شبکه های عصبی مصنوعی

دانشکده مهندسی کامپیوتر
دانشگاه علم و صنعت ایران

منبع: Haykin

ویرایش: حسین علیزاده
http://webpages.iust.ac.ir/halizadeh/

بخش دوم: مدل پرسپترون2

Perceptron - history

43: McCulloch/Pitts: propose artificial neurons

49: Hebb paradigm proposed

58: Rosenblatt-perceptron (First practical application of ANN) fixed preprocessing with masks, learning algorithm, used for picture recognition

60: Widrow/Hoff: ADALINE (ADAptive Linear Neuron)

- Rosenblatt and Hoff proposed **multilayer perceptron**
  - But, not able to modify learning algorithms to train it

69: Minsky/Papert: show the restrictions of the Rosenblatt-perceptron with respect to its representational abilities

86: Rumelhart/McClelland

- Train multilayer perceptron successfully!

Adaline

- ADALINE is an acronym for ADAptive LINear Element (or ADAptive LInear NEuron) developed by Bernard Widrow and Marcian Hoff (1960).

- Variation on the Perceptron Network
  - The output **y** is a linear combination o **x**
  - inputs are +1 or -1, outputs are +1 or -1
  - uses a bias input

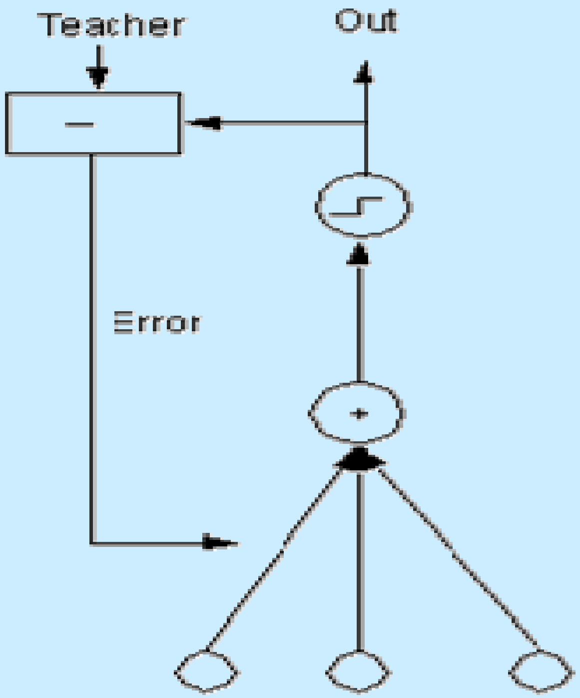
Adaline

Differences:

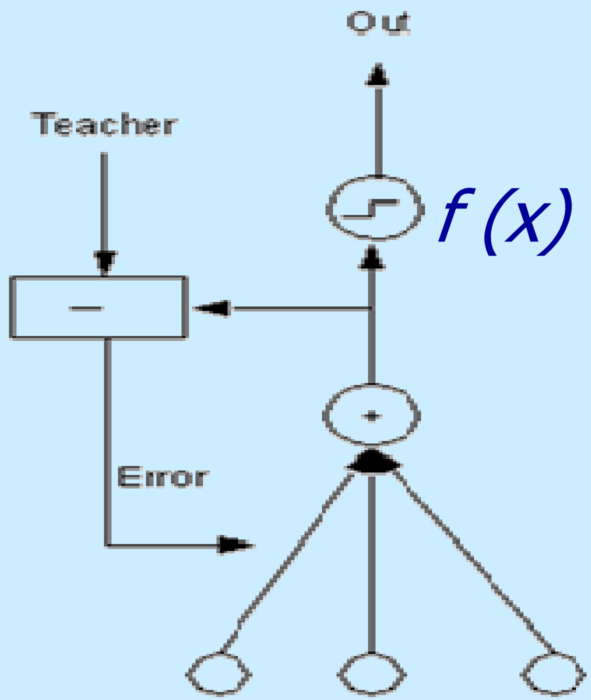
- Weights update is a function of output error
- trained using the Delta Rule
  - also called: Gradient Descent method, Steepest Descent Method , LMS rule (least mean square), Adaline rule, Widrow-Hoff rule (the inventors)
- The step function in the perceptron can be replaced with a continuous (differentiable) function *f*, e.g. the simplest is linear function
- In the case of a hard limiter as the activation function, it is not used during training (i.e. The Delta Rule applies to a Perceptron without a threshold).

Adaline

- With or without the threshold, the Adaline is trained based on the output of the function *f* rather than the final output.



Perceptron learning



Delta Rule

Learning algorithm

- The idea: try to minimize the network error (which is a function of the weights)
- So we have to:
  - Define an error measure
  - Determine the gradient of error w.r.t. changes in weights
  - Define a rule for weight update

$$\boxed{E(\mathrm{w}(\mathrm{n})) = \tfrac{1}{2}\mathrm{e}^2(\mathrm{n})}$$

$$\boxed{\mathrm{e}(\mathrm{n}) = \mathrm{d}(\mathrm{n}) - \sum_{\mathrm{j}=0}^{\mathrm{m}} \mathrm{x}_{\mathrm{j}}(\mathrm{n})\mathrm{w}_{\mathrm{j}}(\mathrm{n})}$$

- We can find the minimum of the error function *E* by means of the Steepest descent method

Gradient Descent Method

- start with an arbitrary point
- find a direction in which E is decreasing most rapidly

$$\boxed{-(\text{gradient of } E(\mathrm{w})) = -\nabla E(\mathrm{w}) = -\left[\frac{\partial E}{\partial w_1}, \ldots, \frac{\partial E}{\partial w_m}\right]}$$

- make a small step in that direction

$$\boxed{\mathrm{w}(\mathrm{n}+1) = \mathrm{w}(\mathrm{n}) - \eta(\nabla\, \mathrm{E}(\mathrm{n}))}$$

