Machine learning introduction

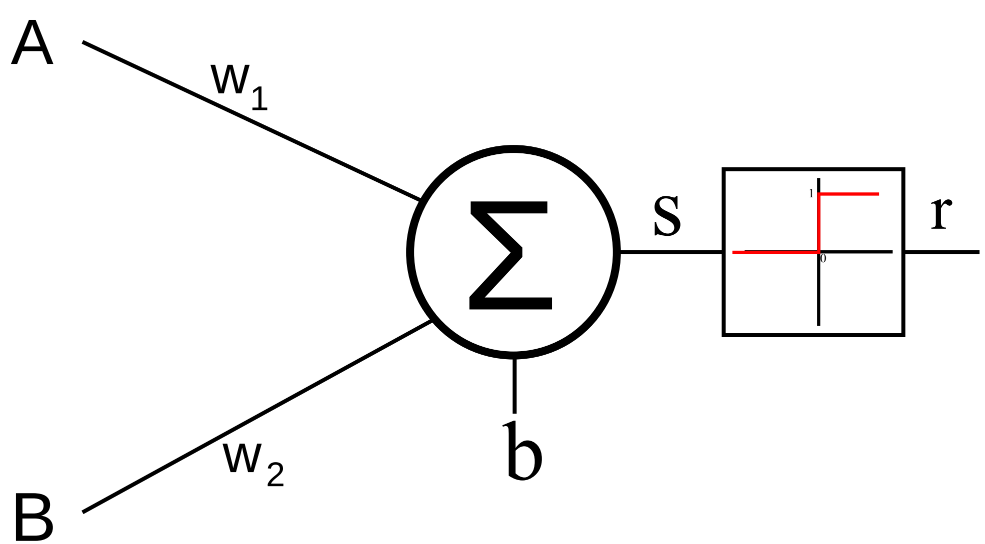
Part I – From neurons to networks

3 - Multi-Layer Networks

Simon Gay

Part I-3 : the multi-layer network

- **Limits of the single layer network**
  - Case study: logic gates
    - We want to implement logic gates AND, OR and XOR with formal neurons
    - A neuron per gate
      - Two inputs A and B and a bias b
      - Threshold activation function: 1 when sum>0, 0 otherwise



Part I-3 : the multi-layer network

- **OR gate**

Table de vérité de OU		
0	0	0
0	1	1
1	0	1
1	1	1

- **A solution (among others) : $w_1=3$, $w_2=3$, $b=-1$**
  - $0 . 3 + 0 . 3 - 1 = -1 \quad \rightarrow 0$
  - $0 . 3 + 1 . 3 - 1 = 2 \quad \rightarrow 1$
  - $1 . 3 + 0 . 3 - 1 = 2 \quad \rightarrow 1$
  - $1 . 3 + 1 . 3 - 1 = 5 \quad \rightarrow 1$

Part I-3 : the multi-layer network

- **AND gate**

Table de vérité de ET		
0	0	0
0	1	0
1	0	0
1	1	1

$$S = A \times w_1 + B \times w_2 + b$$

- **A solution (among others) : $w_1=2$, $w_2=2$, $b=-3$**
  - $0 . 2 + 0 . 2 - 3 = -3 \rightarrow 0$
  - $0 . 2 + 1 . 2 - 3 = -1 \rightarrow 0$
  - $1 . 2 + 0 . 2 - 3 = -1 \rightarrow 0$
  - $1 . 2 + 1 . 2 - 3 = 1 \rightarrow 1$

Part I-3 : the multi-layer network

- **XOR gate**

Table de vérité de XOR		
0	0	0
0	1	1
1	0	1
1	1	0

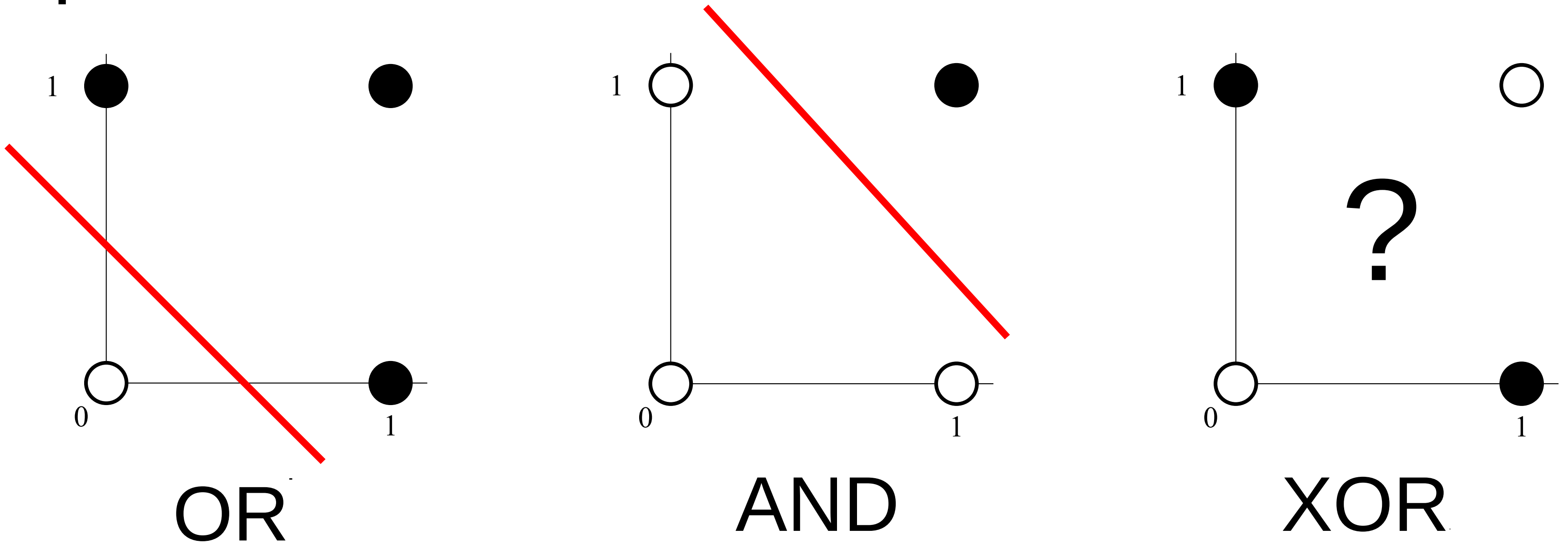
$$S = A \times w_1 + B \times w_2 + b$$

- **Solution :**

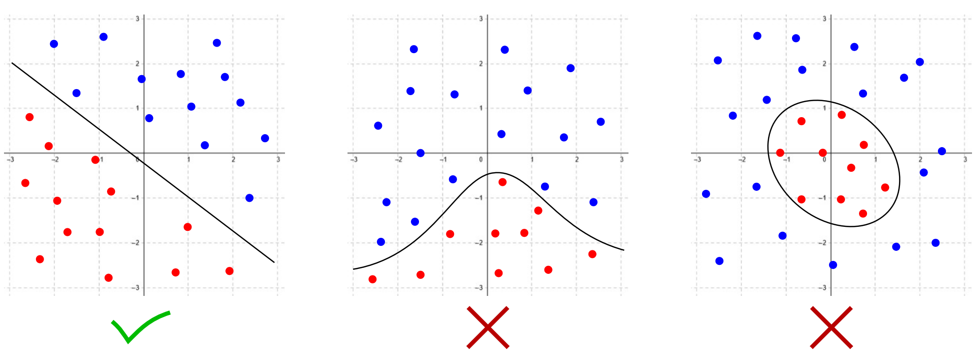
?

