Multilayer Perceptron

Rodrigo Fernandes de Mello

Invited Professor at Télécom ParisTech

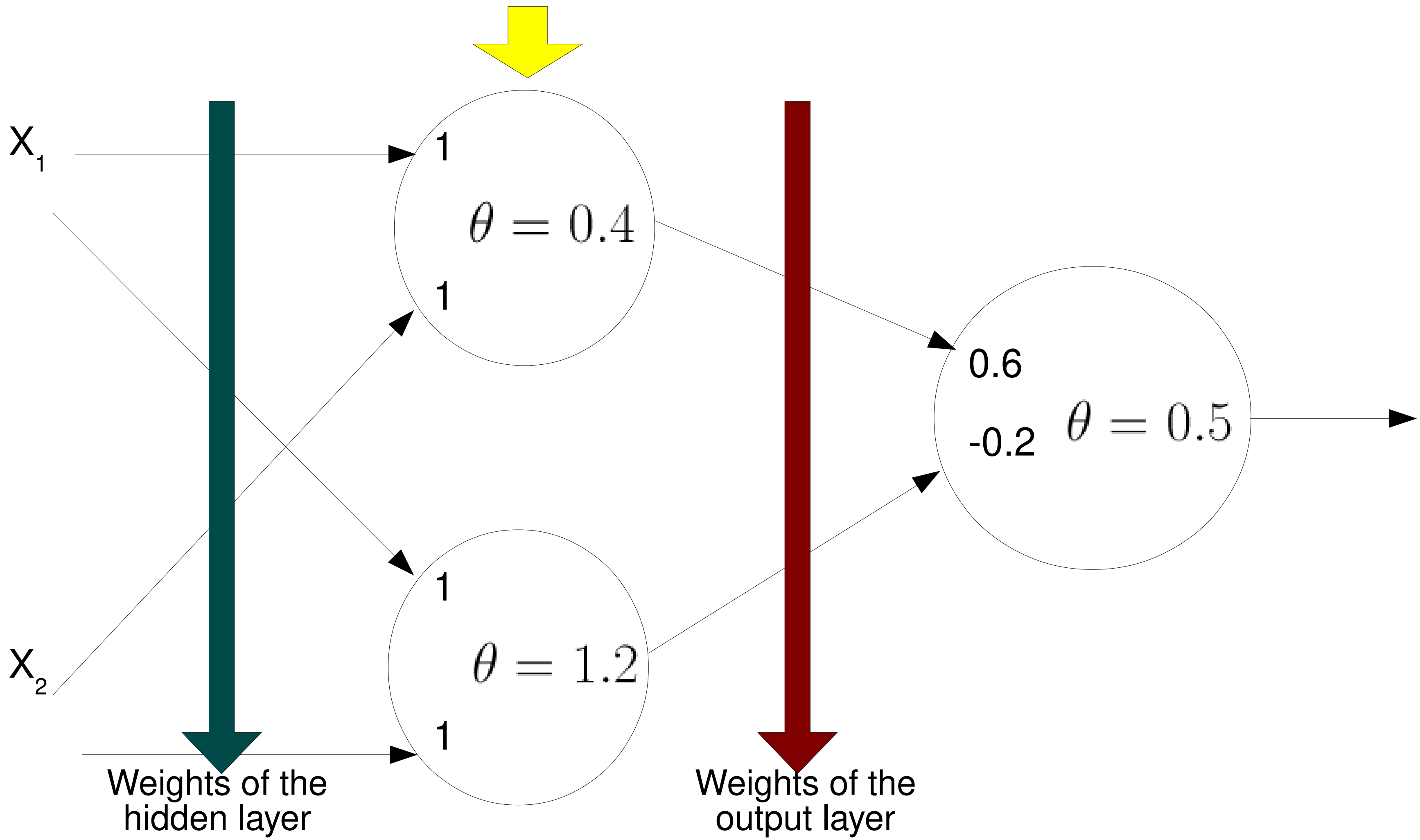
Associate Professor at Universidade de São Paulo, ICMC, Brazil

