Advanced Machine Learning

Bilel GUETARNI, PhD

bilel.guetarni@junia.com

JUNIA ISEN

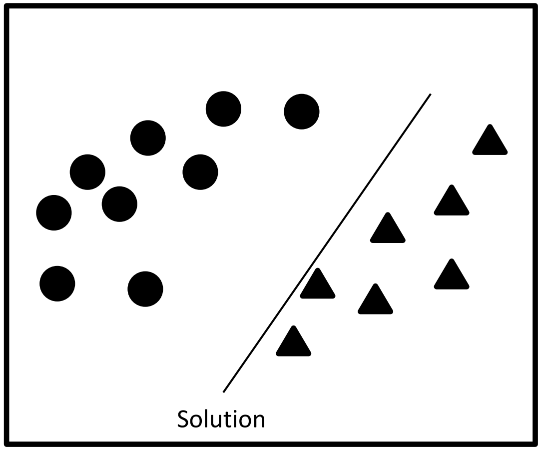
PowerPoint slides from **Halim Benhabiles, PhD** and **Farès BOUGOURZI, PhD**

How to learn with a linear perceptron?

- Training stage ➔ tuning the weights until the desired behavior is reached
- Pay attention to the learning rate parameter η
    - Too big - risk of non convergence
    - Too small - many iterations

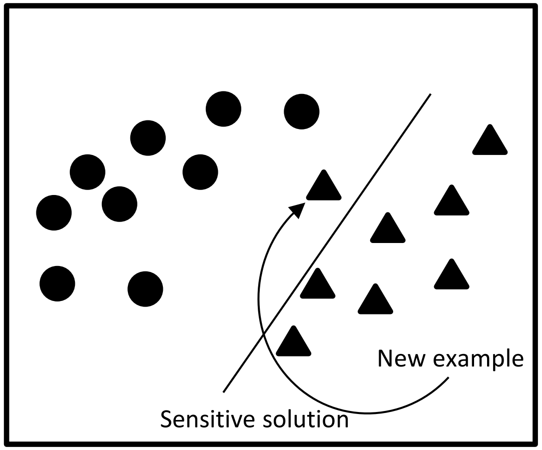
Properties of the linear perceptron

- The final solution (values of weights) depends
  - Initial weights
  - Training learning rate
  - The organization of the training base (appearance order)



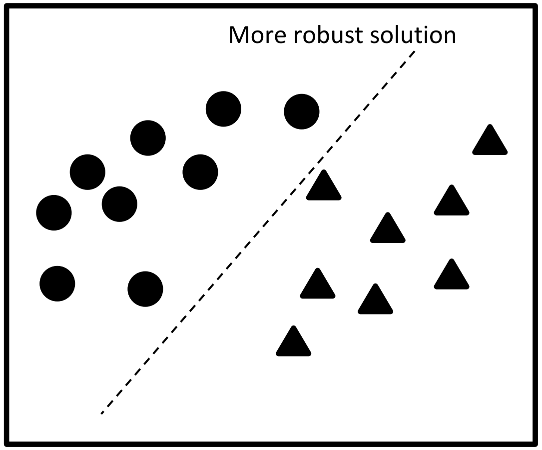
Properties of the linear perceptron

- The final solution (values of weights) depends
  - Initial weights
  - Training learning rate
  - The organization of the training base (appearance order)
- Robustness of solution



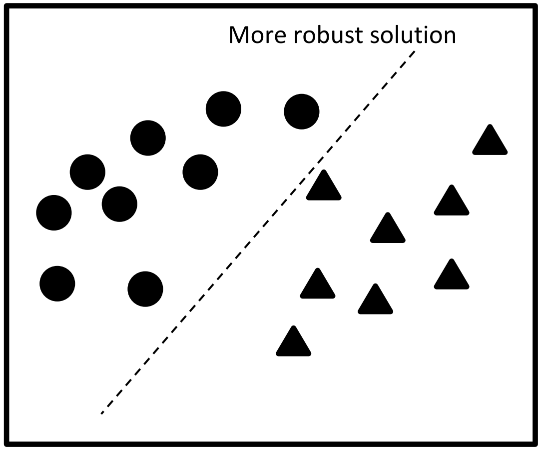
Properties of the linear perceptron

- The final solution (values of weights) depends
  - Initial weights
  - Training learning rate
  - The organization of the training base (appearance order)
- Robustness of solution