Gradient Descent Algorithm

- Approximation of gradient(E)

$$E(\mathrm{w}(\mathrm{n})) = \tfrac{1}{2}\mathrm{e}^2(\mathrm{n}) \qquad \mathrm{e}(\mathrm{n}) = \mathrm{d}(\mathrm{n}) - \sum_{\mathrm{j}=0}^{\mathrm{m}} \mathrm{x}_{\mathrm{j}}(\mathrm{n})\mathrm{w}_{\mathrm{j}}(\mathrm{n})$$

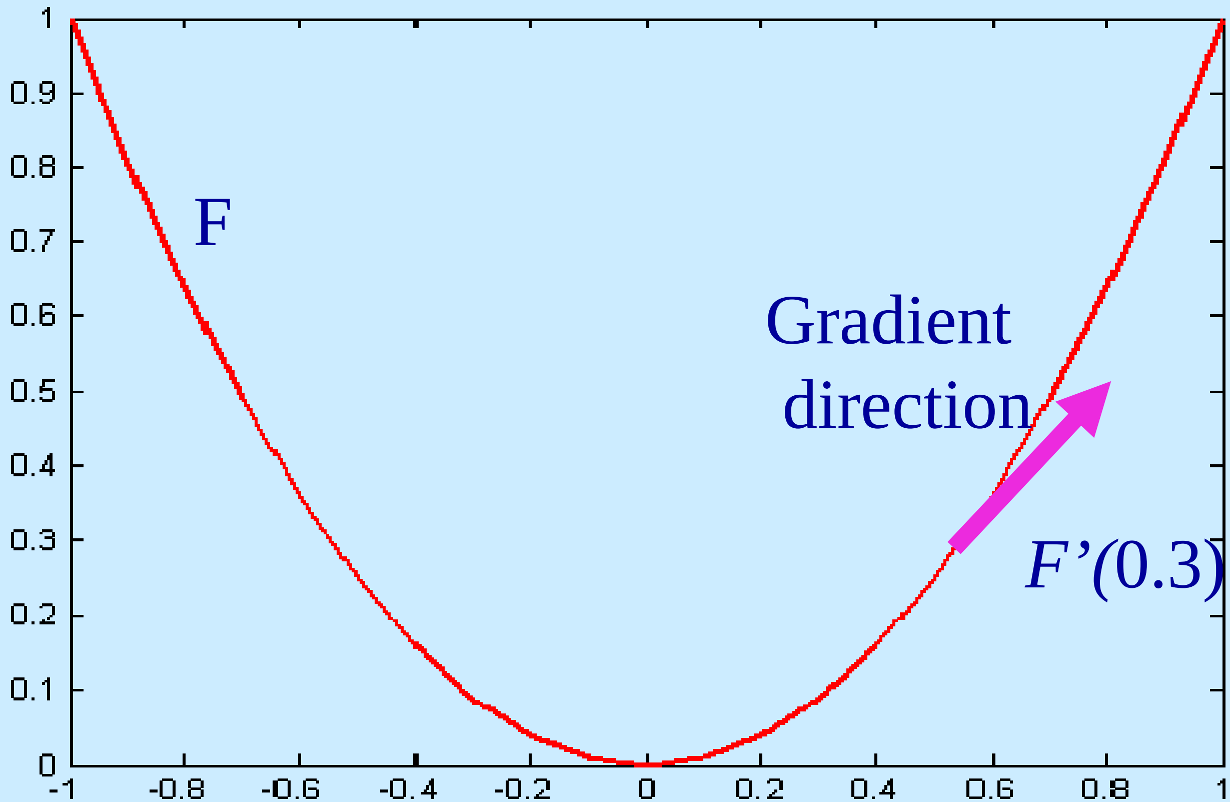
$$\frac{\partial E(\mathrm{w}(\mathrm{n}))}{\partial \mathrm{w}(\mathrm{n})} = \mathrm{e}(\mathrm{n})\frac{\partial \mathrm{e}(\mathrm{n})}{\partial \mathrm{w}(\mathrm{n})} = \mathrm{e}(\mathrm{n})[-\mathrm{x}(\mathrm{n})^{\mathrm{T}}]$$

- Update rule for the weights becomes:

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

Gradient Descent

- Gradient direction is the direction of uphill
  for example, in the Figure, at position 0.3, the gradient is uphill (F is Error, consider 1-dim case)