http://www.icmc.usp.br/~mello

mello@icmc.usp.br

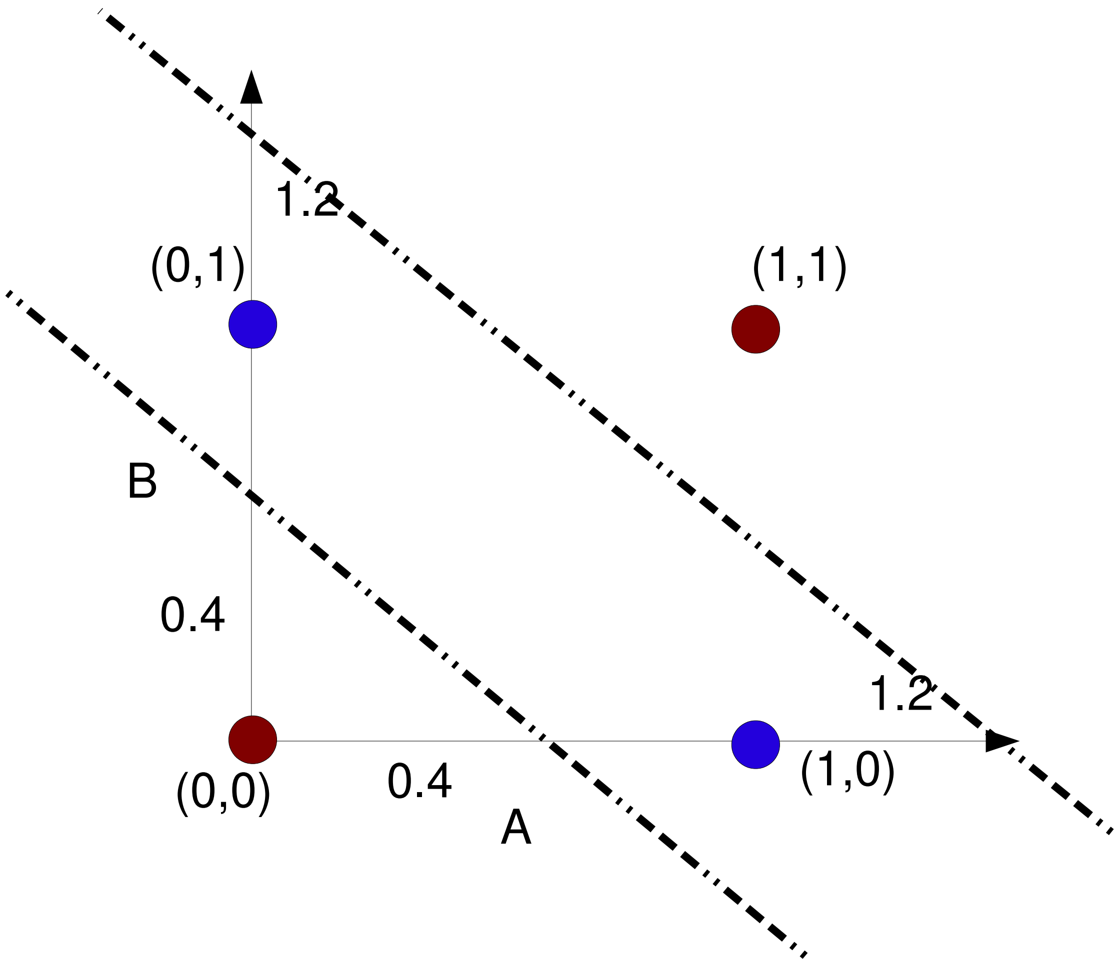
Multilayer Perceptron: Solving XOR

- Implementing XOR
  - Additional layer also called hidden layer → Multilayer Perceptron (MLP)



Multilayer Perceptron: Solving XOR

- Implementing XOR
  - Additional layer also called hidden layer
  - This result was produced by the parameters in the previous slide

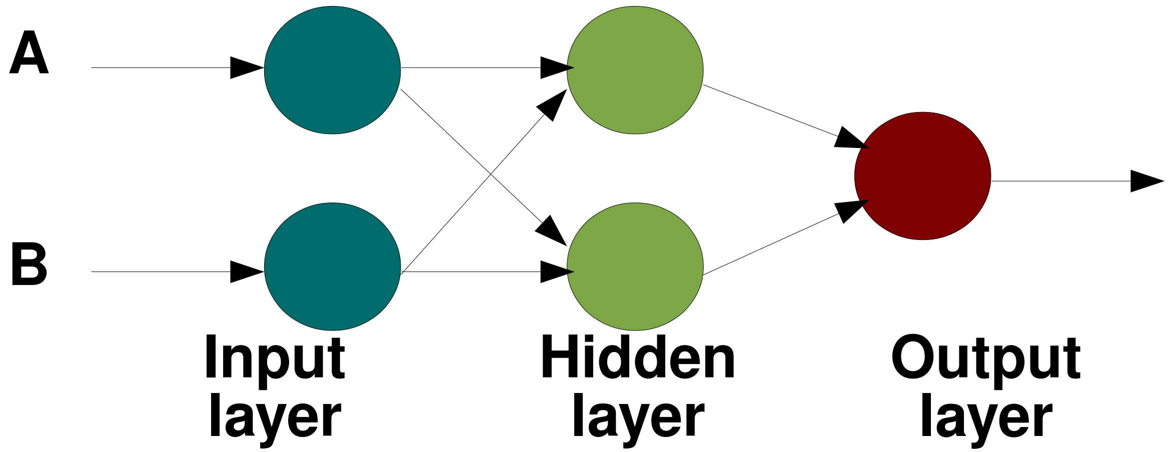


Multilayer Perceptron: Solving XOR

- Implementing XOR
  - However there is a new problem
  - How can we train this network?
    - Training means to update weights to represent input vectors and expected outputs
    - We should find a way to train weights for both layers: hidden and output ones

Multilayer Perceptron: Solving XOR

- Implementing XOR
  - Topology
    - Input length: 2 bits
    - Output length: 1 bit
    - Number of neurons in the output layer: 1
    - Number of neurons in the input layer = Input length, i.e., 2
    - Number of neurons in the hidden layer may vary



