ONLY GOD

1402/ 2023

Neural Network & Deep Learning

Single-Layer Feedforward Networks for Classification

CSE & IT Department

School of ECE

Shiraz University

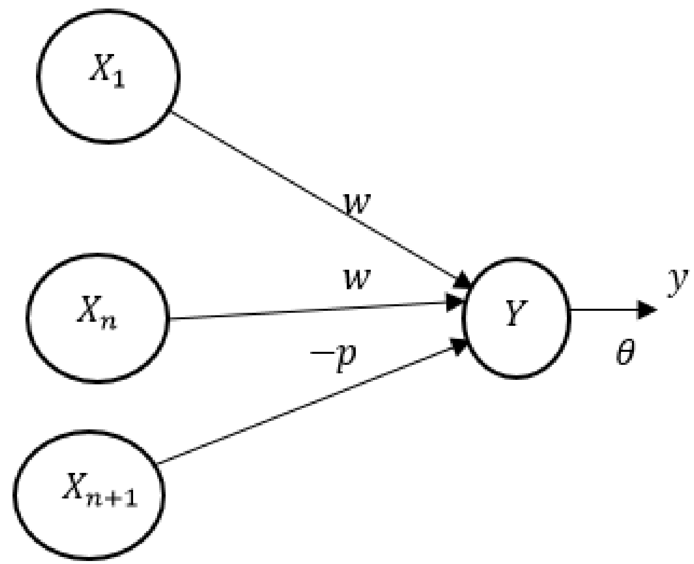
Mc-culloch Pitts Neuron
as
Logic Function

Mc-culloch Pitts (MP) Neuron

mansoori@shirazu.ac.ir

- The earliest artificial neuron
- Its activation is binary

$$y = f(y\_in) = \begin{cases} 1 & \text{if } \; y\_in \geq \theta \\ 0 & \text{if } \; y\_in < \theta \end{cases}$$



- Each weight is excitatory ($w > 0$) or inhibitory ($-p < 0$)
- All excitatory weights into a neuron are equal
- The inhibition is absolute: any inhibitory input will prevent the neuron from firing $\quad y\_in = nw - p < \theta$

MP Neuron as Logic Function

mansoori@shirazu.ac.ir

- By determining the weights and threshold, MP neuron can represent any logic function

Logic gate OR

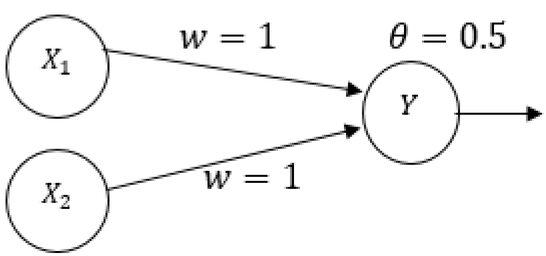
s_1	s_2	t	y_in	y
1	1	1	2	1
1	0	1	1	1
0	1	1	1	1
0	0	0	0	0

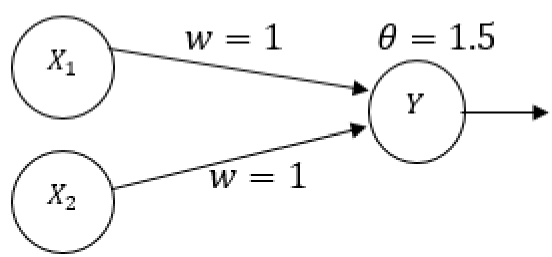
Logic gate AND

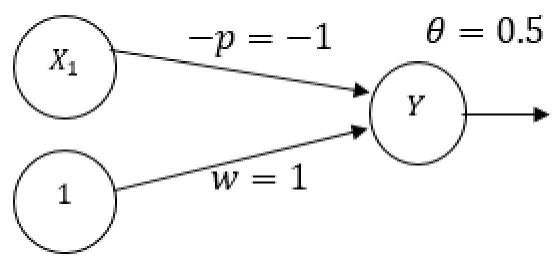
s_1	s_2	t	y_in	y
1	1	1	2	1
1	0	0	1	0
0	1	0	1	0
0	0	0	0	0