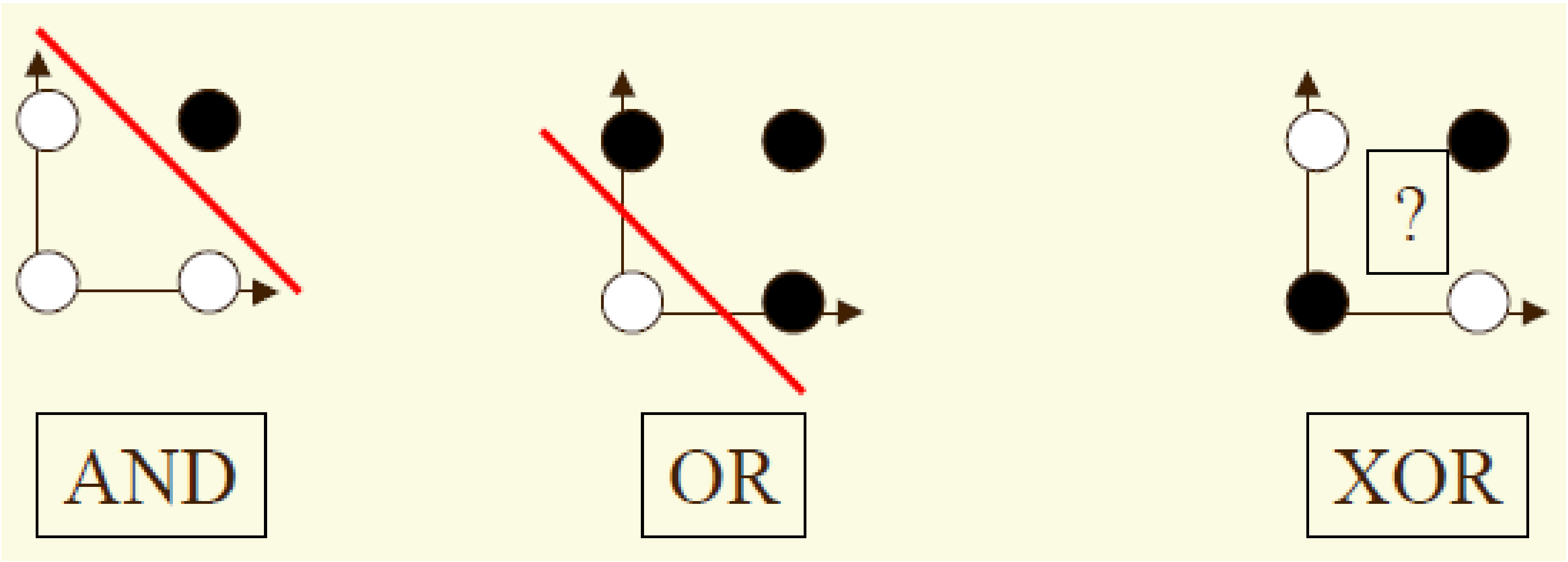
Properties of the linear perceptron

- The final solution (values of weights) depends
  - Initial weights
  - Training learning rate
  - The organization of the training base (appearance order)
- Robustness of solution
- The linear perceptron will not converge if samples are not linearly separable



Limitation of linear perceptron

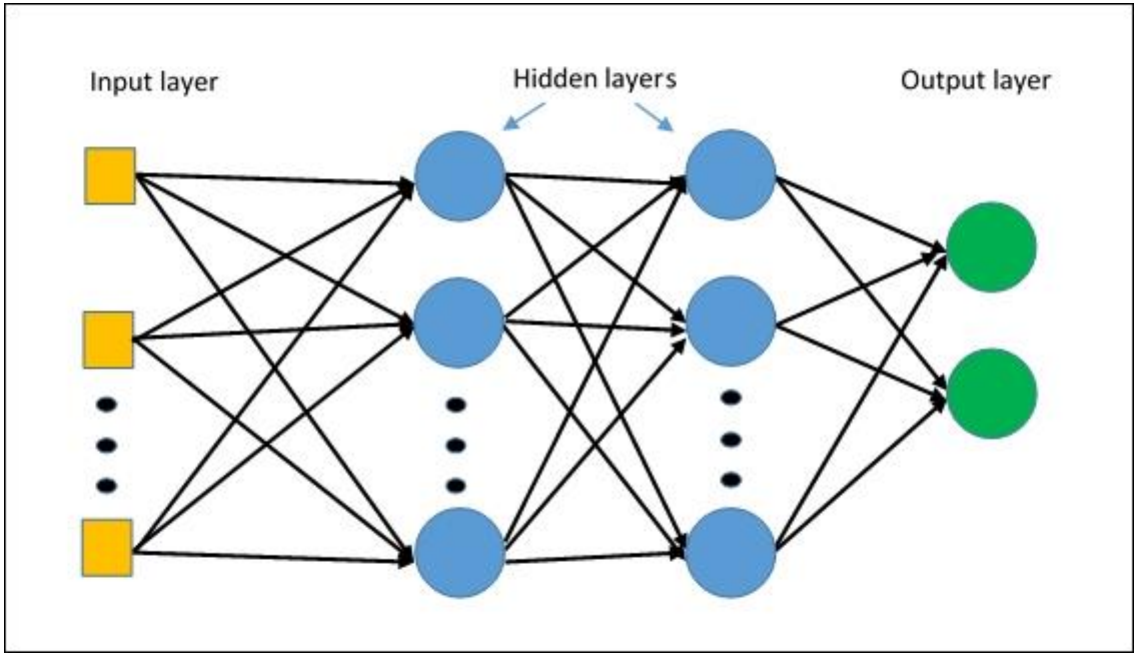
- How to deal with nonlinear data ?