XOR Truth Table

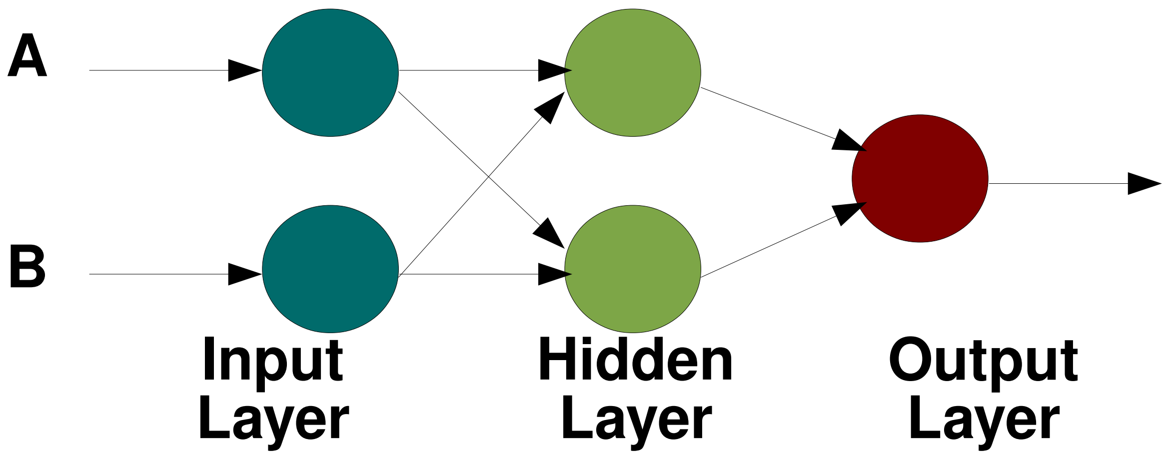
Input		Output
A	B	
0	0	0
0	1	1
1	0	1
1	1	0

Multilayer Perceptron: Solving XOR

- Feedforward network
  - Inputs produce outputs
  - There is no recurrence such as for BAM and Hopfield neural networks

XOR Truth Table

Input		Output
A	B	
0	0	0
0	1	1
1	0	1
1	1	0



- So we can use the Backpropagation algorithm to train MLP
  - Error is propagated from the last layer in the direction of the first and used to update weights

Multilayer Perceptron: Generalized Delta Rule

- Training (weight adaptation) occurs using the error measured at the output layer
- Learning follows the Generalized Delta Rule (GDR)
  - It is a generalization of LMS (Least Mean Squares), seen previously
  - LMS is used for linear regression (separates the space using linear functions)
    - This GDR allows non-linear regression
- Suppose:
  - Pairs of vectors (input, expected output):

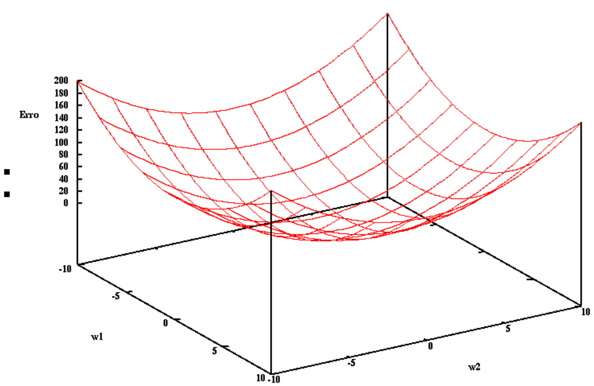
$$(\mathbf{x}_1, \mathbf{y}_1), (\mathbf{x}_2, \mathbf{y}_2), \ldots, (\mathbf{x}_p, \mathbf{y}_p)$$

  - Given:

$$\mathbf{y} = \phi(\mathbf{x}) : \mathbf{x} \in \mathbb{R}^N, \mathbf{y} \in \mathbb{R}^M$$

  - The learning objective is to obtain an approximation:

$$\bar{\mathbf{y}} = \bar{\phi}(\mathbf{x})$$



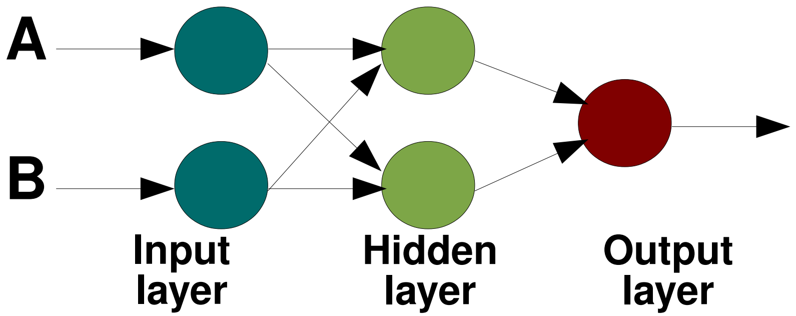
Multilayer Perceptron: Generalized Delta Rule

- The input layer is simple:
  - Neurons only forward values to the hidden layer
- The hidden layer computes:

$$\mathbf{net}^h_{pj} = \sum_{i=1}^{N} w^h_{ji} x_{pi} + \theta^h_j$$

- The output layer computes:

$$\mathbf{net}^o_{pk} = \sum_{j=1}^{L} w_{kj}^{o} i_{pj} + \theta^o_k$$



- In which:

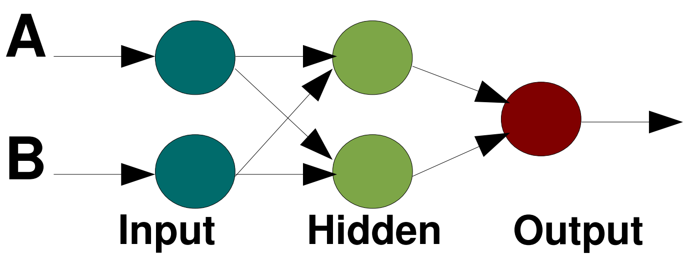
w^h_{ji} é o peso da conexão com o neurônio de entrada i

w^o_{kj} é o peso da conexão com o neurônio j da camada escondida

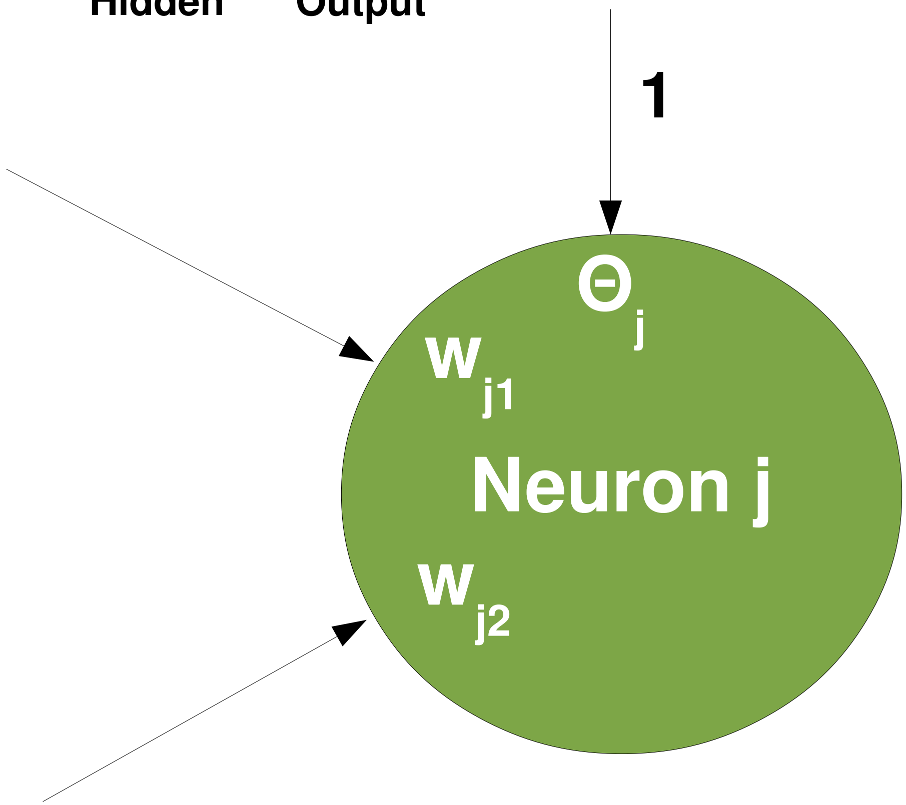
θ^h_j e θ^o_k são os bias

Multilayer Perceptron: Generalized Delta Rule

- For example:
  - Consider one neuron at the hidden layer



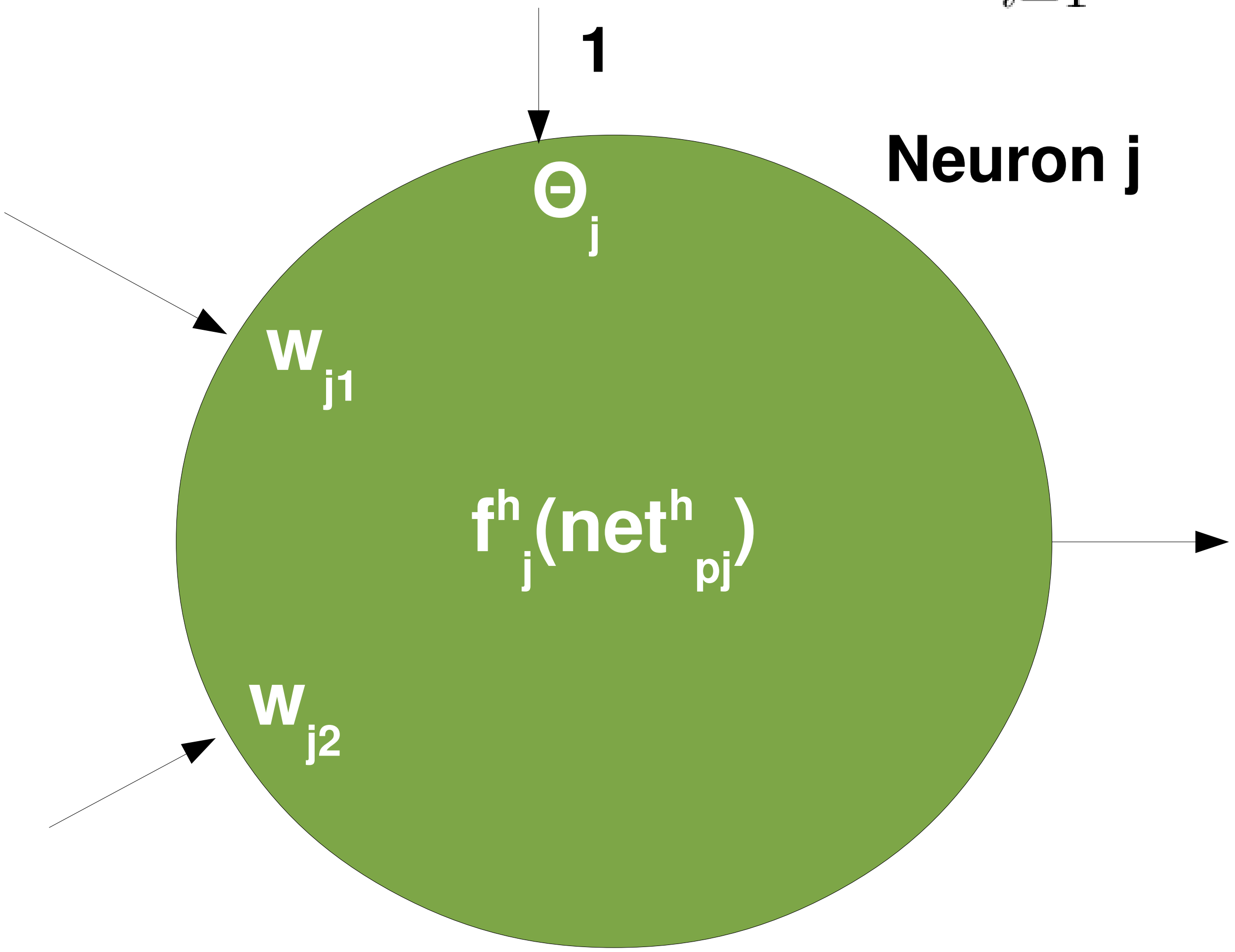
$$\mathbf{net}_{pj}^{h} = \sum_{i=1}^{N} w_{ji}^{h} x_{pi} + \theta_{j}^{h}$$



