IN THE NAME OF ALLAH

Neural Networks

Shahrood University of Technology
Hossein Khosravi

Early Adaptive Networks

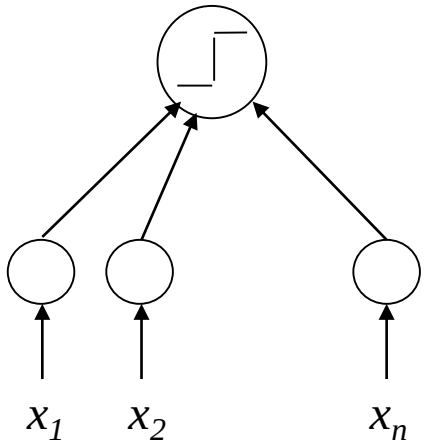
History of NN

- Those who cannot remember the past are condemned to repeat it.
  - Santayana, "The Life of Reason" (1905)
- **Pitts & McCulloch (1943)**
  - First mathematical model of biological neurons
  - All Boolean operations can be implemented by these neuron-like nodes (with different threshold and excitatory/inhibitory connections).
  - Competitor to Von Neumann model for general purpose computing device
- **Hebb (1949)**
  - Hebbian rule of learning: increase the connection strength between neurons i and j whenever both i and j are activated.
  - Or increase the connection strength between nodes i and j whenever both nodes are simultaneously ON or OFF.
  - In short: *cells that fire together, wire together* $\Delta w_i = \eta x_i y$

History of NN

- **Early booming (50's – early 60's)**
  - Rosenblatt (1958)
    - Perceptron: network of threshold nodes for pattern classification
    - Perceptron learning rule
    - Perceptron convergence theorem:

      everything that can be represented by a perceptron can be learned



  - Widrow and Hoff (1960, 1962)
    - Learning rule based on gradient descent (with differentiable unit)
  - Minsky's attempt to build a general purpose machine with Pitts/McCullock units

History of NN

- **The setback (mid 60's – late 70's)**
  - Serious problems with perceptron model (Minsky's book 1969)
    - Single layer perceptron cannot represent (learn) simple functions such as **XOR**
    - Multi-layer of non-linear units may have greater power but there is no learning rule for such nets
    - Scaling problem: connection weights may grow infinitely
  - The first two problems overcame by latter effort in 80's, but the scaling problem persists

History of NN

- **Renewed enthusiasm and flourish (80's – present)**
  - New techniques
    - Backpropagation learning for multi-layer feed forward nets (with non-linear, differentiable node functions)
      - Abbreviation for "Backward Propagation of errors"
      - Paul Werbos (1974 Harvard Ph.D. thesis) is possibly the first one to discover backpropagation algorithm
    - Thermodynamic models (Hopfield net, Boltzmann machine, etc.)
    - Unsupervised learning
  - Impressive application (character recognition, speech recognition, text-to-speech transformation, process control, associative memory, etc.)
  - Traditional approaches face difficult challenges

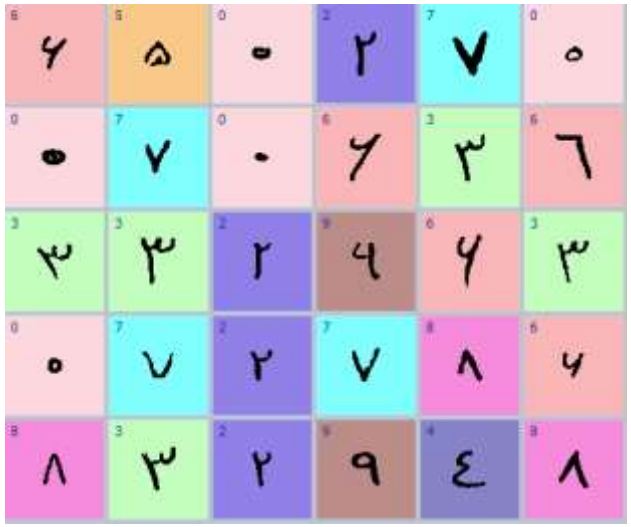
NNs: goal and design

- Knowledge about the learning task is given in the form of a set of examples (dataset) called training examples.
- A NN is specified by:
  - an architecture: a set of neurons and links connecting neurons. Each link has a weight,
  - a neuron model: the information processing unit of the NN,
  - a learning algorithm: used for training the NN by modifying the weights in order to solve the particular learning task correctly on the training examples.

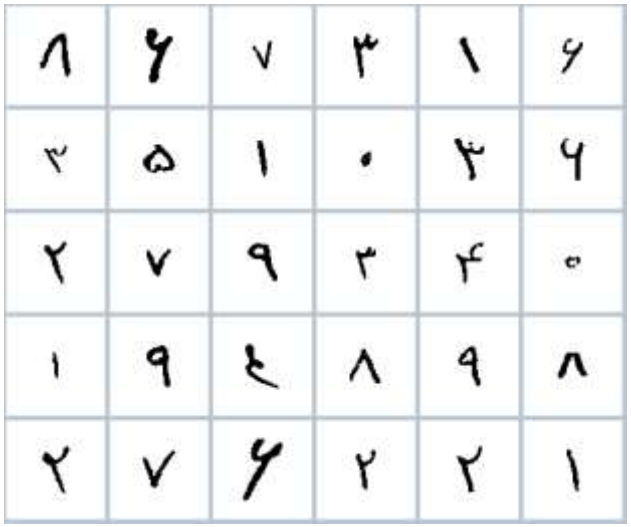
The aim is to obtain a NN that generalizes well, i.e. behaves correctly on new examples of the learning task.

Demo

Training set



Test set



Face Detection Demo:
http://www.facedetection.com/facedetection/online_demos.htm

Network architectures

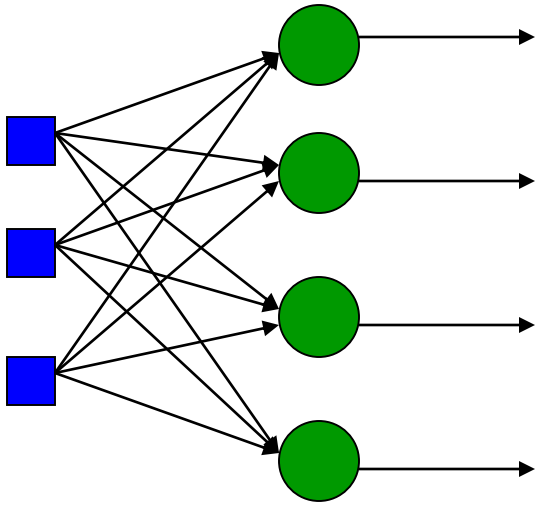
- Three different classes of network architectures
  - single-layer feed-forward
  - multi-layer feed-forward

  } neurons are organized in acyclic layers
  - recurrent

- The architecture of a neural network is linked with the learning algorithm used to train