Multi Layer Perceptron
MLP

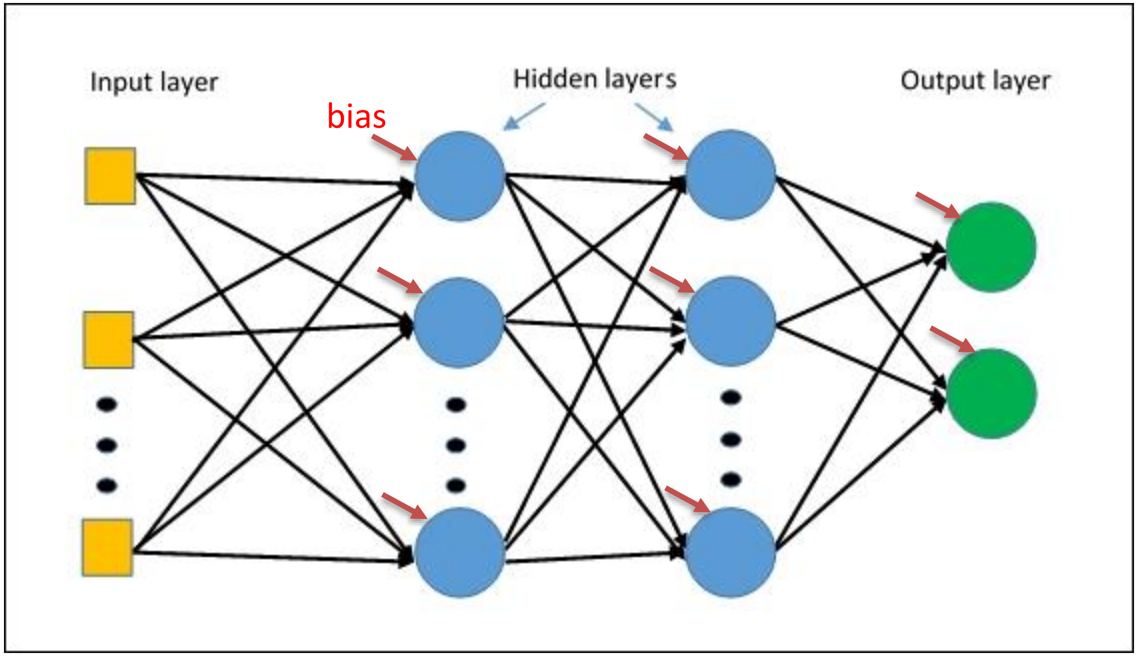
Multi Layer Perceptron MLP

- Multiple layers
  - Neurons distributed over q layers c_0 , c_1,…., c_q
  - c_0 input layer
  - c_q output layer
  - c_1,…., c_{q-1}hidden layers



Multi Layer Perceptron MLP

- Multiple layers
  - Neurons distributed over q layers c_0 , c_1,…., c_q
  - c_0 input layer
  - c_q output layer
  - c_1,…., c_{q-1}hidden layers



