac{f''(a)}{2!}(x-a)^2 + \frac{f^{(3)}(a)}{3!}(x-a)^3 + \ldots + \frac{f^{(n)}(a)}{n!}(x-a)^n + \ldots$$

$$E(w(n+1)) = E(w(n) + \Delta w(n)) \approx E(w(n)) + \frac{\partial E}{\partial w}\Delta w(n)$$

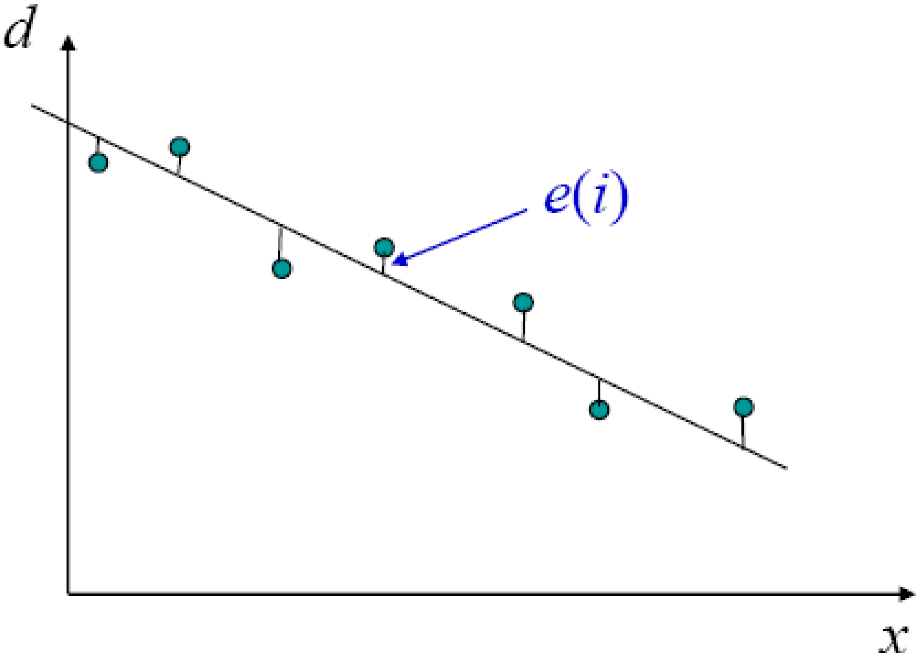
$$\frac{\partial E}{\partial w} = g^T(n) \quad \Longrightarrow \quad \begin{aligned} E(\mathbf{w}(n+1)) &\approx E(\mathbf{w}(n)) + \mathbf{g}^T(n)\Delta\mathbf{w}(n) \\ &\approx E(\mathbf{w}(n)) - \eta\mathbf{g}^T(n)\mathbf{g}(n) \\ &\approx E(\mathbf{w}(n)) - \eta\|\mathbf{g}(n)\|^2 \end{aligned} \qquad \text{holds for small } \eta$$

Therefore the cost function is decreasing as the algorithm progresses from one iteration to the next for a small positive learning rate η.

Linear Regression Problem

Consider that we are trying to fit a linear model to a set of input-output pairs (***x***(1), *d*(1)), (***x***(2), *d*(2)) …, (***x***(*n*), *d*(*n*)) observed in an interval of duration *n*.

$y(x)=w^Tx=w_1x_1+w_2x_2+\ldots+w_mx_m+b$



Cost Function:

$$E(w)=\sum_{i=1}^{n}e(i)^2=\sum_{i=1}^{n}(d(i)-y(i))^2$$

<u>How to find out the optimal parameter such that the cost is minimized?</u>
<u>What is the optimality condition?</u>

$$\frac{\partial E(w)}{\partial w}=0$$

Standard Linear Least Squares

We want to minimize the cost $E(w) = \sum_{i=1}^{n} e(i)^2$

Let's define the error vector: $e = [e(1) \quad e(2) \quad \cdots \quad e(n)]^T$

$$E(w) = \sum_{i=1}^{n} e(i)^2 = e^T e$$

Next, let's express e in terms of the parameter w,

$$e(i) = d(i) - y(i) \quad \Longrightarrow \quad e = d - y$$

$$y = [y(1) \quad y(2) \quad \cdots \quad y(n)]^T \qquad d = [d(1) \quad d(2) \quad \cdots \quad d(n)]^T$$

$$y(i) = w^T x(i) = x(i)^T w$$

$$y = \begin{bmatrix} y(1) \\ y(2) \\ \vdots \\ y(n) \end{bmatrix} = \begin{bmatrix} x(1)^T w \\ x(2)^T w \\ \vdots \\ x(n)^T w \end{bmatrix} = \begin{bmatrix} x(1)^T \\ x(2)^T \\ \vdots \\ x(n)^T \end{bmatrix} w = Xw$$