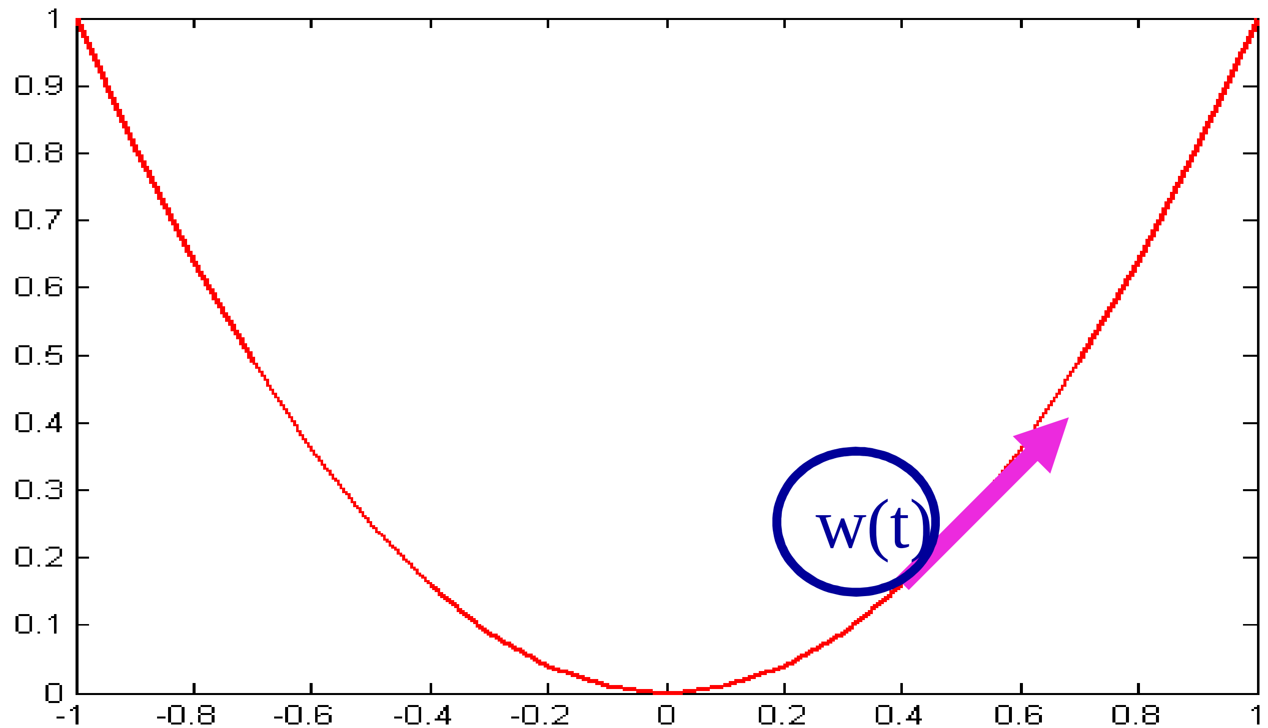


Gradient Descent

- In gradient descent algorithm, we have

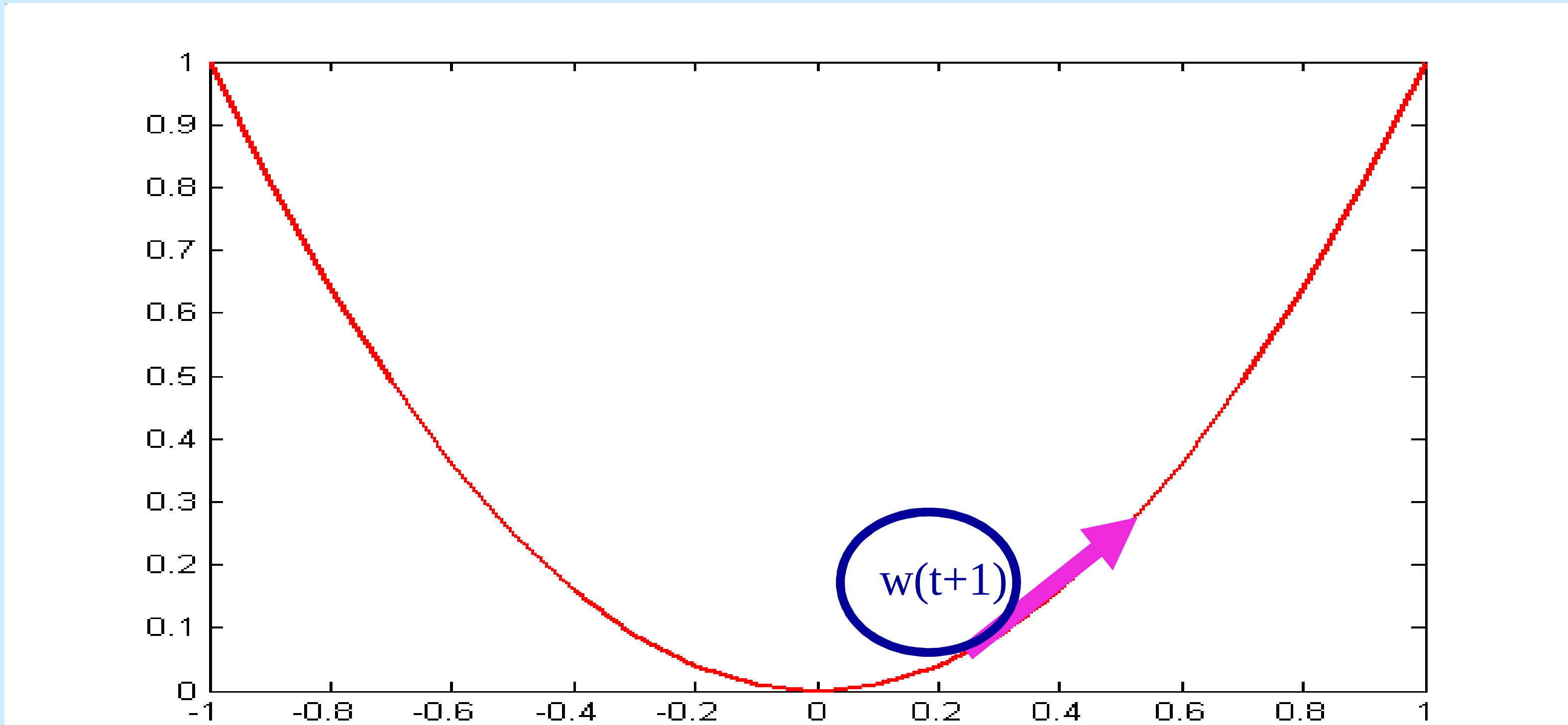
$$\underline{w}(t+1) = \underline{w}(t) - \eta\, \nabla E_{(w(t))}$$

