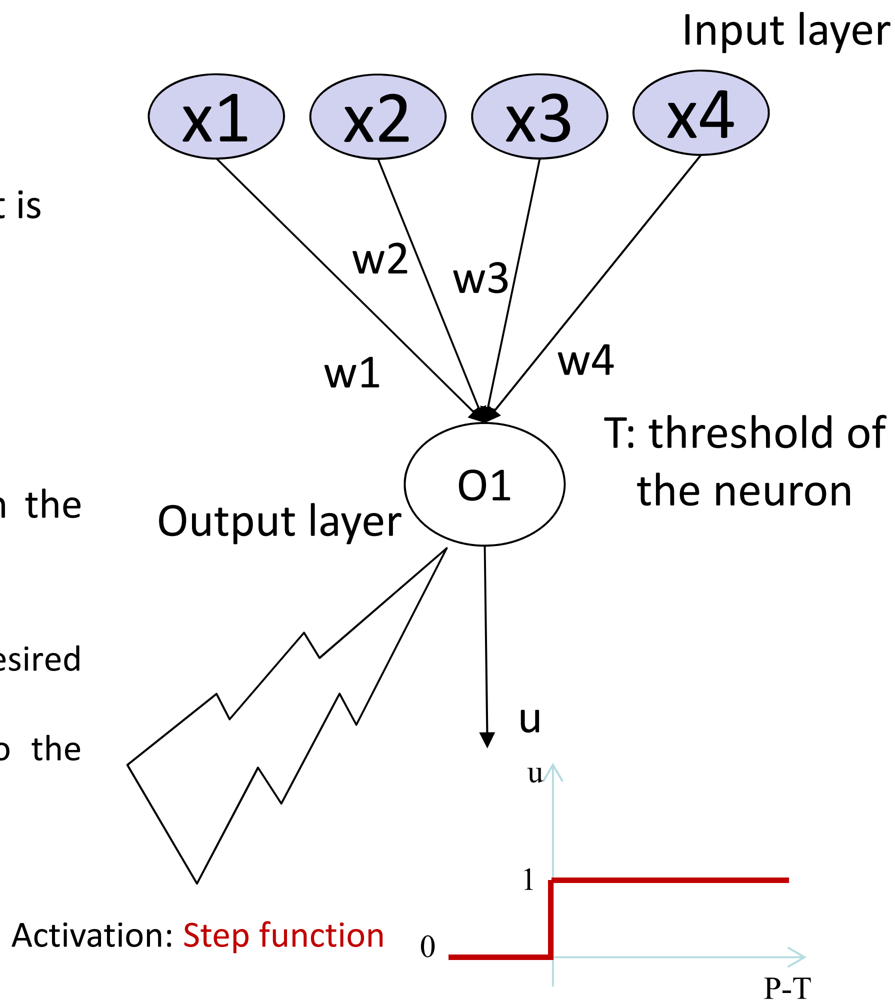
Neuroengineering (I)
2. Perceptron and learning

- **Scuola di Ingegneria Industriale e dell'Informazione** – Politecnico di Milano
- Prof. Pietro Cerveri

Basic network: perceptron

- 2-layer neural network
- Input layer (N>=1 signals) – Output layer (M>=1 signals)
- The set [x1 x2 x3 ,…] of binary values in the input is called the input pattern
  - Biological correspondence: stimulus
- It can be used to:
  - classify categories of patterns

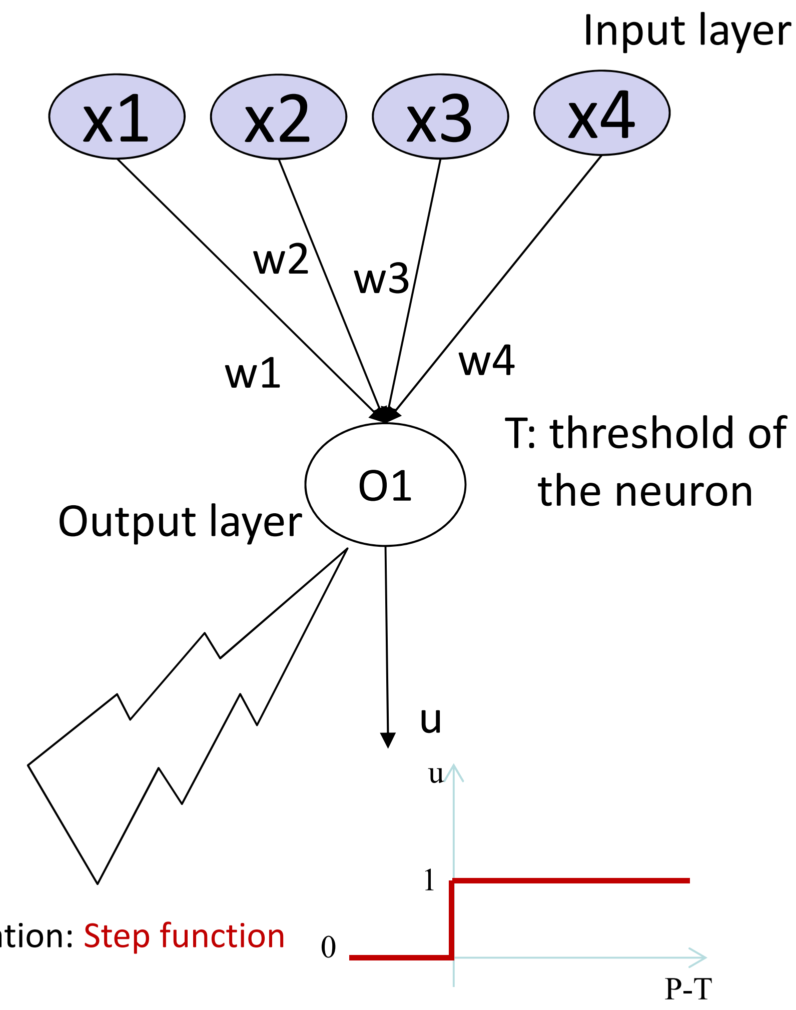
- The simplest Perceptron has one single unit in the output layer
- With binary activation (step) function
  - 1 if the stimulus in input belongs to the desired category
  - 0 if the stimulus in input does not belong to the desired category
- Decision model



Basic network: perceptron

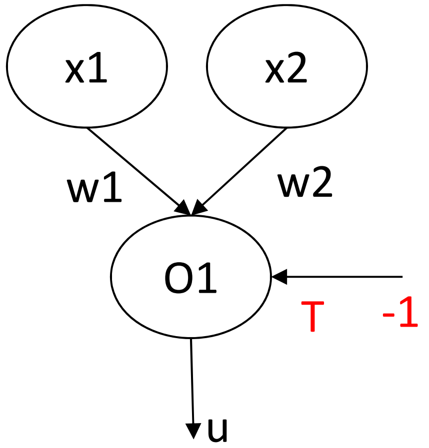
- 2-layer neural network
- Input layer (N>=1 signals) – Output layer (M>=1 signals)
- The set [x1 x2 x3 ,…] of binary values in the input is called the input pattern
  - Biological correspondence: stimulus
- It can be used to:
  - classify categories of patterns

- The simplest Perceptron has one single unit in the output layer
- With binary activation (step) function
  - 1 if the stimulus in input belongs to the desired category
  - 0 if the stimulus in input does not belong to the desired category
- Decision model



Activation: Step function

Action potential and neural threshold



Notice that the threshold can be considered as an additional weight of the neuron with a virtual constant input equal to -1

$$P = \sum_{j=1}^{2} w_j x_j - T = w_1 x_1 + w_2 x_2 + T\left(-1\right)$$

$$P = \sum_{j=1}^{3} w_j x_j = w_1 x_1 + w_2 x_2 + w_3 x_3 \quad with \quad w_3 = \mathrm{T} \; and \; x_3 = -1$$

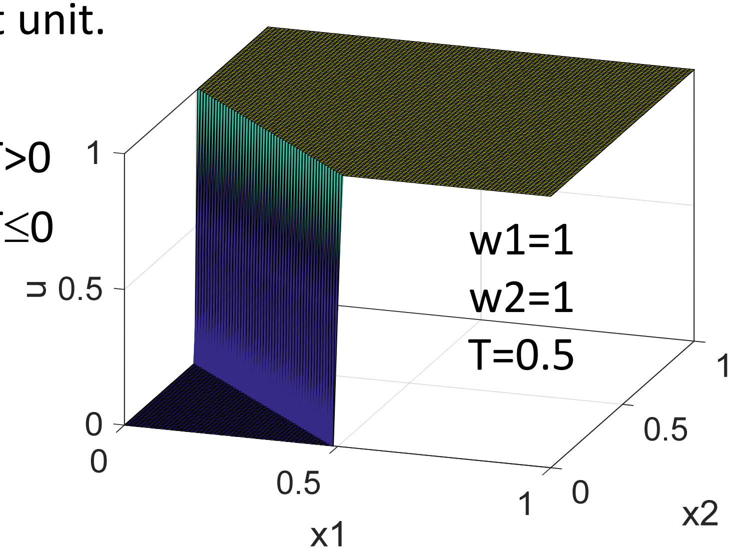
From now on we call *P-T* action potential and we name it with *P*

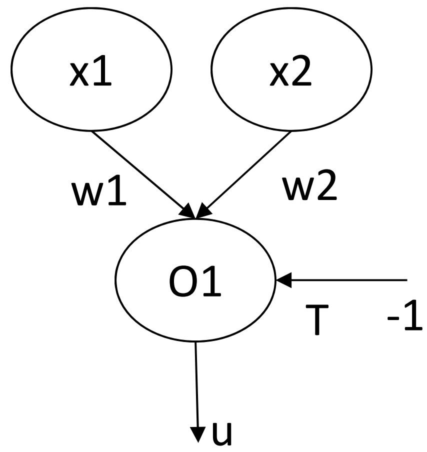
Perceptron with step function in action

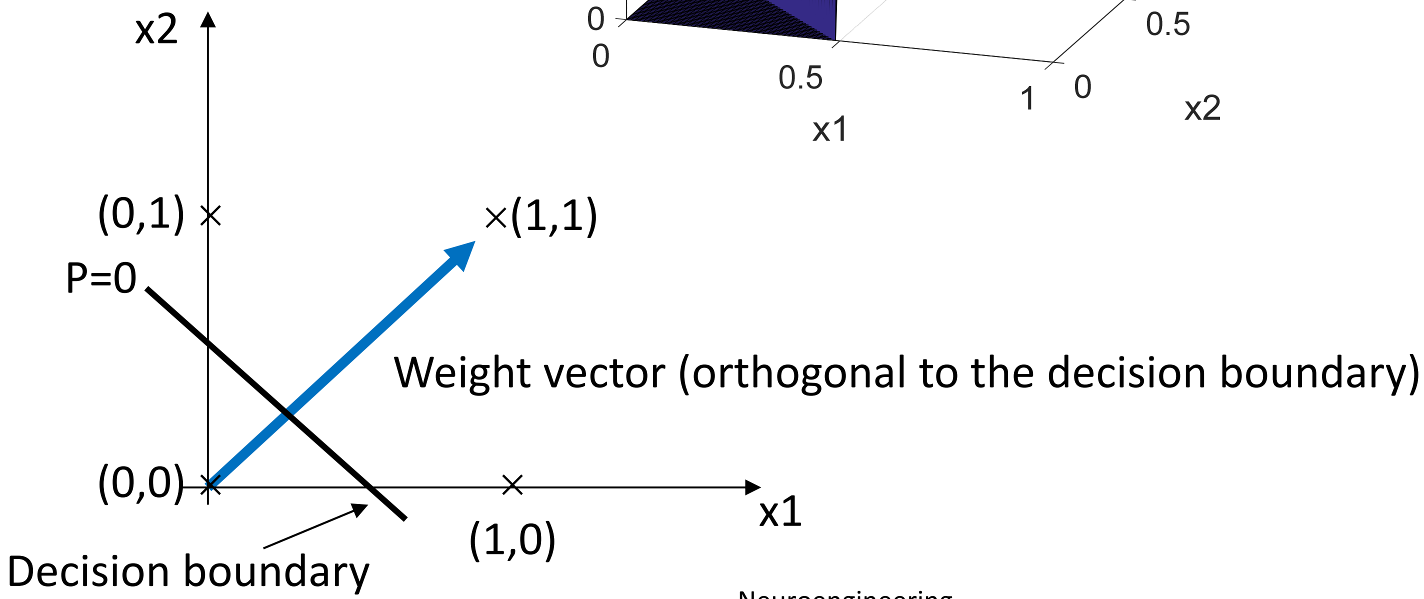
Let assume the perceptron be composed by 2 input units and 1 output unit.

u=1 if w1*x1+w2*x2 – T>0

u=0 if w1*x1+w2*x2 – T≤0





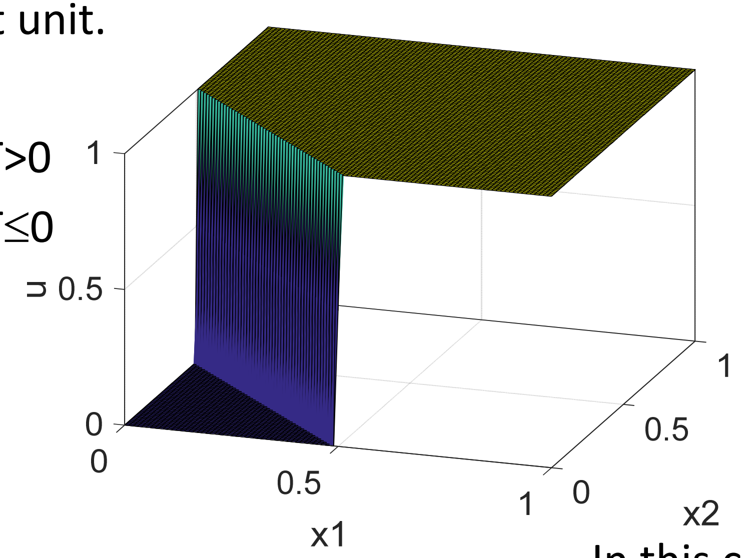


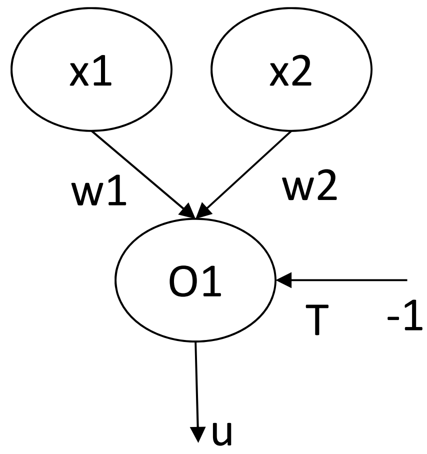
Perceptron with step function in action

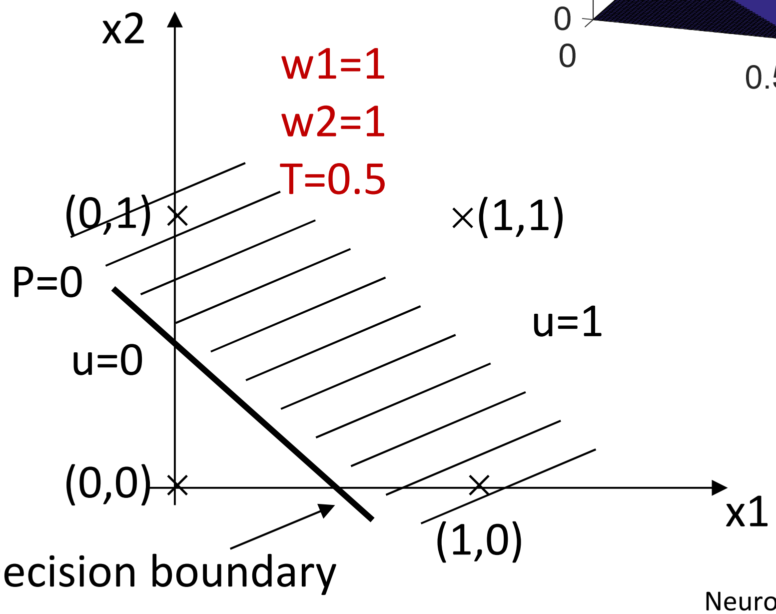
Let assume the perceptron be composed by 2 input units and 1 output unit.