Part I-3 : the multi-layer network

- **Explanations**



- **Single layer networks are limited to linear problems !**

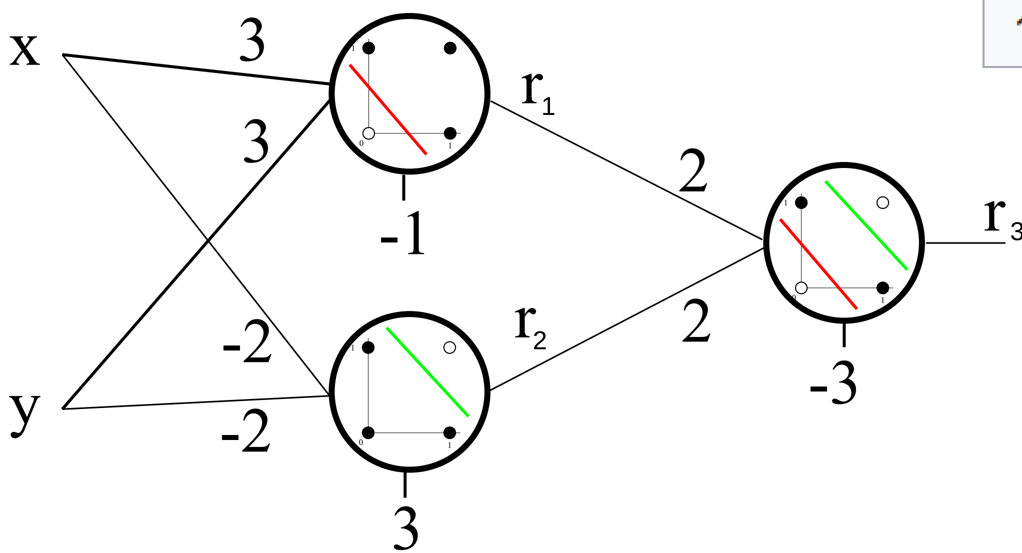


Part I-3 : the multi-layer network

- **Solution for the XOR gate**

  → **with multiple layers of neurons**

Table de vérité de XOR		
0	0	0
0	1	1
1	0	1
1	1	0

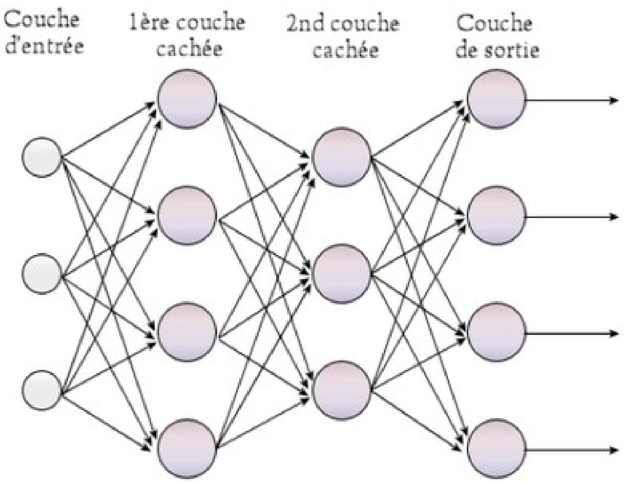


- The neurons of next layers can combine 'separators' from previous layers

  → It is possible to create networks with multiple layers to solve non-linear problemes !

Part I-3 : the multi-layer network

- **Multi-layer networks**
  - First multi-layer network created in 1986 by David Rumelhart (nearly 30 years after the perceptron!)
  - structure:
    - An input layer (input vector)
    - One or more ‘hidden’ layers
    - An output layer

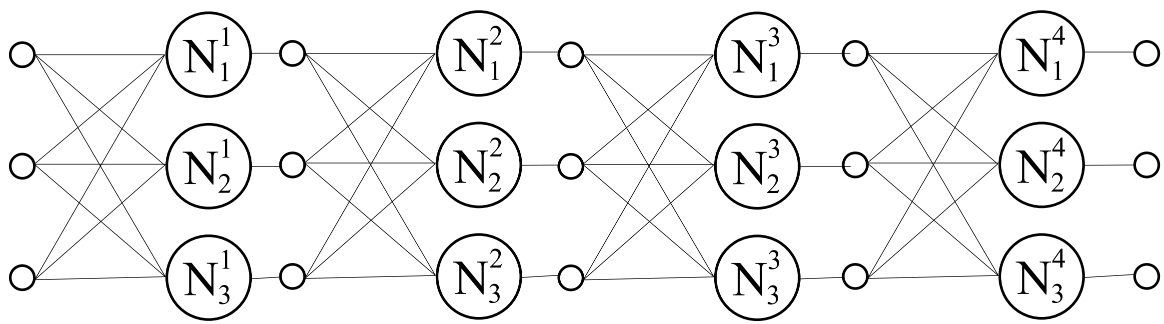


  - How computing the output values ?
  - How updating weights ?

Part I-3 : the multi-layer network

- **How computing output values ?**
  - We start with the input vector X^1
  - The first hidden layer compute outputs y_i^1 of its neurons to define its output vector Y^1
  - The next layer uses the output vector of previous layer as input
    - Thus, $X^k = Y^{k-1}$ (and $x_i^k = y_i^{k-1}$ $\forall$ i)
  - This principle is repeated until the output layer

X^1



Y^4

$Y^1 = X^2$ $Y^2 = X^3$ $Y^3 = X^4$

→ no changes from single layers' principles

Part I-3 : the multi-layer network

- **How updating network's weights ?**

  → it is a more complex problem !

  - We have to define the update for each weight of each neuron
    - Each weight of an hidden layer will influence all neurons of the next layers
  - We will define the influence of a weight variation on the network outputs
    - Thus, it will be possible to reduce the delta value

$$\frac{\partial y_i^L}{\partial w_{ij}^k}$$

  - Algorithm of the *gradient descent* !

Part I-3 : the multi-layer network

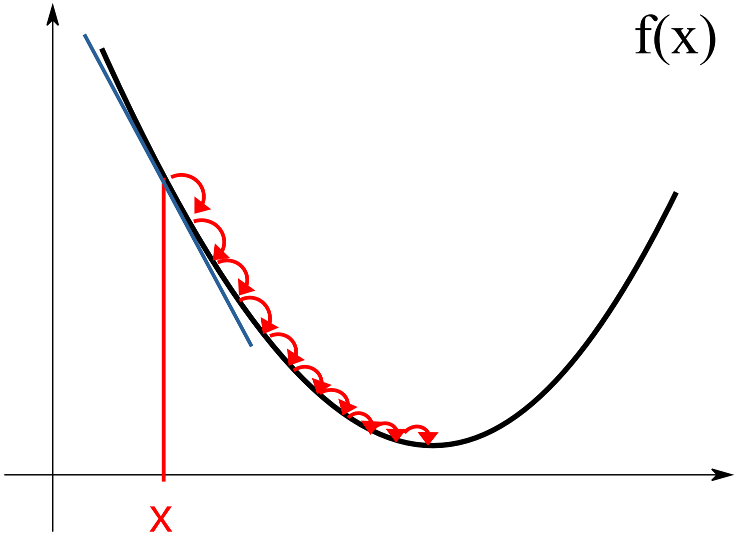
- **The gradient descent algorithm**

  – Illustration: mountain hiking
    - Suddenly: a strong blizzard, reduced visibility
    - The mountain retreat is at the bottom of the valley
    - You are still inside the valley

  – How coming back to the retreat ?
    - Define the direction of the greatest descent under you
    - Move forward a hundred meters
    - Repeat until arriving at the bottom Of the valley