Multilayer Perceptron: Generalized Delta Rule

- More details...

$$\mathbf{net}_{pj}^{h} = \sum_{i=1}^{N} w_{ji}^{h} x_{pi} + \theta_j^h$$



- **Updating weights at the output layer**
- The output layer may contain several neurons
  - The error for a given neuron at this layer is given by:

$$\delta_{pk} = (y_{pk} - o_{pk})$$

  - Having:

y_{pk} saída esperada do neurônio k para vetor de entrada p

o_{pk} saída produzida pelo neurônio k para vetor de entrada p

p identifica o vetor de entrada usado no treinamento

k indica o neurônio da camada de saída

Multilayer Perceptron: Generalized Delta Rule
Updating weights at the output layer

- The objective is to minimize the squared sum of errors to all output units, considering an input p

$$\mathbf{E}_p = \frac{1}{2}\sum_{k=1}^{M} \delta_{pk}^2$$

  - The factor ½ is added to simplify the derivative
    - As we will have another constant to adapt weights, this term ½ does not change anything in terms of concepts, only the step
  - **M** indicates the number of neurons in the output layer
  - Important:
    - The error associated to each input is squared

- Our objective is to “walk” in the direction to reduce the error, which varies according to weights w

$$\mathbf{E}_p = \frac{1}{2}\sum_{k=1}^{M}\delta_{pk}^2$$

$$\delta_{pk} = (y_{pk} - o_{pk})$$

$$\mathbf{E}_p = \frac{1}{2}\sum_{k=1}^{M}(y_{pk} - o_{pk})^2$$

Multilayer Perceptron: Generalized Delta Rule
Updating weights at the output layer

- Deriving the error in the direction of what can be changed (weights w), for that we have (according to the chain rule):

$$\frac{\partial}{\partial x} f(g(x)) = f'(g(x))g'(x) \text{ ou } \frac{\partial y}{\partial x} = \frac{\partial y}{\partial u} \cdot \frac{\partial u}{\partial x}$$

- Simplifying:

$$E_{pk} = \frac{1}{2}(y_{pk} - o_{pk})^2 \text{ em que:}$$
$$f(g(x)) = \frac{1}{2}(y_{pk} - o_{pk})^2$$
$$g(x) = y_{pk} - o_{pk}$$

Multilayer Perceptron: Generalized Delta Rule
Updating weights at the output layer

- Thus:

$$f'(g(x)) = 2 \cdot \frac{1}{2}(y_{pk} - o_{pk})$$
$$g'(x) = 0 - o'_{pk}$$
$$\text{em que } y_{pk} \text{ é uma constante (saída esperada)}$$

- in which:

$$o_{pk} = f_k^o(\mathbf{net}_{pk}^o)$$

- Therefore the derivative will be (also following the chain rule):

$$o'_{pk} = \frac{\partial f_k^o}{\partial \mathbf{net}_{pk}^o} \cdot \frac{\partial \mathbf{net}_{pk}^o}{\partial w_{kj}^o}$$

- Unifying:

$$f'(g(x))g'(x) = \left(2 \cdot \frac{1}{2}(y_{pk} - o_{pk})\right) \cdot \left(0 - \frac{\partial f_k^o}{\partial \mathbf{net}_{pk}^o} \frac{\partial \mathbf{net}_{pk}^o}{\partial w_{kj}^o}\right)$$

- We have:

$$\frac{\partial E_{pk}}{\partial w_{kj}^o} = -(y_{pk} - o_{pk}) \frac{\partial f_k^o}{\partial \mathbf{net}_{pk}^o} \frac{\partial \mathbf{net}_{pk}^o}{\partial w_{kj}^o}$$

- Solving the last term we find:

$$\mathbf{net} = \sum_{j=1}^{L} w_{kj}^o i_{pj} + \theta_k^o$$

$$\frac{\partial \mathbf{net}_{pk}^o}{\partial w_{kj}^o} = \frac{\partial}{\partial w_{kj}^o} \left(\sum_{j=1}^{L} w_{kj}^o i_{pj} + \theta_k^o\right) = i_{pj}$$

Multilayer Perceptron: Generalized Delta Rule
Updating weights at the output layer