therefore the ball goes downhill since $-\nabla E(w(t))$ is downhill direction



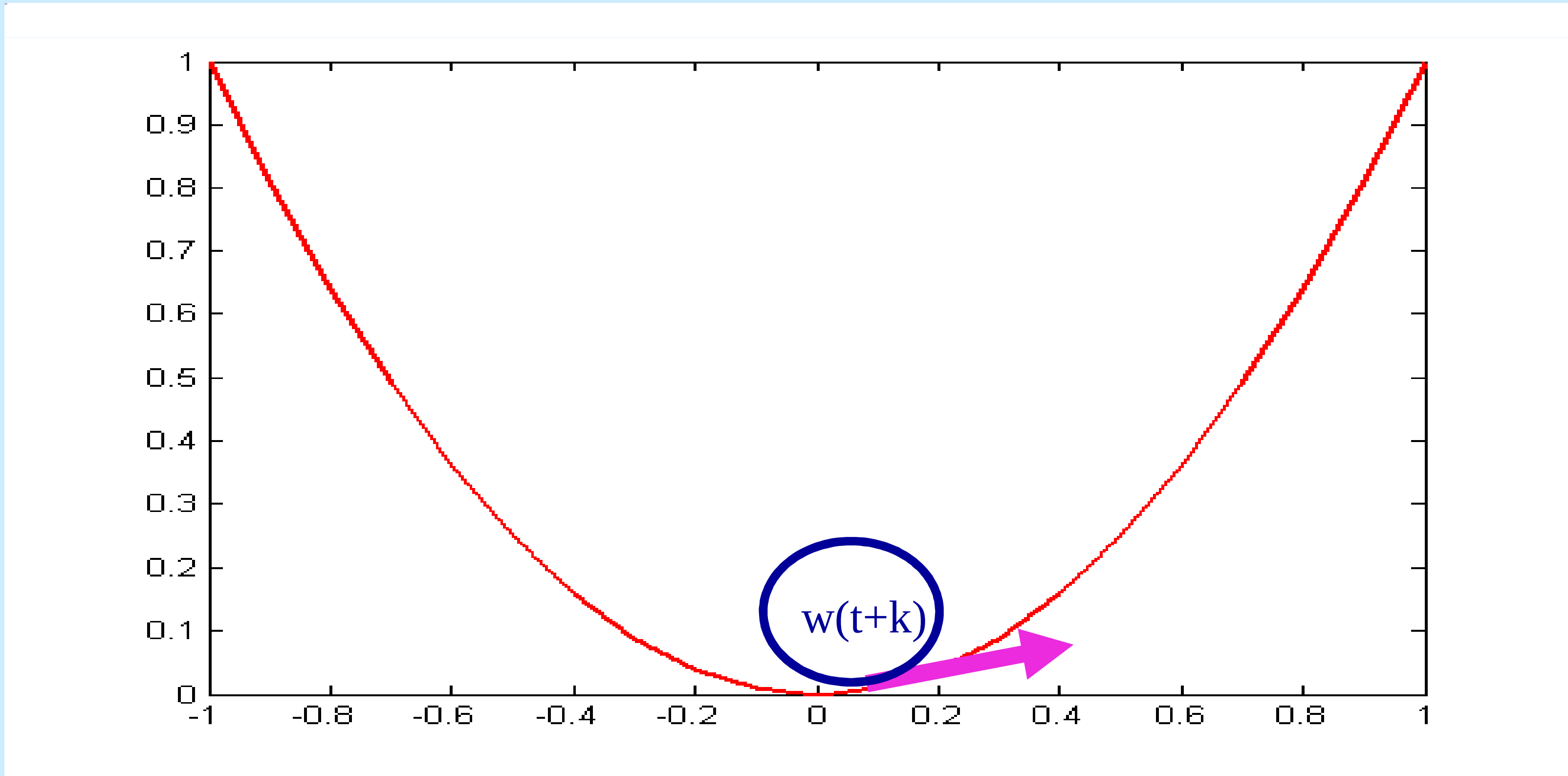
Gradient Descent

- In the next step the ball goes again downhill since $-\nabla E(w(t))$ is downhill direction



Gradient Descent

- Gradually the ball will stop at a local minima where the gradient is zero



Learning Algorithm

- **Step 0**: initialize the weights to small random values and select a learning rate, η
- **Step 1**: for each input vector *s*, with target output, *t* set the inputs to *s*
- **Step 2**: compute the neuron inputs
- **Step 3**: use the delta rule to update the bias and weights
- **Step 4**: stop if the largest weight change across all the training samples is less than a specified tolerance, otherwise cycle through the training set again

Neuron input

$$\mathbf{y = b + \sum x_i w_i}$$

Delta rule

$$\mathbf{b(new) = b(old) + \eta(d - y)}$$

$$\mathbf{w_i(new) = w_i(old) + \eta(d - y)x_i}$$

Running Adaline

- One unique feature of ADALINE is that its activation function is different for training and running

- When running ADALINE use the following:
  - initialize the weights to those found during training
  - compute the net input
  - apply the activation function

Neuron input

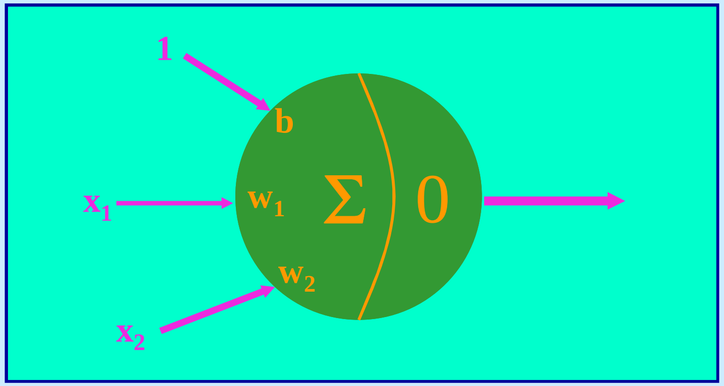
$$y = b + \quad x_i w_i$$

Activation Function

$$y = \begin{cases} 1 \text{ if } y >= 0 \\ -1 \text{ if } y < 0 \end{cases}$$

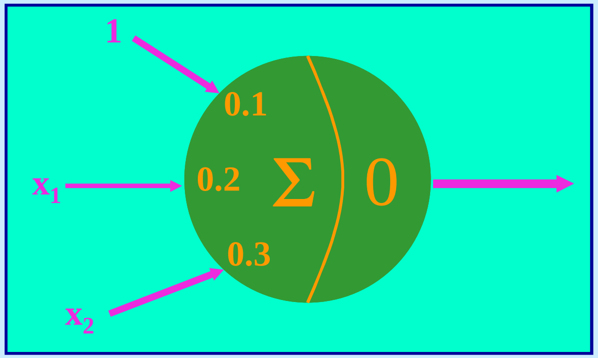
Example – AND function

- Construct an AND function for a ADALINE neuron
  - let $\alpha = 0.1$



x1	x2	bias	Target
1	1	1	1
1	-1	1	-1
-1	1	1	-1
-1	-1	1	-1

Initial Conditions: Set the weights to small random values:

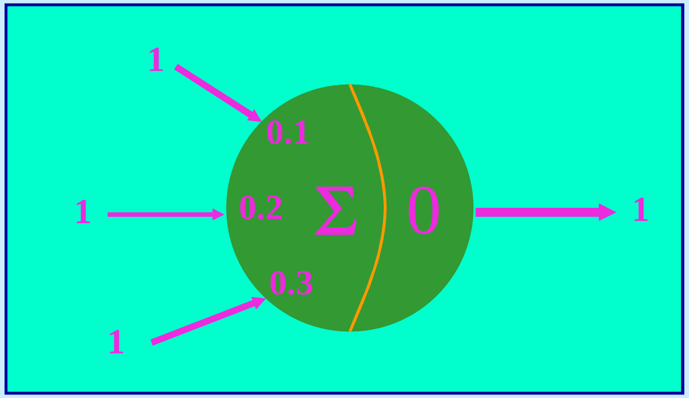


First Training Run

- Apply the input (1,1) with output 1

The net input is:

$$y = 0.1 + 0.2*1 + 0.3*1 = 0.6$$



The new weights are:

$$b = 0.1 + 0.1(1\text{-}0.6) = 0.14$$
$$w_1 = 0.2 + 0.1(1\text{-}0.6)1 = 0.24$$
$$w_2 = 0.3 + 0.1(1\text{-}0.6)1 = 0.34$$

The largest weight change is 0.04

Delta rule

Neuron input

$$y = b + \quad x_i w_i$$

$$b(new) = b(old) + \eta(d - y)$$
$$w_i(new) = w_i(old) + \eta(d - y)x_i$$

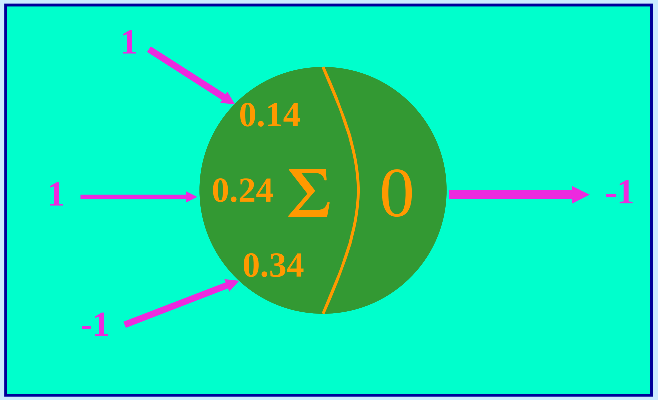
Second Training Run

- Apply the second training set (1 -1) with output -1

The net input is:

$$y = 0.14 + 0.24*1 + 0.34*(-1) = 0.04$$

The new weights are:



$$b = 0.14 + 0.1(-1-0.04) = 0.04$$
$$w_1 = 0.24 + 0.1(-1-0.04)1 = 0.14$$
$$w_2 = 0.34 + 0.1(-1-0.04)(-1) = 0.44$$

The largest weight change is 0.1

Neuron input

$$y = b + \quad x_i w_i$$

Delta rule

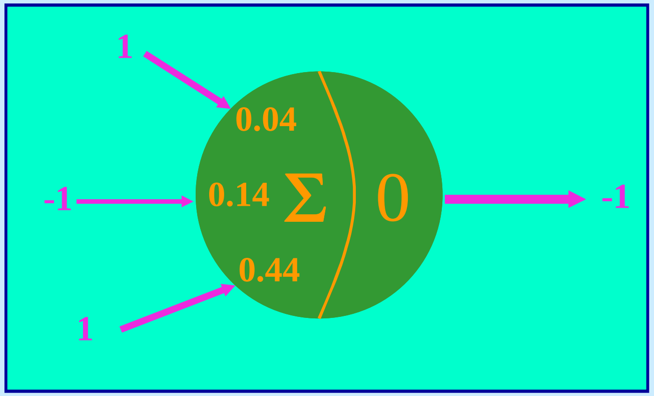
$$b(new) = b(old) + \eta(d - y)$$
$$w_i(new) = w_i(old) + \eta(d - y)x_i$$

Third Training Run

- Apply the third training set (-1 1) with output -1

The net input is:

$$y = 0.04 + 0.14*(-1) + 0.44*1 = 0.34$$



The new weights are:

$$b = 0.04 + 0.1(-1-0.34) = -0.09$$
$$w_1 = 0.14 + 0.1(1+0.34)1 = 0.27$$
$$w_2 = 0.44 + 0.1(-1-0.34)1 = 0.31$$