Part I-3 : the multi-layer network

- **The gradient descent algorithm**



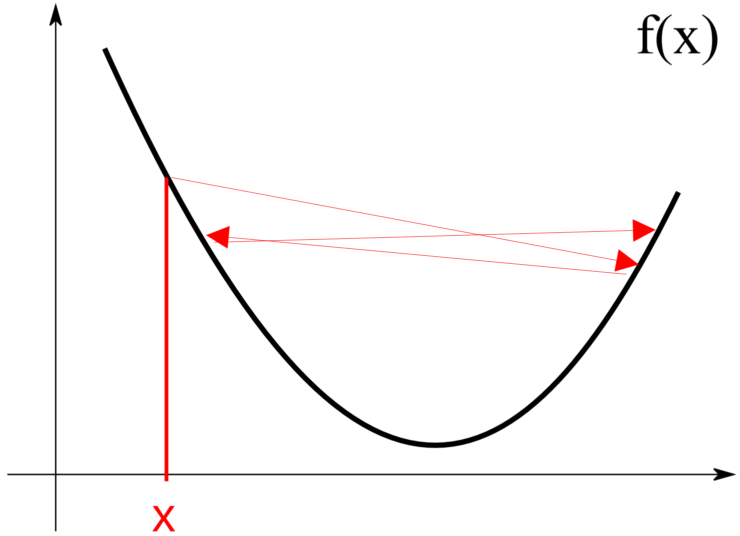
  - A function f to minimize (min($f_{(x)}$))
  - An initial random value x^0
  - The gradient of f at x is computed ($f'_{(x)}$)
  - x is updated using:

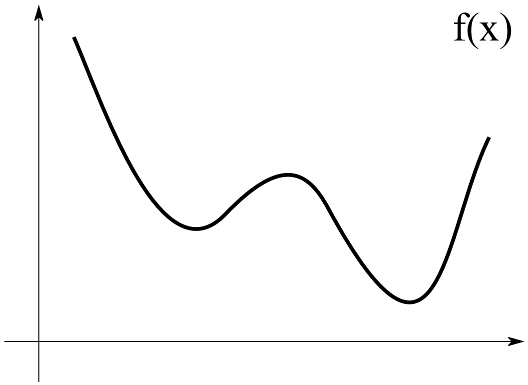
$$x^{t+1} = x^t - \alpha . \frac{d f(x)}{dx}$$

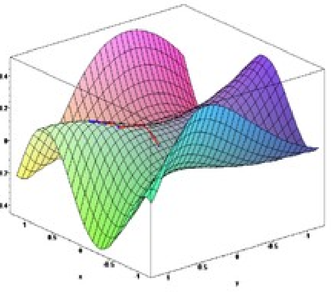
  - Repeat until convergence

Part I-3 : the multi-layer network

- **The gradient descent algorithm**
  - How setting α ?
    - Too high: the value may oscillate around minimum
    - Too low: a large number of iterations may be required
  - Problem of local minimums
    - In practice, with a large number of dimensions, it is very rare to find a point that is minimum on all dimensions







Part I-3 : the multi-layer network

- **Gradient descent applied to neurons**

  – We define an error function:

$$E(y) = \frac{1}{2} \cdot (r - y)^2$$

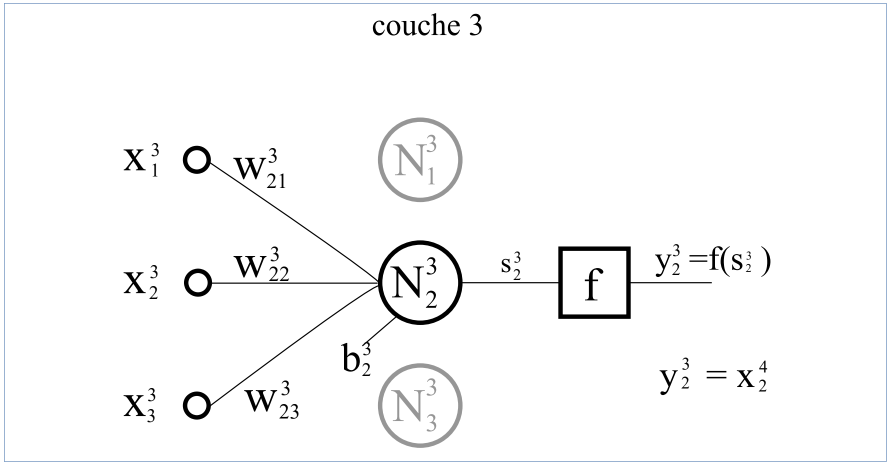
  – For each weight w_{ij}^l , we characterize the impact of a change on E

    - Gradient descent on w_{ij}^l

$$w_{ij}^l \Leftarrow w_{ij}^l - \alpha . \frac{\partial E(y)}{\partial w_{ij}^l}$$

    - We have to find: $\dfrac{\partial E(y)}{\partial w_{ij}^l}$

Part I-3 : the multi-layer network



- **Gradient descent:**
  case of last layer
  - We must find the term:

$$\frac{\partial E\left(y_i^L\right)}{\partial w_{ij}^L}$$

  - We can decompose with intermediate values w → s → y → E
    - Theorem of composed derivative functions

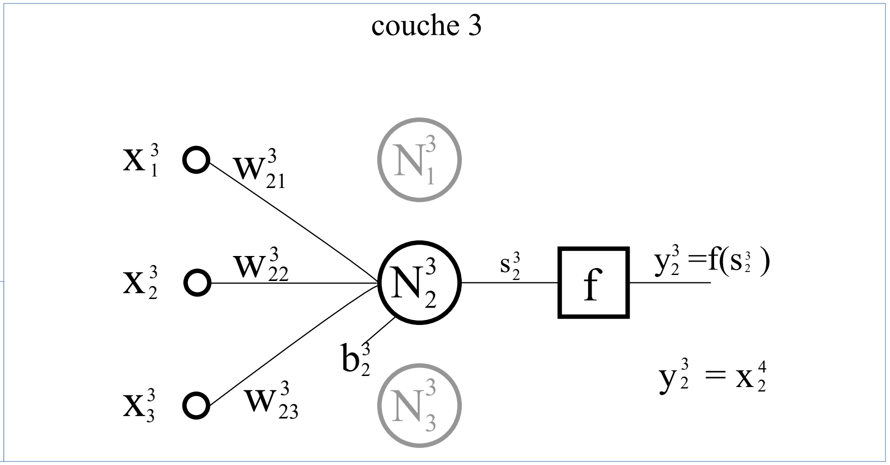
$$\frac{\partial E\left(y_i^L\right)}{\partial w_{ij}^L} = \frac{\partial E\left(y_i^L\right)}{\partial y_i^L} \cdot \frac{\partial y_i^L}{\partial s_i^L} \cdot \frac{\partial s_i^L}{\partial w_{ij}^L}$$

    - Three derivatives to find

Part I-3 : the multi-layer network

- **Gradient descent:**
  case of last layer

$$\frac{\partial E(y_i^L)}{\partial w_{ij}^L} = \frac{\partial E(y_i^L)}{\partial y_i^L} \cdot \frac{\partial y_i^L}{\partial s_i^L} \cdot \frac{\partial s_i^L}{\partial w_{ij}^L}$$



- First term: $\dfrac{\partial E(y_i^L)}{\partial y_i^L}$

- Error function: $E(y) = \dfrac{1}{2}.(r - y)^2$

Derivation of composed functions:
(f o g)’ = (f’ o g).g’

$f(x)=x^2 \rightarrow f'(x)=2.x$
$g(x)=r-x \rightarrow g'(x)=-1$

$$\frac{\partial E(y)}{\partial y} = \frac{1}{2}.\frac{\partial (r-y)^2}{\partial y} = \cancel{\frac{2}{2}}.(r-y).\frac{\partial (r-y)}{\partial y}$$