u=1 if w1*x1+w2*x2 – T>0

u=0 if w1*x1+w2*x2 – T≤0







In this case the perceptron is classifying (0,1) , (1,1) and (1,0) as positive whereas the input pattern (0,0) will be classified as negative

OR port

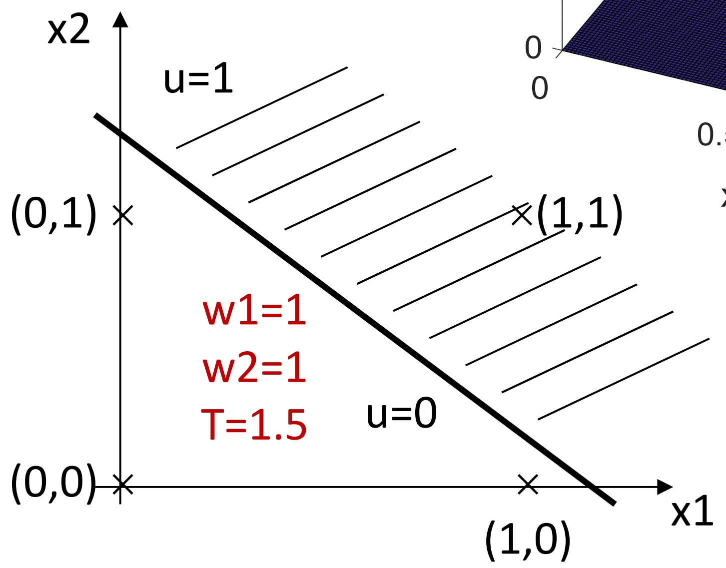
If T=1.5 then it is an AND port

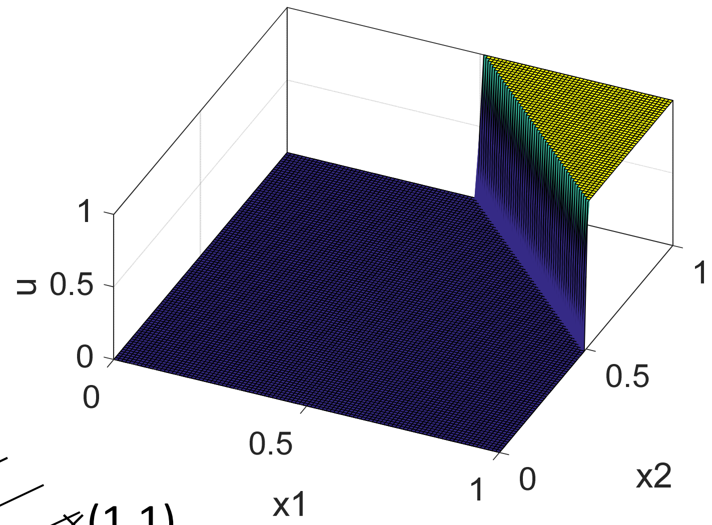
Perceptron with step function in action

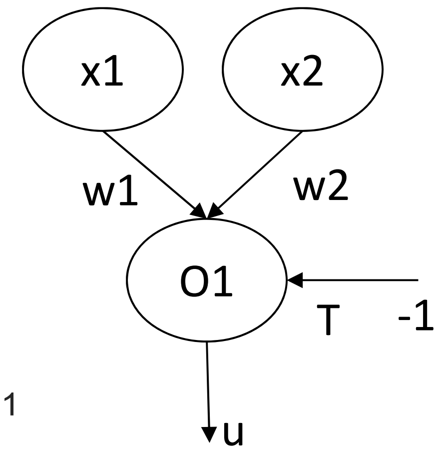
Let assume the perceptron be composed by 2 input units and 1 output unit.

$u=1$ if $1*x1+1*x2 -1.5 > 0$

$u=0$ if $1*x1+1*x2 -1.5 \leq 0$



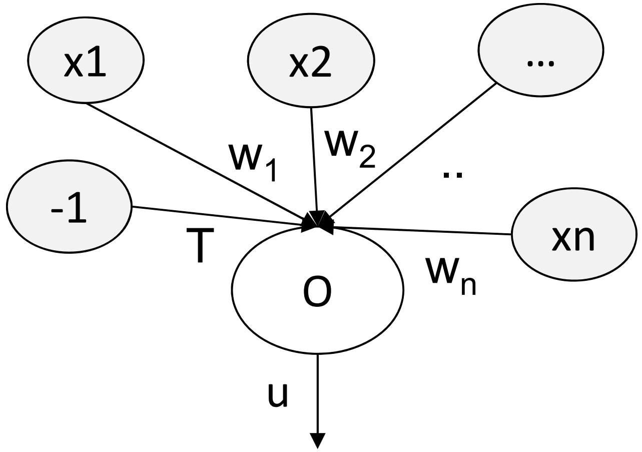




AND port

Training the perceptron

- ANN topology is given
  - Structure
  - Activation function (step)
- Training process: estimation of the neural weights
- **Supervision**: expected outputs are known for each input pattern
- Training dataset
  - Set of input examples (training patterns)
  - Set of corresponding reference output patterns

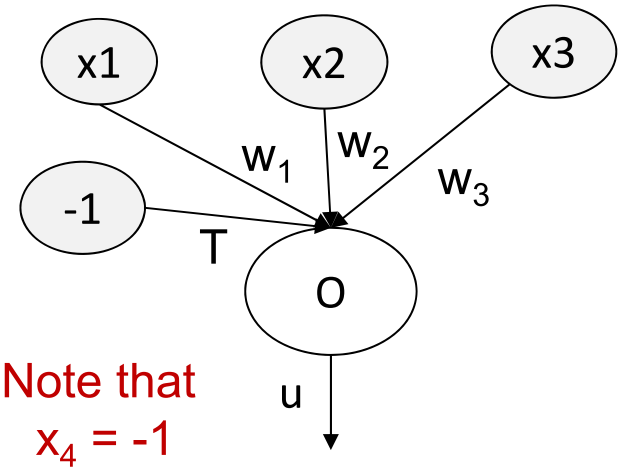


$$\mathbf{w} = \left[w_1 \, w_2 \; ..... \; w_n \, T \right]$$

is the set of weights to be estimated

Training dataset

Network		Pattern
output	$u^{(1)}$	1: $(x^{(1)}_1, x^{(1)}_2, x^{(1)}_3, x^{(1)}_4)$
reference	$t^{(1)}$	
output	$u^{(2)}$	2: $(x^{(2)}_1, x^{(2)}_2, x^{(2)}_3, x^{(2)}_4)$
reference	$t^{(2)}$	
		…
output	$u^{(k)}$	k: $(x^{(k)}_1, x^{(k)}_2, x^{(k)}_3, x^{(k)}_4)$
reference	$t^{(k)}$	
		…
output	$u^{(R)}$	R: $(x^{(R)}_1, x^{(R)}_2, x^{(R)}_3, x^{(R)}_4)$
reference	$t^{(R)}$	



Note that
$x_4 = -1$
$w_4 = T$

- R is the training dataset size (number of patterns)
- Computation of u requires weight initialization

Principle for supervised learning: exploiting the error signal $e = (t-u)$ wrt a specified expected output (reference value) t

Perceptron learning rule
Rosenblatt (1962)

- Activation: step function
- *R* training patterns
- Incremental procedure: $\mathbf{w}_{start}$ (initial weight vector)
  - the weight vector is iteratively updated using online strategy
  - each pattern *k* in the training set contributes to the weight increment vector $\Delta\mathbf{w}$ by means of the error signal e = (t-u)
  - one iteration of the iterative procedure requires the evaluation of all R patterns

$$\mathbf{w}^{(new)} = \mathbf{w}^{(old)} + \Delta\mathbf{w} \quad \text{with} \quad \Delta\mathbf{w} = \eta\left(t^{(k)} - u^{(k)}\right)\mathbf{x}^{(k)}$$

$\eta \leq 1$ is a positive scalar called learning rate

Update principle e = (t-u)

$$\boldsymbol{w}^{(new)} = \boldsymbol{w}^{(old)} + \eta \boldsymbol{e}^{(k)} \boldsymbol{x}^{(k)}$$

If $\boldsymbol{e}^{(k)}$ = +1 then $\mathbf{w}^{(new)} = \mathbf{w}^{(old)} + \eta \mathbf{x}^{(k)}$

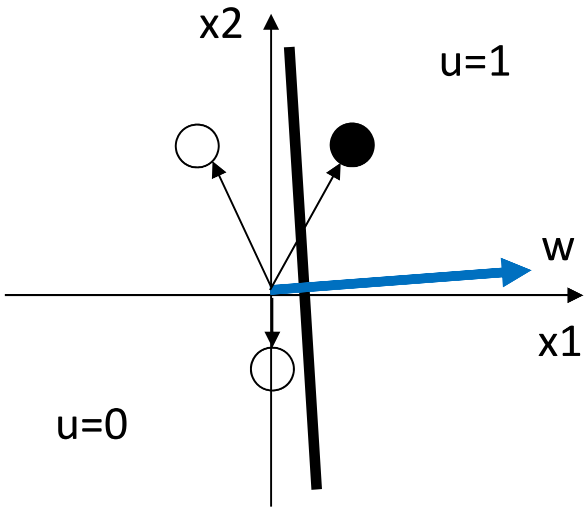
If $\boldsymbol{e}^{(k)}$ = -1 then $\mathbf{w}^{(new)} = \mathbf{w}^{(old)} - \eta \mathbf{x}^{(k)}$

If $\boldsymbol{e}^{(k)}$ = 0 then $\mathbf{w}^{(new)} = \mathbf{w}^{(old)}$

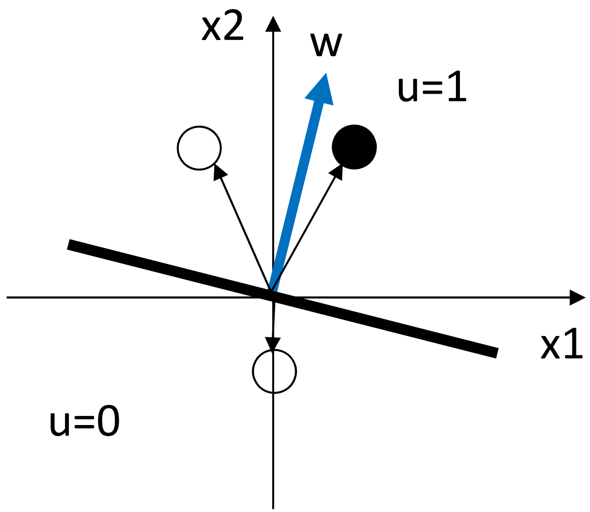
The perceptron learning rule correct the weight vector if and only if a misclassification occurs

Geometric validation

CLASSIFICATION OK

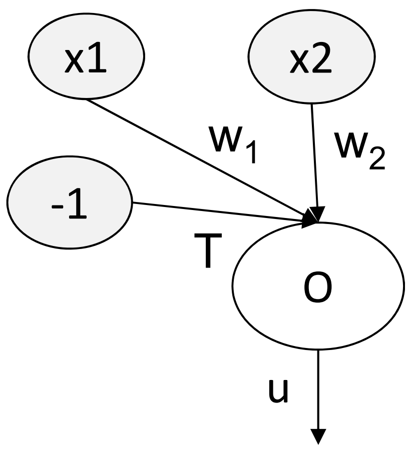


CLASSIFICATION FAILURE



Perceptron rule can be thought of as a way to orient the decision boundary in such a way that the scalar product of weight vector with all the patterns into the first class is positive whereas it is negative with all the patterns into the second class.

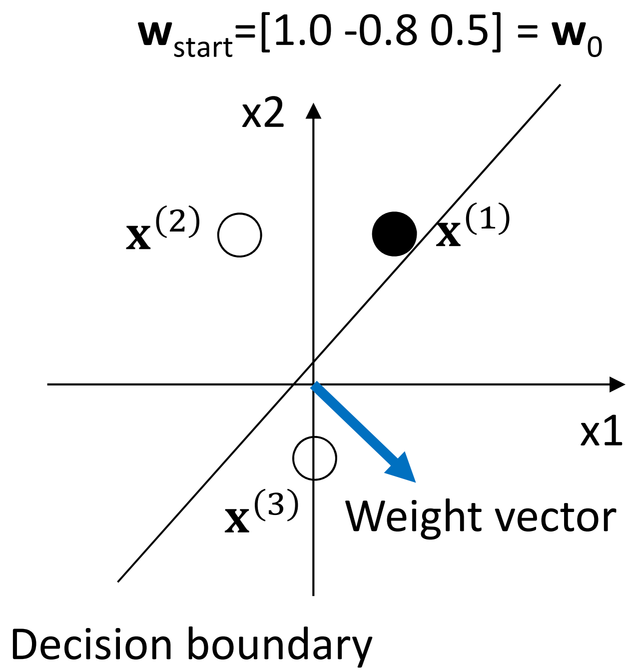
Perceptron learning rule: example



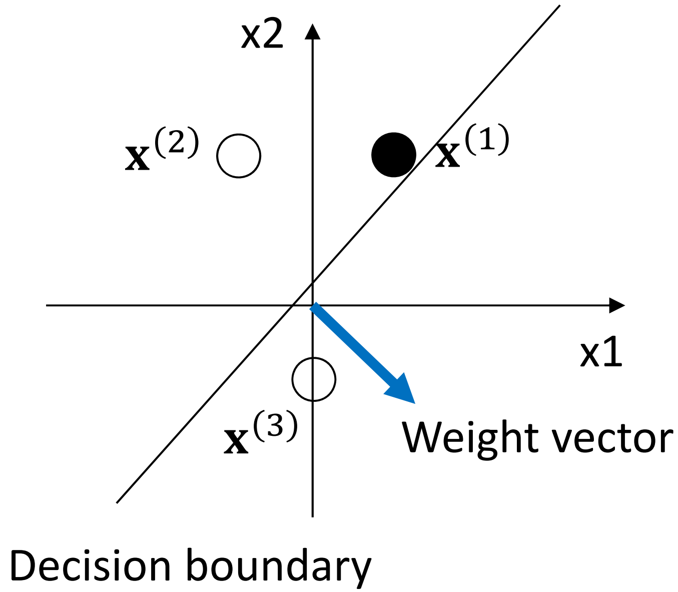
$$\left\{ \mathbf{x}^{(1)} = \begin{bmatrix} 1 \\ 2 \\ -1 \end{bmatrix}, t_1 = 1 \right\}$$

$$\left\{ \mathbf{x}^{(2)} = \begin{bmatrix} -1 \\ 2 \\ -1 \end{bmatrix}, t_2 = 0 \right\}$$

$$\left\{ \mathbf{x}^{(3)} = \begin{bmatrix} 0 \\ -1 \\ -1 \end{bmatrix}, t_3 = 0 \right\}$$