Single Layer Feed-forward

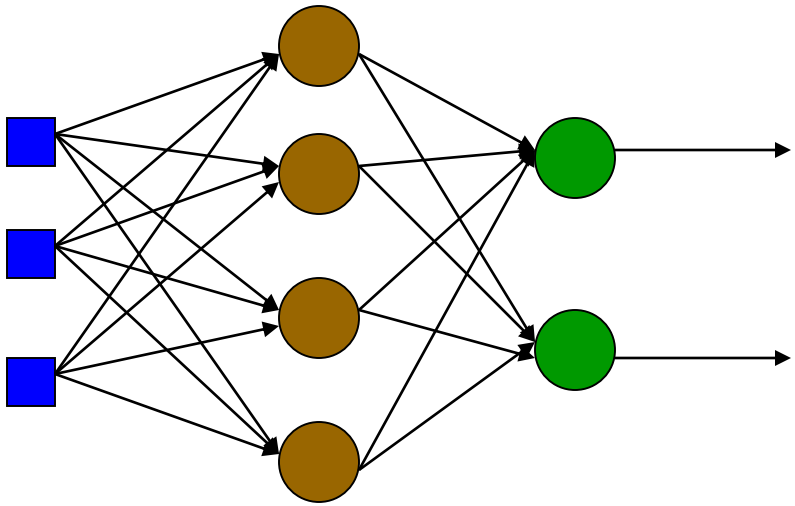
Input layer of source nodes



Output layer of neurons

Multi layer feed-forward

3-4-2 Network



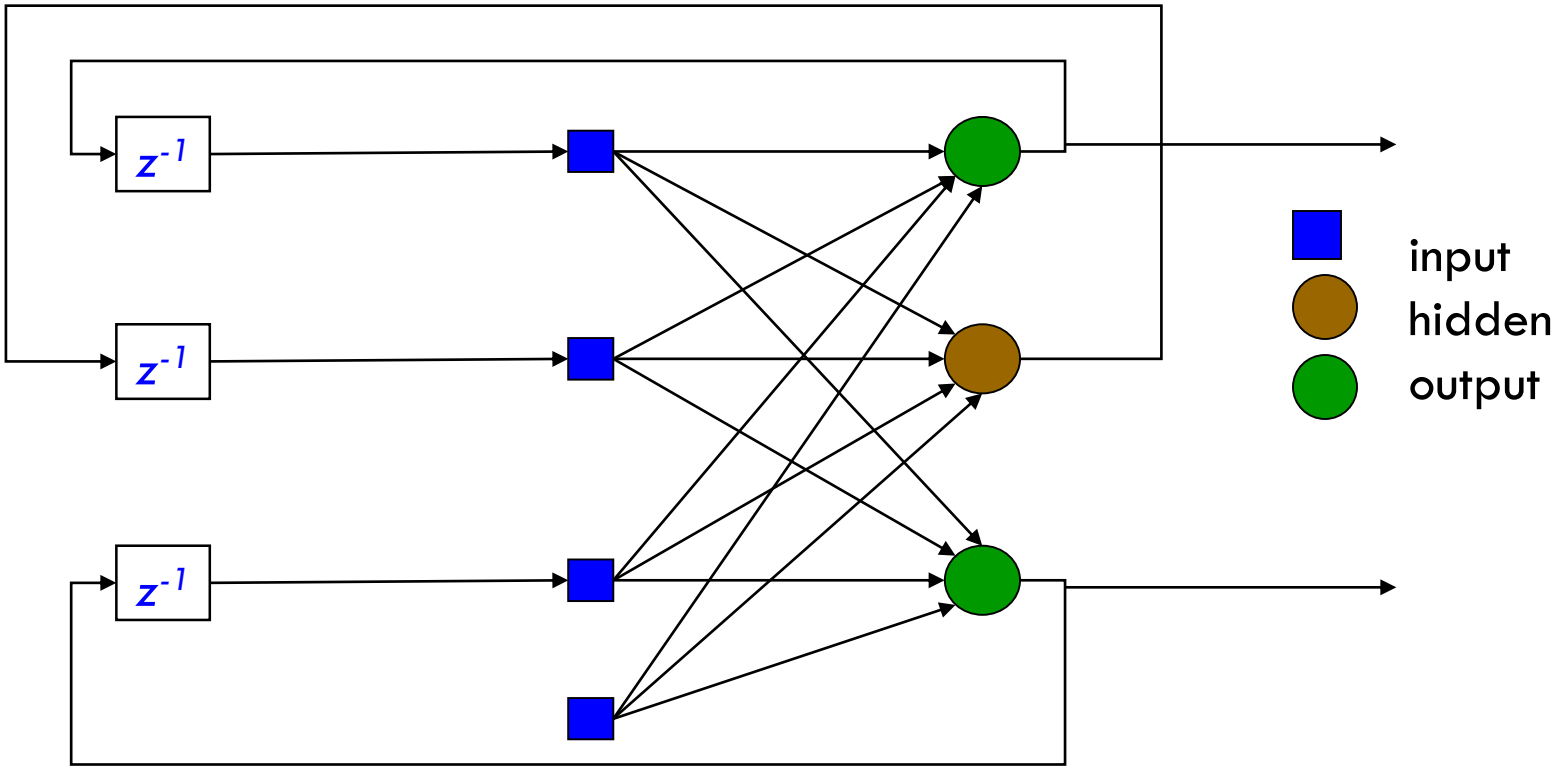
Input layer

Output layer

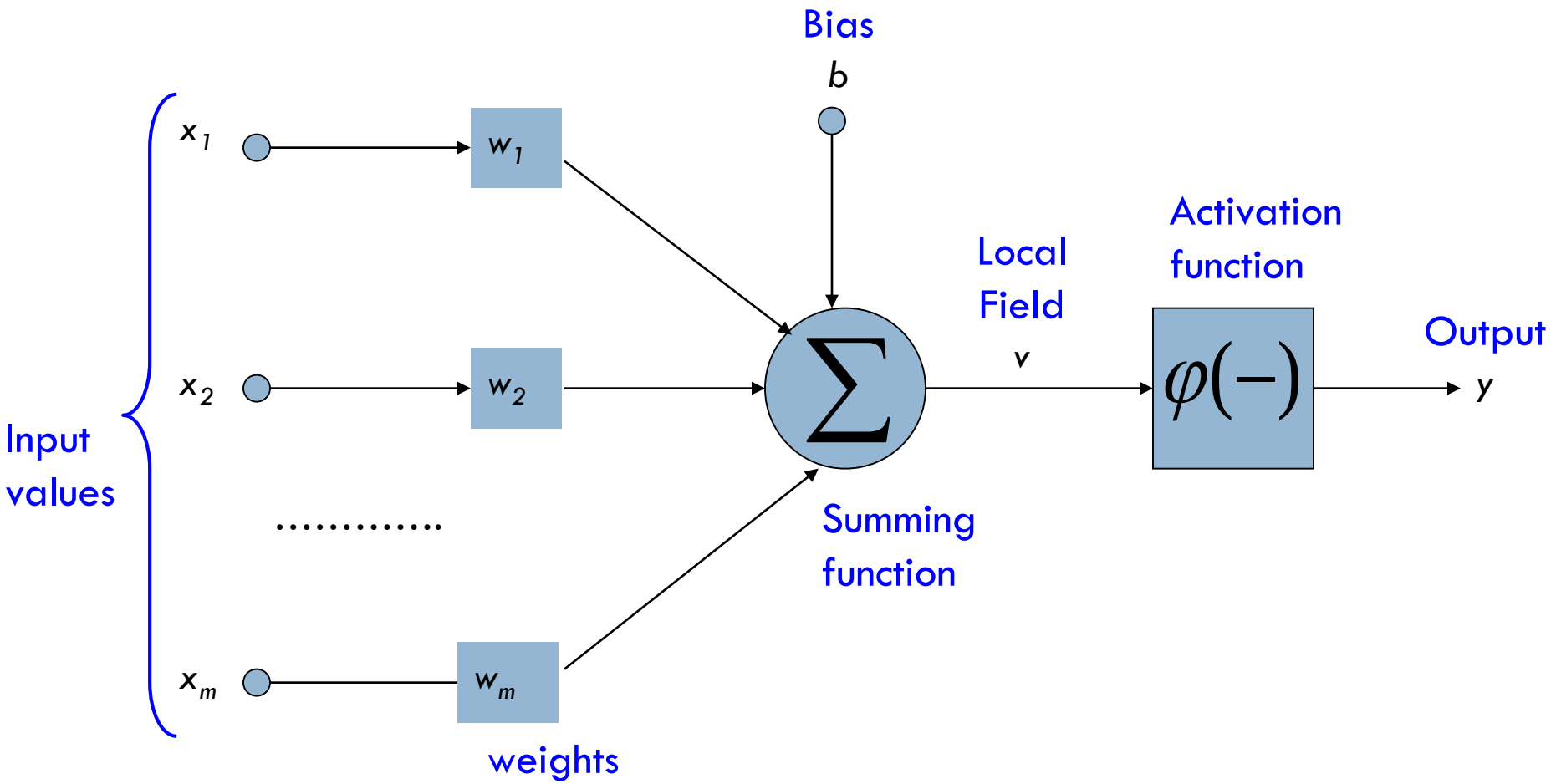
Hidden Layer

Recurrent network