- Substituting:

$$\frac{\partial E_{pk}}{\partial w^o_{pj}} = -(y_{pk} - o_{pk})\frac{\delta f^o_k}{\partial \mathbf{net}^o_{pk}} i_{pj}$$

- We still have to differentiate the activation function
  - So this function MUST be differentiable
  - This avoids the usage of the step function (Perceptron)
- Examples of activation functions:

$$1) \text{ se } f^o_k(\mathbf{net}^o_k) = \mathbf{net}^o_k \text{ então } f'^o_k(\mathbf{net}^o_k) = 1$$
$$\text{assim como } f(x) = x \text{ temos } f'(x) = 1$$

Linear Function

$$2) \text{ se } f^o_k(\mathbf{net}^o_k) = (1 + e^{-\mathbf{net}^o_{jk}})^{-1} \text{ então}$$
$$f'^o_k(\mathbf{net}^o_k) = f^o_k(1 - f^o_k)$$

Sigmoid Function

- Considering the two possibilities for the activation function, we have the weight adaptation as follows:
  - For the **linear function**:

$$f_k^o(\mathbf{net}_{jk}^o) = \mathbf{net}_{jk}^o$$

  - We have:

$$w_{kj}^o(t+1) = w_{kj}^o(t) + \eta(y_{pk} - o_{pk})i_{pj}$$

  - For the **sigmoid function**:

$$f_k^o(\mathbf{net}_{jk}^o) = \frac{1}{1 + e^{-\mathbf{net}_{jk}^o}}$$

  - We have:

$$f_k'^o(\mathbf{net}_k^o) = f_k^o(1 - f_k^o) \text{ logo, neste cenário } f_k'^o(\mathbf{net}_k^o) = o_{pk}(1 - o_{pk})$$

$$w_{kj}^o(t+1) = w_{kj}^o(t) + \eta(y_{pk} - o_{pk})o_{pk}(1 - o_{pk})i_{pj}$$

Multilayer Perceptron: Generalized Delta Rule
Updating weights at the output layer

- We can define the adaptation term in a generic way, i.e., for any activation function:

$$\delta^o_{pk} = (y_{pk} - o_{pk}) f^{o\prime}_k(\mathbf{net}^o_{pk})$$

- And generalize (**Generalized Delta Rule**) the weight adaptation for any activation function, as follows:

$$w^o_{kj}(t+1) = w^o_{kj}(t) + \eta \delta^o_{pk} i_{pj}$$

- **Updating hidden layer weights**
  - How can we know the expected output for each neuron at the hidden layer?
    - At the output layer we know what is expected!
  - In some way, error E_p measured at the output layer must influence in the hidden layer weights

Multilayer Perceptron: Generalized Delta Rule
Updating weights at the hidden layer

- The error measured at the output layer is given by:

$$
\begin{aligned}
E_p &= \frac{1}{2}\sum_k (y_{pk} - o_{pk})^2 \\
&= \frac{1}{2}\sum_k (y_{pk} - f_k^o(\mathbf{net}_{pk}^o))^2 \\
&= \frac{1}{2}\sum_k \left(y_{pk} - f_k^o\left(\sum_j w_{kj}^o i_{pj} + \theta_k^o\right)\right)^2
\end{aligned}
$$

- Term i_{pj} refers to the values produced by the previous (hidden) layer
  - So we can explore this fact to build equations to adapt hidden layer weights

Multilayer Perceptron: Generalized Delta Rule
Updating weights at the hidden layer

- In this manner, we define the error variation in terms of hidden layer weights:

$$\frac{\partial E_p}{\partial w^h_{ji}} = \frac{1}{2}\sum_k \frac{\partial}{\partial w^h_{ji}}(y_{pk} - o_{pk})^2$$

$$= -\sum_k (y_{pk} - o_{pk}) \frac{\partial o_{pk}}{\partial \mathbf{net}^o_{pk}} \frac{\partial \mathbf{net}^o_{pk}}{\partial i_{pj}} \frac{\partial i_{pj}}{\partial \mathbf{net}^h_{pj}} \frac{\partial \mathbf{net}^h_{pj}}{\partial w^h_{ji}}$$

- From that we obtain:

$$\frac{\partial E_p}{\partial w^h_{ji}} = -\sum_k (y_{pk} - o_{pk}) f^{o\prime}_k(\mathbf{net}^o_{pk}) w^o_{kj} f^{h\prime}_j(\mathbf{net}^h_{pj}) x_{pi}$$

Multilayer Perceptron: Generalized Delta Rule
Updating weights at the hidden layer

- In this manner, we define the error variation in terms of hidden layer weights:

$$\frac{\partial E_p}{\partial w^h_{ji}} = \frac{1}{2}\sum_k \frac{\partial}{\partial w^h_{ji}}(y_{pk} - o_{pk})^2$$

$$= -\sum_k (y_{pk} - o_{pk}) \boxed{\frac{\partial o_{pk}}{\partial \mathbf{net}^o_{pk}}} \frac{\partial \mathbf{net}^o_{pk}}{\partial i_{pj}} \boxed{\frac{\partial i_{pj}}{\partial \mathbf{net}^h_{pj}}} \frac{\partial \mathbf{net}^h_{pj}}{\partial w^h_{ji}}$$

Derivative of the activation function at the output layer in the direction of net

Derivative of the activation function at the hidden layer in the direction of net