$\mathbf{w}_0$=[1.0 -0.8 0.5]



Step1

$\mathbf{x}^{(1)}$=[1, 2,-1] ; t_1=1

a. Compute u_1

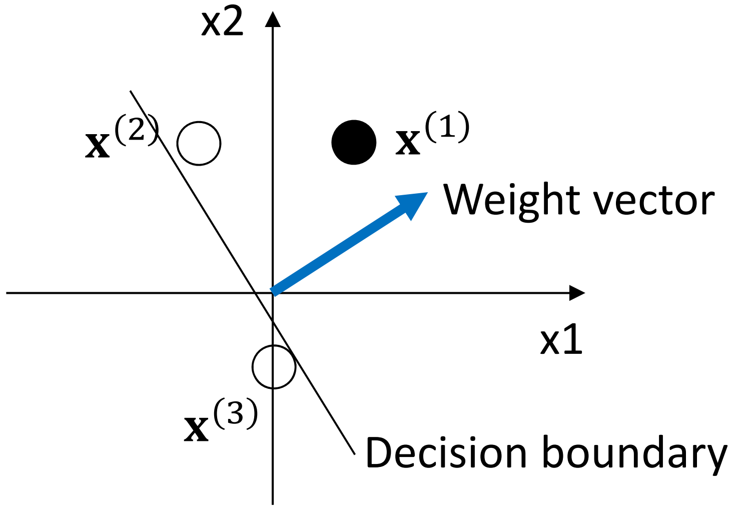
u_1 = step($\mathbf{w}_0$* $\mathbf{x}^{(1)}$)=H(-1.1) = 0

$e_1 = t_1 - u_1 = 1 - 0 = 1$

b. Compute $\mathbf{w}_1$

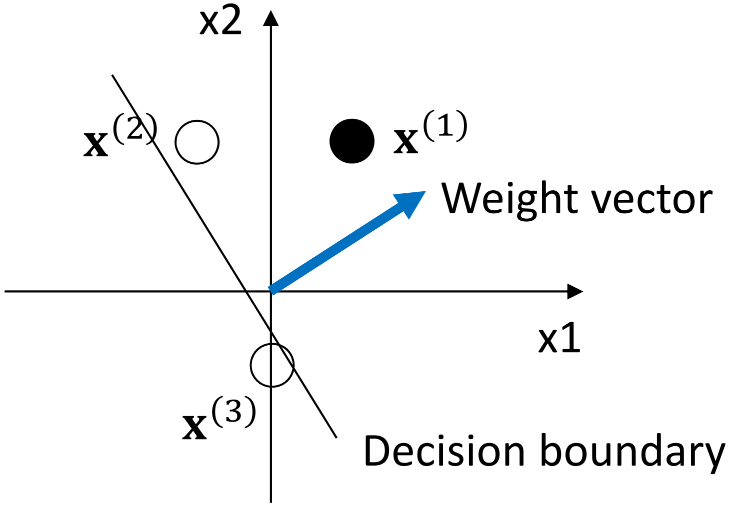
$\mathbf{w}_1 = \mathbf{w}_0 + e_1 * \mathbf{x}^{(1)}$

= [2.0 1.2 -0.5]



Let us assume η=1

$\mathbf{w}_1$=[2.0 1.2 -0.5]



Step2

$\mathbf{x}^{(2)}$=[-1, 2,-1] ; t_2=0

a. Compute u_2

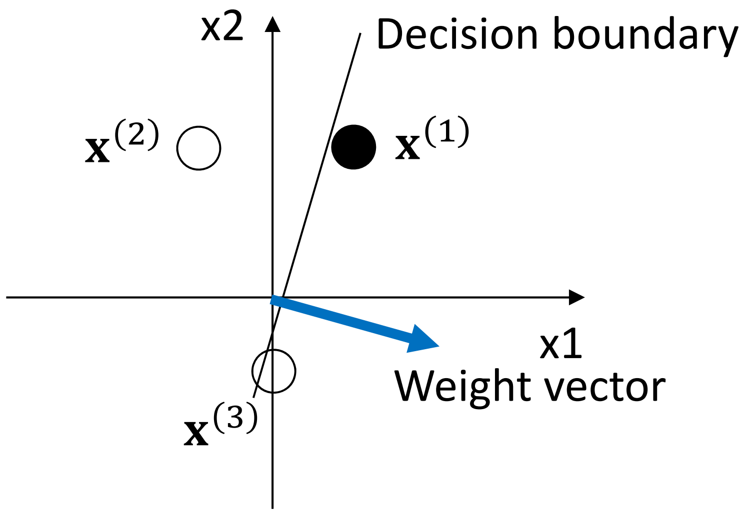
u_2 = step($\mathbf{w}_1$* $\mathbf{x}^{(2)}$)=H(0.9)=1

$e_2 = t_2 - u_2 = 0 - 1 = -1$

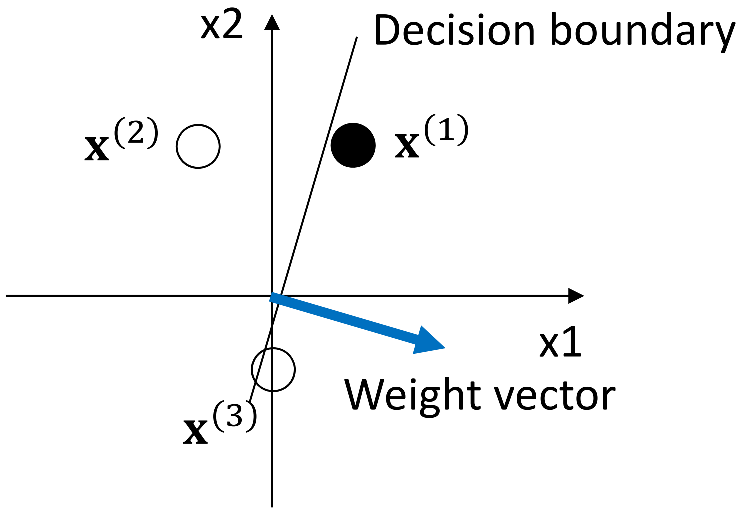
b. Compute $\mathbf{w}_2$

$\mathbf{w}_2 = \mathbf{w}_1 + e_2 * \mathbf{x}^{(2)}$

= [3.0 -0.8 0.5]



$\mathbf{w}_2$=[3.0 -0.8 0.5]



Step3

$\mathbf{x}^{(3)}$=[0, -1,-1] ; t_3=0

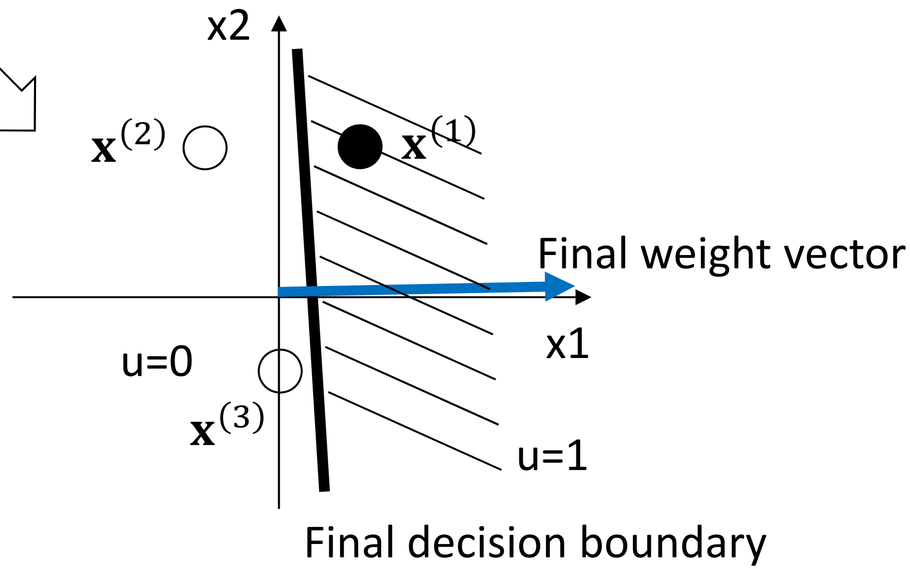
a. Compute u_3

u_3 = step($\mathbf{w}_2$* $\mathbf{x}^{(3)}$)=H(0.3)=1
$e_3 = t_3 - u_3 = 1 - 0 = 1$

b. Compute $\mathbf{w}_1$

$\mathbf{w}_3 = \mathbf{w}_2 + e_3 * \mathbf{x}^{(3)}$

= [3.0 0.2 1.5]



Learning ok

$\mathbf{w}_3$= [3.0 0.2 1.5]

$$\left\{ \mathbf{x}^{(1)} = \begin{bmatrix} 1 \\ 2 \\ -1 \end{bmatrix}, t_1 = 1 \right\}$$

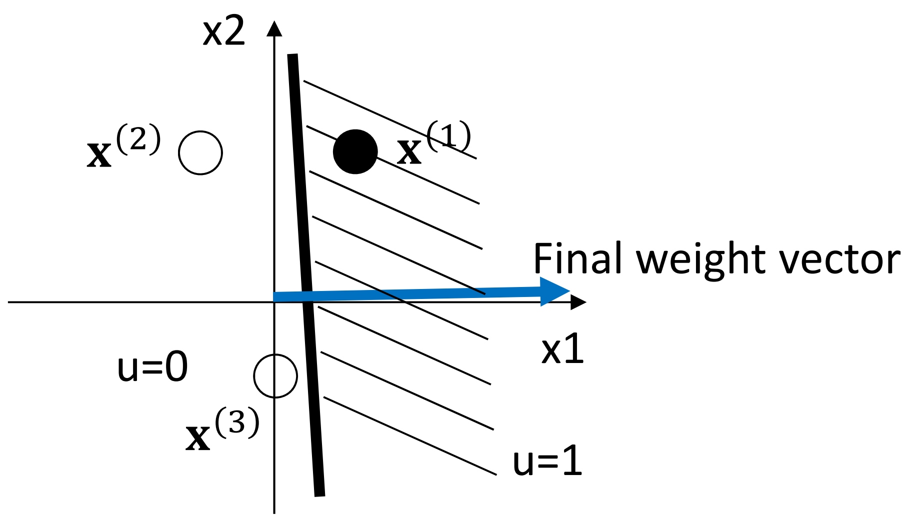
$$\left\{ \mathbf{x}^{(2)} = \begin{bmatrix} -1 \\ 2 \\ -1 \end{bmatrix}, t_2 = 0 \right\}$$

$$\left\{ \mathbf{x}^{(3)} = \begin{bmatrix} 0 \\ -1 \\ -1 \end{bmatrix}, t_3 = 0 \right\}$$

u_1 = step($\mathbf{w}_3$ * $\mathbf{x}^{(1)}$)=H(1.9) = 1

u_2 = step($\mathbf{w}_3$ * $\mathbf{x}^{(2)}$)=H(-4.9) = 0

u_3 = step($\mathbf{w}_3$ * $\mathbf{x}^{(3)}$)=H(-1.7) = 0



Final decision boundary

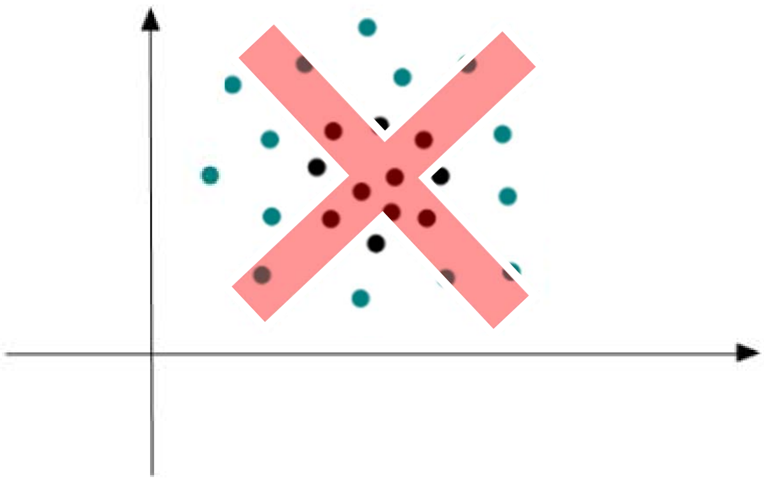
Perceptron learning rule

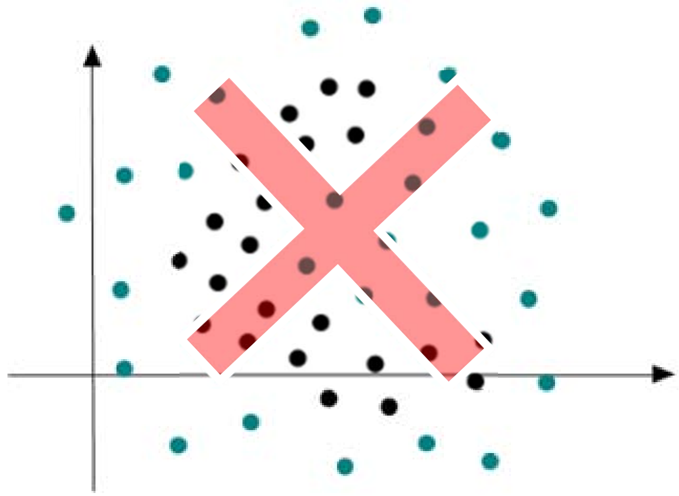
The decision boundary is always orthogonal to the weight vector.