- Recurrent Network with **hidden neuron**: unit delay operator z^{-1} is used to model a dynamic system



The Neuron

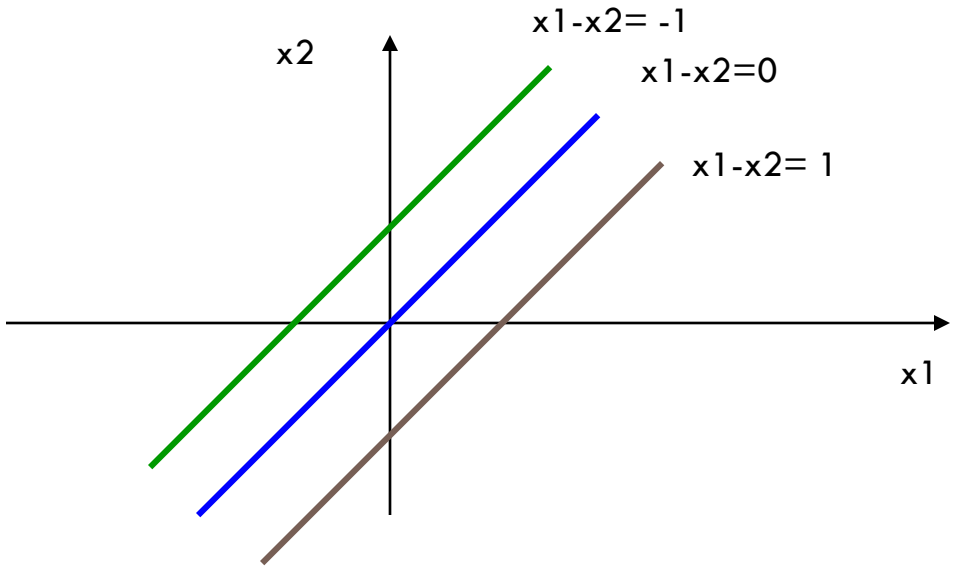

Bias
b
x_1
w_1
x_2
w_2
x_m
w_m
Input values
weights
Summing function
Local Field
v
Activation function
$\varphi(-)$
Output
y

Bias of a Neuron

- The bias ***b*** has the effect of applying a translation to the weighted sum ***u***