Logic gate NOT

s_1	t	y_in	y
1	0	0	0
0	1	1	1





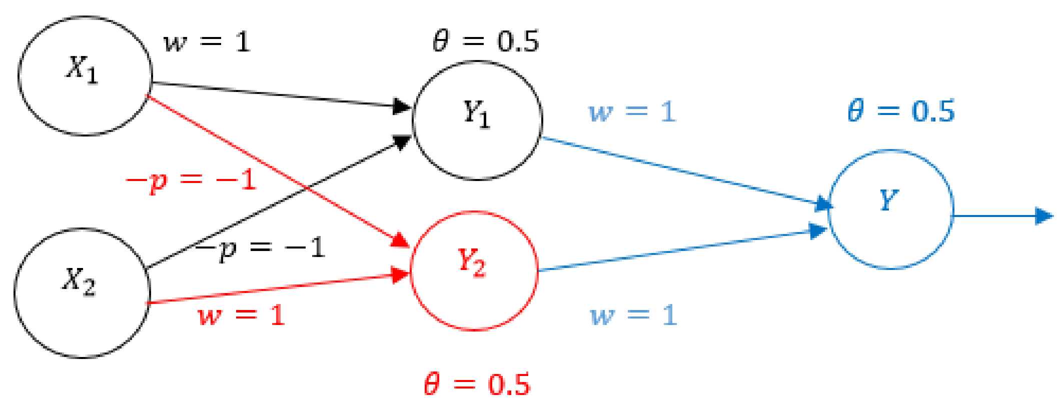


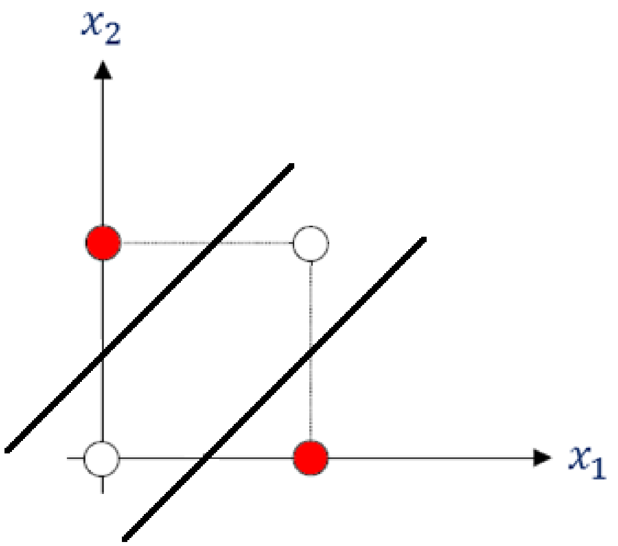
Network of MP Neurons

m a n s o o r i @shirazu.ac.ir

- One neuron can't do much on its own
- Usually, a network of many neurons is built while activation flows between neurons via synapses with different strengths
- A two-layer net of MP neurons can represent XOR logic function

s_1	s_2	t	y_in	y
1	1	0	0	0
1	0	1	1	1
0	1	1	1	1
0	0	0	0	0





SLFN as Classifier (Hebb Rule)

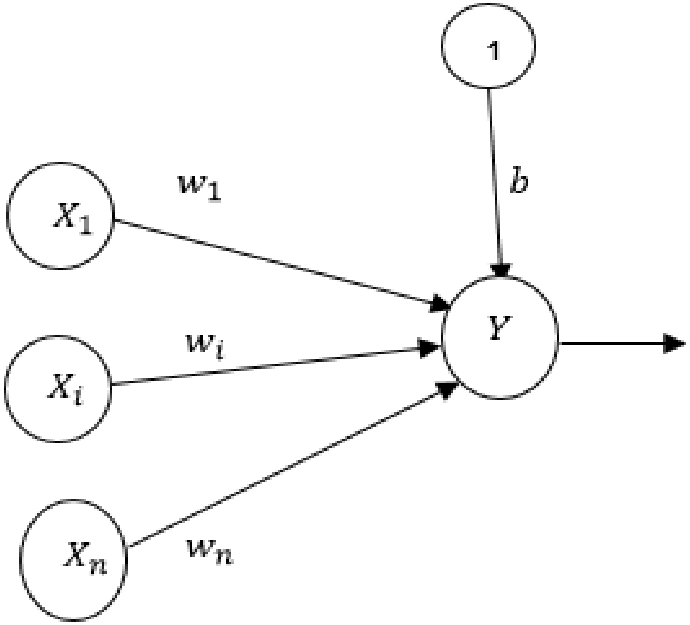
2-class NN Classifier

mansoori@shirazu.ac.ir

- Using single-layer NN with one output neuron for two classes

$$y\_in = b + \sum_{i=1}^{n} x_i w_i$$

$$y = f(y\_in) = \begin{cases} 1 & \text{if} \quad y\_in \geq 0 \\ -1 & \text{if} \quad y\_in < 0 \end{cases}$$