Regression matrix: $X = \begin{bmatrix} x(1)^T \\ x(2)^T \\ \vdots \\ x(n)^T \end{bmatrix}$

So we have $e = d - Xw$

Standard Linear Least Squares

We want to minimize the cost $E(w) = \sum_{i=1}^{n} e(i)^2 = e^T e$

Do you know how to calculate $\frac{\partial E}{\partial e}$? $\frac{\partial E}{\partial e} = 2e^T$

We also have , $e = d - Xw$

Do you know how to calculate $\frac{\partial e}{\partial w}$? $\frac{\partial e}{\partial w} = -X$

How to calculate $\frac{\partial E}{\partial w}$?

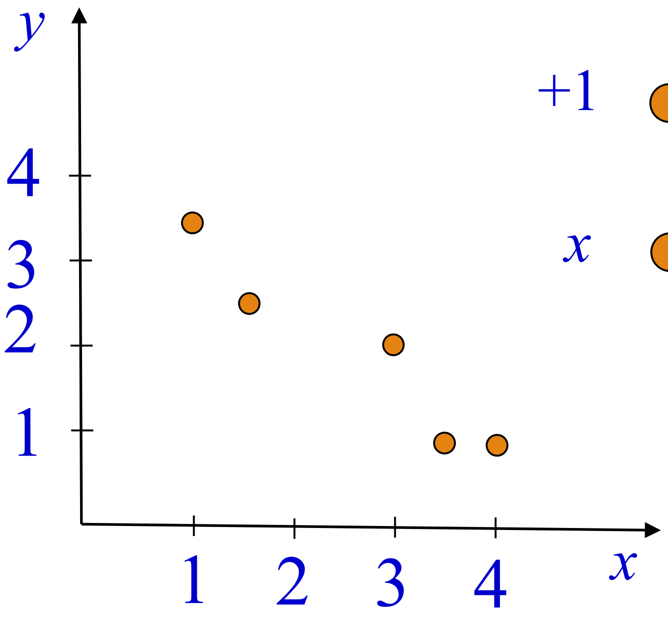
By chain rule, $\frac{\partial E}{\partial w} = \frac{\partial E}{\partial e}\frac{\partial e}{\partial w} \Longrightarrow \frac{\partial E}{\partial w} = -2e^T X = 0$

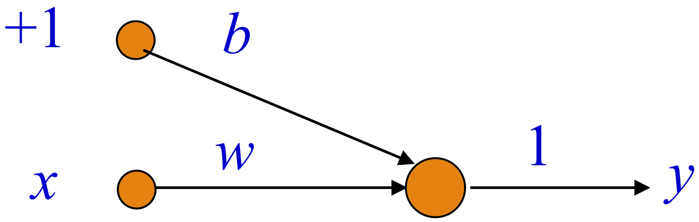
$$e^T X = (d - Xw)^T X = (d^T - w^T X^T)X = d^T X - w^T X^T X = 0$$

$$\Longrightarrow \quad w^T X^T X = d^T X \quad \Longrightarrow \quad X^T X w = X^T d$$

$$w = (X^T X)^{-1} X^T d$$

- Example: Consider that we are trying to fit a linear model to a set of input-output pairs {(1, 3.5), (1.5, 2.5), (3, 2), (3.5, 1),(4, 1)}