$$v = u + b$$

- ***v*** is called **induced field** of the neuron



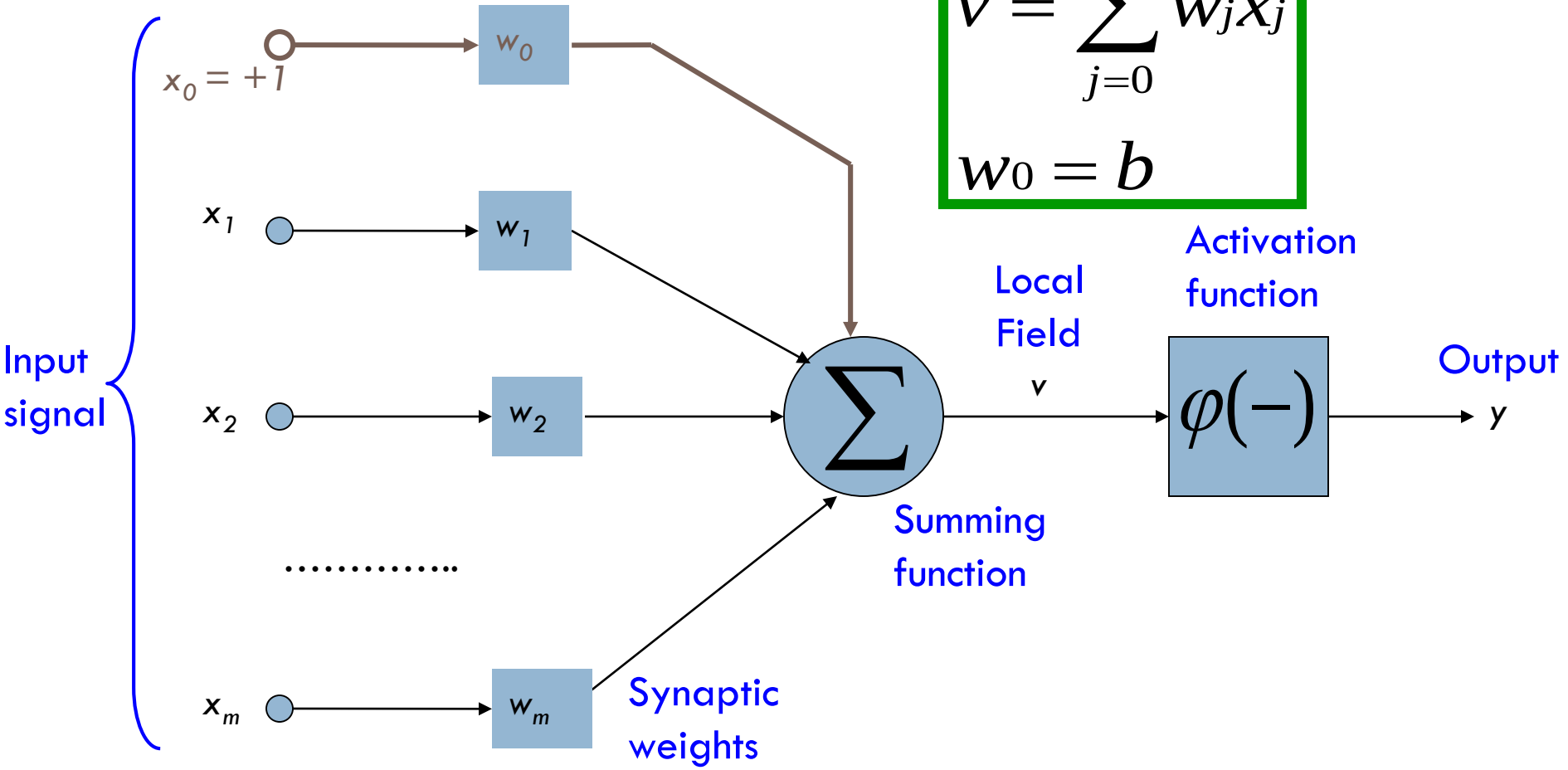
$$u = x1 - x2$$

Bias as extra input

• The bias is an external parameter of the neuron. *It* can be modeled by adding an extra input.

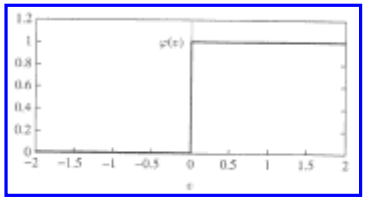
$$v = \sum_{j=0}^{m} w_j x_j$$

$$w_0 = b$$

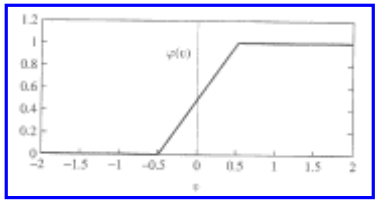


Activation Function

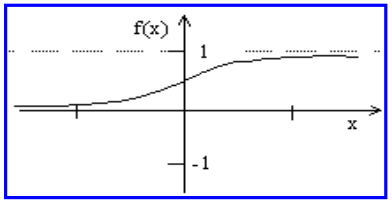
There are different activation functions used in different applications. The most common ones are:



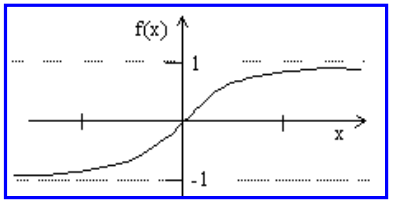
Hard-limiter



Piecewise linear



Sigmoid



Hyperbolic tangent

$$\varphi(v)=\begin{cases}1 & if\ v\geq 0\\ 0 & if\ v<0\end{cases}$$

$$\varphi(v)=\begin{cases}1 & if\ v\geq 1/2\\ v & if\ 1/2\geq v\geq -1/2\\ 0 & if\ v\leq -1/2\end{cases}$$

$$\varphi(v)=\frac{1}{1+\exp(-av)}$$

$$\varphi(v)=\tanh(v)$$

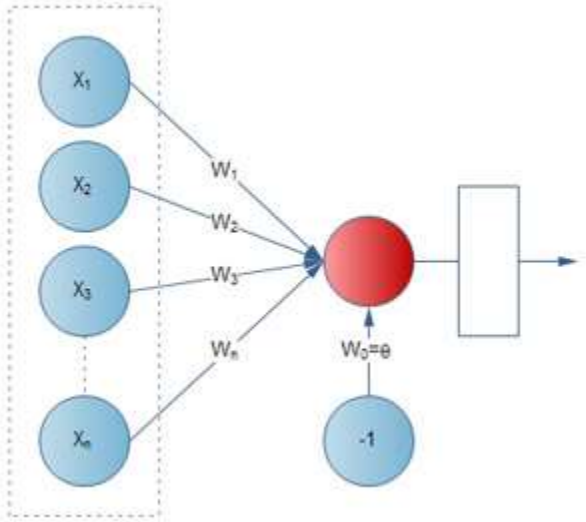
Learning Algorithms

Depend on the network architecture:

- Error correcting learning (perceptron)
- Delta rule (AdaLine, Back propagation)
- Competitive Learning (Self Organizing Maps)

Perceptron

- A type of artificial neural network invented in 1957 at the Cornell Aeronautical Laboratory by Frank Rosenblatt
- The simplest kind of feedforward neural network: a linear classifier.
- A binary classifier which maps its input x to an output value f(x) (a single binary value):



$$f(x) = \begin{cases} 1 & \text{if } w \cdot x + b > 0 \\ 0 & \text{else} \end{cases}$$

Perceptron

- f(x) (0 or 1) is used to classify x as either a positive or a negative instance: in the case of a binary classification problem.
- The bias alters the position (not the orientation) of the decision boundary.
- The perceptron learning algorithm does not terminate if the learning set is not linearly separable.

Perceptron Training

- How can we train a perceptron for a classification task?

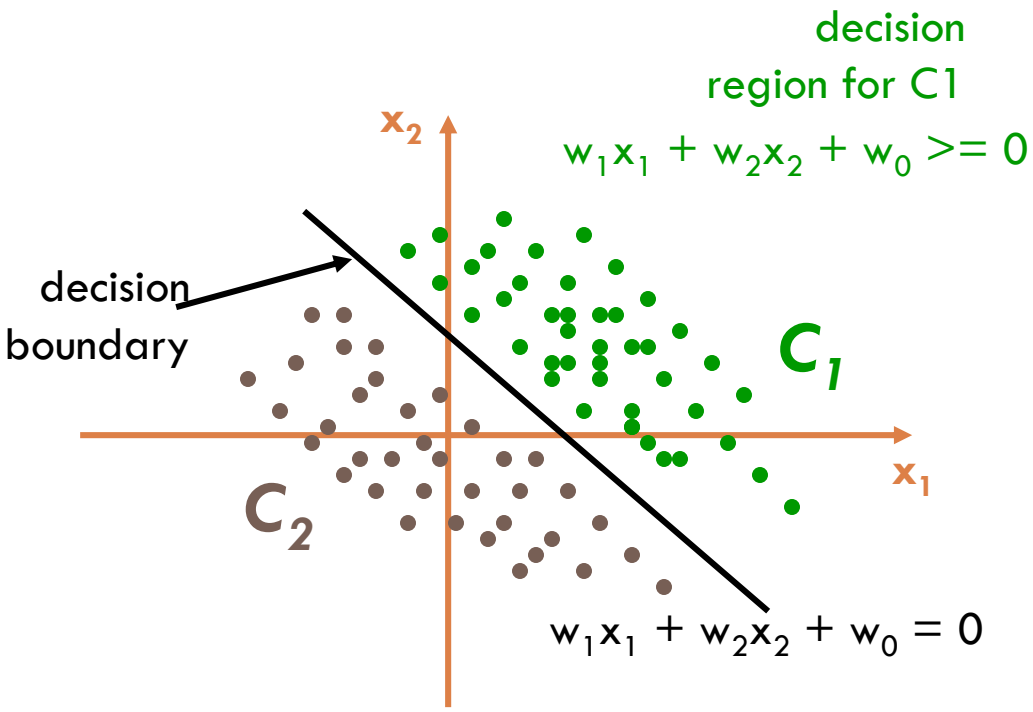
- We try to find suitable values for the weights in such a way that the training examples are correctly classified.