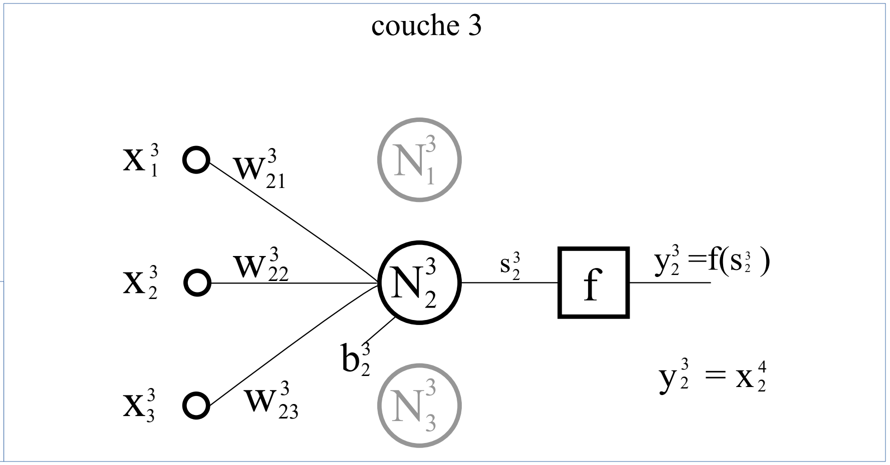
-1

$$\boxed{\frac{\partial E(y)}{\partial y} = -(r-y)}$$

Part I-3 : the multi-layer network

- **Gradient descent:**
  case of last layer

$$\frac{\partial E(y_i^L)}{\partial w_{ij}^L} = \underline{\frac{\partial E(y_i^L)}{\partial y_i^L}} \cdot \frac{\partial y_i^L}{\partial s_i^L} \cdot \frac{\partial s_i^L}{\partial w_{ij}^L}$$



- second term: $$\frac{\partial y_i^L}{\partial s_i^L} = \frac{\partial f(s_i^L)}{\partial s_i^L}$$

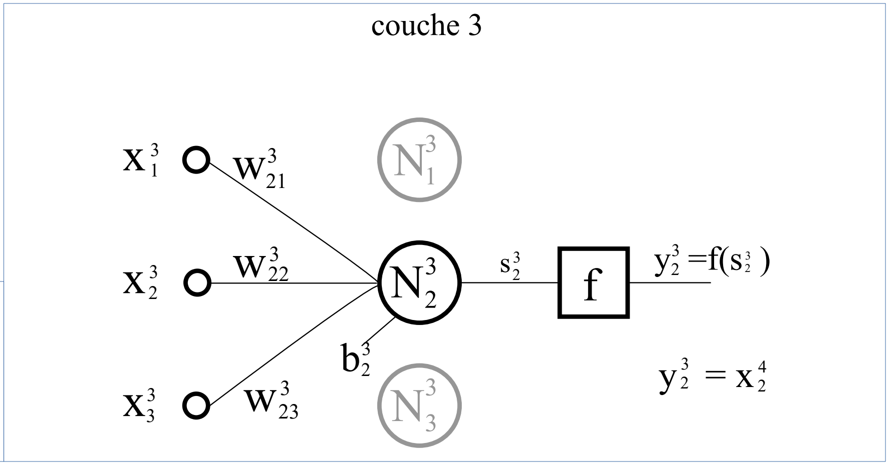
- It is just the derivative of activation function !

$$\boxed{\frac{\partial y_i^L}{\partial s_i^L} = f'_{(s_i^L)}}$$

Part I-3 : the multi-layer network

- **Gradient descent:**
  case of last layer

$$\frac{\partial E(y_i^L)}{\partial w_{ij}^L} = \underline{\frac{\partial E(y_i^L)}{\partial y_i^L}} \cdot \underline{\frac{\partial y_i^L}{\partial s_i^L}} \cdot \frac{\partial s_i^L}{\partial w_{ij}^L}$$



- Third term: $\frac{\partial s_i^L}{\partial w_{ij}^L}$

$f(x) = a.x \rightarrow f'(x) = a$

$f(x) = a \rightarrow f'(x) = 0$

'constants'

$$\frac{\partial s_i^L}{\partial w_{ij}^L} = \frac{\partial(w_{i1}^L . x_1^L + w_{i2}^L . x_2^L + \ldots + w_{ij}^L . x_i^L + \ldots + b_i^L)}{\partial w_{ij}^L} = \frac{\partial(w_{ij}^L . x_j^L)}{\partial w_{ij}^L}$$

$$\boxed{\frac{\partial s_i^L}{\partial w_{ij}^L} = x_j^L}$$

Part I-3 : the multi-layer network

- **Gradient descent (more details in ANNEX 2) :**

For output layer : $$\frac{\partial E(y_i^L)}{\partial w_{ij}^L} = -(r - y_i^L) \;.\; f'_{(s_i^L)} \;.\; x_j^L$$

We note : $$\delta_i^L = (r - y_i^L) \;.\; f'_{(s(...))}$$

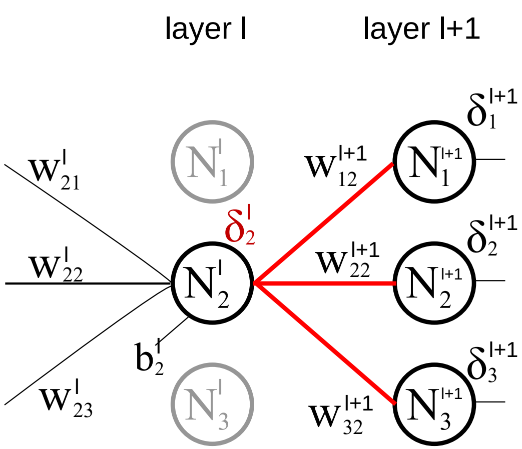
Thus : $$\boxed{w_{ij}^L \Leftarrow w_{ij}^L + \alpha . x_j^L . \delta_i^L}$$

$$w_{ij}^l \Leftarrow w_{ij}^l - \alpha . \frac{\partial E(y)}{\partial w_{ij}^l}$$

For hidden layers : $$\frac{\partial E(y_i^L)}{\partial w_{ij}^l} = -f'_{(s_i^l)} \;.\; x_j^l \;.\; \sum_{k \in [1,n]} \delta_k^{l+1} \;.\; w_{ki}^{l+1}$$

We note : $$\delta_i^l = f'_{s_i^l} \;.\; \sum_{k \in [1,n]} \delta_k^{l+1} \;.\; w_{ki}^{l+1}$$

Thus (again) $$\boxed{w_{ij}^l \Leftarrow w_{ij}^l + \alpha . x_j^l . \delta_i^l}$$



Part I-3 : the multi-layer network

- **Network reinforcement: 3 steps**
  - Forward propagation: output values of neurons are computed successively, from input layer to output layer:

$$y_i^l = f\left(\sum_{k \in [0,n]} w_{ik}^l . x_k^l + b_i^l \right)$$

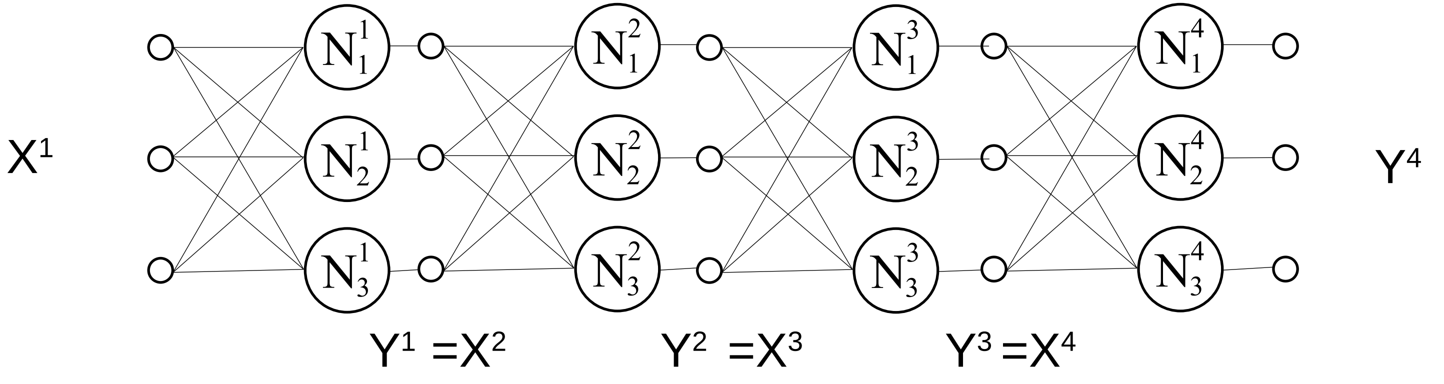
  - Backward propagation: the delta values of neurons of output layers are computed, then the deltas of hidden layers are recursively computed **from output to input.**
    - Output layer $\delta_i^L = \left(r - y_i^L\right) . f'_{(s(\ldots))}$
    - Hidden layers $\delta_i^l = f'_{s_i^l} . \sum_{k \in [1,n]} \delta_k^{l+1} . w_{ki}^{l+1}$
  - Weights are updated using previously computed deltas

$$w_{ij}^l \Leftarrow w_{ij}^l + \alpha . x_j^l . \delta_i^l$$

$$b_i^l \Leftarrow b_i^l + \alpha . \delta_i^l$$

Part I-3 : the multi-layer network

- **In practice:**
  - If weights are initialized to 0,
    - Output vectors Y^2 to Y^n will remain to 0
    - Learning is not possible
  - Weights must be initialized with random values
    - Each training will lead to a different result