Decision boundary: $b + \sum_{i=1}^{n} x_i \, w_i = 0$

Learning Classifier by Hebb Rule

mansoori@shirazu.ac.ir

Algorithm: Hebb learning rule for two-class pattern classification

1. Initialize all weights, bias and learning rate

$$w_i = 0 \quad (i = 1, \dots, n), \quad b = 0, \quad \alpha = 1$$

2. for all training patterns $(p = 1, \dots, P)$

   2.1. Select p^{th} pattern

   $$< \vec{s}\,, t > = < \vec{s}(p)\,, t(p) >$$

   2.2. Set activation to input and output units

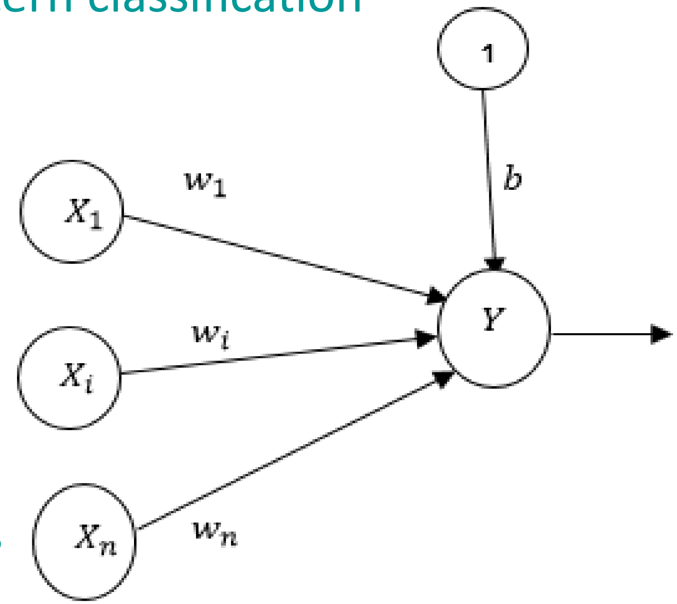
   $$x_i = s_i \quad (i = 1, \dots, n), \quad y = t$$

   2.3. Adjust weights and bias

   $$w_i(new) = w_i(old) + \alpha\, x_i\, y \quad (i = 1, \dots, n)$$

   $$b(new) = b(old) + \alpha\, y$$

3. Stop



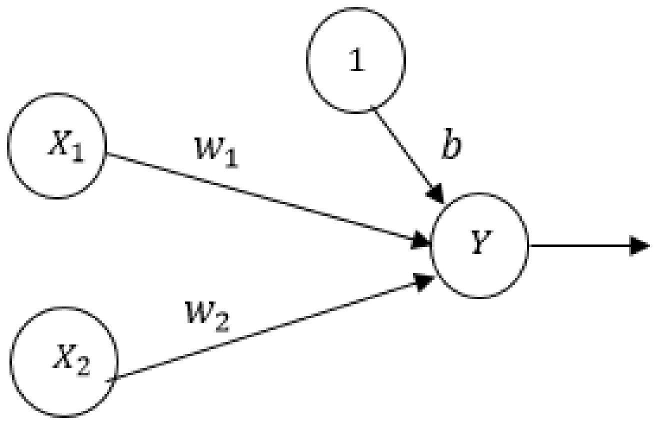
Note: order of presentation is not important

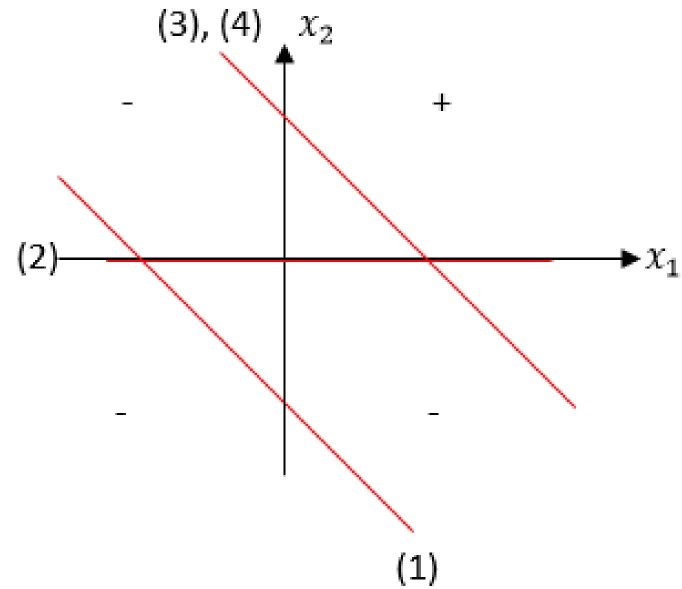
Ex. of Hebbian Classifier

mansoori@shirazu.ac.ir

Example: AND logic function

s_1	s_2	t
1	1	1
1	-1	-1
-1	1	-1
-1	-1	-1





Initials: $w_1 = 0,\ w_2 = 0,\ b = 0,\ \alpha = 1$

No.	x_1	x_2	1	y	Δw_1	Δw_2	Δb	w_1	w_2	b	$x_2 = -\frac{w_1}{w_2}x_1 - \frac{b}{w_2}$
(1)	1	1	1	1	1	1	1	1	1	1	$x_2 = -x_1 - 1$
(2)	1	-1	1	-1	-1	1	-1	0	2	0	$x_2 = 0$
(3)	-1	1	1	-1	1	-1	-1	1	1	-1	$x_2 = -x_1 + 1$
(4)	-1	-1	1	-1	1	1	-1	2	2	-2	$x_2 = -x_1 + 1$