The largest weight change is 0.13

Neuron input

$$y = b + \quad x_i w_i$$

Delta rule

$$b(new) = b(old) + \eta(d - y)$$
$$w_i(new) = w_i(old) + \eta(d - y)x_i$$

Fourth Training Run

- Apply the fourth training set (-1 -1) with output -1

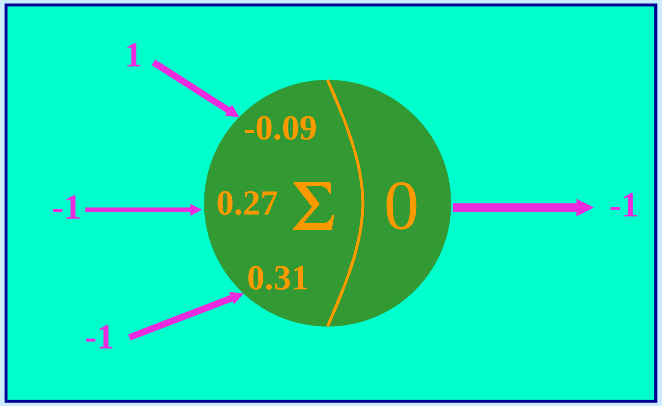
The net input is:

$$y = -0.09 - 0.27*1 - 0.31*1 = -0.67$$

The new weights are:

$$b = -0.09 + 0.1(-1-0.67) = -0.27$$
$$w_1 = 0.27 + 0.1(-1-0.67)(-1) = 0.43$$
$$w_2 = 0.31 + 0.1(-1-0.67)(-1) = 0.47$$

The largest weight change is 0.16



Neuron input

$$y = b + \quad x_i w_i$$

Delta rule

$$b(new) = b(old) + \eta(d - y)$$
$$w_i(new) = w_i(old) + \eta(d - y)x_i$$

Result

- Continue to cycle through the four training inputs until the largest change in the weights over a complete cycle is less than some small number (say 0.01)

- In this case, the solution becomes
  - $b = -0.5$
  - $w_1 = 0.5$
  - $w_2 = 0.5$

Gradient Descent

-converge to the minimum error point

-independently of Linearly/Nonlinearly separable problems

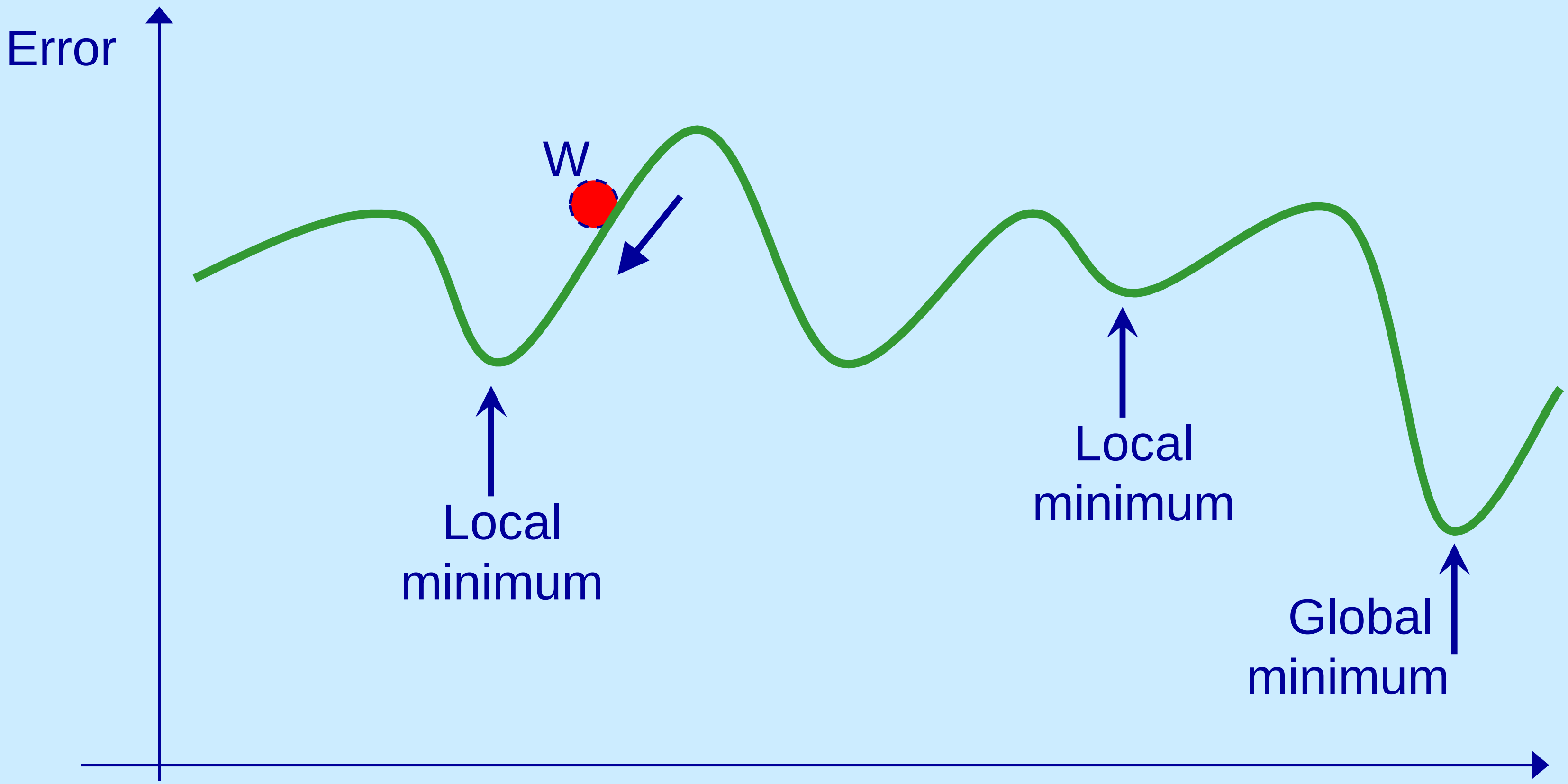
There can be problems with Gradient Descent

a) Convergence to a local minimum can be slow (e.g. 1000s of steps).

b) If there are many local minima on the error surface, then there is no guarantee that the global minimum is found.