- Geometrically, we try to find a hyper-plane that separates the examples of the two classes.

Perceptron Geometric View

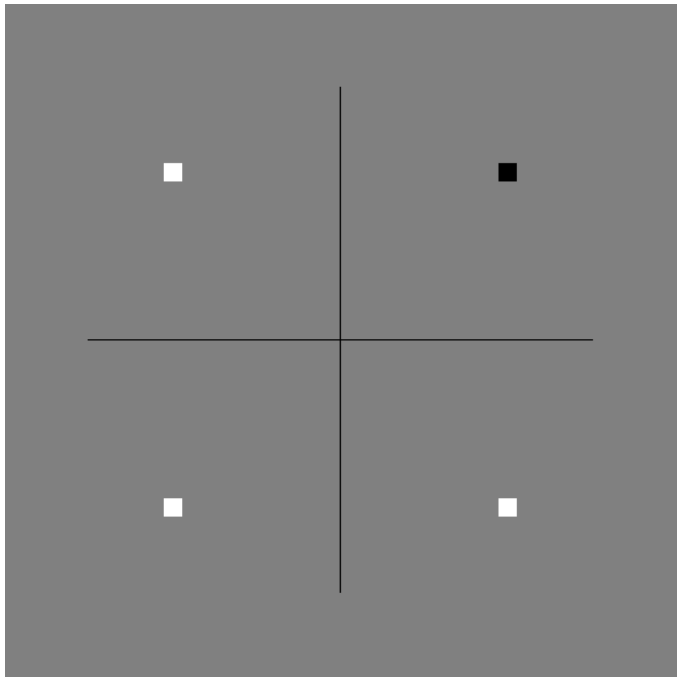
The equation below describes a (hyper-)plane in the input space consisting of real valued 2D vectors. The plane splits the input space into two regions, each of them describing one class.

$$\sum_{i=1}^{2} w_i x_i + w_0 = 0$$



Example: AND

- Here is a representation of the AND function
- White means *false*, black means *true* for the output
- -1 means *false*, +1 means *true* for the input



-1 AND -1 = false

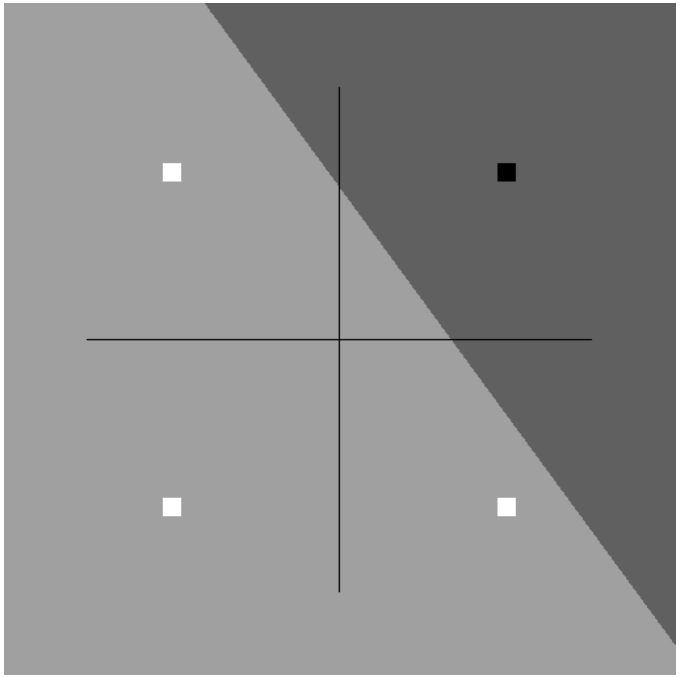
-1 AND +1 = false

+1 AND -1 = false

+1 AND +1 = true

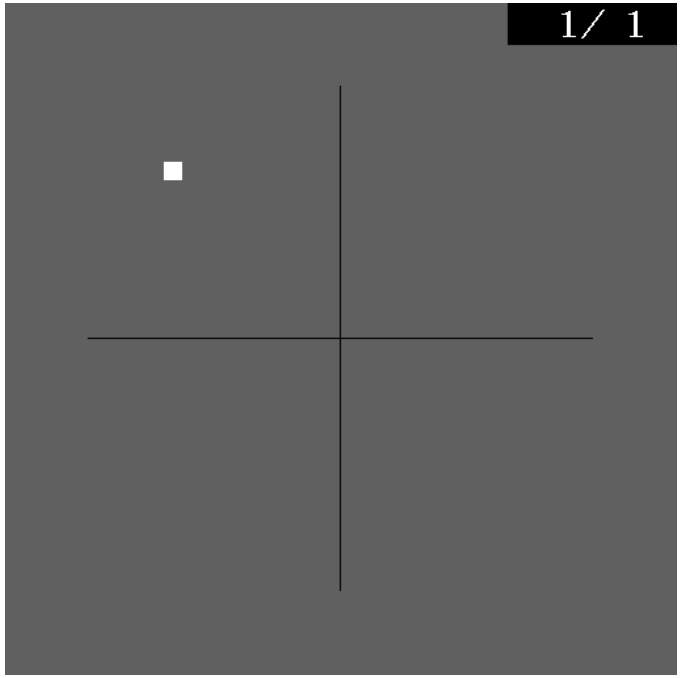
Example: AND continued

- A linear decision surface separates *false* from *true* instances



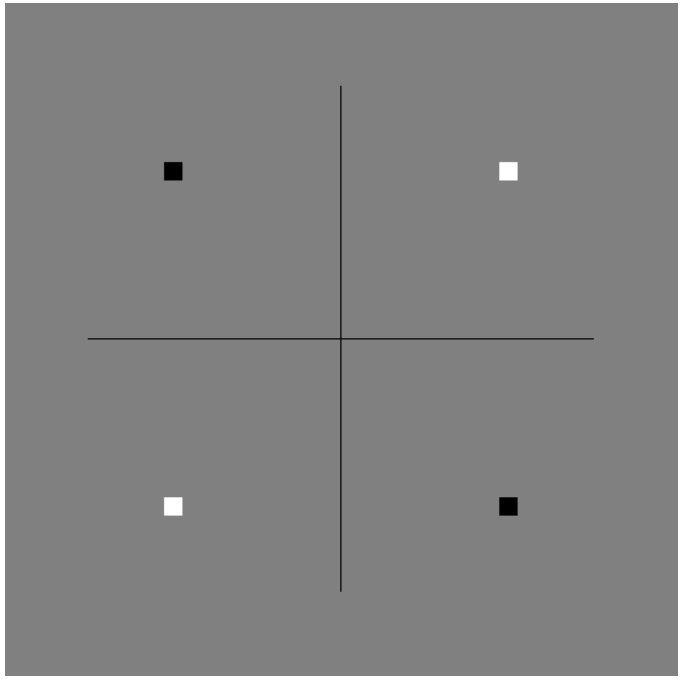
Example: AND continued

- Watch a perceptron learn the AND function:



Example: XOR

- Here's the XOR function:



-1 XOR -1 = *false*

-1 XOR +1 = *true*

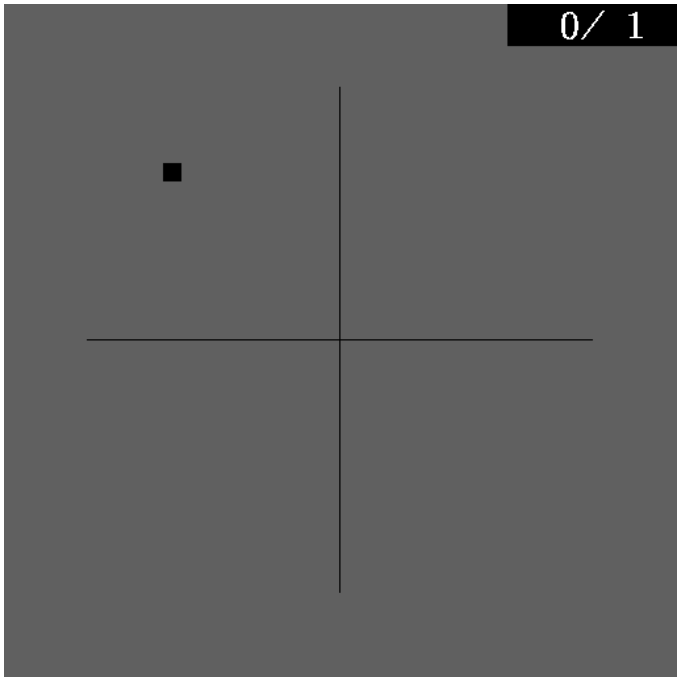
+1 XOR -1 = *true*

+1 XOR +1 = *false*

Perceptron cannot learn such *linearly not separable* functions

Example: XOR continued

- Watch a perceptron try to learn XOR



Example

-1	-1	-1	-1	-1	-1	-1	-1
-1	-1	+1	+1	+1	+1	-1	-1
-1	-1	-1	-1	-1	+1	-1	-1
-1	-1	-1	+1	+1	+1	-1	-1
-1	-1	-1	-1	-1	+1	-1	-1
-1	-1	-1	-1	- 1	+1	-1	-1
-1	-1	+1	+1	+1	+1	-1	-1
-1	-1	-1	-1	-1	-1	-1	-1

Example

- How to train a perceptron to recognize this 3?
- Assign –1 to weights of input values that are equal to -1, +1 to weights of input values that are equal to +1, and –63 to the bias.
- Then the output of the perceptron will be 1 when presented with a “prefect” 3, and at most –1 for all other patterns.

Example

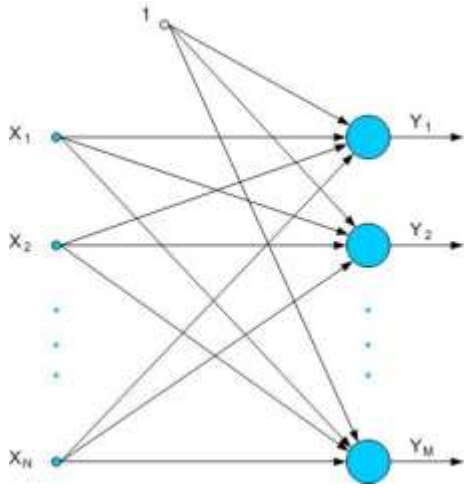
-1	-1	-1	-1	-1	-1	-1	-1
-1	-1	+1	+1	+1	+1	-1	-1
-1	-1	-1	-1	-1	+1	-1	-1
-1	-1	-1	+1	+1	+1	-1	-1
-1	+1	-1	-1	-1	+1	-1	-1
-1	-1	-1	-1	- 1	+1	-1	-1
-1	-1	+1	+1	+1	+1	-1	-1
-1	-1	-1	-1	-1	-1	-1	-1

Example

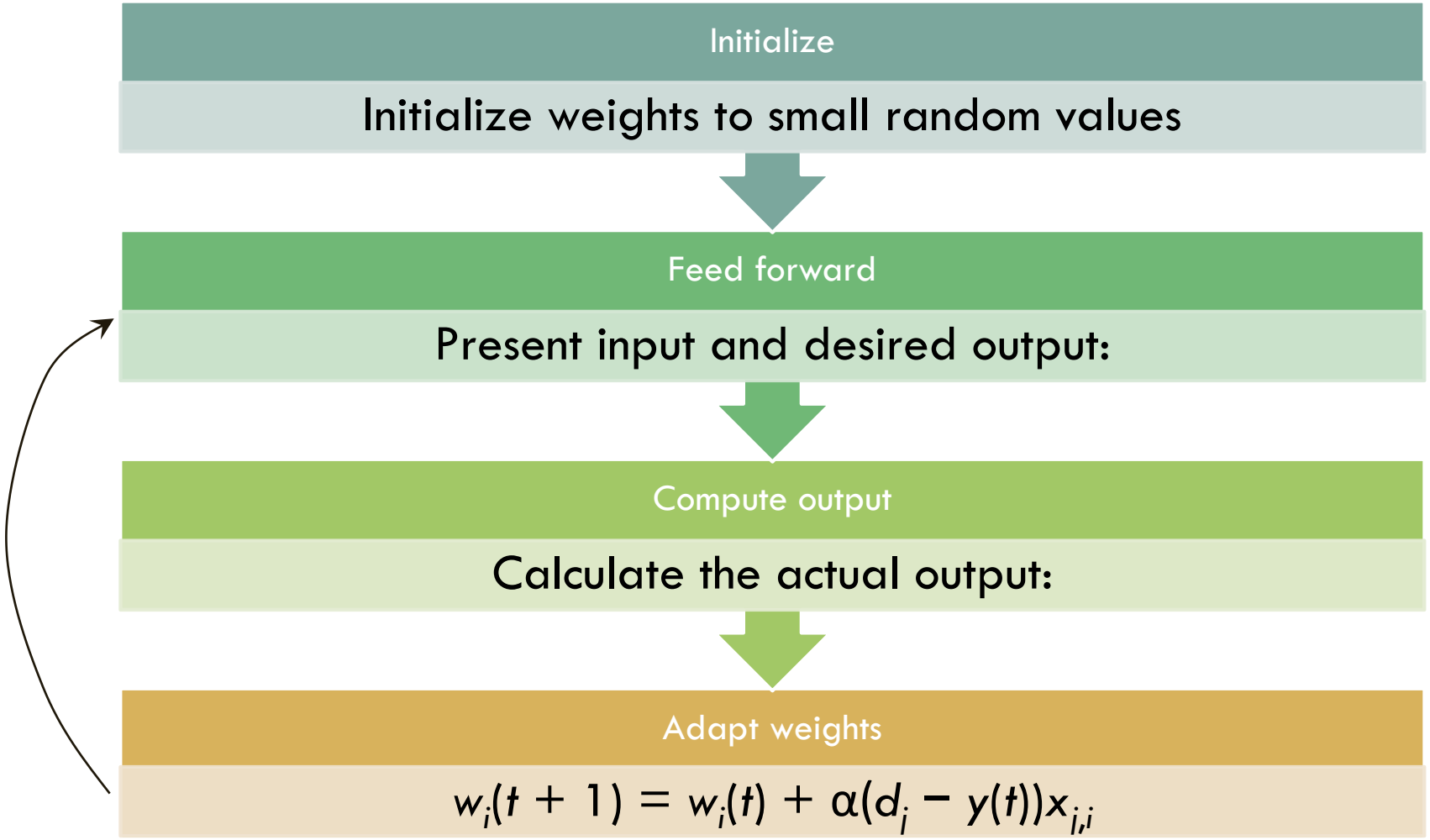
- What if a slightly different 3 is to be recognized, like the one in the previous slide?
- The original 3 with one bit corrupted would produce a sum equal to –1.
- If the bias is set to –61 then also this corrupted 3 will be recognized, as well as all patterns with one corrupted bit.