The perceptron learning rule is guaranteed to converge to a solution in a finite number of steps, as long as a solution exists.

but …

linearly separable problem only

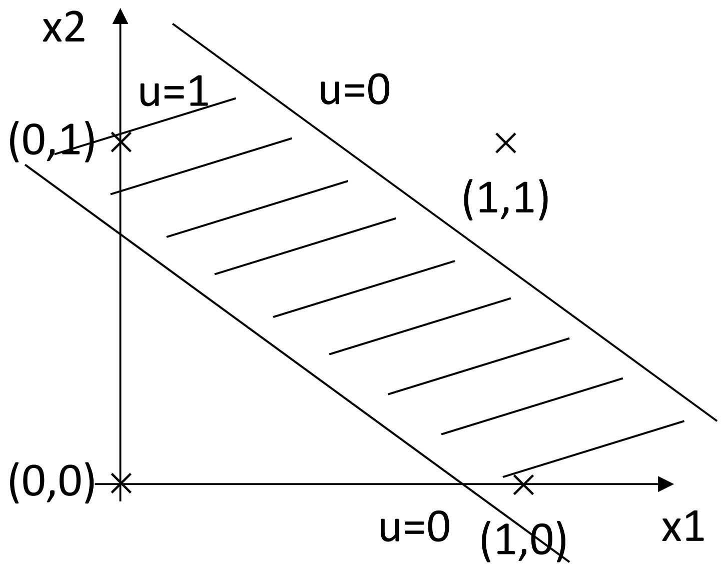




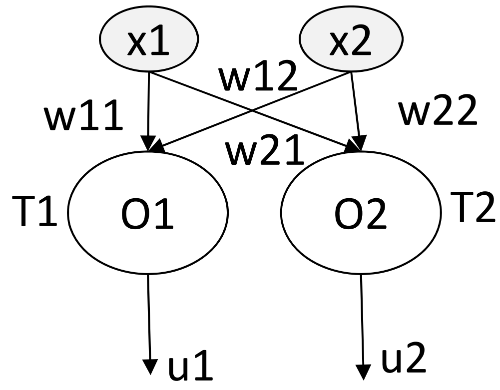
Perceptron as a linear binary classifier

- Perceptron can only solve problems that are linearly separable
  - In N-dimensional space it must exist a hyper-plane w1x1+w2x2+w3x3+…=0 that separates completely the patterns in between
- The perceptron with one single neuron cannot solve the XOR classification

  - What if we assume a perceptron with 2 output units?

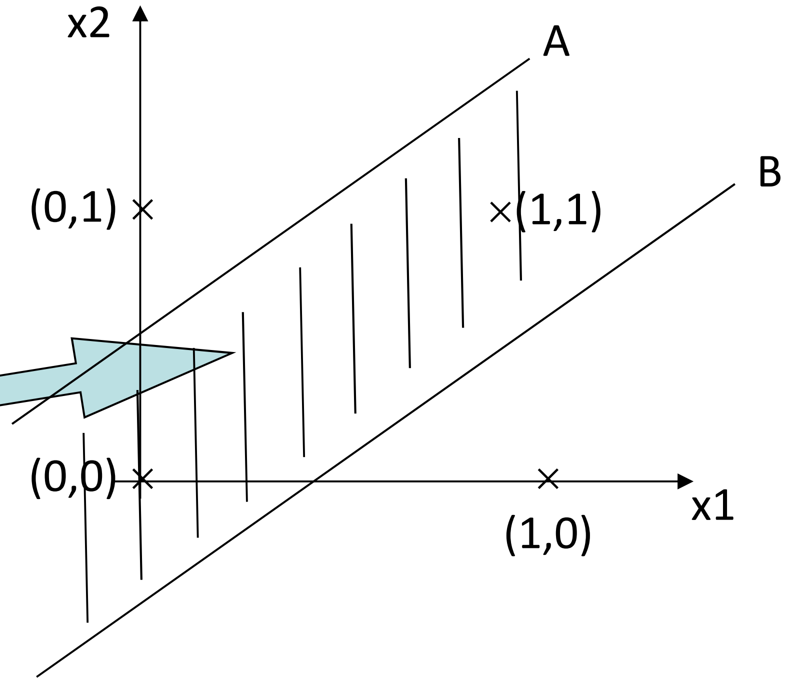


Multiple output perceptron



- With this architecture the network allows up to 4 categories to be classified
- (No, No) (Yes, Yes) (Yes, No) (No, Yes)

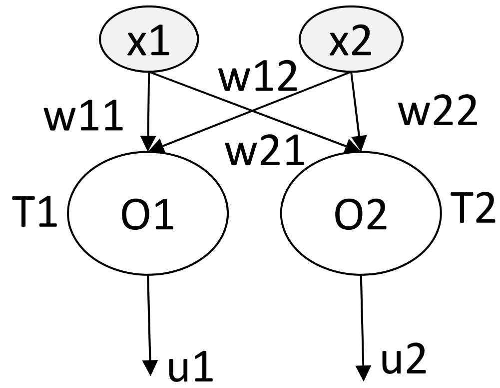
w11=1, w12=-1, T1=-0.5
w21=-1, w22=1, T2=-0.5
A: x1-x2+0.5=0
B: -x1+x2+0.5=0



x1=1 x2=1-> u1=1 u2=1
x1=0 x2=0-> u1=1 u2=1

x1=1 x2=0-> u1=1 u2=0
x1=0 x2=1-> u1=0 u2=1

Multiple output perceptron

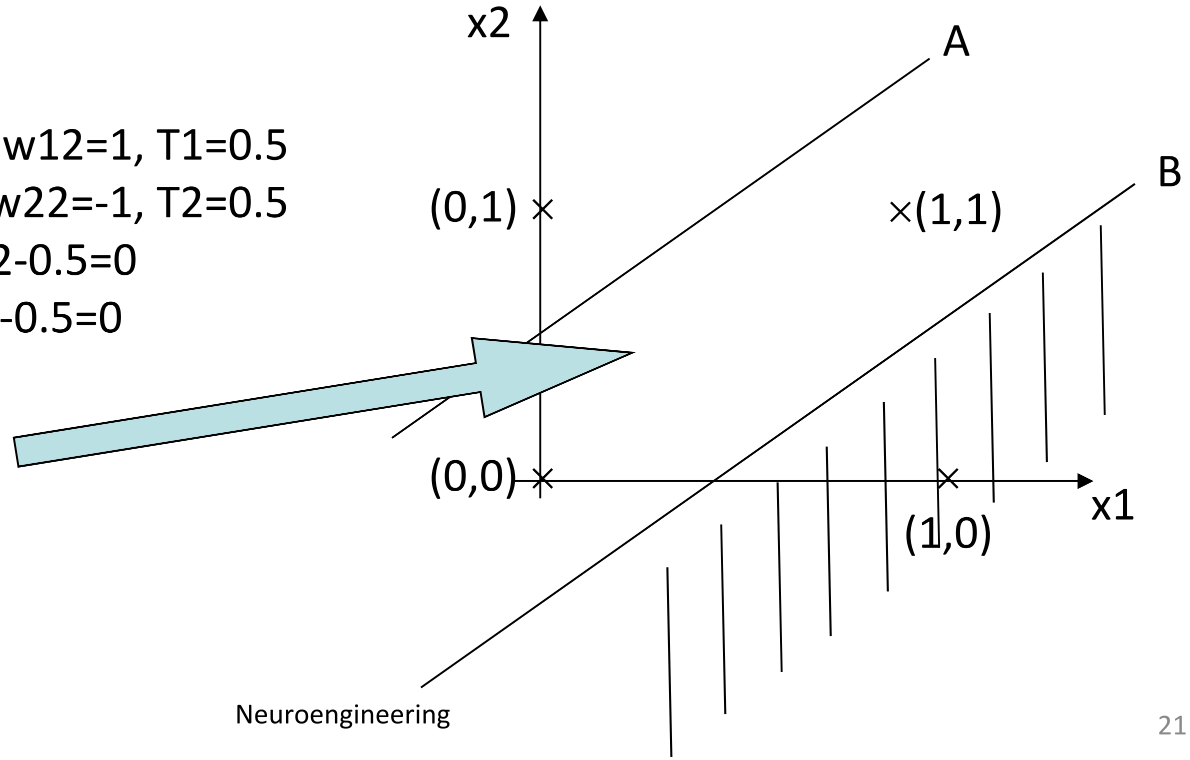


- With this architecture the network allows up to 4 categories to be classified
- (No, No) (Yes, Yes) (Yes, No) (No, Yes)

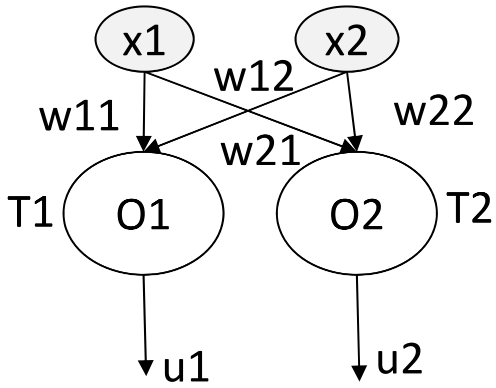
w11=-1, w12=1, T1=0.5
w21=1, w22=-1, T2=0.5
A: -x1+x2-0.5=0
B: x1-x2-0.5=0

x1=1 x2=1-> u1=0 u2=0
x1=0 x2=0-> u1=0 u2=0

x1=1 x2=0-> u1=0 u2=1
x1=0 x2=1-> u1=1 u2=0

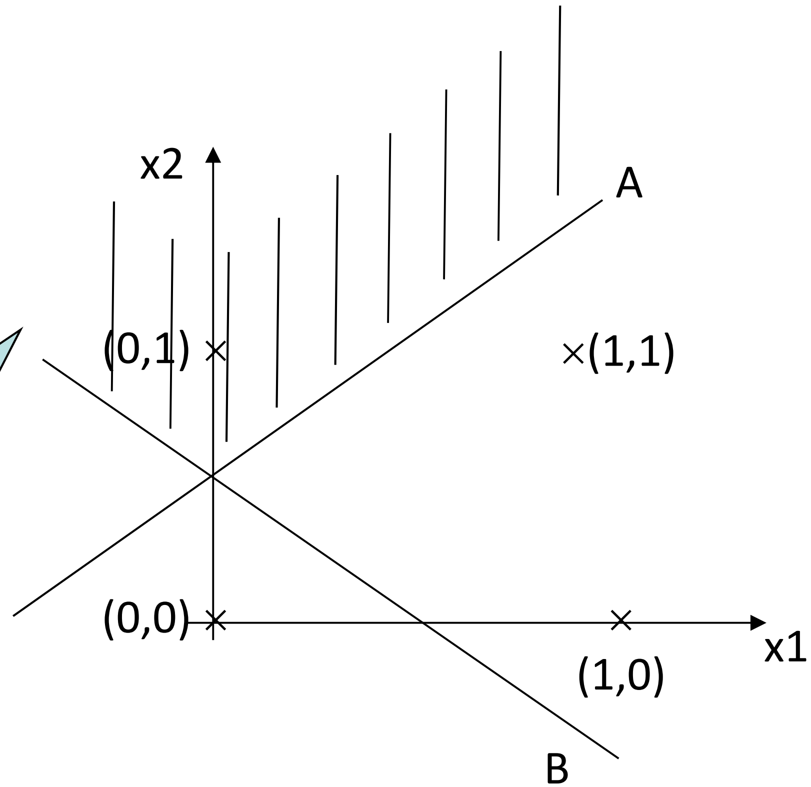


Multiple output perceptron



- With this architecture the network allows up to 4 categories to be classified
- (No, No,) (Yes, Yes) (Yes, No) (No, Yes)

w11=-1, w12=1, T1=0.5
w21=1, w22=1, T2=0.5
A: -x1+x2-0.5=0
B: x1+x2-0.5=0



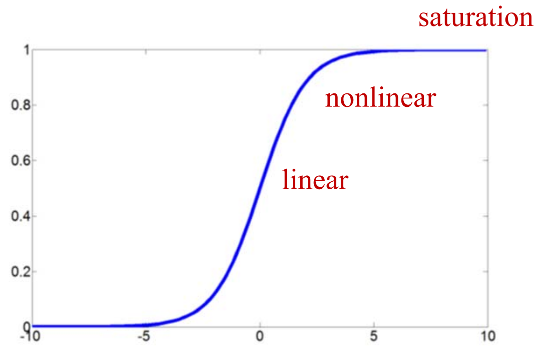
x1=1 x2=1-> u1=0 u2=1
x1=0 x2=0-> u1=0 u2=0

x1=1 x2=0-> u1=0 u2=1
x1=0 x2=1-> u1=1 u2=1

Extending the perceptron

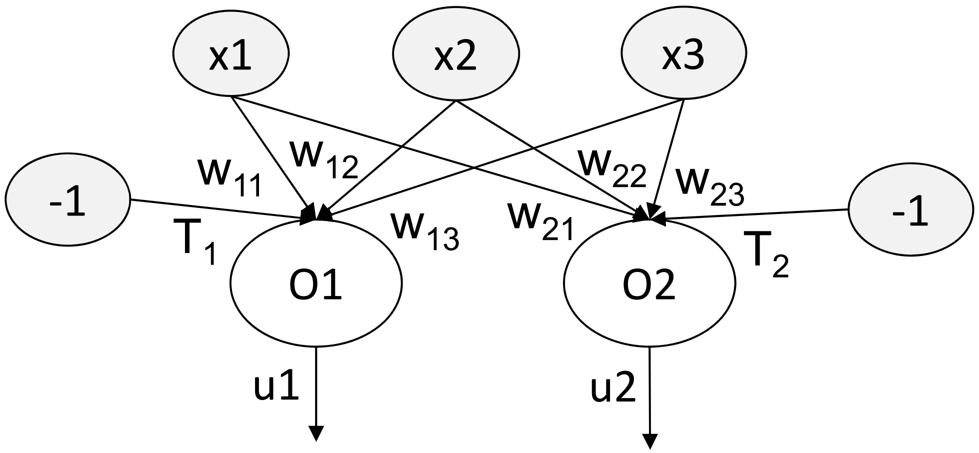
$$P_i = \sum_{j=1}^{M} w_{ij} x_j(t) = w_{i1}x_1 + w_{i2}x_2 + w_{i3}x_3 + .. + w_{iM}x_M + T_i(-1)$$

- Continuous input/output
- **Non-linear continuous activation function**
- Smooth transition near to 0
- x_s: signals of the input (1:M)
- u_s : signal of the output neurons (1:N)
- w_{ij} connection weight between the j^{th} input and i^{th} output neuron