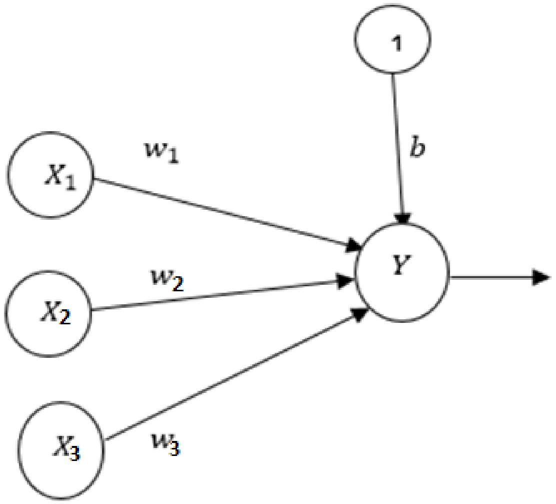
Limitations of Hebb Rule

mansoori@shirazu.ac.ir

Hebb rule has limitations

Example: 3-input AND

s_1	s_2	s_3	t
1	1	1	1
-1	1	1	-1
1	-1	1	-1
1	1	-1	-1



$$\text{if } \vec{w} = \begin{bmatrix} 0 \\ 0 \\ 0 \\ 0 \end{bmatrix} \xrightarrow{\text{learning}} \vec{w} = \begin{bmatrix} 0 \\ 0 \\ 0 \\ -2 \end{bmatrix} : \text{can not classify patterns}$$

$$\vec{w} = \begin{bmatrix} 1 \\ 1 \\ 1 \\ -2 \end{bmatrix} : \text{can classify patterns}$$

Discrete-neuron Perceptron (Perceptron Rule)

Discrete-neuron Perceptron

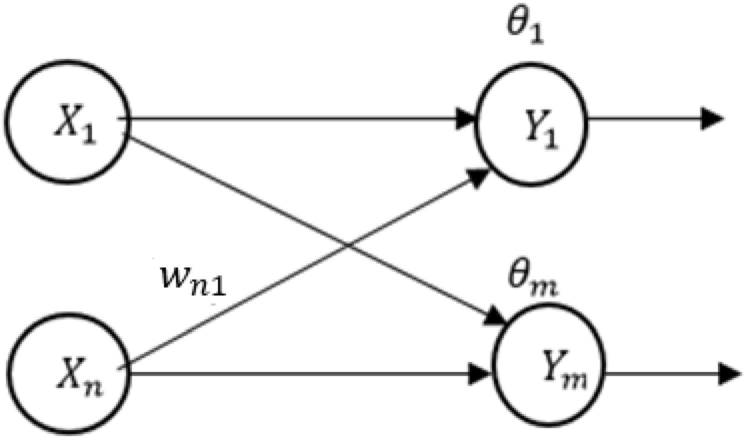
mansoori@shirazu.ac.ir

- Simplest and one of the earliest NN model proposed by Rosenblatt in 1958, 1962
- Used for pattern recognition
- Name is in use both for a particular artificial neuron model and for entire systems built from these neurons
- Heavily criticized by Minsky and Papert (1969)
  - Caused a recession in ANN-research that lasted for more than a decade
  - Until the advent of back-propagation learning for multi-layer networks (Rumelhart 1986) and recurrent networks (Hopfield 1982-85)

Discrete-neuron Perceptron

mansoori@shirazu.ac.ir

- A discrete-neuron single-layer feedforward network
  - Arrangement of one input layer of MP neurons feeding forward to one output layer of MP neurons
  - An input layer of n real-valued input nodes (not neurons)
  - An output layer of m neurons

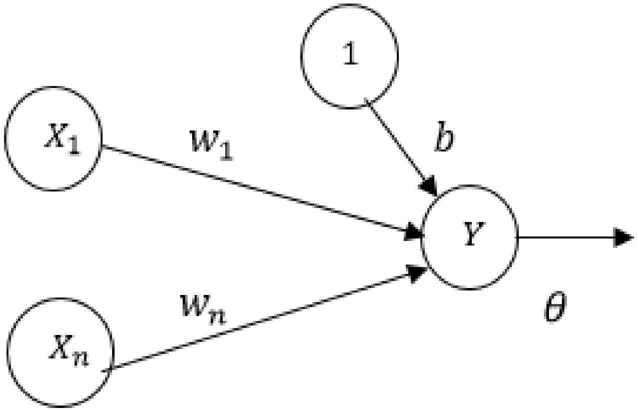


  - Each neuron has a real-valued threshold and n real-valued weights
- Computes a vector function $f : \mathcal{R}^n \longrightarrow \{-1,0,1\}^m$
- Performs classification of linearly separable patterns

Perceptron as Classifier

mansoori@shirazu.ac.ir

- Inputs: binary or bipolar
- θ : fixed
- Weights and bias: adjustable
- No sensitive to initial value of weights and bias



- Weights are updated only for patterns that do not produce correct value
- More training patterns produce the correct response, less learning occurs

Learning Classifier by Perceptron Rule

mansoori@shirazu.ac.ir

Algorithm: perceptron rule for two-class pattern classification

1. Initialize all weights and bias

$$w_i = 0 \quad (i = 1, \dots, n), \qquad b = 0 \qquad \text{(for simplicity)}$$

2. Set learning rate α, $(0 < \alpha \leq 1)$

$$\alpha = 1 \qquad \text{(for simplicity)}$$

3. While weights change do

3.1. For all training patterns $(p = 1, \dots, P)$

3.1.1. Select the p^{th} pattern

$$< \vec{s}\,, t > = < \vec{s}(p)\,, t(p) >$$

3.1.2. Set activation for input units

$$x_i = s_i \quad (\, i = 1, \dots, n)$$

Learning Classifier by Perceptron Rule

mansoori@shirazu.ac.ir

3.1.3. Compute response of output units

$$y\_in = b + \sum_{i=1}^{n} x_i w_i \Rightarrow y = f(y\_in) = \begin{cases} 1 & \text{if} \quad y\_in > \theta \\ 0 & \text{if} \; -\theta \le y\_in \le \theta \\ -1 & \text{if} \quad y\_in < -\theta \end{cases}$$