Multi Layer Perceptron MLP

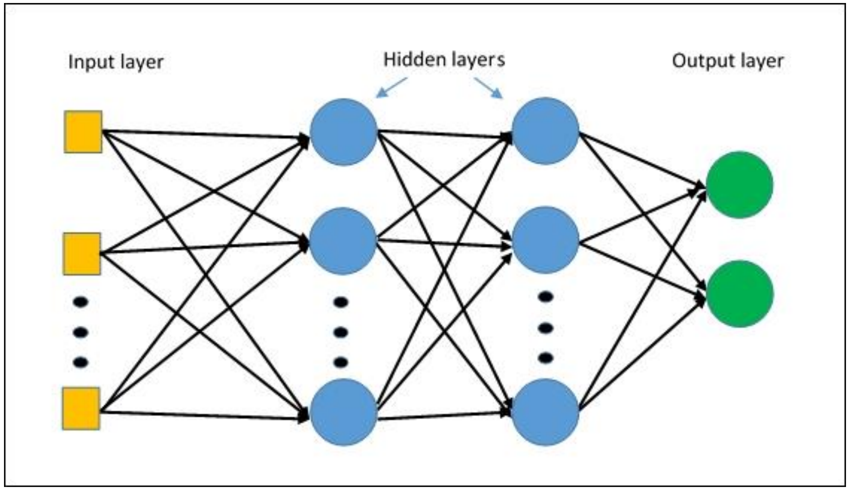
Activation functions

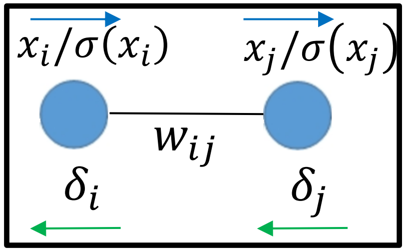
- Sigmoid [0,1]
  - $\sigma(x) = \frac{1}{1+\ e^{-x}}$
  - $\sigma'(x) = \sigma(x) * (1 - \sigma(x))$
- Hyperbolic tangent [-1,1]
  - $\sigma(x) = \tanh(x)$
  - $\sigma'(x) = 1 - \sigma^2(x)$
- Relu
  Rectified linear unit
  - $\sigma(x) = \max(x, 0)$
  - $\sigma'(x) = 1$

Multi Layer Perceptron MLP algorithm (forward backward propagation) – Stochastic Gradient Descent

One Epoch

- Set randomly weights w_{ij} in [-0.5, 0.5]
- Repeat
  - Take an example (X,c) from S
  - Calculate $\sigma(x_i)$ for each hidden neurone i
  - Calculate the outputs o_i
  - For each output neuron i
    - $\delta_i = \underbrace{o_i * (1 - o_i)}_{\sigma'(x) \text{ the derivative of sigmoid log in this example}} * \overbrace{abs(c - o_i)}^{\text{The loss}}$
  - For each hidden layer (q-1 to 1) [backpropagation]
    - For each hidden neuron i of current layer
      - $\delta_i = \underbrace{\sigma(x_i) * (1 - \sigma(x_i))}_{\sigma'(x_i)} * \sum_{j \in succ(i)} \delta_j * w_{ij}$
  - For each weight w_{ij} [update weights based on the gradient]
    - $w_{ij} = w_{ij} - \underbrace{\eta * \delta_j * \sigma(x_i)}_{\text{The gradient descente rule}}$, where $\sigma(x_i) = x_i$ for input layer
- EndRepeat when covering all examples from S
- Calculate the **MSE**
  - If (MSE ≠ 0 and total epochs not reached) then go to Repeat

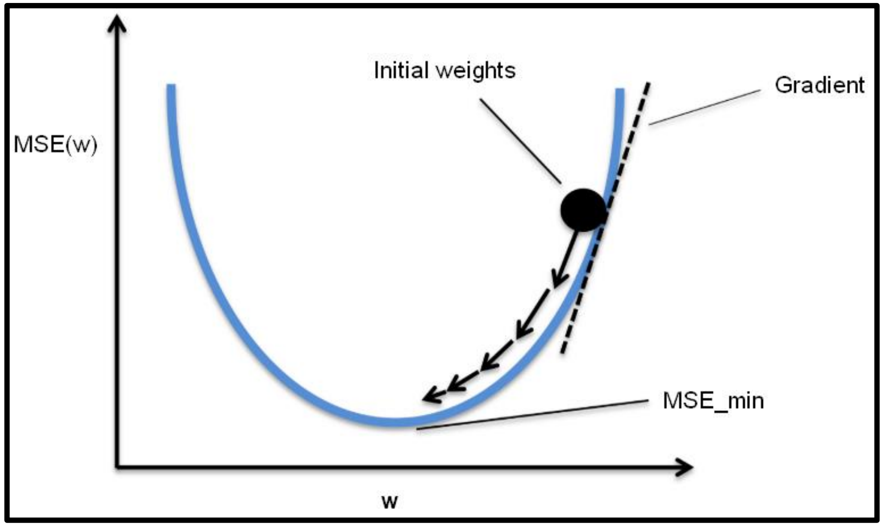




MLP and gradient descent for optimization

- Find the best weights that minimizes the Mean Square Error - ***MSE*** (**<u>Gradient descent method</u>**)

$$\mathrm{MSE}(w) = \frac{1}{n}\sum_{(X,c)\ in\ S}(c - \sigma)^2$$



Lab session

- Implement your MLP algorithm to learn ‘XOR’ function and test it
  - using normal distribution of weights [0, 1]
  - using 3 different architectures (network depth and hidden layer size)
  - using different learning rates [20,2,0.2, 0.02, 0.002]
  - by replacing sigmoid log activation function by Relu
  - by changing the order of the examples

- Provide your own analysis based on the different results