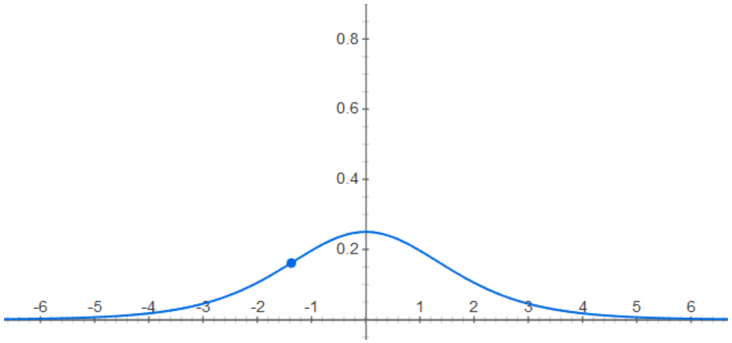


Part I-3 : the multi-layer network

- **In practice:**

  – Properties of the sigmoid function:

    - Derivative of sigmoid:



$$\sigma'(x) \;=\; \sigma(x) \;.\; (1-\sigma(x))$$

  – Terms σ(x) where already computed during *forward propagation* (neurons' outputs)

    - It is possible to simplify the equations of deltas:

$$\delta_i^L \;=\; (r-y_i^L) \;.\; y_i^L \;.\; (1-y_i^L)$$

$$\delta_i^l \;=\; y_i^l \;.\; (1-y_i^l) \;.\; \sum_{k\in[1,n]} \delta_k^{l+1} \;.\; w_{ki}^{l+1}$$

Part I-3 : the multi-layer network

- **How implementing a multi-layer network ?**
  - Forward propagation does not need changes
    - Each layer computes its output vector independently

  - Learning function must be split into three different functions
    - A function to compute output layer's deltas

$$\delta_i^L = \left(r - y_i^L\right) . f'_{(s(\ldots))}$$

    - A function to compute hidden layers' deltas

$$\delta_i^l = f'_{s_i^l} . \sum_{k \in [1,n]} \delta_k^{l+1} . w_{ki}^{l+1}$$

    - A function to update weights

$$w_{ij}^l \Leftarrow w_{ij}^l + \alpha . x_j^l . \delta_i^l$$

$$b_i^l \Leftarrow b_i^l + \alpha . \delta_i^l$$

Part I-3 : the multi-layer network

- **How implementing a multi-layer network**
  - The network is built layer by layers (here with a vector of layers)

```
// initialize structures
layers=new Layer[3];
layers[0]=new Layer(size_x*size_y, nb_layer1);
layers[1]=new Layer(nb_layer1, nb_layer2);
layers[2]=new Layer(nb_layer1, nb_layer3);
```

  - learning/exploitation process applies forward propagation, backward propagation and update on layers in the right order :

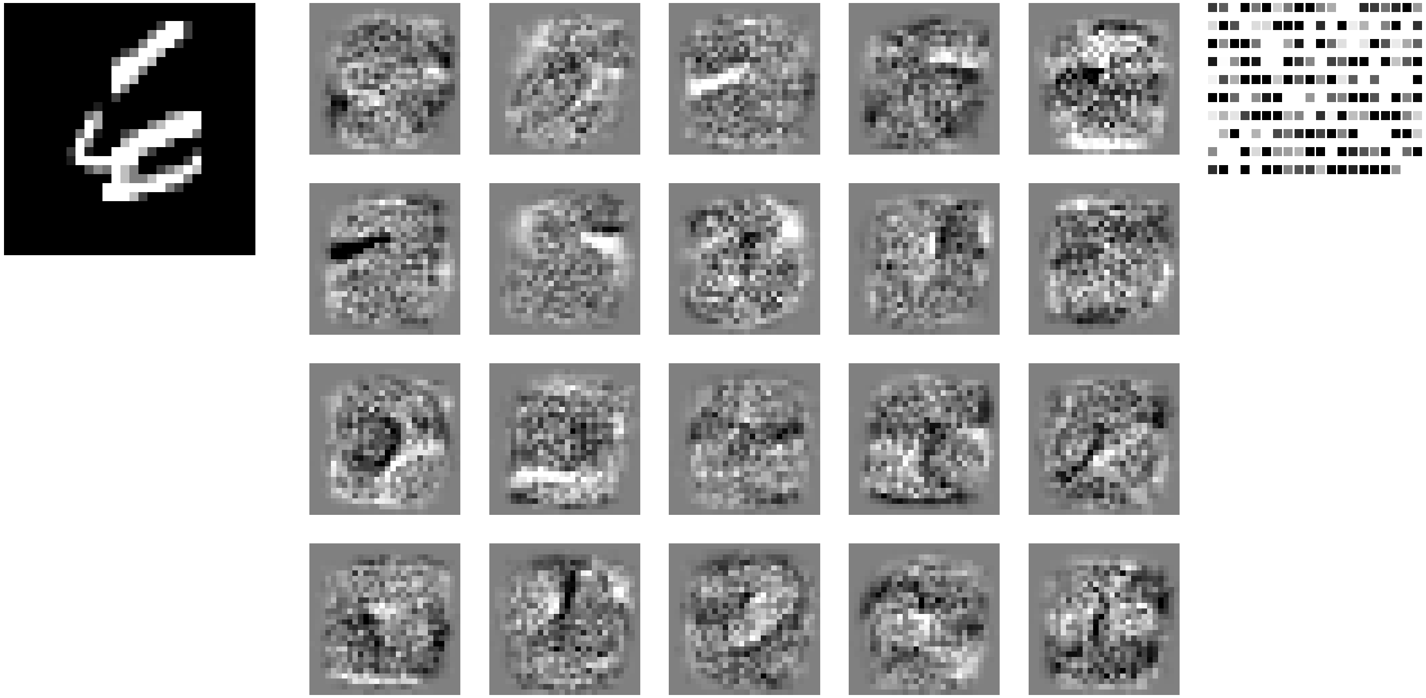
```
// compute layers
layers[0].compute(matrixImages[test]);
layers[1].compute(layers[0].results);
layers[2].compute(layers[0].results);

// backpropagate delta
layers[2].setLastDeltas(output);
layers[1].SetDeltas(layers[2]);
layers[0].SetDeltas(layers[1]);

// learn error
layers[0].learn(matrixImages[test]);
layers[1].learn(layers[0].results);
layers[2].learn(layers[1].results);
```