3.1.4. if error occurred $(y \neq t)$

Update weights and bias

$$w_i(new) = w_i(old) + \alpha\, t\, x_i\,, \quad b(new) = b(old) + \alpha\, t$$

else

$$w_i(new) = w_i(old)\,, \qquad b(new) = b(old)$$

4. Stop

Learning Classifier by Perceptron Rule

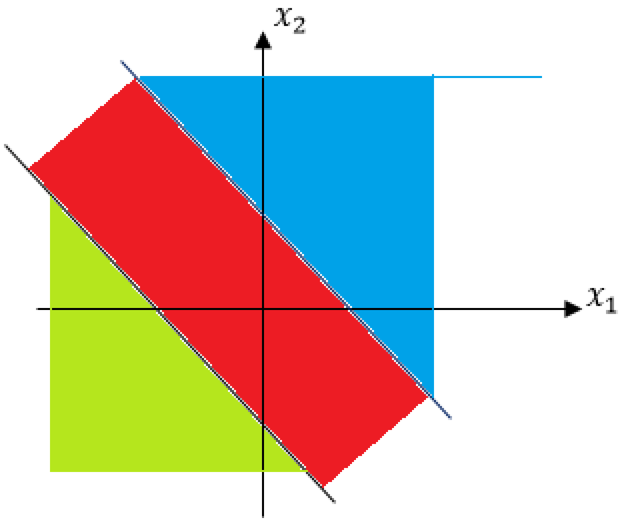
if $n = 2$

$$w_1x_1 + w_2x_2 + b > \theta \qquad \text{positive region}$$

$$\theta \geq w_1x_1 + w_2x_2 + b \geq -\theta \qquad \text{undecided region}$$

$$w_1x_1 + w_2x_2 + b < -\theta \qquad \text{negitive region}$$

$w_1 = 1, \quad w_2 = 1,$
$b = 0, \quad \theta = 1$

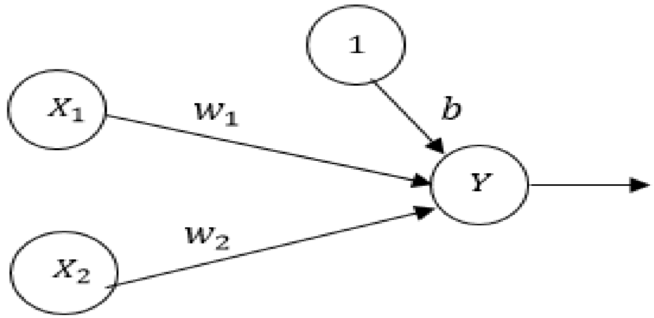


Ex. of Perceptron as Classifier

mansoori@shirazu.ac.ir

Example: NAND logic function

s_1	s_2	t
1	1	-1
1	-1	1
-1	1	1
-1	-1	1



$\alpha = 1$

$\theta = 0$

$w_1 = 0$

$w_2 = 0$

$b = 0$

Separating line: $x_2 = -x_1 + 1$

Epoch 1:

x_1	x_2	1	y_in	y	t	Δw_1	Δw_2	Δb	w_1	w_2	b
1	1	1	0	0	-1	-1	-1	-1	-1	-1	-1
1	-1	1	-1	-1	1	1	-1	1	0	-2	0
-1	1	1	-2	-1	1	-1	1	1	-1	-1	1
-1	-1	1	3	1	1	0	0	0	-1	-1	1

Epoch 2:

x_1	x_2	1	y_in	y	t	Δw_1	Δw_2	Δb	w_1	w_2	b
1	1	1	-1	-1	-1	0	0	0	-1	-1	1
1	-1	1	1	1	1	0	0	0	-1	-1	1
-1	1	1	1	1	1	0	0	0	-1	-1	1
-1	-1	1	3	1	1	0	0	0	-1	-1	1