Gradient Descent

- Problem of local minima



The Learning Rate, η

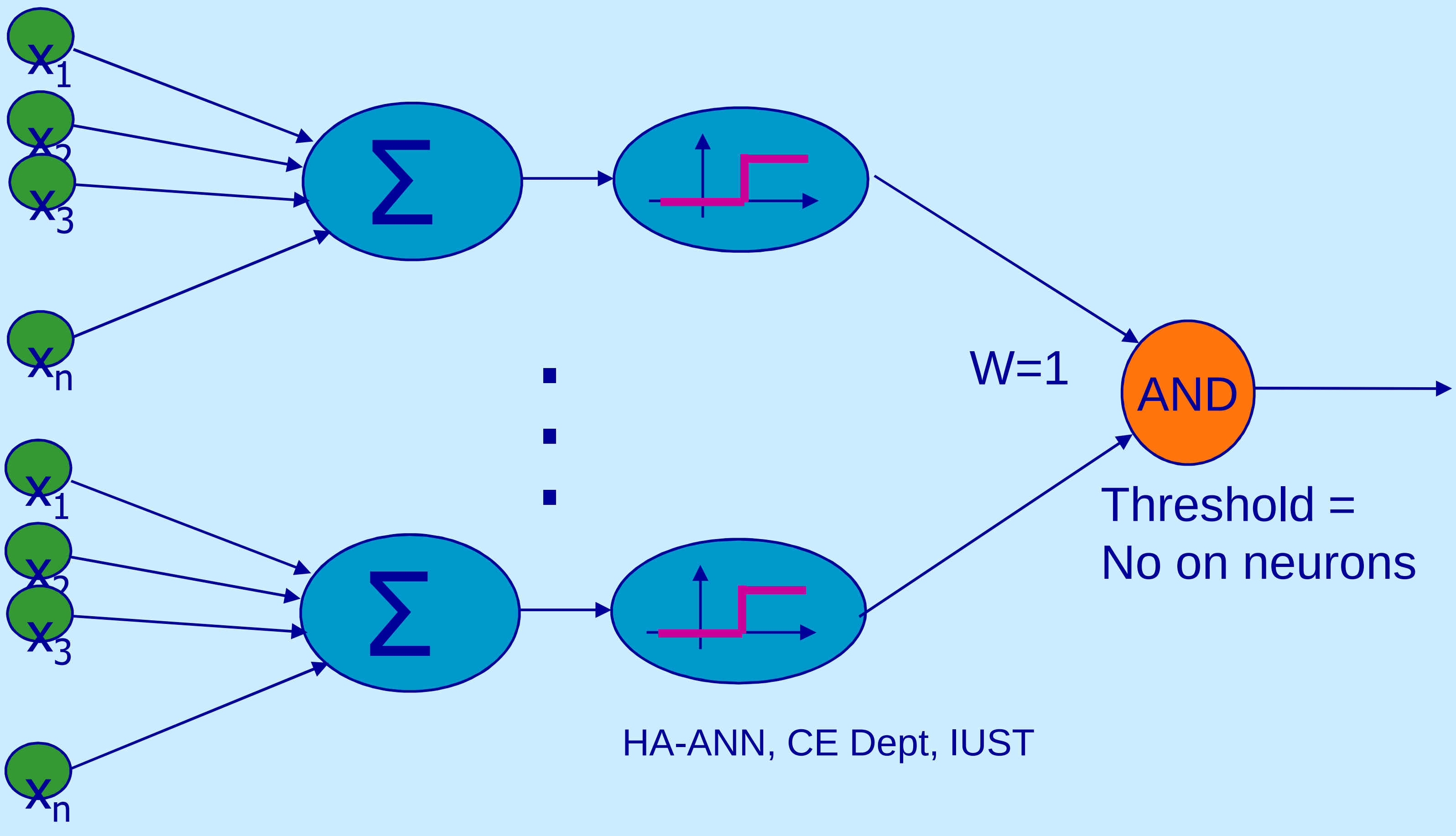
- The performance of an ADALINE neuron depends heavily on the choice of the learning rate
  - if it is too large the system will not converge
  - if it is too small the convergence will take to long

- Typically, η is selected by trial and error
  - typical range: $0.01 < \eta < 10.0$
  - often start at 0.1
  - sometimes it is suggested that:
    $0.1 < n^*\eta < 1.0$ (where n is the number of inputs)
  - Sometimes it is a fixed value, or a decreasing parameter:

$$\eta(t) = \frac{\eta_0}{1 + t/n}$$

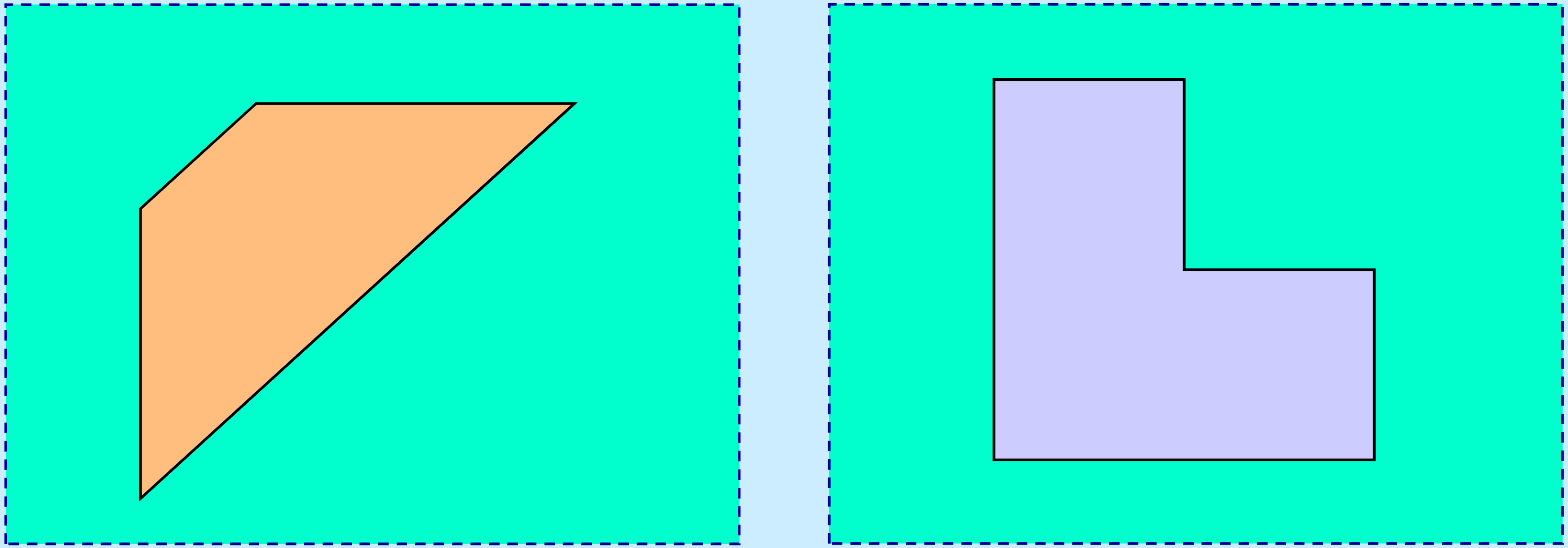
Madaline

Several Adaline in parallel give a Madaline



Madaline

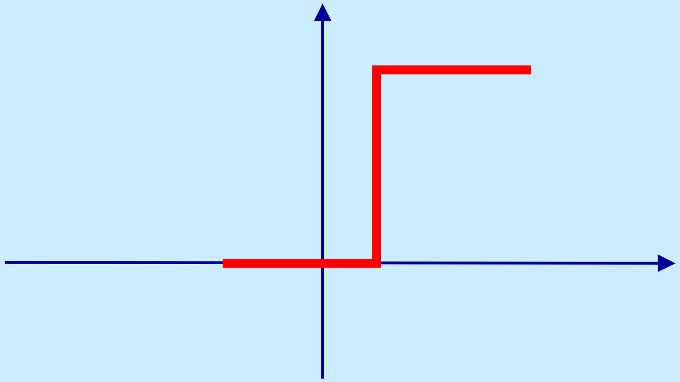
Separable regions:



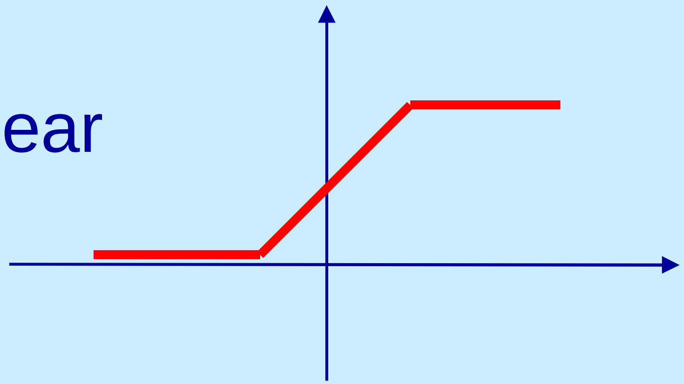
Other points

-Choice of activation function

-Step function (Hard limiter)



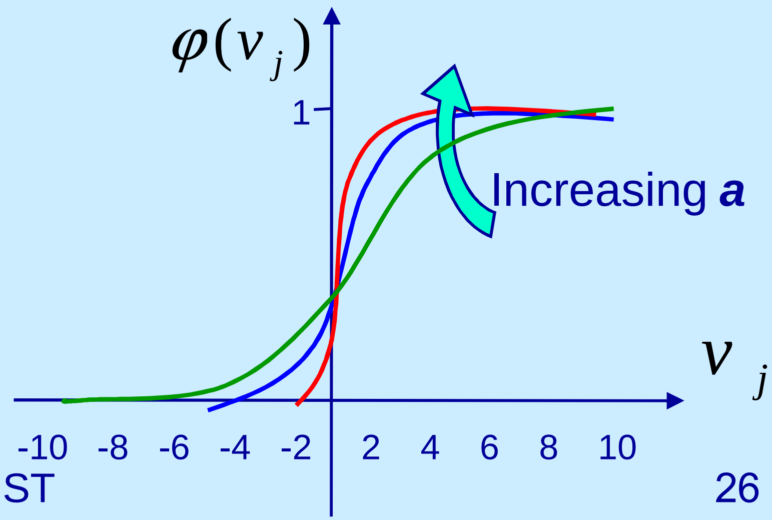
-Piecewise linear



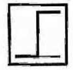
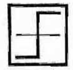
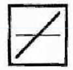
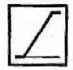
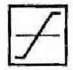
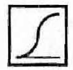
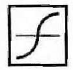
-Sigmoid (logistic)

$$\varphi(\mathrm{v}_{\mathrm{j}}) = \frac{1}{1+e^{-av_j}} \quad \text{with} \quad a > 0$$

$$\mathrm{v}_{\mathrm{j}} = \sum_i w_{\mathrm{ji}} y_{\mathrm{i}}$$



Other functions

Name	Input/Output Relation	Icon	MATLAB Function
Hard Limit	$a = 0 \quad n < 0$ $a = 1 \quad n \geq 0$		hardlim
Symmetrical Hard Limit	$a = -1 \quad n < 0$ $a = +1 \quad n \geq 0$		hardlims
Linear	$a = n$		purelin
Saturating Linear	$a = 0 \quad n < 0$ $a = n \quad 0 \leq n \leq 1$ $a = 1 \quad n > 1$		satlin
Symmetric Saturating Linear	$a = -1 \quad n < -1$ $a = n \quad -1 \leq n \leq 1$ $a = 1 \quad n > 1$		satlins
Log-Sigmoid	$a = \frac{1}{1 + e^{-n}}$		logsig
Hyperbolic Tangent Sigmoid	$a = \frac{e^{n} - e^{-n}}{e^{n} + e^{-n}}$		tansig
Positive Linear	$a = 0 \quad n < 0$ $a = n \quad 0 \leq n$		poslin
Competitive	$a = 1$ neuron with max n $a = 0$ all other neurons	C	compet