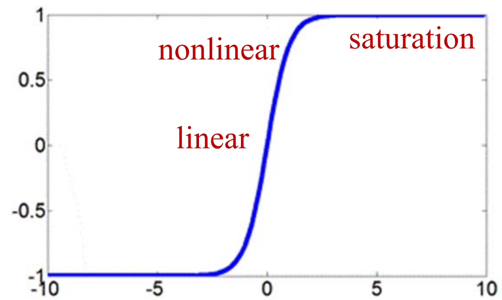
Sigmoidal

$$u = \mathrm{logsig}(P_i) = \frac{1}{1+e^{-P_i}}$$

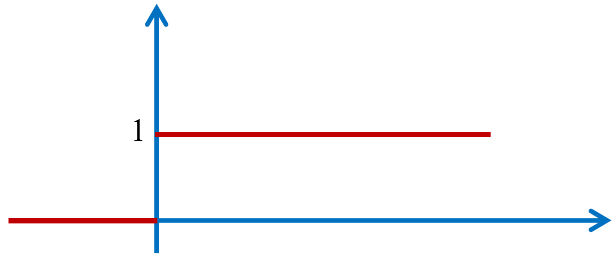


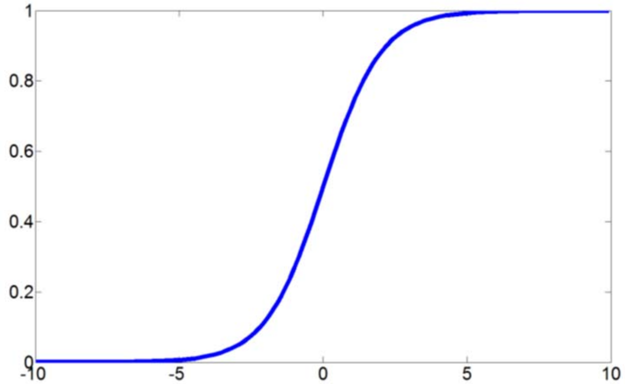
Hyperbolic

$$u = f(P_i) = \tanh(P_i) = \frac{e^{P_i} - e^{-P_i}}{e^{P_i} + e^{-P_i}}$$

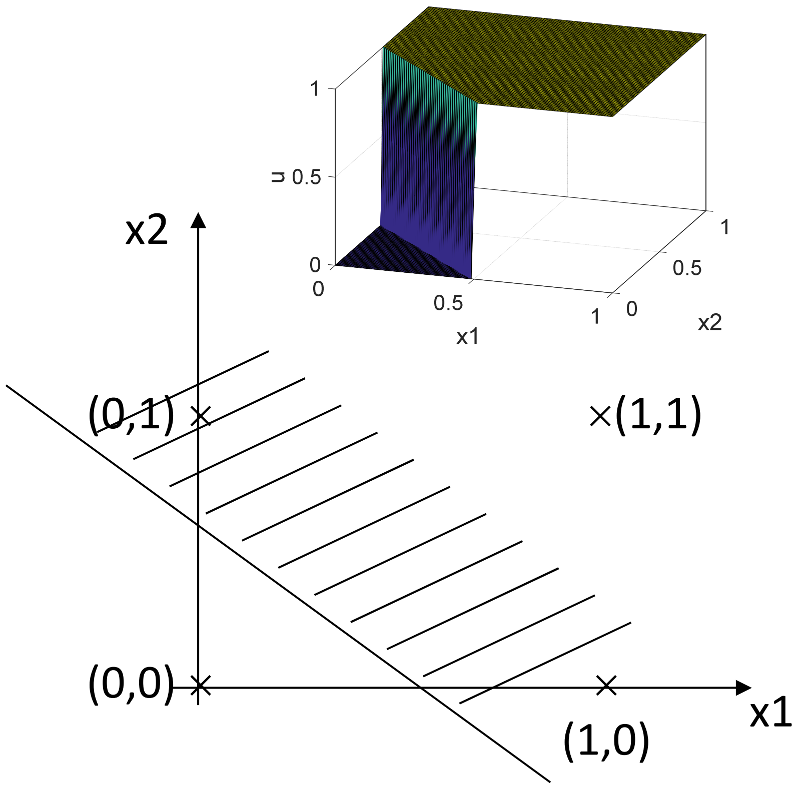


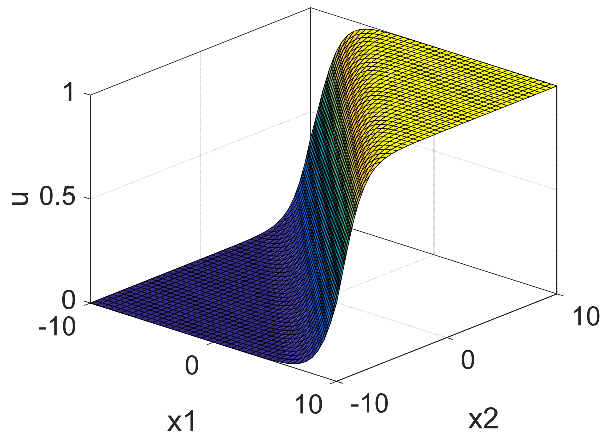
Sign -> Sigmoidal





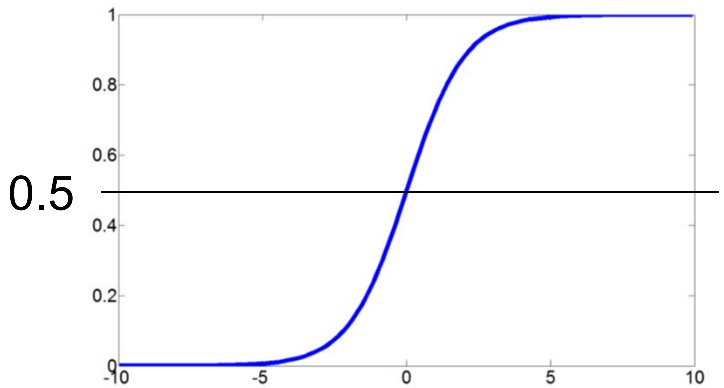
$$u = \mathrm{logsig}(P) = \frac{1}{1+e^{-P}}$$

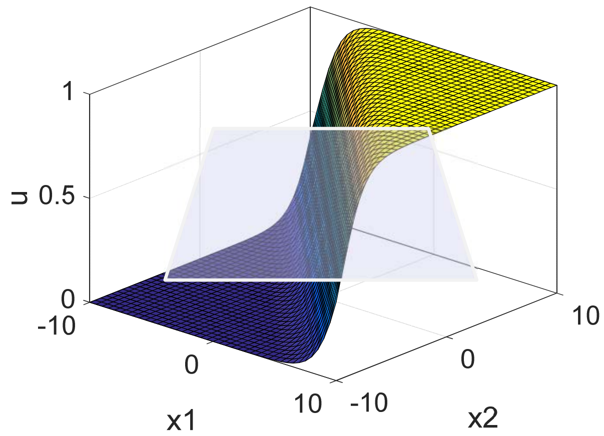




Binary classification with Sigmoidal

Need a manual threshold on the output





Class 0: u >=0.5
Class 1: u <0.5

More general solution: using Softmax network (we will see in the following)

Learning with C^1 activation functions

Pattern: input dataset
Supervised learning: the expected(reference) is the output dataset $\{t\}$

- **Error signal E**
  - Training set (R patterns)
  - $u^{(k)}_i$ is the i^{th} output neuron for the k^{th} pattern

$$E^{(k)} = \sum_{i=1}^{N} \frac{1}{2}\left(t_i^{(k)} - u_i^{(k)}\right)^2$$

is the error of the network for k^{th} input pattern

$$E = \sum_{k=1}^{R} E^{(k)} = \sum_{k=1}^{R}\sum_{i=1}^{N} \frac{1}{2}\left(t_i^{(k)} - u_i^{(k)}\right)^2$$

over all the R patterns

Network		Pattern
output	$u^{(1)}_1$, $u^{(1)}_2$	1: ($x^{(1)}_1$, $x^{(1)}_2$, $x^{(1)}_3$)
reference	$t^{(1)}_1$, $t^{(1)}_2$	
output	$u^{(2)}_1$, $u^{(2)}_2$	2: ($x^{(2)}_1$, $x^{(2)}_2$, $x^{(2)}_3$)
reference	$t^{(2)}_1$, $t^{(2)}_2$	
		…
output	$u^{(k)}_1$, $u^{(k)}_2$	*k*: ($x^{(k)}_1$, $x^{(k)}_2$, $x^{(k)}_3$)
reference	$t^{(k)}_1$, $t^{(k)}_2$	
		…
output	$u^{(R)}_1$, $u^{(R)}_2$	*R*: ($x^{(R)}_1$, $x^{(R)}_2$, $x^{(R)}_3$)
reference	$T^{(R)}_1$, $t^{(R)}_2$	

R: number of patterns
M=3: input *lines*
N=2: output lines

Delta rule: Widrow and Hoff (1960)

- Minimization of the error signal E
  - Square makes error positive and penalizes large errors more
  - Use E to update the weights: Δw_{ij} function of E

- Main criteria
  - If E increases with the increase of w_{ij} then the variation Δw_{ij} is to be negative
  - If E increases with the decrease of w_{ij} then the variation Δw_{ij} is to be positive

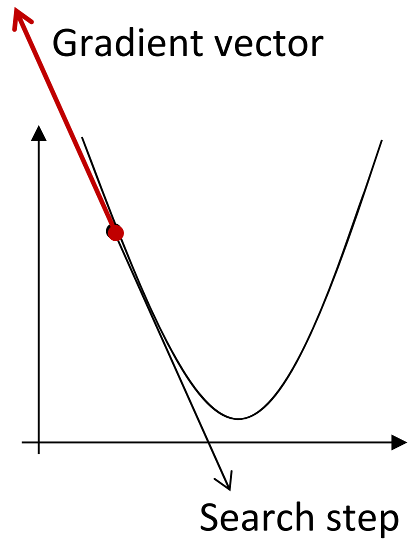
- Gradient descent method
  - Equation for Δw_{ij} ?
  - Iterative until E is below a predefined threshold (or weight convergence)
  - w_{ij} at the initial step??
  - Stop criterion

Minimize error on the training set of examples

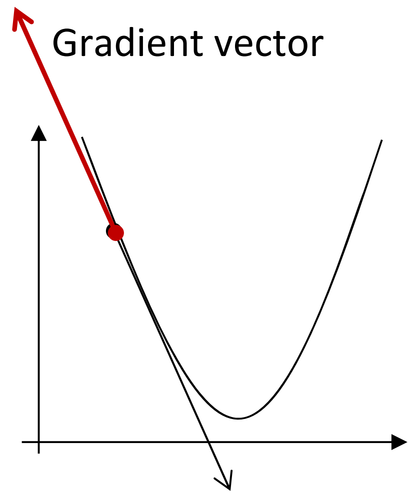
$$E = \sum_{k=1}^{R} E^{(k)} = \sum_{k=1}^{R} \sum_{i=1}^{N} \frac{1}{2} \left(t_i^{(k)} - u_i^{(k)} \right)^2$$

Gradient descent

- Gradient descent is an optimization algorithm that approaches a local minimum of a function by taking steps proportional to the negative of the gradient of the function as the current point



Gradient descent

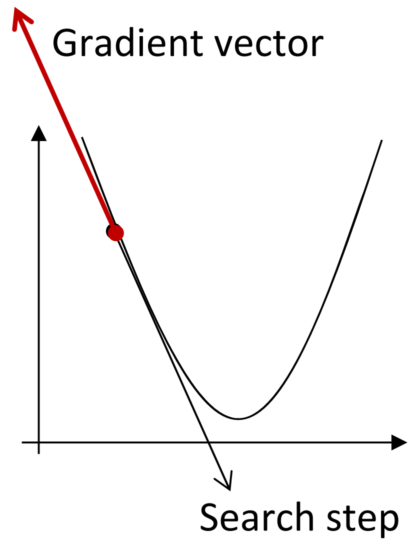


- Gradient descent is an optimization algorithm that approaches a local minimum of a function by taking steps proportional to the negative of the gradient of the function as the current point

- So, calculate the derivative (gradient) of the Cost Function with respect to the weights, and then change each weight by a small increment in the negative (opposite) direction to the gradient

$$\frac{\partial E}{\partial w} = \frac{\partial E}{\partial y}\frac{\partial y}{\partial w} = -(\bar{y} - y)x = -\delta\, x \quad \text{with} \quad \delta = (desired - measured)$$

Gradient descent



- Gradient descent is an optimization algorithm that approaches a local minimum of a function by taking steps proportional to the negative of the gradient of the function as the current point

- So, calculate the derivative (gradient) of the Cost Function with respect to the weights, and then change each weight by a small increment in the negative (opposite) direction to the gradient

$$\frac{\partial E}{\partial w} = \frac{\partial E}{\partial y}\frac{\partial y}{\partial w} = -(\bar{y} - y)x = -\delta\, x \quad \text{with} \quad \delta = (desired - measured)$$

- In order to reduce E by gradient descent, you update the weights in the negative direction of the gradient vector
- η is the learning rate to modulate the amplitude of the gradient vector (to be chosen a priori)
  - $0<\eta<1$

$$\Delta w = f\left(\frac{\partial E}{\partial w}\right)$$

$$w_{new} = w_{old} + \Delta w$$

$$\Delta w = \eta\,\delta\, x$$

$$E = \sum_{k=1}^{R} E^{(k)} = \sum_{k=1}^{R}\sum_{i=1}^{N}\frac{1}{2}\left(t_i^{(k)} - u_i^{(k)}\right)^2$$

$$\frac{\partial E}{\partial w_{ij}} = \frac{\partial\left(\frac{1}{2}\left(t_i - u_i\right)^2\right)}{\partial w_{ij}} \quad \Rightarrow \quad \frac{\partial\left(\frac{1}{2}\left(t_i - u_i\right)^2\right)}{\partial u_i}\frac{\partial u_i}{\partial w_{ij}}$$

Apply Chain rule

$$-\left(t_i - u_i\right)\frac{\partial u_i}{\partial w_{ij}}$$ with $$u_i = f\left(P_i\right) = f\left(\sum_{j}^{M} w_{ij}x_j\right)$$

$$\Rightarrow \quad -\left(t_i - u_i\right)\frac{\partial u_i}{\partial P_i}\frac{\partial P_i}{\partial w_{ij}} \quad \Rightarrow \quad -\left(t_i - u_i\right) f'\left(P_i\right)\frac{\partial P_i}{\partial w_{ij}}$$

$$\Rightarrow \quad -\left(t_i - u_i\right) f'\left(P_i\right)\frac{\partial\left(\sum_{j=1}^{M} w_{ij}x_j\right)}{\partial w_{ij}} \quad \Rightarrow \quad -\left(t_i - u_i\right) f'\left(P_i\right)x_j$$

Delta rule recap

$$\Delta w_{ij} \approx \left(\frac{\partial E}{\partial w_{ij}} \right)$$

$$\Delta w_{ij} = \eta \sum_{k=1}^{R} \left(t_i^{(k)} - u_i^{(k)} \right) f' \left(P_i^{(k)} \right) x_j^{(k)}$$

Linear

$$f(P_i) = aP_i$$

$$f'(P_i) = a$$

Hyperbolic

$$f(P_i) = \tanh(P_i) = \frac{e^{P_i} - e^{-P_i}}{e^{P_i} + e^{-P_i}}$$

$$f'(P_i) = 1 - (\tanh(P_i))^2$$

Logistic

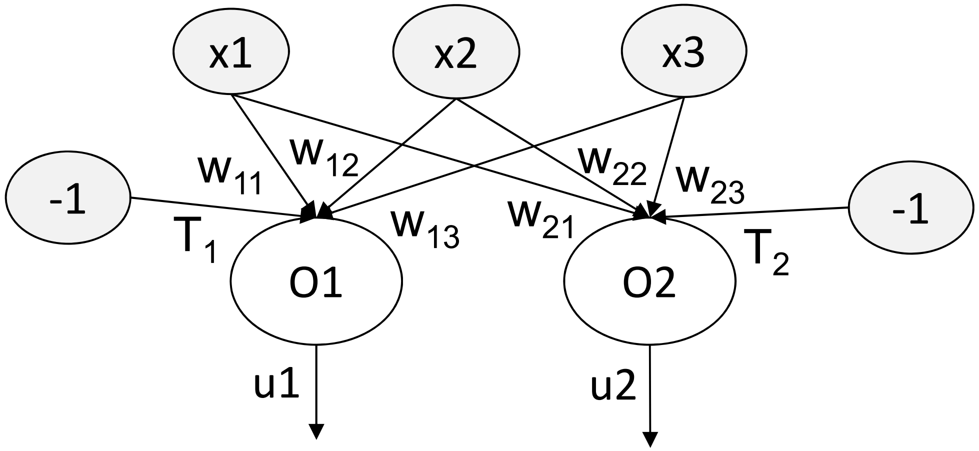
$$f(P_i) = \frac{1}{1 + e^{-P_i}}$$

$$f'(P_i) = f(P_i)(1 - f(P_i))$$

Heaviside

$$f(P_i) = \begin{cases} 0, P_i \leq 0 \\ 1, P_i > 0 \end{cases}$$

$$f'(P_i) = \delta(P_i) \begin{cases} 1, P_i = 0 \\ 0, elsewhere \end{cases}$$



$$\Delta w_{ij} \approx \left(\frac{\partial E}{\partial w_{ij}} \right)$$

$$\Delta w_{ij} = \eta \sum_{k=1}^{R} \left(t_i^{(k)} - u_i^{(k)} \right) f'\left(P_i^{(k)} \right) x_j^{(k)}$$