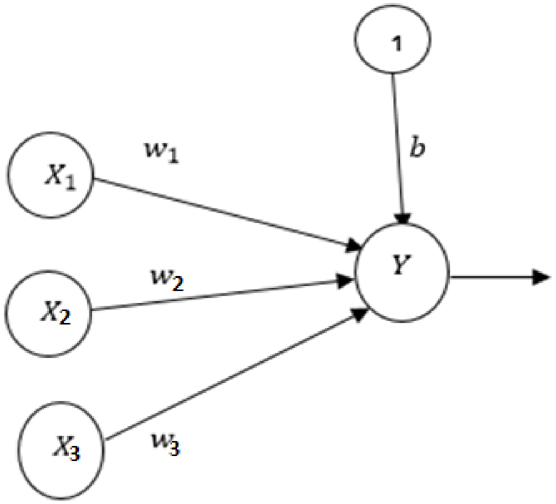
Capability of Perceptron Rule

mansoori@shirazu.ac.ir

Perceptron rule is more powerful than Hebb rule

Example: 3-input AND

s_1	s_2	s_3	t
1	1	1	1
-1	1	1	-1
1	-1	1	-1
1	1	-1	-1



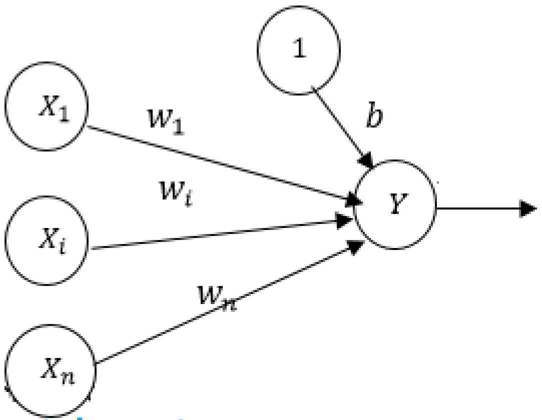
$$\text{if } \vec{w} = \begin{bmatrix} 0 \\ 0 \\ 0 \\ 0 \end{bmatrix} \xrightarrow{\text{after learning in 8 epochs}} \vec{w} = \begin{bmatrix} 2 \\ 2 \\ 2 \\ -4 \end{bmatrix} : \text{can classify patterns}$$

mansoori@shirazu.ac.ir

ADALINE
(Delta Rule)

ADALINE: ADAptive LInear NEuron

mansouri@shirazu.ac.ir



- Developed by Widrow and Hoff in 1960
- Uses bipolar representation for input and output units
- Weights and bias are adjustable
- During training, the activation function is the identity

  $$y = f(y\_in) = y\_in$$

- Trained using the delta rule: $\Delta w_i = \alpha\,(t - y)\,x_i$
- After training, a threshold function ($\theta = 0$) is used as activation

  function $y = f(y\_in) = \begin{cases} +1 & \text{if } \; y\_in > 0 \\ 0 & \text{if } \; y\_in = 0 \\ -1 & \text{if } \; y\_in < 0 \end{cases}$, $\quad y = sgn(y\_in)$

- Can model any linearly separable problem

Learning ADALINE by Delta Rule

mansoori@shirazu.ac.ir

Algorithm: training ADALINE for two-class pattern classification

1. Initialize weights and bias: $w_i\, , b$: small random values $(i = 1, \dots, n)$
2. Set training rate $\alpha\, , (0 < \alpha \leq 1)$: too slow: $0.1 \leq n\alpha \leq 1$:not converge
3. While the largest weight change is greater than a tolerance do

   3.1. For all training patterns $(p = 1, \dots, P)$

   3.1.1. Select the p^{th} pattern: $< \vec{s}\, , t > = < \vec{s}(p)\, , t(p) >$

   3.1.2. Set activation for input units

   $$x_i = s_i \qquad (i = 1, \dots, n)$$

   3.1.3. Compute the activation of output unit

   $$y\_in = b + \sum_{i=1}^{n} x_i w_i \Rightarrow y = f(y\_in) = y\_in$$

   3.1.4. Update the weights and bias

   $$w_i(new) = w_i(old) + \alpha\,(t - y)x_i$$

   $$b(new) = b(old) + \alpha\,(t - y)$$

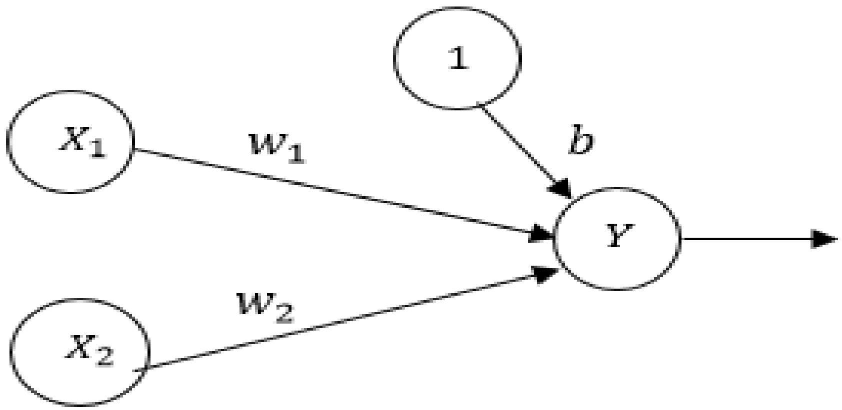
4. Stop

Ex. of ADALINE

mansoori@shirazu.ac.ir

Example: AND logic function

s_1	s_2	t
1	1	1
1	-1	-1
-1	1	-1
-1	-1	-1



$$\alpha = 0.2, \quad w_1 = 0.1\,, w_2 = 0.3\,, b = 0.2$$

x_1	x_2	1	t	y_in	Δw_1	Δw_2	Δb	w_1	w_2	b
1	1	1	1	0.60	0.08	0.08	0.08	0.18	0.38	0.28
1	-1	1	-1	0.08	-0.22	0.22	-0.22	-0.04	0.60	0.06
-1	1	1	-1	0.70	0.34	-0.34	-0.34	0.30	0.26	-0.28
-1	-1	1	-1	-0.84	0.03	0.03	-0.03	0.33	0.29	-0.31