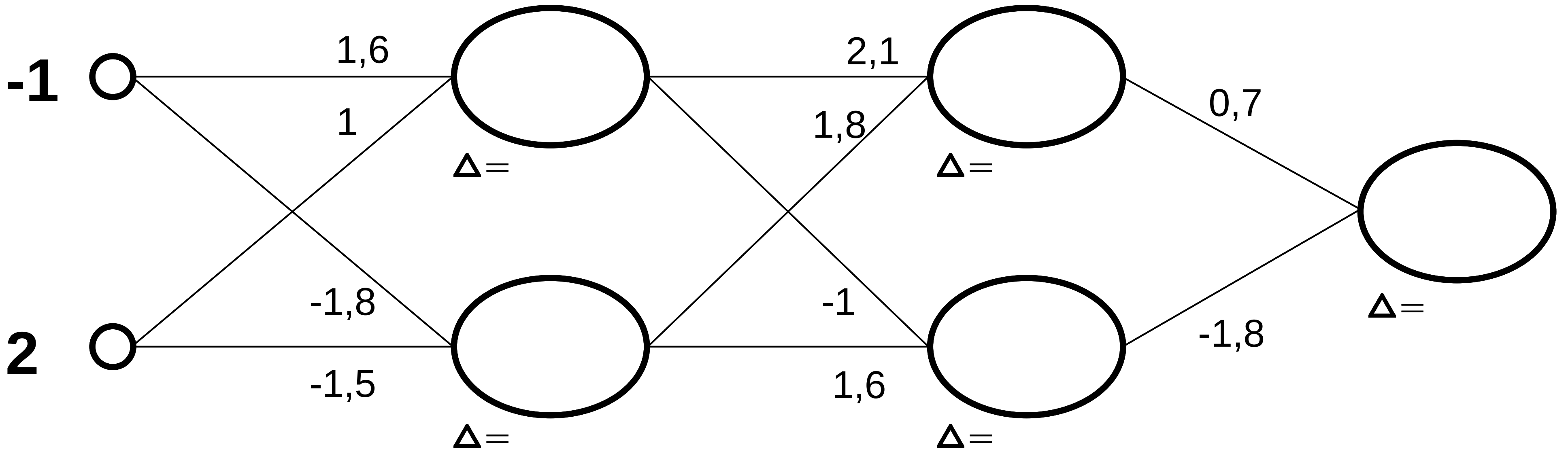
Part I-3 : the multi-layer network

- **How implementing a multi-layer network**
  - Example : 2 layers of 20 and 10 neurons
  - Lower layers defines pertinent features
  - Higher level combine features to recognize elements
  - Features are different at each training (weights randomly initialized)
  - On MNIST number, error ratio around 5 %



Part I-3 : the multi-layer network

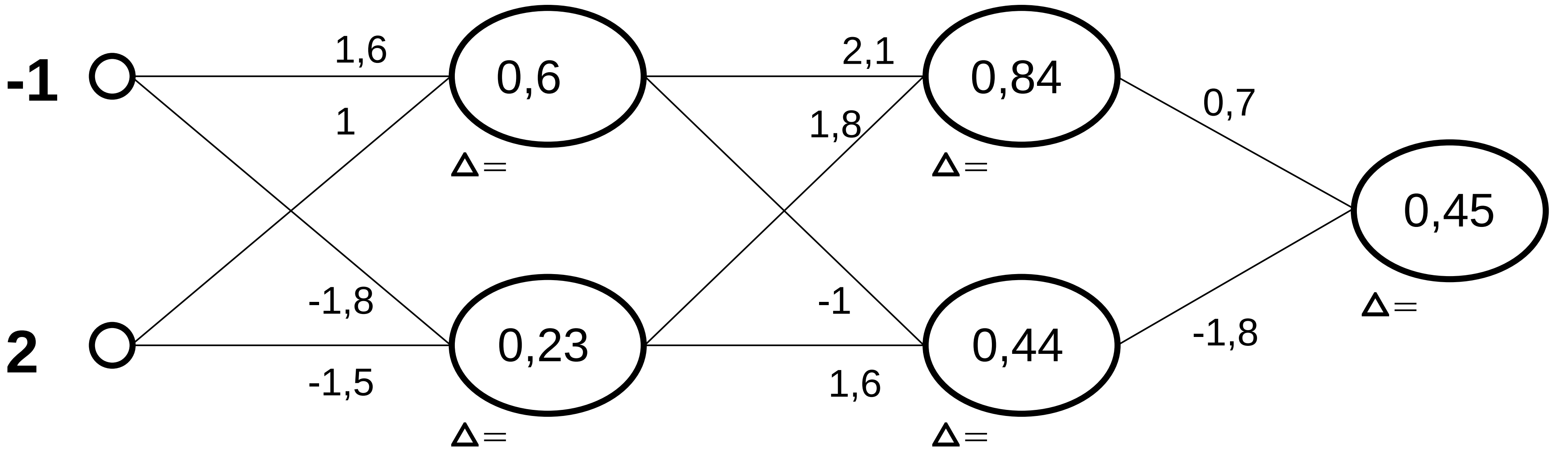
- **A little example**



- Input vector : [-1;2], expected output : 1
- Simplification: no bias
- Weights randomly initialized

Part I-3 : the multi-layer network

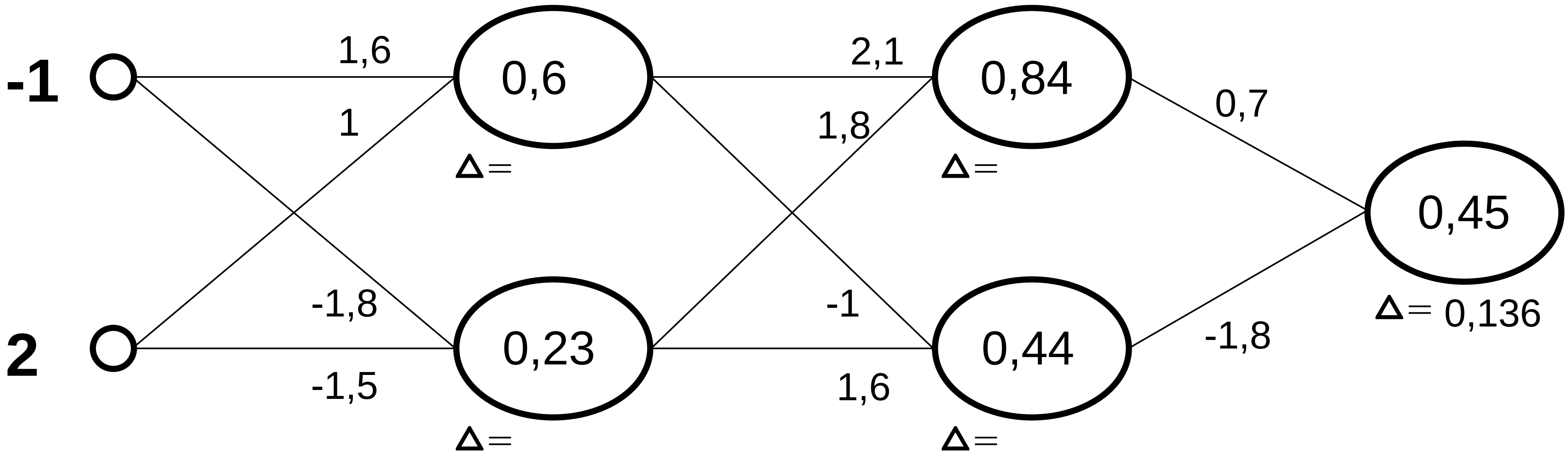
- **A little example**



- Forward propagation
  - $y_1^1 = s(\ 1{,}6 \times -1 \ + \ 1 \ \times 2\ ) \ = \ s(\ 0{,}4\ ) \ = 0{,}6$
  - $y_2^1 = s(\ -1{,}8 \times -1 \ + \ -1{,}5 \times 2\ ) \ = \ s(\ -1{,}2) \ = 0{,}23$
  - $y_1^2 = s(\ 2{,}1 \times 0{,}6 \ + \ 1{,}8 \times 0{,}23\ ) \ = \ s(\ 1{,}67\ ) \ = 0{,}84$
  - $y_2^2 = s(\ -1 \ \times 0{,}6 \ + \ 1{,}6 \times 0{,}23\ ) \ = \ s(\ -0{,}22) \ = 0{,}44$
  - $y_1^3 = s(\ 0{,}7 \times 0{,}84 \ + \ -1{,}8 \times 0{,}44\ ) = s(\ 0{,}21) \ = 0{,}45$