R: number of patterns in the training set

$$\Delta w_{11} = \eta \sum_{k=1}^{R} \left(t_1^{(k)} - u_1^{(k)} \right) f'\left(P_1^{(k)} \right) x_1^{(k)}$$

$$\Delta w_{21} = \eta \sum_{k=1}^{R} \left(t_2^{(k)} - u_2^{(k)} \right) f'\left(P_2^{(k)} \right) x_1^{(k)}$$

$$\Delta w_{12} = \eta \sum_{k=1}^{R} \left(t_1^{(k)} - u_1^{(k)} \right) f'\left(P_1^{(k)} \right) x_2^{(k)}$$

$$\Delta w_{22} = \eta \sum_{k=1}^{R} \left(t_2^{(k)} - u_2^{(k)} \right) f'\left(P_2^{(k)} \right) x_2^{(k)}$$

$$\Delta w_{13} = \eta \sum_{k=1}^{R} \left(t_1^{(k)} - u_1^{(k)} \right) f'\left(P_1^{(k)} \right) x_3^{(k)}$$

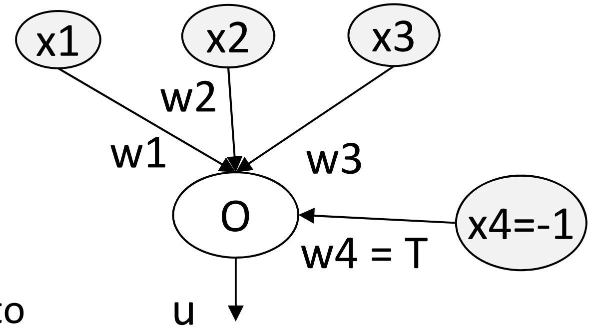
$$\Delta w_{23} = \eta \sum_{k=1}^{R} \left(t_2^{(k)} - u_2^{(k)} \right) f'\left(P_2^{(k)} \right) x_3^{(k)}$$

$$\Delta w_{14} = \Delta T_1 = \eta \sum_{k=1}^{R} \left(t_1^{(k)} - u_1^{(k)} \right) f'\left(P_1^{(k)} \right) (-1)$$

$$\Delta w_{24} = \Delta T_2 = \eta \sum_{k=1}^{R} \left(t_2^{(k)} - u_2^{(k)} \right) f'\left(P_2^{(k)} \right) (-1)$$

Example

Learning AND with logsig activation using the Delta Rule



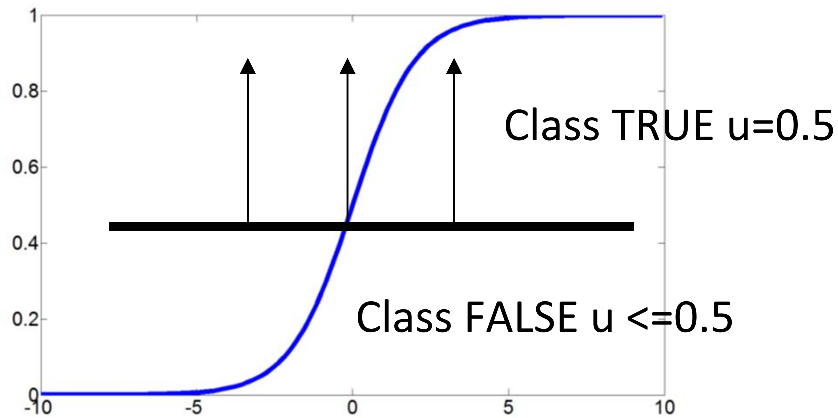
Output function is sigmoid (logistics)

	x1	x2	x3	x4
p1	0	0	0	-1
p2	0	0	1	-1
p3	0	1	0	-1
p4	1	0	0	-1
p5	1	1	0	-1
p6	0	1	1	-1
p7	1	0	1	-1
p8	1	1	1	-1

t
0
0
0
0
0
0
0
1

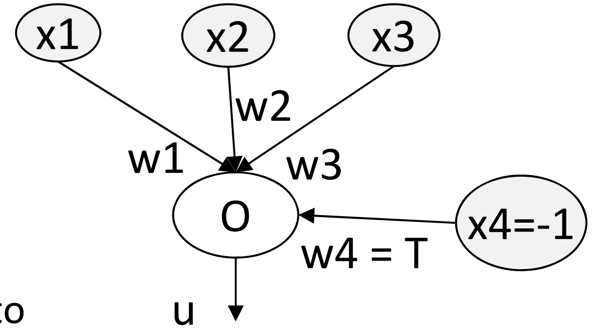
- Three input
- 8 patterns
- Weights are initialized to random (Gaussian)
- **On-line updating**
- Learning rate η =0.95

$\mathbf{w}_{start}$ = [-0.35 0.63 1.45 0.95]



Example

Learning AND with logsig activation using the Delta Rule

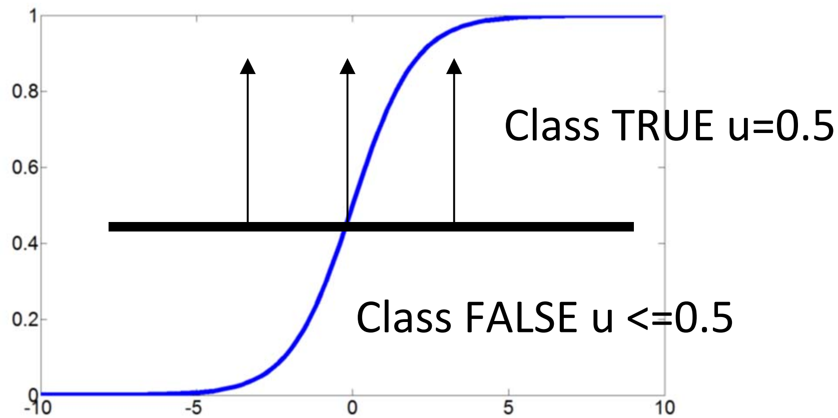


Output function is sigmoid (logistics)

	x1	x2	x3	x4
p1	0	0	0	-1
p2	0	0	1	-1
p3	0	1	0	-1
p4	1	0	0	-1
p5	1	1	0	-1
p6	0	1	1	-1
p7	1	0	1	-1
p8	1	1	1	-1

t
0
0
0
0
0
0
0
1

- Three input
- 8 patterns
- Weights are initialized to random (Gaussian)
- **On-line updating**
- Learning rate η =0.95



$\mathbf{w}_{start}$ = [-0.35 0.63 1.45 0.95]

Neuron output for each pattern using $\mathbf{w}_{start}$:

$u^{(1)}$=0.27 $u^{(2)}$=0.62 $u^{(3)}$=0.42 $u^{(4)}$=0.21 $u^{(5)}$=0.33 $u^{(6)}$=0.75 $u^{(7)}$=0.53 $u^{(8)}$=0.68

Applying Delta rule

$$\left(\Delta w_{ij}\right)^{(k)} = \eta\left(t_i^{(k)} - u_i^{(k)}\right) f'\left(P_i^{(k)}\right) x_j^{(k)}$$

$$P = w_1 x_1 + w_2 x_2 + w_3 x_3 + w_4(-1)$$

$$f\left(P_i\right) = \frac{1}{1+e^{-P_i}}$$

$$f'\left(P_i\right) = f\left(P_i\right)\left(1 - f\left(P_i\right)\right)$$

- **Step 1**: I pattern (**p1**)

w1	w2	w3	w4	P	u	t	e	Δw1	Δw2	Δw3	Δw4	nw1	nw2	nw3	nw4
-0.35	0.63	1.45	0.95	-0.95	0.27	0	-0.27	0	0	0	0.05	-0.35	0.63	1.45	1.00

P =x1*w1+x2*w2+x3*w3+w4(-1)

logsig(P)

t-u

η * e* (logsig(P)*(1-logsig(p))) * **p1**(x1)

η * e* (logsig(P)*(1-logsig(p))) * **p1**(x2)

η * e* (logsig(P)*(1-logsig(p))) * **p1**(x3)

η * e* (logsig(P)*(1-logsig(p))) * **p1**(x4)

$$w_{new} = w_{old} + \Delta w$$

Applying Delta rule

$$\left(\Delta w_{ij}\right)^{(k)} = \eta\left(t_i^{(k)} - u_i^{(k)}\right) f'\left(P_i^{(k)}\right) x_j^{(k)}$$

$$P = w_1 x_1 + w_2 x_2 + w_3 x_3 + w_4 (-1)$$

$$f(P_i) = \frac{1}{1+e^{-P_i}}$$

$$f'(P_i) = f(P_i)\left(1 - f(P_i)\right)$$

- **Step 2**: II pattern (**p2**)

w1	w2	w3	w4	P	u	t	e	Δw1	Δw2	Δw3	Δw4	nw1	nw2	nw3	nw4
-0.35	0.63	1.45	0.95	-0.95	0.27	0	-0.27	0	0	0	0.05	-0.35	0.63	1.45	1.00

w1	w2	w3	w4	P	u	t	e	Δw1	Δw2	Δw3	Δw4	nw1	nw2	nw3	nw4
-0.35	0.63	1.45	1.00	-0.44	0.61	0	-0.61	0	0	-0.13	0.13	-0.35	0.63	1.31	1.14

Applying Delta rule

$$\left(\Delta w_{ij}\right)^{(k)} = \eta\left(t_i^{(k)} - u_i^{(k)}\right) f'\left(P_i^{(k)}\right) x_j^{(k)}$$

$$P = w_1 x_1 + w_2 x_2 + w_3 x_3 + w_4(-1)$$

$$f(P_i) = \frac{1}{1+e^{-P_i}}$$

$$f'(P_i) = f(P_i)\left(1 - f(P_i)\right)$$

- **Step 3**: III pattern **(p3)**

w1	w2	w3	w4	P	u	t	e	Δw1	Δw2	Δw3	Δw4	nw1	nw2	nw3	nw4
-0.35	0.63	1.45	0.95	-0.95	0.27	0	-0.27	0	0	0	0.05	-0.35	0.63	1.45	1.00

w1	w2	w3	w4	P	u	t	e	Δw1	Δw2	Δw3	Δw4	nw1	nw2	nw3	nw4
-0.35	0.63	1.45	1.00	-0.44	0.61	0	-0.61	0	0	-0.13	0.13	-0.35	1.31	1.45	1.14

w1	w2	w3	w4	P	u	t	e	Δw1	Δw2	Δw3	Δw4	nw1	nw2	nw3	nw4
-0.35	1.31	1.45	1.14	-0.51	0.37	0	-0.37	0	-0.08	0	0.08	-0.35	0.54	1.31	1.22

....

Applying Delta rule

$$\left(\Delta w_{ij}\right)^{(k)} = \eta\left(t_i^{(k)} - u_i^{(k)}\right) f'\left(P_i^{(k)}\right) x_j^{(k)}$$

$$P = w_1 x_1 + w_2 x_2 + w_3 x_3 + w_4(-1)$$

$$f(P_i) = \frac{1}{1+e^{-P_i}}$$

$$f'(P_i) = f(P_i)\left(1 - f(P_i)\right)$$

- **Step 8**: VIII pattern (**p8**)

w1	w2	w3	w4	P	u	t	e	Δw1	Δw2	Δw3	Δw4	nw1	nw2	nw3	nw4
-0.35	0.63	1.45	0.95	-0.67	0.33	0	-0.33	-0.07	0	-0.07	0.07	-0.49	0.36	1.10	1.50

w1	w2	w3	w4	P	u	t	e	Δw1	Δw2	Δw3	Δw4	nw1	nw2	nw3	nw4
-0.49	0.36	1.10	1.50	-0.52	0.37	**1**	0.62	0.13	0.13	0.13	-0.13	-0.35	0.50	1.24	1.36

Applying Delta rule

$$\left(\Delta w_{ij}\right)^{(k)} = \eta\left(t_i^{(k)} - u_i^{(k)}\right) f'\left(P_i^{(k)}\right) x_j^{(k)}$$

$$P = w_1 x_1 + w_2 x_2 + w_3 x_3 + w_4(-1)$$

$$f(P_i) = \frac{1}{1+e^{-P_i}}$$

$$f'(P_i) = f(P_i)\left(1 - f(P_i)\right)$$

!!!! NO CONVERGENCE

- **Step 8**: VIII pattern (**p8**)

w1	w2	w3	w4	P	u	t	e	Δw1	Δw2	Δw3	Δw4	nw1	nw2	nw3	nw4
-0.35	0.63	1.45	0.95	-0.67	0.33	0	-0.33	-0.07	0	-0.07	0.07	-0.49	0.36	1.10	1.50

w1	w2	w3	w4	P	u	t	e	Δw1	Δw2	Δw3	Δw4	nw1	nw2	nw3	nw4
-0.49	0.36	1.10	1.50	-0.52	0.37	**1**	0.62	0.13	0.13	0.13	-0.13	-0.35	0.50	1.24	1.36

Neuron output for each pattern using $\mathbf{w}_{\mathbf{end}}$:

$u^{(1)}=0.20$ $u^{(2)}=0.46$ $u^{(3)}=0.29$ $u^{(4)}=0.15$ $u^{(5)}=0.22$ $u^{(6)}=0.62$ $u^{(7)}=0.33$ $u^{(8)}=0.37$

Applying Delta rule

Let us try to reinforce the learning

Feeding the network several times with the same training set of the 8 patterns

Ex: **10** iterations

$\mathbf{w}_{end}$ = [0.28 0.61 0.97 2.28]

Neuron output for each pattern using $\mathbf{w}_{end}$:

$u^{(1)}$=0.00 $u^{(2)}$=0.21 $u^{(3)}$=0.15 $u^{(4)}$=0.11 $u^{(5)}$=0.22 $u^{(6)}$=0.29 $u^{(7)}$=0.20 $u^{(8)}$=0.27

Applying Delta rule

Let us try to reinforce the learning

Feeding the network several times with the same training set of the 8 patterns

Ex: **25** iterations

$\mathbf{w}_{end}$ = [1.00 1.04 1.23 3.06]

Neuron output for each pattern using $\mathbf{w}_{end}$:

$u^{(1)}=0.00$ $u^{(2)}=0.13$ $u^{(3)}=0.11$ $u^{(4)}=0.11$ $u^{(5)}=0.26$ $u^{(6)}=0.28$ $u^{(7)}=0.25$ $u^{(8)}=0.42$

Applying Delta rule

Let us try to reinforce the learning

Feeding the network several times with the same training set of the 8 patterns

Ex: **25** iterations

$\mathbf{w}_{\mathbf{end}}$ = [1.00 1.04 1.23 3.06]

Neuron output for each pattern using $\mathbf{w}_{\mathbf{end}}$:

$u^{(1)}$=0.00 $u^{(2)}$=0.13 $u^{(3)}$=0.11 $u^{(4)}$=0.11 $u^{(5)}$=0.26 $u^{(6)}$=0.28 $u^{(7)}$=0.25 $u^{(8)}$=0.42

Applying Delta rule

Let us try to reinforce the learning

Feeding the network several times with the same training set of the 8 patterns

Ex: **50** iterations

$\mathbf{w}_{\mathbf{end}}$ = [1.57 1.50 1.66 4.15]

Neuron output for each pattern using $\mathbf{w}_{\mathbf{end}}$:

$u^{(1)}$=0.01 $u^{(2)}$=0.07 $u^{(3)}$=0.06 $u^{(4)}$=0.06 $u^{(5)}$=0.25 $u^{(6)}$=0.25 $u^{(7)}$=0.24 $u^{(8)}$=0.54

!!!! FINALLY CONVERGENCE (what is the role of η?)