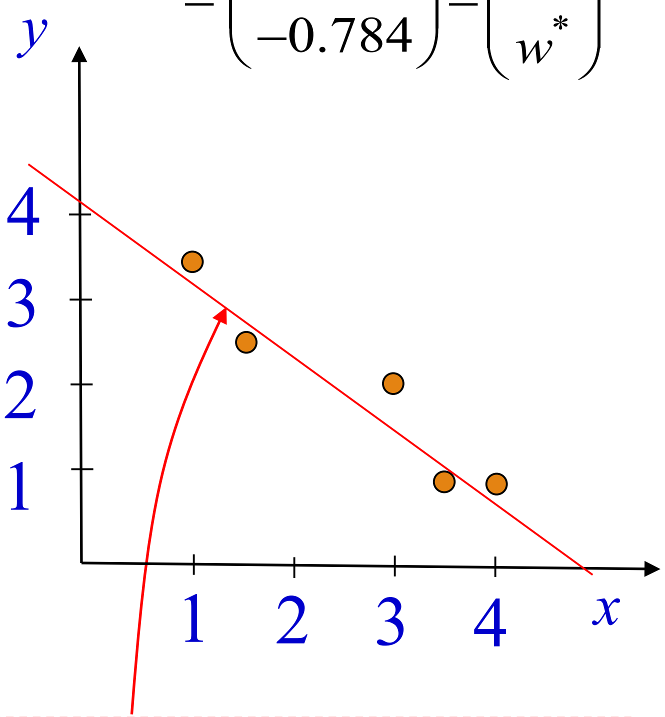


$$\boldsymbol{d} = \begin{pmatrix} 3.5 \\ 2.5 \\ 2 \\ 1 \\ 1 \end{pmatrix} \qquad \boldsymbol{X} = \begin{pmatrix} 1 & 1 \\ 1 & 1.5 \\ 1 & 3 \\ 1 & 3.5 \\ 1 & 4 \end{pmatrix}$$

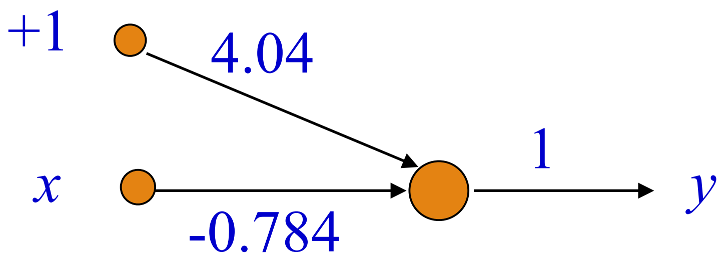
- The solution:

$$\boldsymbol{w}^* = \left(\overset{2\times5}{\begin{pmatrix}1&1&1&1&1\\1&1.5&3&3.5&4\end{pmatrix}} \overset{5\times2}{\begin{pmatrix}1&1\\1&1.5\\1&3\\1&3.5\\1&4\end{pmatrix}} \right)^{-1} \overset{2\times5}{\begin{pmatrix}1&1&1&1&1\\1&1.5&3&3.5&4\end{pmatrix}} \overset{5\times1}{\begin{pmatrix}3.5\\2.5\\2\\1\\1\end{pmatrix}}$$

$$= \overset{2\times1}{\begin{pmatrix}4.04\\-0.784\end{pmatrix}} = \begin{pmatrix}b^*\\w^*\end{pmatrix}$$



$$\boxed{y = w^*x + b^* = \text{-}0.784x + 4.04}$$



Linear Regression Problem

Consider that we are trying to fit a linear model to a set of input-output pairs (***x***(1), *d*(1)), (***x***(2), *d*(2)) …, (***x***(*n*), *d*(*n*)) observed in an interval of duration *n*.

$$y(x)=w_1x_1+w_2x_2+\ldots+w_mx_m+b$$

The standard linear least squares: $w=(X^TX)^{-1}X^Td$

Regression matrix: $X=\begin{bmatrix} x(1)^T \\ x(2)^T \\ \vdots \\ x(n)^T \end{bmatrix}$

What is the dimension of the regression matrix X?

$$n\times(m+1)$$

It may be out of memory if n is very large!

How do we deal with the "BIG DATA"?

Linear Regression Problem

Consider that we are trying to fit a linear model to a set of input-output pairs ($\boldsymbol{x}(1)$, $d(1)$), ($\boldsymbol{x}(2)$, $d(2)$) …, ($\boldsymbol{x}(n)$, $d(n)$) observed in an interval of duration n.