Learning Algorithm

- Assume we have a training set: $D = \{(\mathbf{x}_1, d_1), \ldots, (\mathbf{x}_m, d_m)\}$
  - x_j is a n-dimensional input vector & d_j is desired output
- Target:
  - Desired output
- Output:
  - Actual output



Learning Algorithm



Learning Algorithm

- Steps 3 and 4 are repeated until the iteration error $d_j - y(t)$ is less than a user-specified threshold, or a predetermined number of iterations have been completed.

- Novikoff (1962) proved that the perceptron algorithm converges after a finite number of iterations if the data set is linearly separable.

- α is learning rate, which affects the learning time
  - Learning is usually slower for small values and faster for larger values of α
  - α is usually something below 1, e.g. 0.1

Pseudo Code for Learning Algorithm

- Variables and parameters at iteration n of the learning algorithm:

  x (n) = input vector

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

  w(n) = weight vector

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

  b(n) = bias

  a(n) = actual response from the perceptron

  d(n) = desired response

  η= learning rate parameter (real number)

  - Too small an η produces slow convergence.
  - Too large of an η can cause oscillations in the process.

Pseudo Code for Learning Algorithm

k=1;

initialize $\mathbf{w}_i(k)$ randomly;

while (there is a misclassified training example)

Select the misclassified example ($\mathbf{x}(n)$,*d(n)*)

$\mathbf{w}_i(k+1) = \mathbf{w}_i(k) + \Delta w_i$

where $\Delta w_i = \eta \{d(n)-a(n)\} \cdot \mathbf{x}_i(n)$;

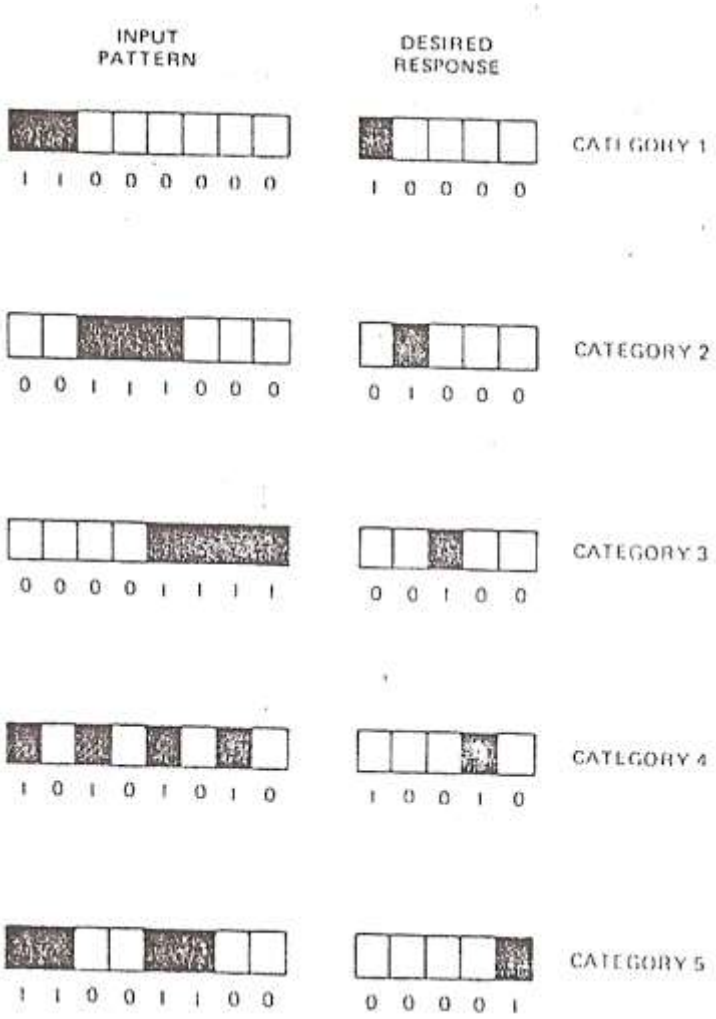
k = k+1;

end-while;

η= learning rate parameter (real number)

Example

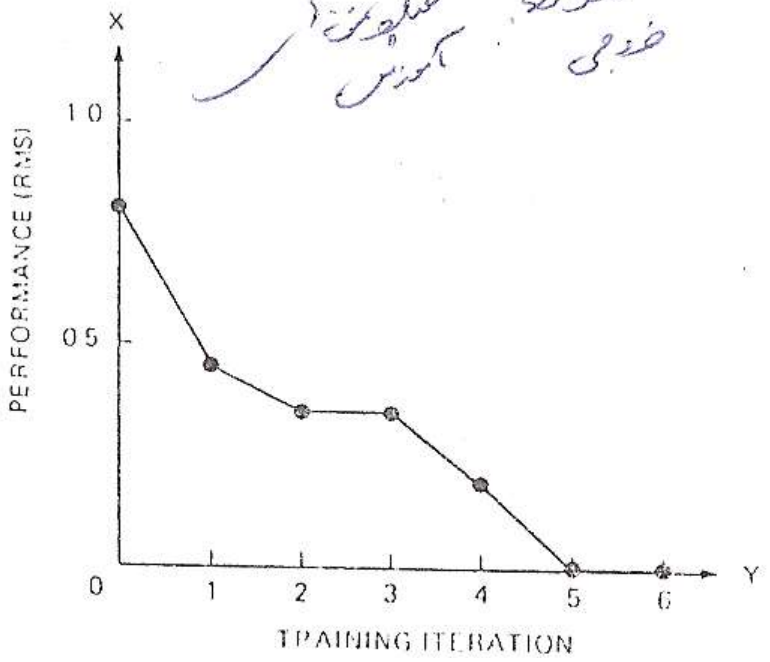
- The goal of training is to arrive at a single set of weights that allow each of mappings to be done successfully by the network.