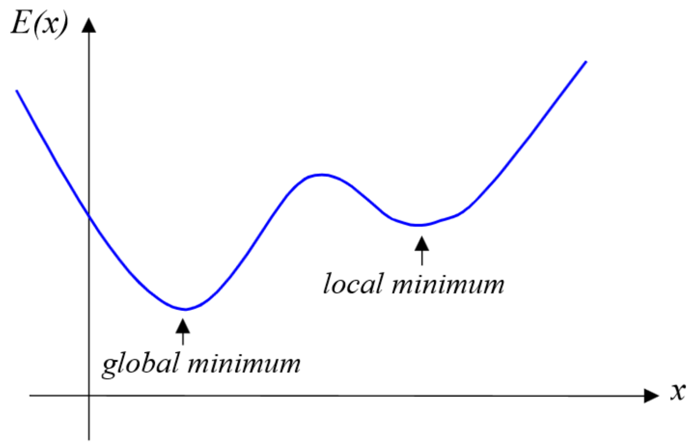
$\Downarrow$

Number of iterations is critical if we do not use any other stop criterion

Notes on delta rule

- <u>Local minima</u>
  - Dependence on the starting guess (initial values of the weights)

If we start nearby a the local minimum, we may end up at the that point rather than the global minimum. Starting with a range of different initial weight sets increases our chances of finding the global minimum. Any variation from true gradient descent will also increase our chances of stepping into the deeper valley



Notes on delta rule

- Local minima
- Dependence on the activation function

Differentiable activation function is important for the gradient descent algorithm to work

Linear functions are suitable for continuous output but may lead to parameter unbound (some workaround is needed)

The logistic function ranges from 0 to 1

Better is having a range between -1 and 1 (using Tanh) increasing the learning ability

Notes on delta rule

- Local minima
- Dependence on the activation function
- <u>Weight initialization</u>

Heuristic rule:

- Random in a small range about zero
  - Sigmoidal function can easily saturate for great values of the weights (again some workaround is needed)

Notes on delta rule

- Local minima
- Dependence on the activation function
- Weight initialization
- <u>Weight updating</u>
  - On-line updating implies that each pattern error contributes sequentially to the weight updating (selection can be random)

$$\Delta w_{ij} = \eta (t_i - u_i) f'(P_i) x_j$$

Notes on delta rule

- Local minima
- Dependence on the activation function
- Weight initialization
- Weight updating
  - Batch updating implies that all the pattern errors are cumulated before updating the other weights

$$\Delta w_{ij} = \eta \sum_{k=1}^{R} \left(t_i^{(k)} - u_i^{(k)} \right) f'\left(P_i^{(k)} \right) x_j^{(k)}$$

Notes on delta rule

- Local minima
- Dependence on the activation function
- Weight initialization
- Weight updating

  – Mini-batch updating implies the use of a subset S of the overall training dataset R

One iteration

$$\Delta w_{ij}^{(1)} = \eta \sum_{k=1}^{S} \left(t_i^{(k)} - u_i^{(k)}\right) f'\left(P_i^{(k)}\right) x_j^{(k)} \qquad w_{ij} = w_{ij} + \Delta w_{ij}^{(1)}$$

$$\Delta w_{ij}^{(2)} = \eta \sum_{k=S+1}^{2*S} \left(t_i^{(k)} - u_i^{(k)}\right) f'\left(P_i^{(k)}\right) x_j^{(k)} \qquad w_{ij} = w_{ij} + \Delta w_{ij}^{(2)}$$

$$\ldots.up\ to\ R$$

$$n = floor(\frac{R}{S})$$

$$\text{One iteration}\left\{\begin{array}{ll}
\Delta w_{ij}^{(1)} = \eta \sum_{k=1}^{S} \left(t_i^{(k)} - u_i^{(k)}\right) f'\left(P_i^{(k)}\right) x_j^{(k)} & w_{ij} = w_{ij} + \Delta w_{ij}^{(1)} \\
\Delta w_{ij}^{(2)} = \eta \sum_{k=S+1}^{2*S} \left(t_i^{(k)} - u_i^{(k)}\right) f'\left(P_i^{(k)}\right) x_j^{(k)} & w_{ij} = w_{ij} + \Delta w_{ij}^{(2)} \\
\qquad\qquad\qquad \dots & \\
\Delta w_{ij}^{(n)} = \eta \sum_{k=(n-1)S+1}^{n*S} \left(t_i^{(k)} - u_i^{(k)}\right) f'\left(P_i^{(k)}\right) x_j^{(k)} & w_{ij} = w_{ij} + \Delta w_{ij}^{(n)} \\
\Delta w_{ij}^{(n+1)} = \eta \sum_{k=n*S+1}^{R-n*S} \left(t_i^{(k)} - u_i^{(k)}\right) f'\left(P_i^{(k)}\right) x_j^{(k)} & w_{ij} = w_{ij} + \Delta w_{ij}^{(n+1)}
\end{array}\right.$$

Notes on delta rule

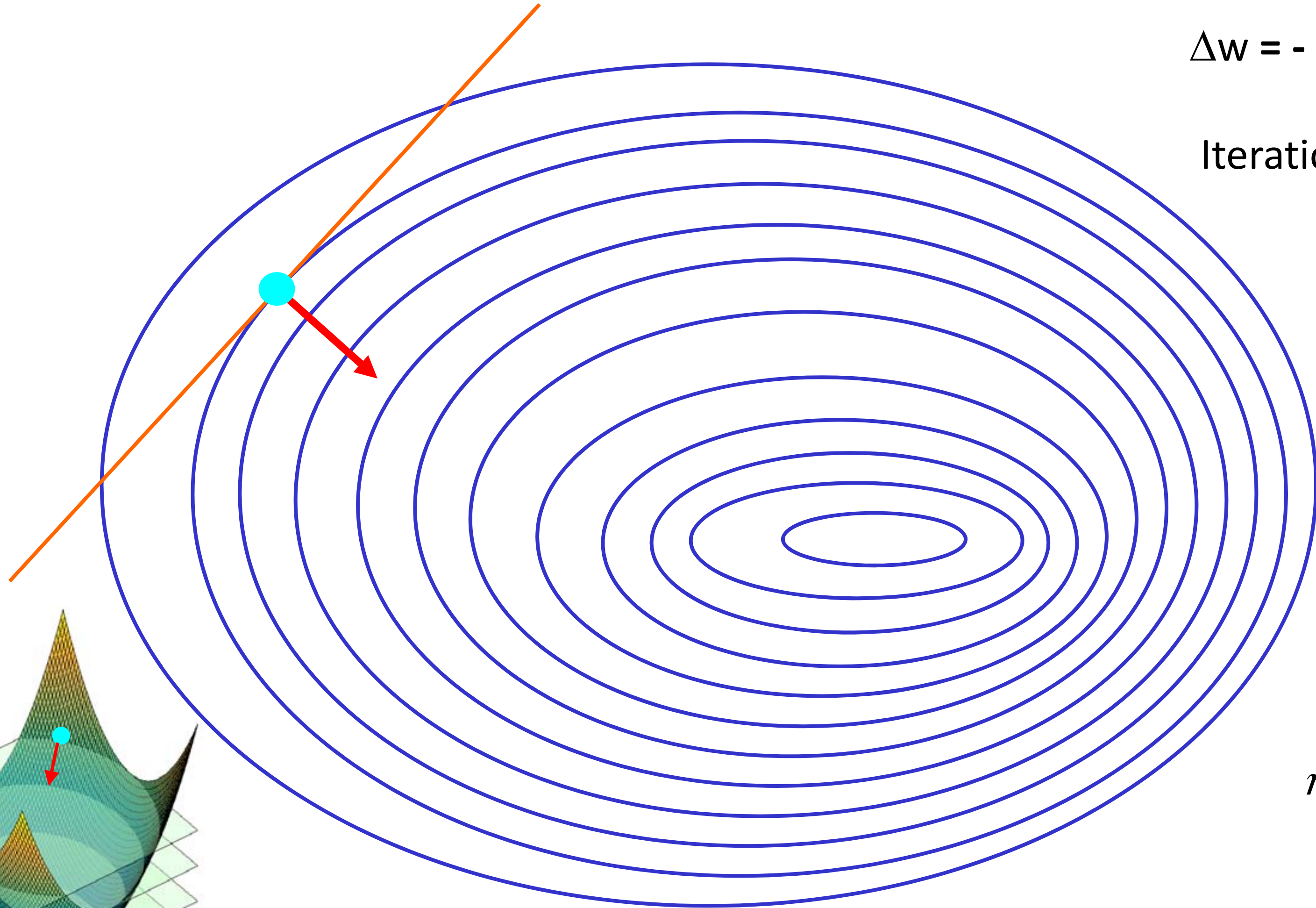
- Local minima
- Dependence on the activation function
- Weight initialization
- Weight updating
- <u>Learning rate</u>

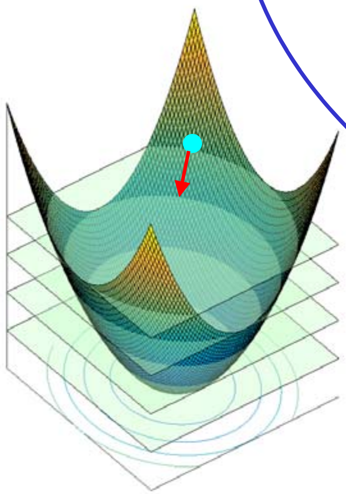
Scale down the gradient vector amplitude but..