- From that we obtain:

$$\frac{\partial E_p}{\partial w^h_{ji}} = -\sum_k (y_{pk} - o_{pk}) \boxed{f^{o\prime}_k(\mathbf{net}^o_{pk})} w^o_{kj} \boxed{f^{h\prime}_j(\mathbf{net}^h_{pj})} x_{pi}$$

Multilayer Perceptron: Generalized Delta Rule
Updating weights at the hidden layer

- Having:

$$\mathbf{net}^o_{pk} = \sum_{j=1}^{L} w^o_{kj} i_{pj} + \theta^o_k$$

$$\mathbf{net}^h_{pi} = \sum_{j=1}^{L} w^h_{kj} x_{pi} + \theta^h_k$$

$$\begin{aligned}\frac{\partial E_p}{\partial w^h_{ji}} &= \frac{1}{2}\sum_k \frac{\partial}{\partial w^h_{ji}} (y_{pk} - o_{pk})^2 \\ &= -\sum_k (y_{pk} - o_{pk}) \frac{\partial o_{pk}}{\partial \mathbf{net}^o_{pk}} \boxed{\frac{\partial \mathbf{net}^o_{pk}}{\partial i_{pj}}} \frac{\partial i_{pj}}{\partial \mathbf{net}^h_{pj}} \boxed{\frac{\partial \mathbf{net}^h_{pj}}{\partial w^h_{ji}}}\end{aligned}$$

$$\frac{\partial E_p}{\partial w^h_{ji}} = -\sum_k (y_{pk} - o_{pk}) f^{o\prime}_k(\mathbf{net}^o_{pk}) \boxed{w^o_{kj}} f^{h\prime}_j(\mathbf{net}^h_{pj}) \boxed{x_{pi}}$$

Multilayer Perceptron: Generalized Delta Rule
Updating weights at the hidden layer

- So we can compute the weight adaptation for the hidden layer in form:

$$\triangle_p w_{ji}^h = \eta f_j^{h\prime}(\mathbf{net}_{pj}^h) x_{pi} \sum_k (y_{pk} - o_{pk}) f_k^{o\prime}(\mathbf{net}_{pk}^o) w_{kj}^o$$

$$\triangle_p w_{ji}^h = \eta f_j^{h\prime}(\mathbf{net}_{pj}^h) x_{pi} \sum_k \delta_{pk}^o w_{kj}^o$$

- In this manner, the weight adaptation at the hidden layer depends on the error at the output layer
  - The term **Backpropagation** comes from this notion of dependency on the error at the output layer

Multilayer Perceptron: Generalized Delta Rule
Updating weights at the hidden layer

- So we can compute the term delta for the hidden layer in the same way we did for the output layer:

$$\delta^h_{pj} = f^{h\prime}_j(\mathbf{net}^h_{pj}) \sum_k \delta^o_{pk} w^o_{kj}$$

- In this way, the weight adaptation is given by:

$$w^h_{ji}(t+1) = w^h_{ji}(t) + \eta \delta^h_{pj} x_i$$

- Finally, we can implement the MLP learning algorithm

Multilayer Perceptron

- **Backpropagation learning algorithm**
- Essential functions to update weights:
  - Considering the sigmoid activation function (this is the default):

$$f(x) = \frac{1}{1+e^{-x}} \qquad f'(x) = f(x)\cdot(1-f(x))$$

- Output layer:

$$\delta^o_{pk} = (y_{pk} - o_{pk}) f^{o\prime}_k(\mathbf{net}^o_{pk})$$

$$w^o_{kj}(t+1) = w^o_{kj}(t) + \eta\delta^o_{pk} i_{pj}$$

- Hidden layer:

$$\delta^h_{pj} = f^{h\prime}_j(\mathbf{net}^h_{pj}) \sum_k \delta^o_{pk} w^o_{kj}$$

$$w^h_{ji}(t+1) = w^h_{ji}(t) + \eta\delta^h_{pj} x_i$$

This is very important!!!

First, we MUST compute all deltas so then we update weights!!!

- **Backpropagation learning algorithm**
- Essential functions to update weights:
  - Considering the sigmoid activation function (this is the default):

$$f(x) = \frac{1}{1+e^{-x}} \qquad f'(x) = f(x) \cdot (1 - f(x))$$

- Output layer:

k identifies the neuron

$$\delta^o_{pk} = (y_{pk} - o_{pk}) f^{o\prime}_k(\mathbf{net}^o_{pk})$$

$$w^o_{kj}(t+1) = w^o_{kj}(t) + \eta \delta^o_{pk} i_{pj}$$

- Hidden layer:

$$\delta^h_{pj} = f^{h\prime}_j(\mathbf{net}^h_{pj}) \sum_k \delta^o_{pk} w^o_{kj}$$

$$w^h_{ji}(t+1) = w^h_{ji}(t) + \eta \delta^h_{pj} x_i$$

- **Backpropagation learning algorithm**
- Essential functions to update weights:
  - Considering the sigmoid activation function (this is the default):

$$f(x) = \frac{1}{1+e^{-x}} \qquad f'(x) = f(x) \cdot (1 - f(x))$$

This refers to the layer (hidden or output)