Training

- Performance function: RMS Error

$$E(RMS) = \sqrt{\frac{\Sigma_p \Sigma_j (t_{jp} - x_{jp})^2}{n_p n_o}}$$



Example

- Consider the 2-dimensional training set $C_1 \cup C_2$,
  - **C_1 = {(1,1), (1, -1), (0, -1)}** **with class label 1**
  - **C_2 = {(-1,-1), (-1,1), (0,1)}** **with class label 0**

- Train a perceptron on $C_1 \cup C_2$

Example

Add bias constant value (1):

C1: {(1, 1, 1), (1, 1, -1), (1, 0, -1)}
C2: {(1, -1,-1), (1, -1,1), (1, 0,1)}
$\alpha = 1$
Fill out this table sequentially (First pass):

Input	Weight	Desired	Actual	Update?	New weight
(1, 1, 1)	(1, 0, 0)	1	1	No	(1, 0, 0)
(1, 1, -1)	(1, 0, 0)	1	1	No	(1, 0, 0)
(1,0, -1)	(1, 0, 0)	1	1	No	(1, 0, 0)
(1,-1, -1)	(1, 0, 0)	0	1	Yes	(0, 1, 1)
(1,-1, 1)	(0, 1, 1)	0	0	No	(0, 1, 1)
(1, 0, 1)	(0, 1, 1)	0	1	Yes	(-1, 1, 0)

Example

C1: {(1, 1, 1), (1, 1, -1), (1, 0, -1)}
C2: {(1, -1,-1), (1, -1,1), (1, 0,1)}

Fill out this table sequentially (Second pass):

Input	Weight	Desired	Actual	Update?	New weight
(1, 1, 1)	(-1, 1, 0)	1	0	Yes	(0, 2, 1)
(1, 1, -1)	(0, 2, 1)	1	1	No	(0, 2, 1)
(1,0, -1)	(0, 2, 1)	1	0	Yes	(1, 2, 0)
(1,-1, -1)	(1, 2, 0)	0	0	No	(1, 2, 0)
(1,-1, 1)	(1, 2, 0)	0	0	No	(1, 2, 0)
(1, 0, 1)	(1, 2, 0)	0	1	Yes	(0, 2, -1)

Example

C1: {(1, 1, 1), (1, 1, -1), (1, 0, -1)}

C2: {(1, -1,-1), (1, -1,1), (1, 0,1)}

Fill out this table sequentially (Third pass):

Input	Weight	Desired	Actual	Update?	New weight
(1, 1, 1)	(0, 2, -1)	1	1	No	(0, 2, -1)
(1, 1, -1)	(0, 2, -1)	1	1	No	(0, 2, -1)
(1,0, -1)	(0, 2, -1)	1	1	No	(0, 2, -1)
(1,-1, -1)	(0, 2, -1)	0	0	No	(0, 2, -1)
(1,-1, 1)	(0, 2, -1)	0	0	No	(0, 2, -1)
(1, 0, 1)	(0, 2, -1)	0	0	No	(0, 2, -1)

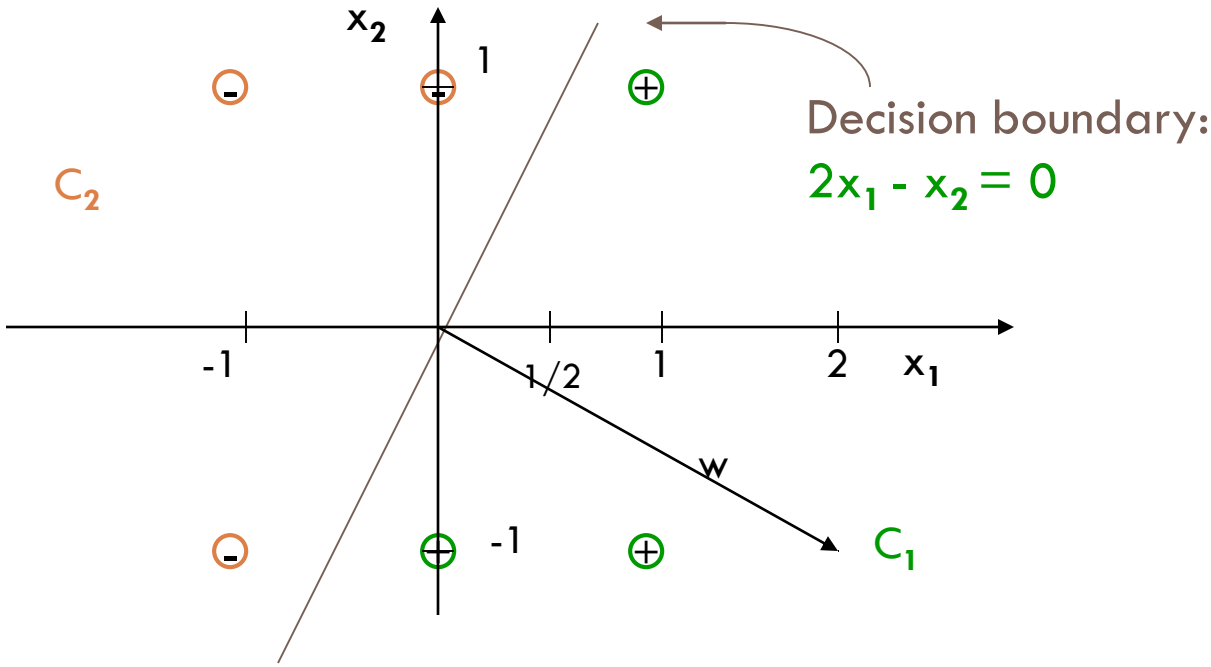
At epoch 3 no weight changes.

$\Rightarrow$ stop execution of algorithm.

Final weight vector: (0, 2, -1).

$\Rightarrow$ decision hyper-plane is $2x_1 - x_2 = 0$.

Result



Some Unhappiness About Perceptron Training

- The perceptron learning rule fails to converge if examples are not linearly separable
  - Can only model linearly separable classes, like (those described by) the following Boolean functions:
    - AND, OR, NAND & NOR
- When a perceptron gives the right answer, no learning takes place.
- Anything below the threshold is interpreted as “no”, even if it just below the threshold.
  - Might it be better to train the neuron based on **how far** below the threshold it is?

Break

اگر هر روز راه را عوض کنی هرگز به مقصد نمی رسی. بالانش

اگر همان کاری را انجام دهید که همیشه انجام می دادید همان نتیجه ای را می گیرید که همیشه می گرفتید.

قضاوت فوری درباره چیزهایی که جنبه های مختلف دارند، دلیل کم عقلی و دیوانگی است. مونتین

هر که گره از کار مسلمانی بگشاید خداوند در دنیا و آخرت گره از کارش خواهد گشود. امام حسین «ع»