Derivative-based Direct Search : the Gradient Method

$\Delta w = -\ \eta \cdot \nabla f(x)$

Iteration # 1





Δw = Step

η = Learning Rate

Contour plot of the cost function

Derivative-based Direct Search :
the Gradient Method

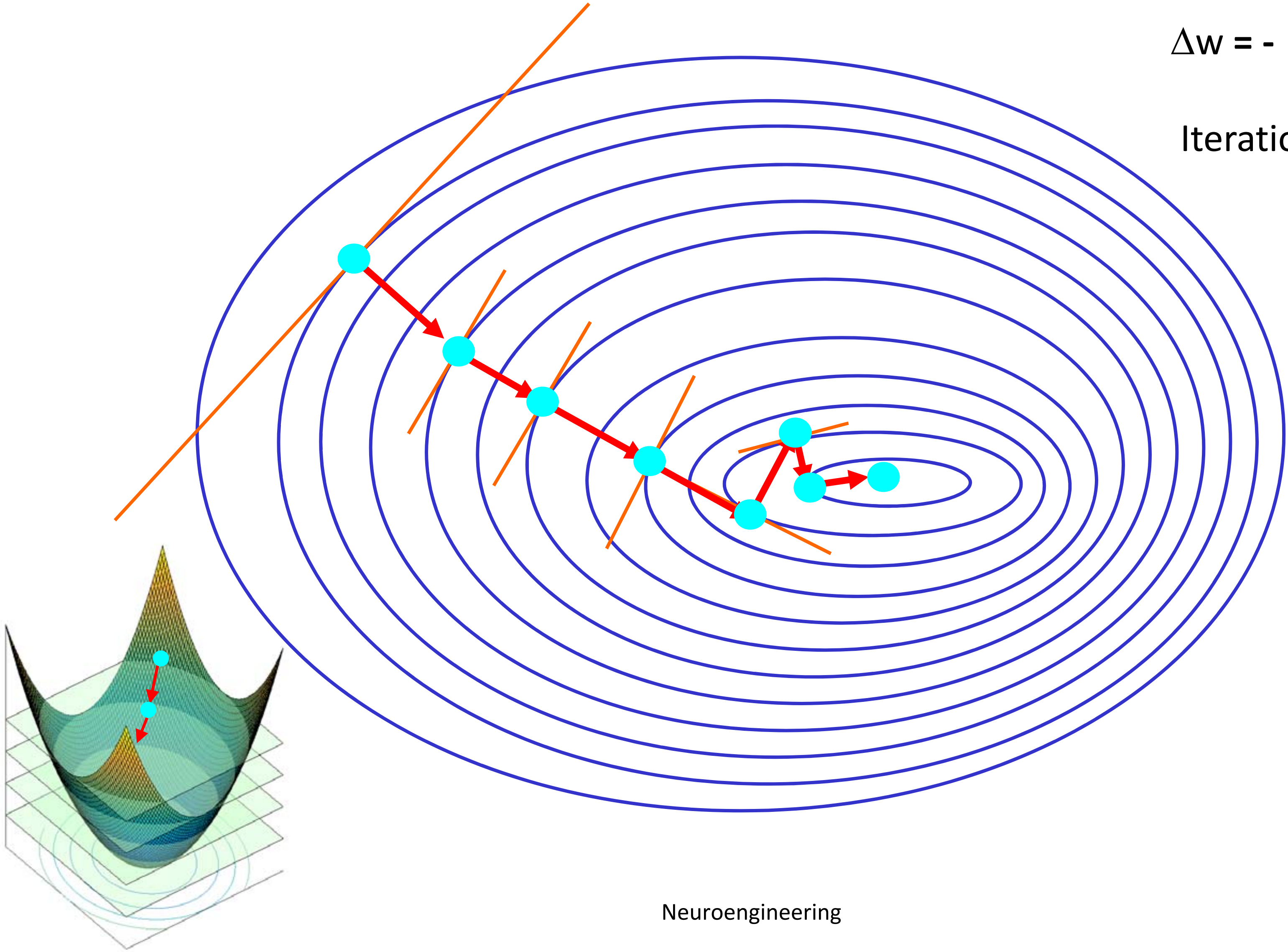
$\Delta w = - \eta \cdot \nabla f(x)$

Iteration # 1

2

3

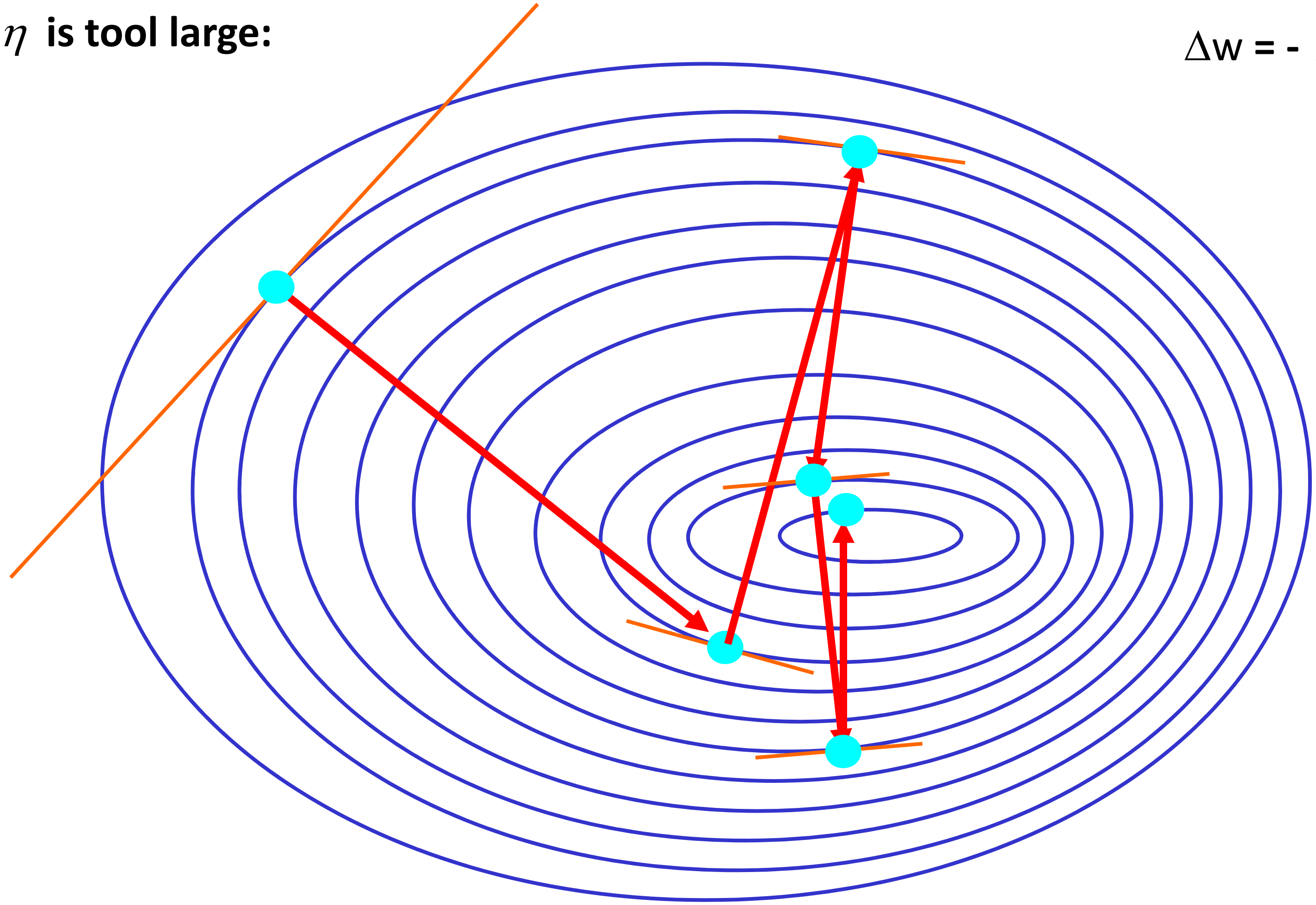
…



Derivative-based Direct Search :
the Gradient Method

If η is tool large:

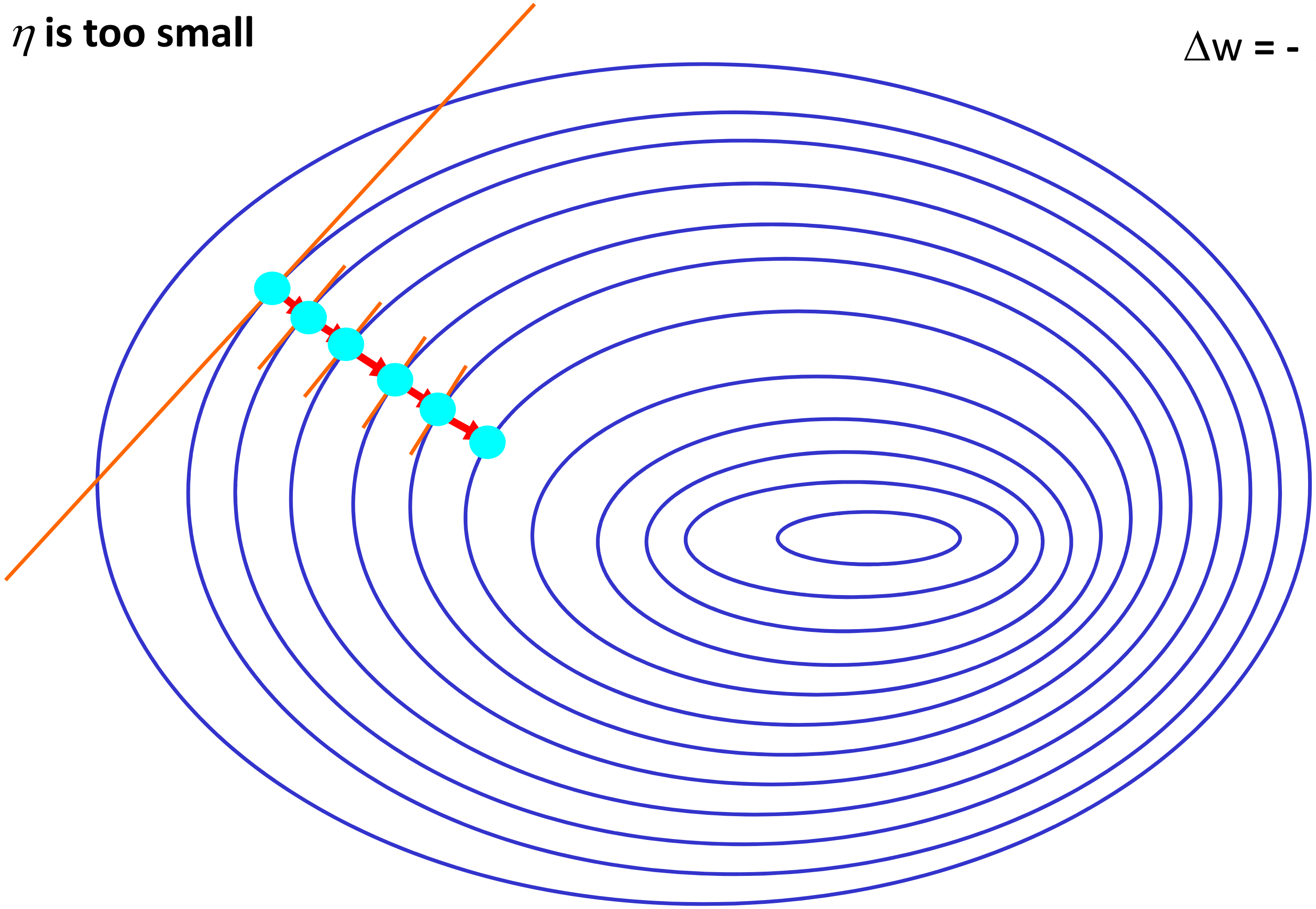
$\Delta w = - \eta \cdot \nabla f(x)$



Derivative-based Direct Search :
the Gradient Method

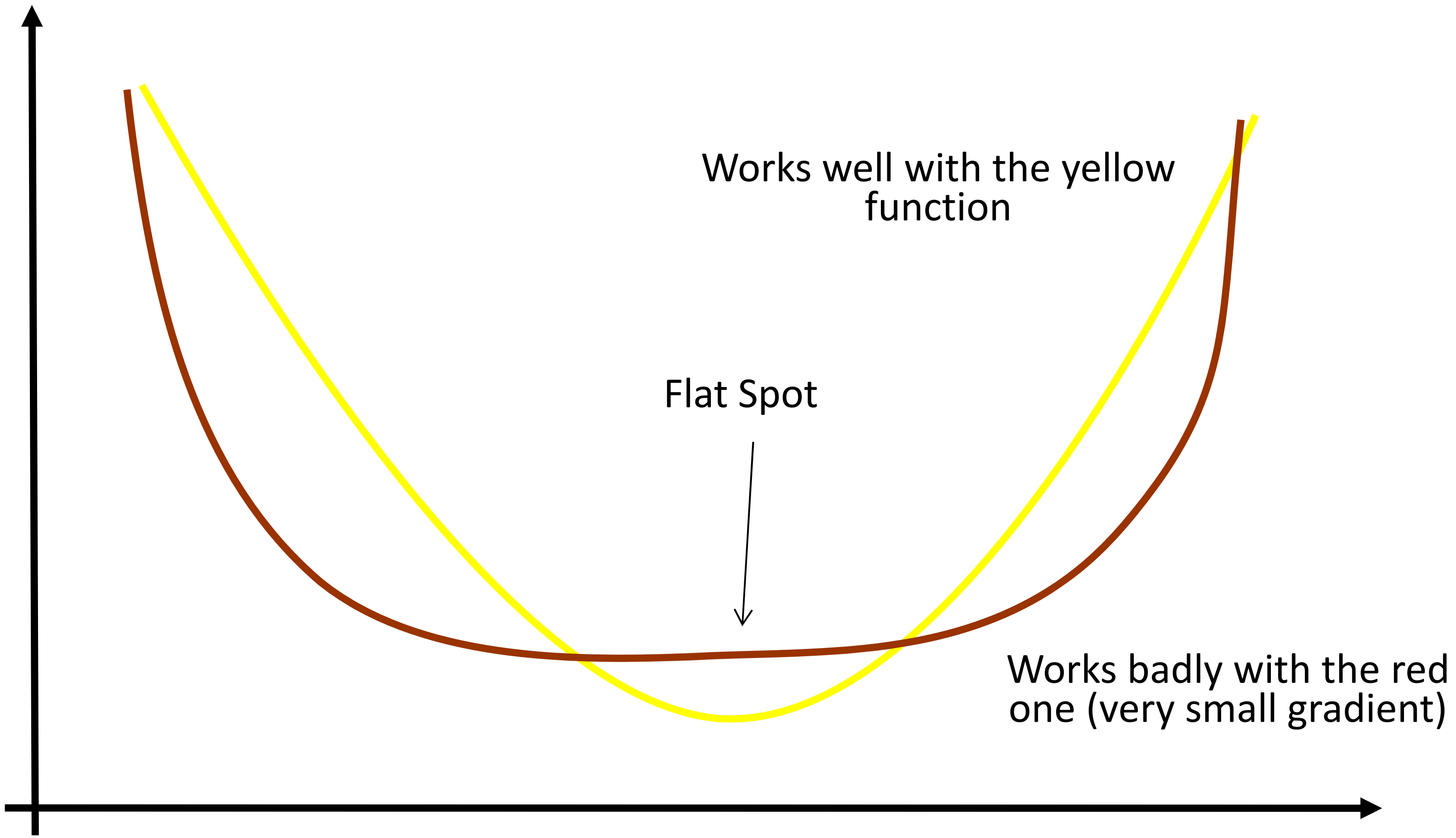
If η is too small

$\Delta w = - \eta \cdot \nabla f(x)$



Derivative-based Direct Search :
the Gradient Method

$\Delta w = - \eta \cdot \nabla f(x)$



Notes on delta rule

- Local minima
- Dependence on the activation function
- Weight initialization
- Weight updating
- Learning rate
- <u>Stop criteria</u>

Maximum number of iterations $t < T_{\max}$

Euclidean norm of the gradient vector less than a predefined threshold $\left\| \frac{\partial E}{\partial w} \right\| < \delta$

Error function less a predefined threshold $E < \varepsilon$

Hybrid criterion $\alpha \left\| \frac{\partial E}{\partial w} \right\| + \beta E < \gamma$

Notes on delta rule

- Local minima
- Dependence on the activation function
- Weight initialization
- Weight updating
- Learning rate
- Stop criteria
- Loss/cost function

Introducing a regularization factor (this is a smart workaround to previous issues)

typically $||w||^2$ for keeping weights small as much as possible

Composition of error function component and regularization factor:

$$f = \eta \sum_{i=1}^{N} \frac{1}{2}\left(t_i^{(k)} - u_i^{(k)}\right)^2 + \beta\|w\|^2$$ being β the regularization rate

Notes on delta rule

- Local minima
- Dependence on the activation function
- Weight initialization
- Weight updating
- Learning rate
- Stop criteria
- Loss/cost function
- <u>Training data</u>

Heuristic rules

- training data should be representative for the target task
- avoiding many examples of one type at the expense of another
- if one class of pattern is easy to learn, having large numbers of patterns from that class in the training set will only slow down the over-all learning process
- rescale input values (zero mean and std normalization)

- <u>Fitting against extrapolation error</u>

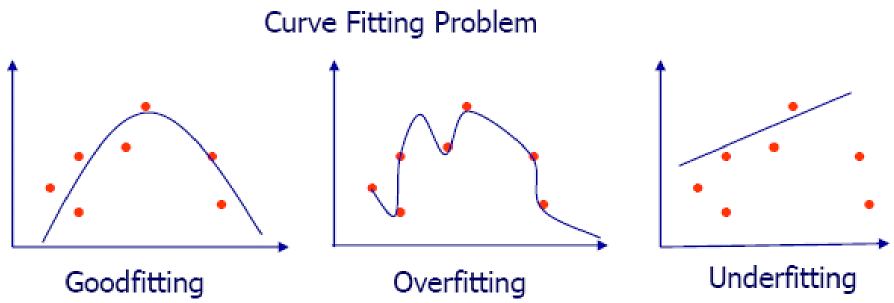
Training set	Validation set	Test set
Compute the weights	Fitting control	Extrapolation quality

To prevent under-fitting

- The network must have a sufficient number of hidden units
- Convergence threshold

To prevent over-fitting

- Avoid too much layers and units
- Additional noise superimposed to the training patterns
- The training can be stopped before convergence



Learning and generalization

Empower generalization properties

- Use of the network with patterns (Test set) not being included in the training set
- The network has good generalization capabilities if its performance on the Test set is similar to that one obtained on the training set
  - Small residual error on the training set does not guarantee good generalization properties
  - Validation during training on a set different from that one used for compute weights
  - The available pattern set is partitioned in **training** and **validation (usually 10-20%)** sets
  - The training is performed only on training set and evaluated both on the training set and validation set
  - The training is stopped when the error on the validation set overcomes a predefined **threshold**

Final remarks

- Training data should be representative
  - it should not contain too many examples of one type at the expense of another
  - if one class of pattern is easy to learn, having large numbers of patterns from that class in the training set will only slow down the over-all learning process
  - In case of continuous input data
    - rescale the input values (zero mean and std normalization)
- Weights initialized randomly in a small range about zero
  - Sigmoidal function can easily saturate for great values of the weights
- Batch mode
  - Small pattern errors can be smoothed out
- On-line mode
  - More sensitive to pattern errors
  - But … random pattern presentation (shuffle the order of the training data each epoch) makes the search in the weight space more stochastic
    - This reduces the probability to be trapped in local minima