- Output layer:

$$\delta^o_{pk} = (y_{pk} - o_{pk}) f^{o\prime}_k(\mathbf{net}^o_{pk})$$

$$w^o_{kj}(t+1) = w^o_{kj}(t) + \eta \delta^o_{pk} i_{pj}$$

- Hidden layer:

$$\delta^h_{pj} = f^{h\prime}_j(\mathbf{net}^h_{pj}) \sum_k \delta^o_{pk} w^o_{kj}$$

$$w^h_{ji}(t+1) = w^h_{ji}(t) + \eta \delta^h_{pj} x_i$$

Multilayer Perceptron

- **Backpropagation learning algorithm**
- Essential functions to update weights:
  - Considering the sigmoid activation function (this is the default):

$$f(x) = \frac{1}{1+e^{-x}} \qquad f'(x) = f(x) \cdot (1 - f(x))$$

- Output layer:

p identifies the input vector

$$\delta_{pk}^{o} = (y_{pk} - o_{pk}) f_k^{o\prime}(\mathbf{net}_{pk}^{o})$$

$$w_{kj}^{o}(t+1) = w_{kj}^{o}(t) + \eta \delta_{pk}^{o} i_{pj}$$

- Hidden layer:

$$\delta_{pj}^{h} = f_j^{h\prime}(\mathbf{net}_{pj}^{h}) \sum_k \delta_{pk}^{o} w_{kj}^{o}$$

$$w_{ji}^{h}(t+1) = w_{ji}^{h}(t) + \eta \delta_{pj}^{h} x_i$$

Multilayer Perceptron

- **Backpropagation learning algorithm**
- Essential functions to update weights:
  - Considering the sigmoid activation function (this is the default):

$$f(x) = \frac{1}{1+e^{-x}} \qquad f'(x) = f(x) \cdot (1 - f(x))$$

- Output layer:

Input values produced by the hidden layer

$$\delta^o_{pk} = (y_{pk} - o_{pk}) f^{o\prime}_k(\mathbf{net}^o_{pk})$$

$$w^o_{kj}(t+1) = w^o_{kj}(t) + \eta \delta^o_{pk} i_{pj}$$

- Hidden layer:

Input values produced by the input layer

$$\delta^h_{pj} = f^{h\prime}_j(\mathbf{net}^h_{pj}) \sum_k \delta^o_{pk} w^o_{kj}$$

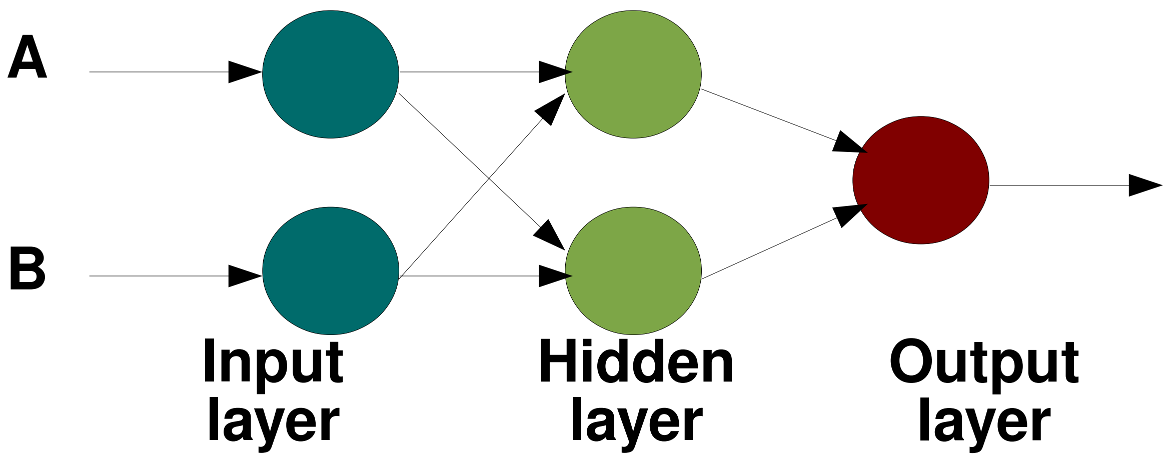
$$w^h_{ji}(t+1) = w^h_{ji}(t) + \eta \delta^h_{pj} x_i$$

Multilayer Perceptron

- Implementing
  - XOR

XOR Truth Table

Input		Output
A	B	
0	0	0
0	1	1
1	0	1
1	1	0

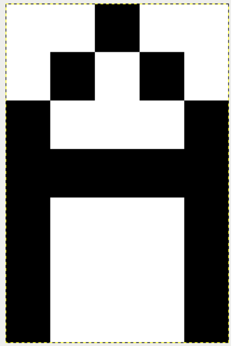


Multilayer Perceptron

- Implementing
  - OCR (Optical Character Recognition)
  - Solution:
    - We can build a Hash Table (Bits → ASCII Code)
      - Problem: when one or more bits are random or present noise
        - How can we deal with noise in such scenario?
    - MLP
      - Capable of learning and generalize knowledge
        - Even in the presence of noise, it will produce good results

Multilayer Perceptron

- Implementing
  - OCR (Optical Character Recognition)
  - Input is given by a 7x5 matrix
    - Example (separate letter A from letter B)



  - Extend this example to separate any character
  - Extend this example for any classification problem

Multilayer Perceptron

- Implementing
  - Face recognition
  - Song classification according to genre

- Consider other datasets:
  - UCI (http://archive.ics.uci.edu/ml/datasets.html):
    - Iris
    - Reuters-21578 Text Categorization Collection
    - CMU Face Images
  - Million Song Dataset
    - http://labrosa.ee.columbia.edu/millionsong/pages/getting-dataset