$$y(x)=w_1x_1+w_2x_2+\ldots+w_mx_m+b$$

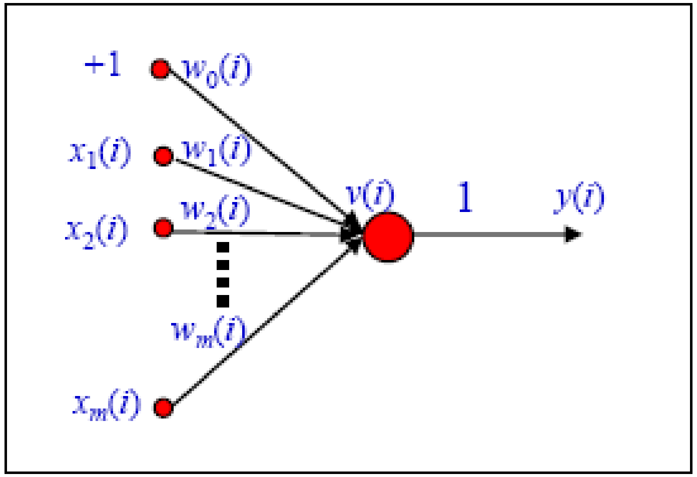
Can we directly use Rosenblatt's percetron to solve this linear regression problem?

No. The output of perceptron is either 1 or 0 due to the hard limiter!

Can we modify the percetron a little bit such that it can match the linear model?

We can just replace the hard-limiter by the linear function :

$$y(i)=v(i)=w^T(i)x(i)$$



Can the linear neuron learn the function by itself just like the percetron?

Least-Mean-Square Algorithm

Proposed by Widrow and Hoff at Stanford University in 1960.

Based on the instantaneous cost function at step n,

$$E(w) = \frac{1}{2} e^2(n)$$

where $e(n)$ is the error signal measured at step n.

$$e(n) = d(n) - x^T(n) w(n)$$

The chain rule,

$$\boxed{\frac{\partial E}{\partial w} = \frac{\partial E}{\partial e} \cdot \frac{\partial e}{\partial w}}$$

$$\frac{\partial E}{\partial e} = e(n) \qquad \frac{\partial e(n)}{\partial w(n)} = -x^T(n) \qquad \Longrightarrow \qquad \frac{\partial E(w)}{\partial w(n)} = -e(n) x^T(n)$$

The gradient of E(w),

$$g(n) = \left(\frac{\partial E(w)}{\partial w(n)}\right)^T = -e(n) x(n)$$

Applying steepest descent method, we have

$$w(n+1) = w(n) - \eta g(n) = w(n) + \eta e(n) x(n)$$

η is the learning-rate parameter.

This is sometimes called "Widrow-Hoff learning rule" or "incremental gradient algorithm".

Summary of the LMS algorithm:

Given n training samples: $\{\boldsymbol{x}(i), d(i)\}$, $i = 1, 2,\ldots,n$

where $\boldsymbol{x}(i)$ is an input vector, $d(i)$ is the corresponding desired response.
User-selected parameter: learning rate η

Weights initialization.

Computation (LMS rule):

- For $i = 1, 2,\ldots,$ compute

$$e(n) = d(n) - w^T(n)x(n)$$
$$w(n+1) = w(n) + \eta e(n)x(n)$$

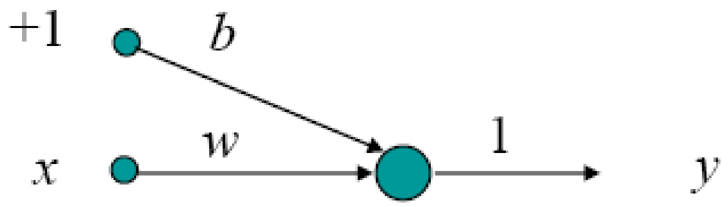
Isn't it the same learning algorithm as that for the Perceptron?

YES!

Example: LMS algorithm

- Training sample, ($x(i)$, $d(i)$): {(1, 3.5), (1.5, 2.5), (3, 2), (3.5, 1), (4, 1)}.

  Initial weight is chosen as $\mathbf{w}'(1) = [2, 0]^T$. Learning rate η is 0.1.



Find the solution using LMS algorithm for 50 epochs.

Solution:

1. Learning process

Step 1: $\mathbf{x}^{(1)} = \begin{pmatrix} 1 \\ 1 \end{pmatrix}$

$e(1)$=$d^{(1)}$-$\mathbf{w}(1)^{\mathrm{T}}\mathbf{x}^{(1)}$=3.5-2 = 1.5

$$\mathbf{w}(2) = \mathbf{w}(1) + \eta \mathbf{x}^{(1)} e(1)$$

$$= \begin{pmatrix} 2 \\ 0 \end{pmatrix} + 0.1 \begin{pmatrix} 1 \\ 1 \end{pmatrix} 1.5 = \begin{pmatrix} 2.15 \\ 0.15 \end{pmatrix}$$