Part I-3 : the multi-layer network

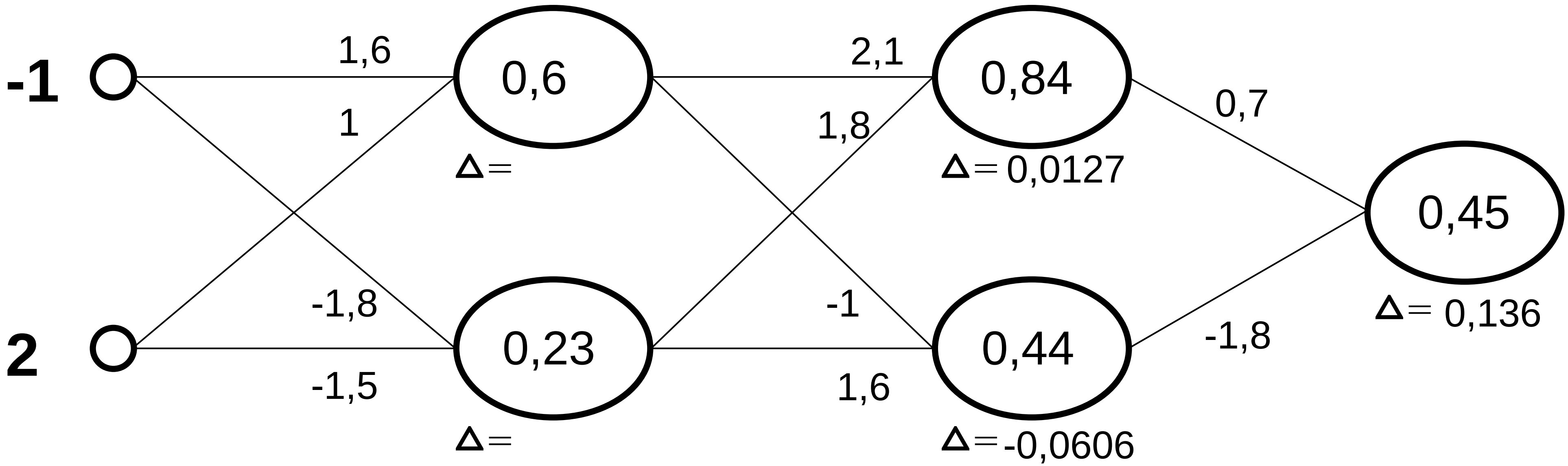
- **A little example**



- Backward propagation (r=1)
  - first delta : $$\delta_i^L = \left(r - y_i^L\right) \;.\; y_i^L \;.\; \left(1 - y_i^L\right)$$
  - Δ_1^3 = (1 – 0,45) x 0,45 x (1-0,45) = 0,136

Part I-3 : the multi-layer network

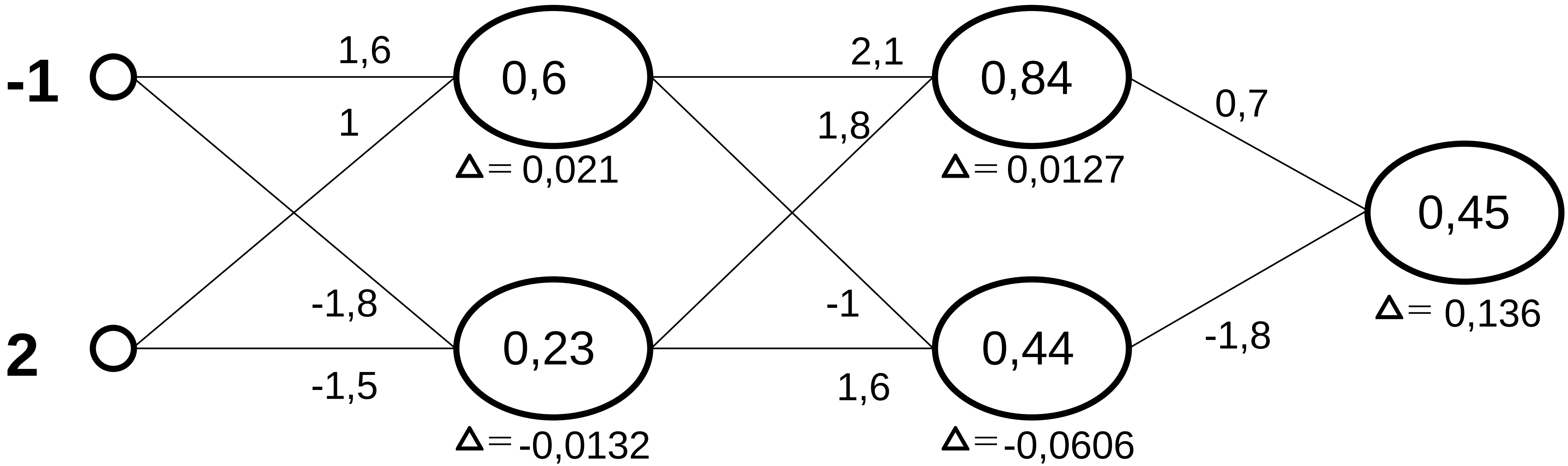
- **A little example**



- Backward propagation
  - propagation: $$\delta_i^l = y_i^l \ . \ \left(1 - y_i^l\right) \ . \sum_{k \in [1,n]} \delta_k^{l+1} \ . \ w_{ki}^{l+1}$$
  - Δ_1^2 = 0,84 x (1 – 0,84) x 0,136 x 0,7 = 0,0127
  - Δ_2^2 = 0,44 x (1 – 0,44) x 0,136 x -1,8 = -0,0606

Part I-3 : the multi-layer network

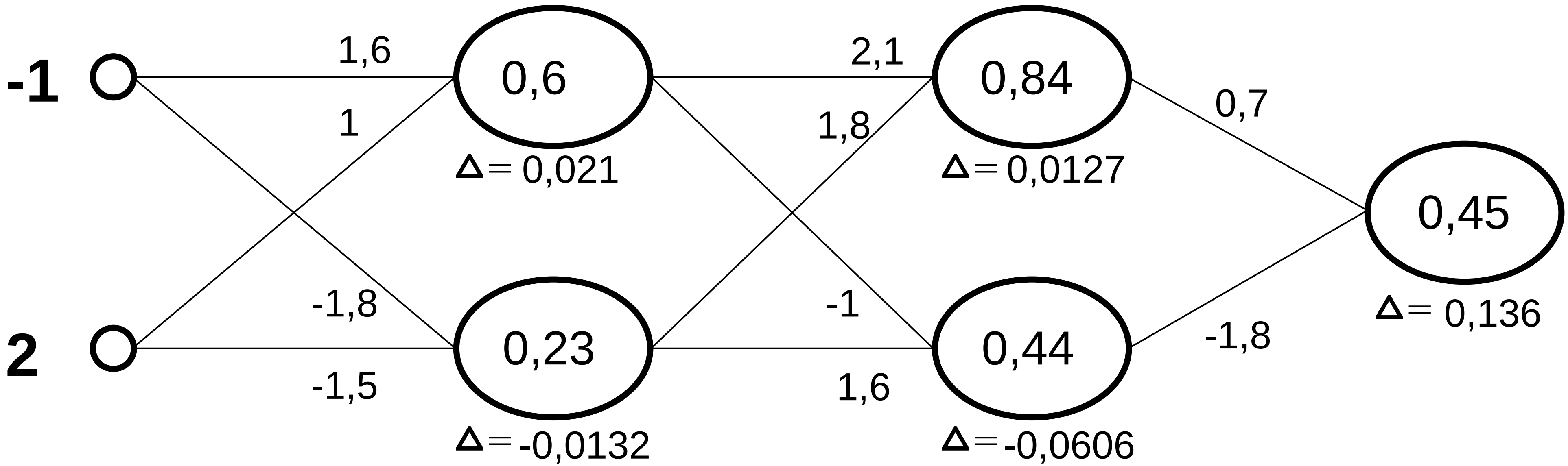
- **A little example**



- Backward propagation
  - propagation: $$\delta_i^l = y_i^l \ . \ \left(1 - y_i^l\right) \ . \ \sum_{k \in [1,n]} \delta_k^{l+1} \ . \ w_{ki}^{l+1}$$
  - Δ_1^1 = 0,6 x (1 – 0,6) x (0,0127 x 2,1 + -0,0606 x -1) = 0,021
  - Δ_2^1 = 0,23 x (1 – 0,23) x (0,0127 x 1,8 + -0,0606 x 1,6) = -0,0132