Ex. of ADALINE

mansoori@shirazu.ac.ir

$$E = \sum_{p=1}^{4}\big(t(p) - y(p)\big)^2 = \sum_{p=1}^{4}\big(t(p) - y\_in(p)\big)^2$$
$$= \big(1 - (w_1 + w_2 + b)\big)^2 + \big(-1 - (w_1 - w_2 + b)\big)^2$$
$$+ \big(-1 - (-w_1 + w_2 + b)\big)^2 + \big(-1 - (-w_1 - w_2 + b)\big)^2 \Longrightarrow$$
$$E = 4({w_1}^2 + {w_2}^2 + b^2 + 1 - w_1 - w_2 + b)$$

$$\frac{\partial E}{\partial \vec{w}} = 0 \ \Longrightarrow \begin{cases} \dfrac{\partial E}{\partial w_1} = 0 \ \rightarrow 2w_1 - 1 = 0 \ \rightarrow \ w_1 = \dfrac{1}{2} \\ \dfrac{\partial E}{\partial w_2} = 0 \ \rightarrow 2w_2 - 1 = 0 \ \rightarrow \ w_2 = \dfrac{1}{2} \\ \dfrac{\partial E}{\partial b} = 0 \ \ \rightarrow 2b + 1 = 0 \ \rightarrow \ b = -\dfrac{1}{2} \end{cases}$$

Separating line: $x_2 = -x_1 + 1$

Continuous-neuron Perceptron (Delta Rule)

mansoori@shirazu.ac.ir

Limitations of Discrete-neuron Perceptron

- Only Boolean-valued functions can be computed
- A simple learning algorithm for multi-layer discrete-neuron perceptron is lacking
- The computational capabilities of single-layer discrete-neuron perceptron is limited
- These disadvantages disappear when we consider multi-layer continuous-neuron perceptron

mansoori@shirazu.ac.ir

Continuous-neuron Perceptron

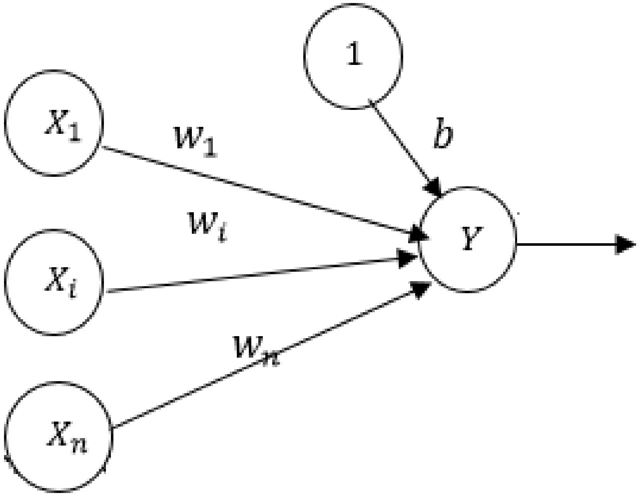
- A continuous-neuron perceptron with n inputs and m outputs computes:
  - A function $\mathcal{R}^n \longrightarrow [-1,1]^m$, when the bipolar sigmoid activation function is used
  - A function $\mathcal{R}^n \longrightarrow \mathcal{R}^m$, when a linear activation function is used
- The learning rules are based on optimization techniques for error-functions (delta rule)
  - This requires a continuous and differentiable error function
- Can model any linearly separable problem

Continuous-neuron Perceptron

mansoori@shirazu.ac.ir

- Uses real-valued/binary/bipolar representation for input and output units

- Weights and bias are adjustable



- Trained using the delta rule:

$$\Delta w_i = \alpha\,(t - y) f'(y\_in)\, x_i$$

Continuous-neuron Perceptron

mansoori@shirazu.ac.ir

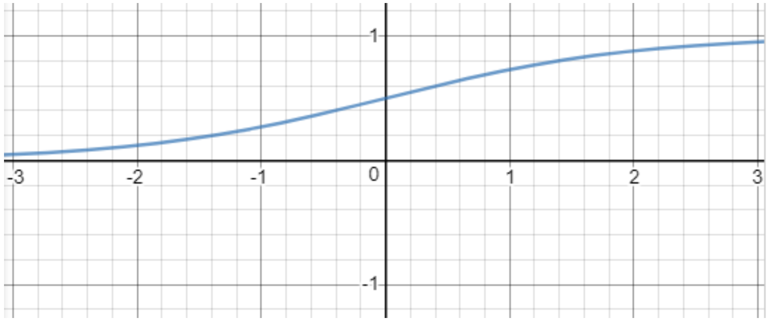
Activation functions:

Binary sigmoid: $f(y\_in) = \frac{1}{1+e^{-y\_in}}$

$=> \; f'(y\_in) = f(y\_in)\big(1 - f(y\_in)\big)$

$=> f'(y\_in) = y(1-y)$

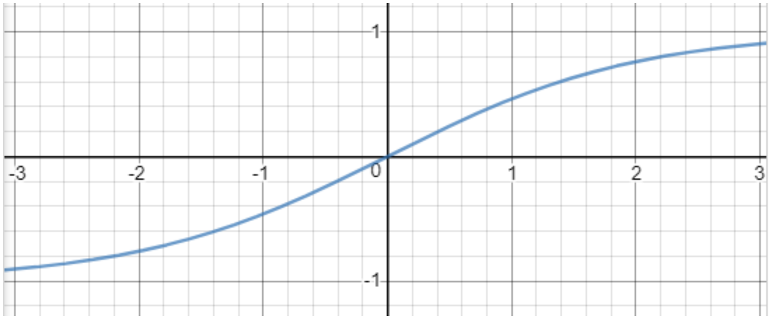
$$\Delta w_i = \alpha\,(t-y)\,y\,(1-y)\,x_i$$



Bipolar sigmoid: $f(y\_in) = \frac{1-e^{-y\_in}}{1+e^{-y\_in}}$

$=> \; f'(y\_in) = \frac{1}{2}\big(1 + f(y\_in)\big)\big(1 - f(y\_in)\big)$

$=> f'(y\_in) = \frac{1}{2}(1+y)(1-y)$



$$\Delta w_i = \alpha\,(t-y)\,(1-y^2)\,x_i$$

Learning Perceptron by Delta Rule

mansoori@shirazu.ac.ir

Algorithm: training Perceptron for two-class pattern classification

1. Initialize weights and bias: $w_i\,, b$: small random values $(i = 1, \dots, n)$

2. Set training rate $\alpha\,, (0 < \alpha \leq 1)$:

3. While the largest weight change is greater than a tolerance do

   3.1. For all training patterns $(p = 1, \dots, P)$

   3.1.1. Select the p^{th} pattern: $< \vec{s}\,, t > = < \vec{s}(p)\,, t(p) >$

   3.1.2. Set activation for input units

   $$x_i = s_i \qquad (i = 1, \dots, n)$$

   3.1.3. Compute the activation of output unit

   $$\text{y\_in} = b + \sum_{i=1}^{n} x_i w_i \;\; \Rightarrow \;\; y = f(y\_in)$$

   3.1.4. Update the weights and bias

   $$w_i(new) = w_i(old) + \alpha(t - y)y(1 - y)x_i$$

   $$b(new) = b(old) + \alpha(t - y)y(1 - y)$$

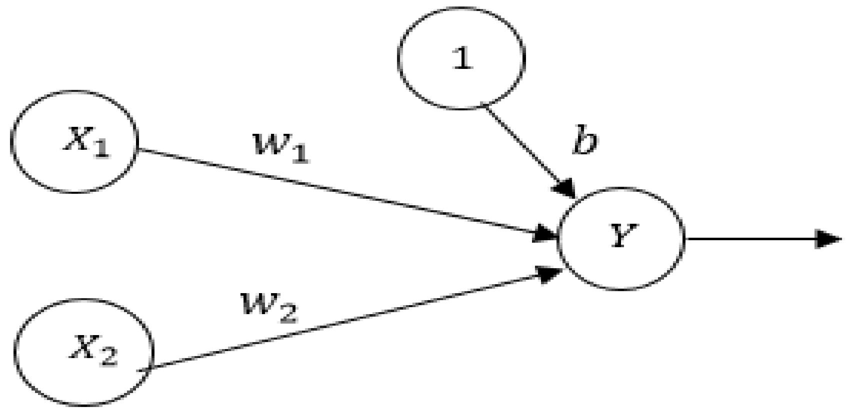
4. Stop

Ex. of Perceptron

mansoori@shirazu.ac.ir

Example: AND logic function

s_1	s_2	t
1	1	1
1	-1	-1
-1	1	-1
-1	-1	-1



Activation function of Y: bipolar sigmoid

$\alpha = 0.2, \quad w_1 = 0.1\,, w_2 = 0.3\,, b = 0.2$

x_1	x_2	1	t	y_in	y	Δw_1	Δw_2	Δb	w_1	w_2	b
1	1	1	1	0.600	0.291	0.029	0.029	0.029	0.129	0.329	0.229
1	-1	1	-1	0.029	0.015	-0.003	0.003	-0.003	0.126	0.332	0.226
-1	1	1	-1	0.432	0.213	0.041	-0.041	-0.041	0.167	0.291	0.185
-1	-1	1	-1	-0.273	-0.136	-0.027	-0.027	0.027	0.140	0.264	0.212

NN Classifier (Review)

mansoori@shirazu.ac.ir

- NN classifiers learn decision boundaries from training data
- One can train networks by iteratively updating their weights
- Trained networks are expected to generalize, i.e. deal appropriately with input data they were not trained on
- Single neuron perceptron can classify the inputs into one of two classes
- In general, an m neuron perceptron can classify the inputs into 2^m classes
- Simple Perceptron can only cope with linearly separable problems
- The Perceptron learning rule will find weights for linearly separable problems in a finite number of epochs