Step 2: $\mathbf{x}^{(2)} = \begin{pmatrix} 1 \\ 1.5 \end{pmatrix}$

$e(2)$=$d^{(2)}$-$\mathbf{w}(2)^{\mathrm{T}}\mathbf{x}^{(2)}$=0.125

$$\mathbf{w}(3) = \mathbf{w}(2) + \eta \mathbf{x}^{(2)} e(2)$$

$$= \begin{pmatrix} 2.1625 \\ 0.1688 \end{pmatrix}$$

Step 3: $\mathbf{x}^{(3)} = \begin{pmatrix} 1 \\ 3 \end{pmatrix}$

$e(3)$=$d^{(3)}$-$\mathbf{w}(3)^{\mathrm{T}}\mathbf{x}^{(3)}$=-0.6688

$$\mathbf{w}(4) = \mathbf{w}(3) + \eta \mathbf{x}^{(3)} e(3)$$

$$= \begin{pmatrix} 2.0956 \\ -0.0319 \end{pmatrix}$$

Step 4: $\mathbf{x}^{(4)} = \begin{pmatrix} 1 \\ 3.5 \end{pmatrix}$

$e(4)$=$d^{(4)}$-$\mathbf{w}(4)^{\mathrm{T}}\mathbf{x}^{(4)}$=-0.9841

$$\mathbf{w}(5) = \mathbf{w}(4) + \eta \mathbf{x}^{(4)} e(4)$$

$$= \begin{pmatrix} 1.9972 \\ -0.3763 \end{pmatrix}$$

Step 5: $\mathbf{x}^{(5)} = \begin{pmatrix} 1 \\ 4 \end{pmatrix}$

$e(5)$=$d^{(5)}$-$\mathbf{w}(5)^{\mathrm{T}}\mathbf{x}^{(5)}$=0.508

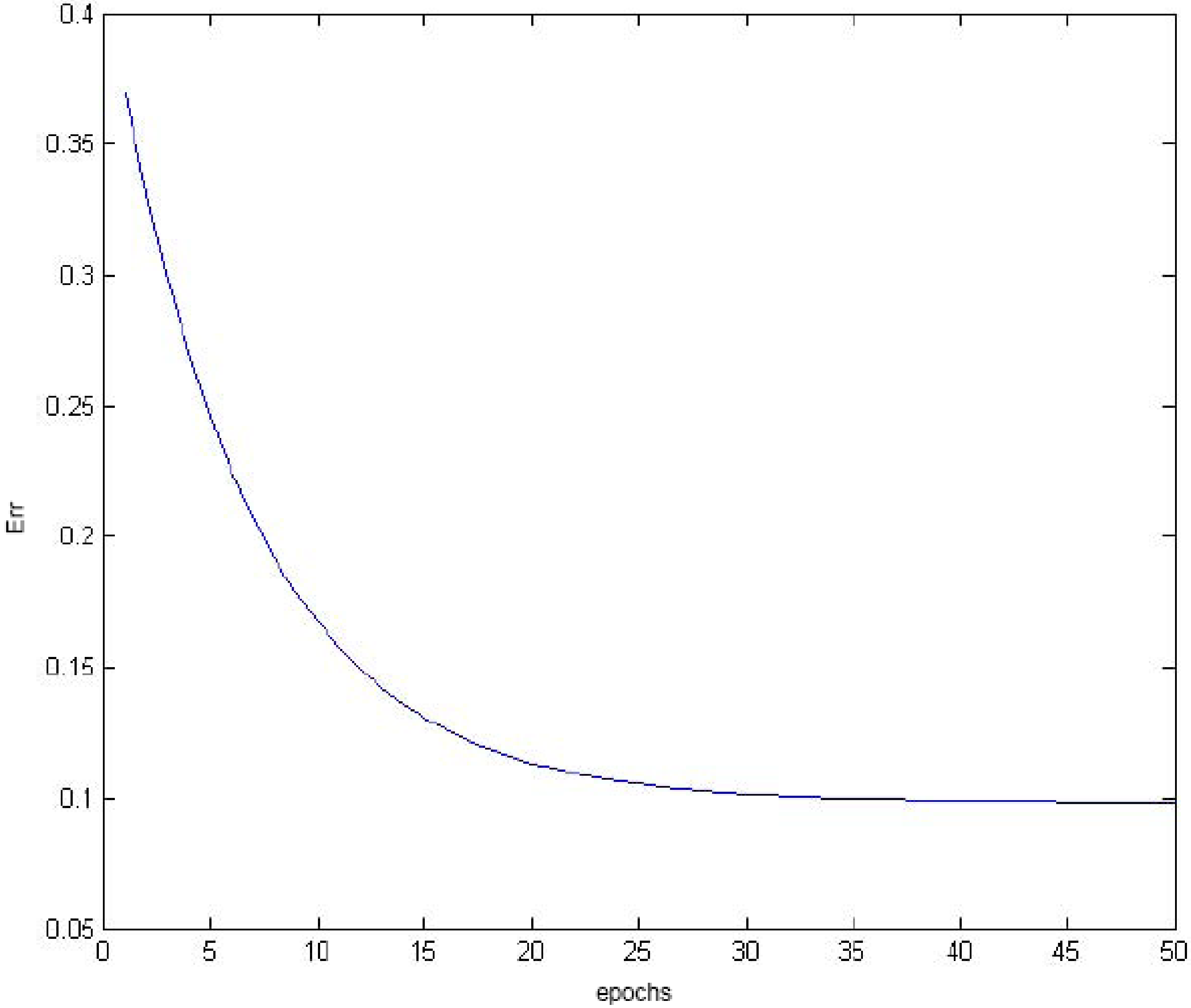
$$\mathbf{w}(6) = \mathbf{w}(5) + \eta \mathbf{x}^{(5)} e(5)$$