Part I-3 : the multi-layer network

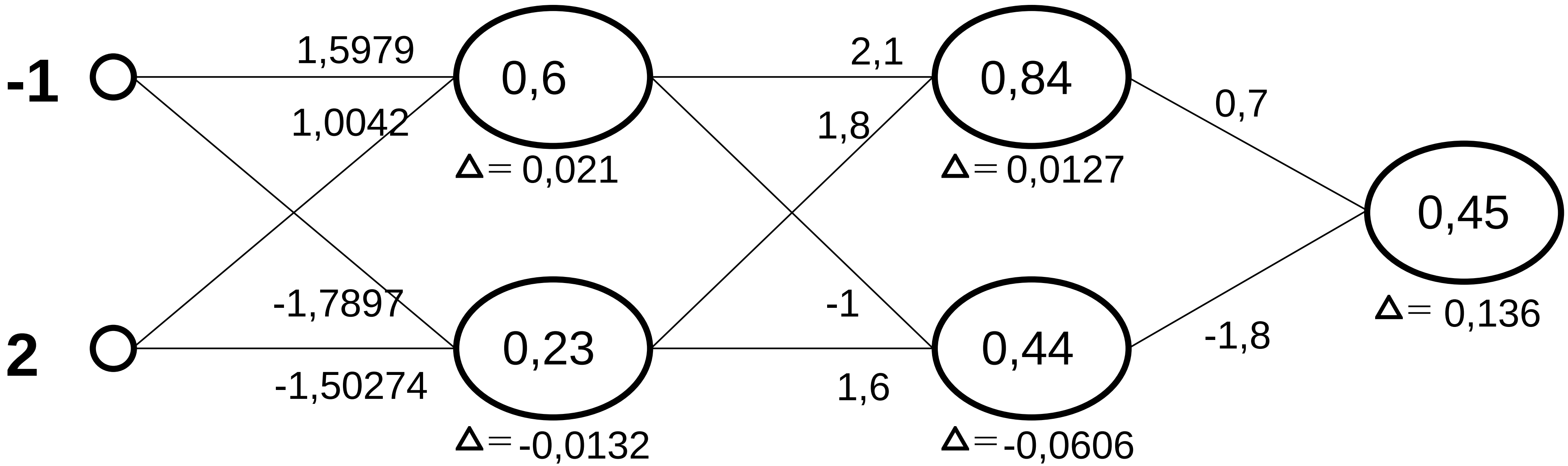
- **A little example**



- Weights update (α = 0,1) $$w_{ij}^{l} \Leftarrow w_{ij}^{l} + \alpha . x_{j}^{l} . \delta_{i}^{l}$$
  - $W_{11}^{1} \rightarrow$ 1,6 + 0,1 x -1 x 0,021 = 1,6 + -0,0021 = 1,5979
  - $W_{12}^{1} \rightarrow$ 1 + 0,1 x 2 x 0,021 = 1 + 0,0042 = 1,0042
  - $W_{21}^{1} \rightarrow$ -1,8 + 0,1 x -1 x -0,0132 = -1,8 + 0,00132 = -1,79868
  - $W_{22}^{1} \rightarrow$ -1,5 + 0,1 x 2 x -0,0132 = 1 + -0,00274 = -1,50274

Part I-3 : the multi-layer network

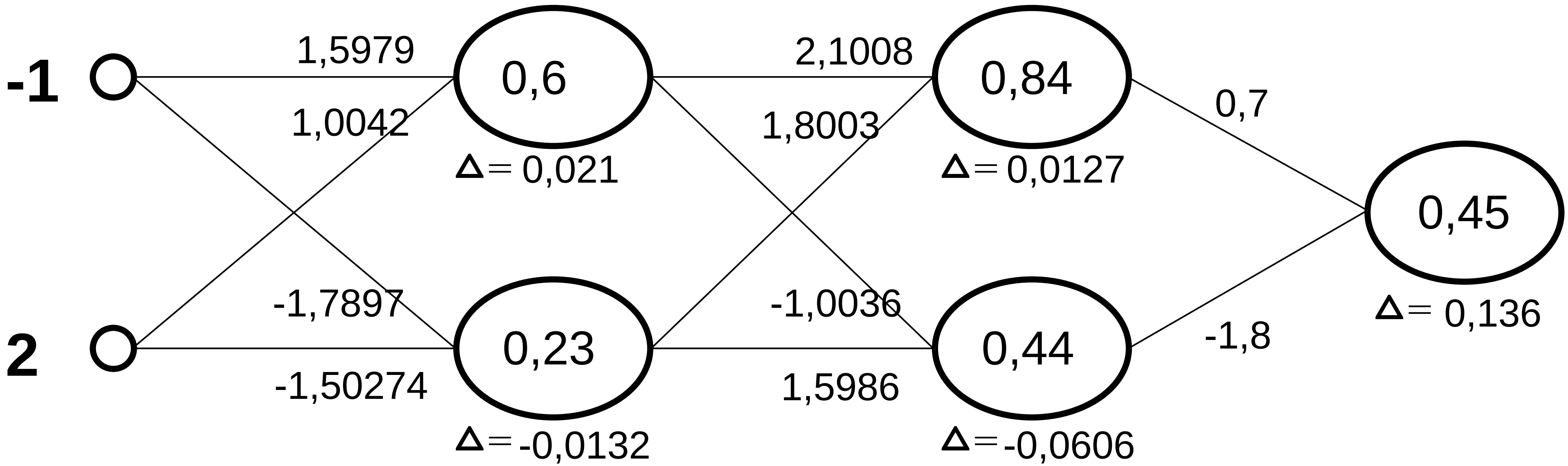
- **A little example**



- Weights update (α = 0,1) $$w_{ij}^{l} \Leftarrow w_{ij}^{l} + \alpha \, . \, x_{j}^{l} \, . \, \delta_{i}^{l}$$
  - $W_{11}^{2} \rightarrow$ 2,1 + 0,1 x 0,6 x 0,0127 = 2,1 + 0,0008 = 2,1008
  - $W_{12}^{2} \rightarrow$ 1,8 + 0,1 x 0,23 x 0,0127 = 1,8 + 0,0003 = 1,8003
  - $W_{21}^{2} \rightarrow$ -1 + 0,1 x 0,6 x -0,0606 = -1 + -0,0036 = -1,0036
  - $W_{22}^{2} \rightarrow$ 1,6 + 0,1 x 0,23 x -0,0606 = 1,6 + -0,0014 = -1,5986

Part I-3 : the multi-layer network

- **A little example**



- Weights update (α = 0,1) $\quad w_{ij}^{l} \Leftarrow w_{ij}^{l} + \alpha . x_{j}^{l} . \delta_{i}^{l}$
  - $W_{11}^{3} \rightarrow$ 0,7 + 0,1 x 0,84 x 0,0136 = 0,7 + 0,01149 = 0,71149
  - $W_{12}^{3} \rightarrow$ -1,8 + 0,1 x 0,44 x -0,0606 = -1,8 + 0,00605 = -1,79395

- Learning process is repeated for each input sample

Part I-3 : the multi-layer network

http://playground.tensorflow.org