$$= \begin{pmatrix} 2.048 \\ -0.1731 \end{pmatrix}$$

Up to now, all training patterns have been used once. We say “one epoch” of the training is completed.

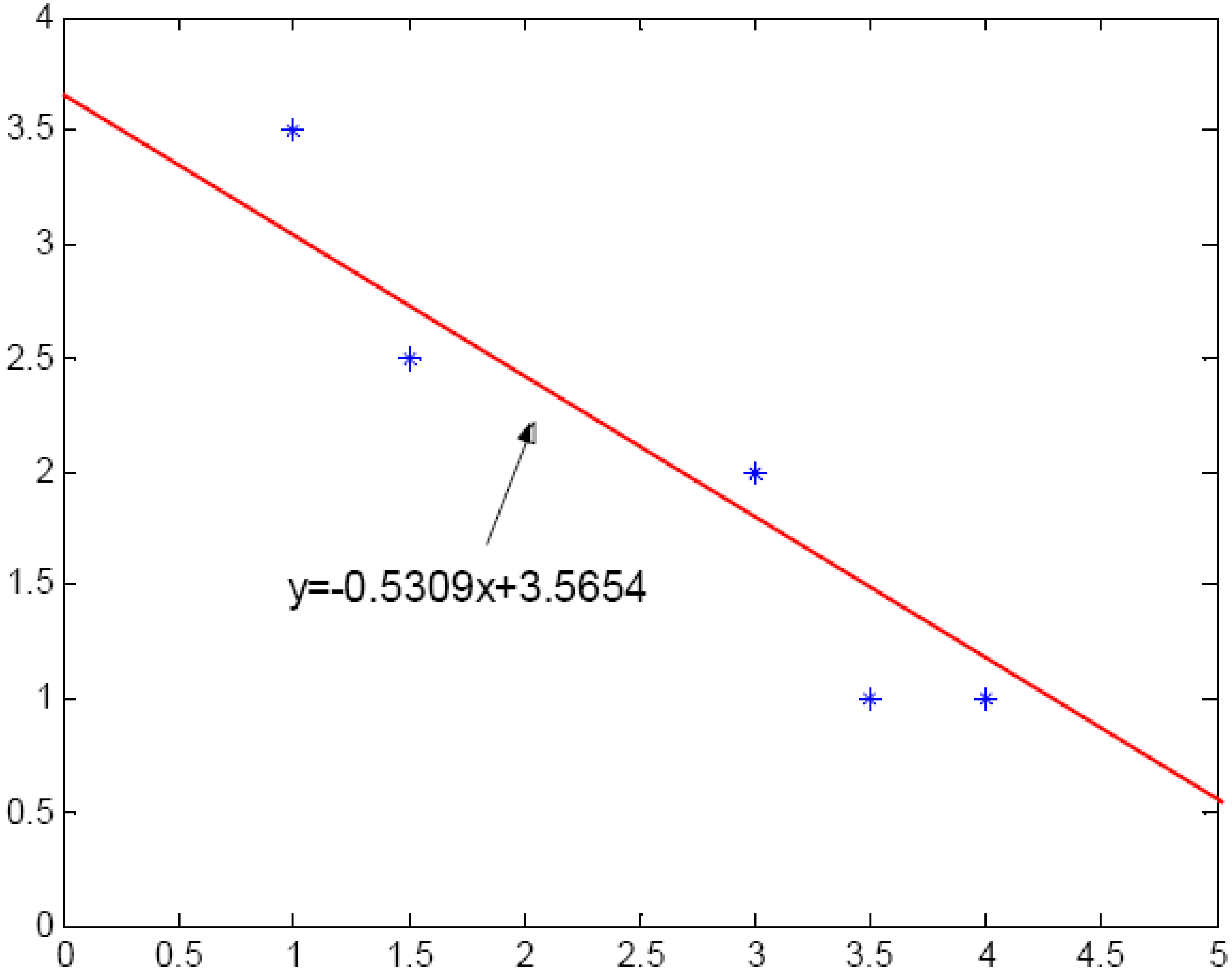
More epochs needed until satisfactory fitting is obtained.

2. The error function: Average of the cost $e^2/2$ at the end of each epoch.



3. Result

After 50 epochs, the weight vector is w’ = [3.5654, -0.5309]’. The figure below plots the result of the LMS algorithm for this example.

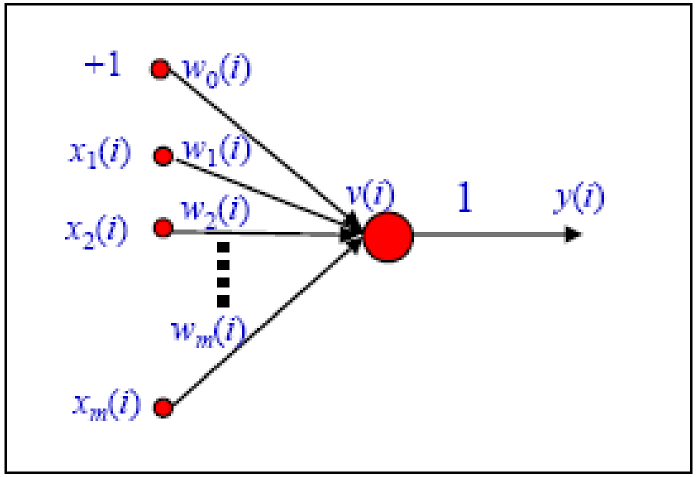


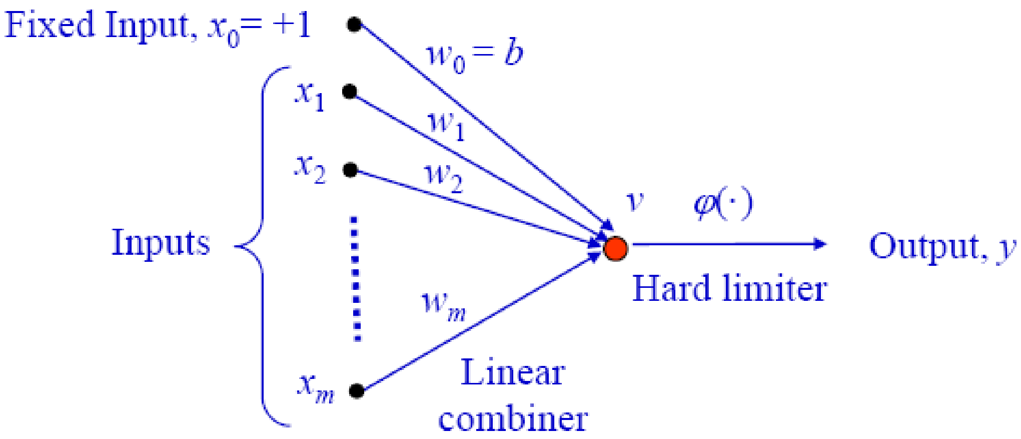
Perceptron v.s. LMS algorithm

The perceptron and the LMS algorithm emerged roughly about the same time, during the late 1950s.

They represent different implementations of single-layer perceptron based on *error-correction-learning*.

The LMS algorithm uses a linear neuron!



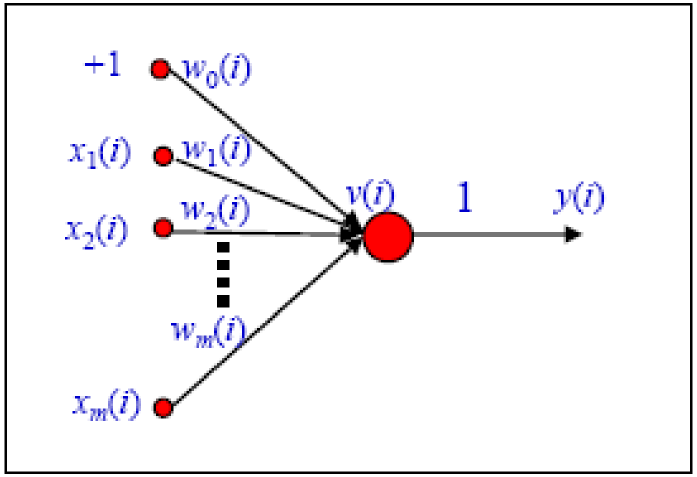


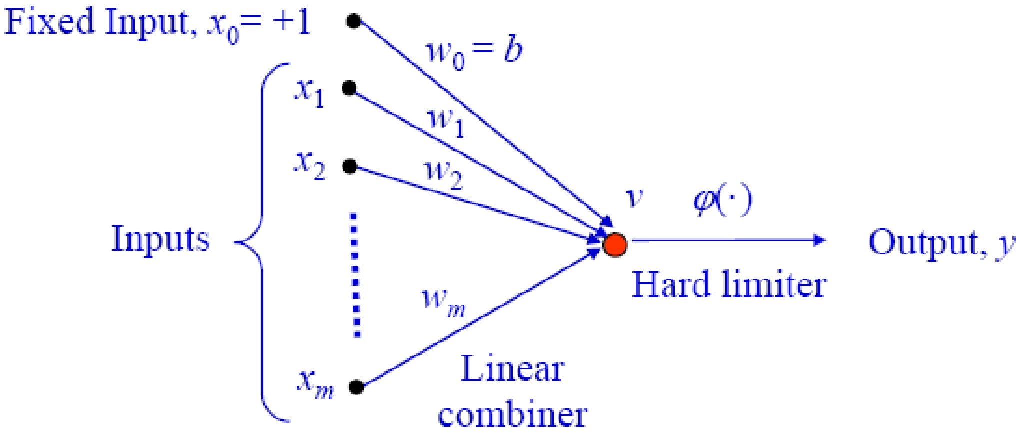
The learning process in the perceptron stops after a finite number of iterations.

How about LMS algorithm, does it converge in finite time?

In contrast, LMS algorithm usually does not stop unless an extra stopping rule is applied because perfect fitting is normally impossible!

The learning of the perceptrons

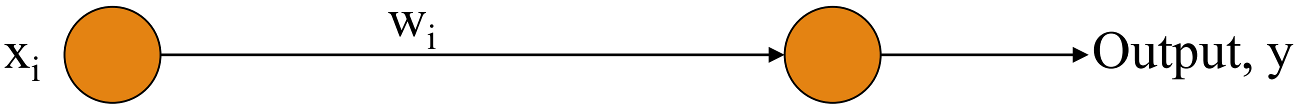




$$w(n+1) = w(n) + \eta e(n) x(n)$$

Let's take a closer look at what happens to each synaptic weight:

$$w_i(n+1) = w_i(n) + \eta e(n) x_i(n)$$



The adjustment of the synaptic weight only depends upon the information of the input neuron and the output neuron, and nothing else.

The adjustment of the synaptic weight is proportional to both the input signal and the output error.

<u>Does the weight adjustment indirectly depend upon other synaptic weights?</u>

Yes. Because the output of the neuron depends upon all the connected neurons!

The Tragical ending of the Perceptron

Frank Rosenblatt(1928-1969)

Frank was overly optimistic about the power of the perceptron and predicted that "perceptron may eventually be able to learn, make decisions, and translate languages."

In 1969, Marvin Minsky and Seymour Papert showed that it was impossible for perceptron to learn an XOR function. They conjectured (incorrectly) that a similar result would hold for a perceptron with three or more layers.

Frank (1946) and Marvin(1945) were schoolmates at the Bronx High School of Science in New York City.

Rosenblatt died tragically in a boating accident in 1969, shortly after Minsky's book was published.

Mysteriously, McCulloch and Pitts died in the same year, 1969.

In 2004 the IEEE established the Frank Rosenblatt Award.

So he will live forever!

Q & A…

THANK YOU !