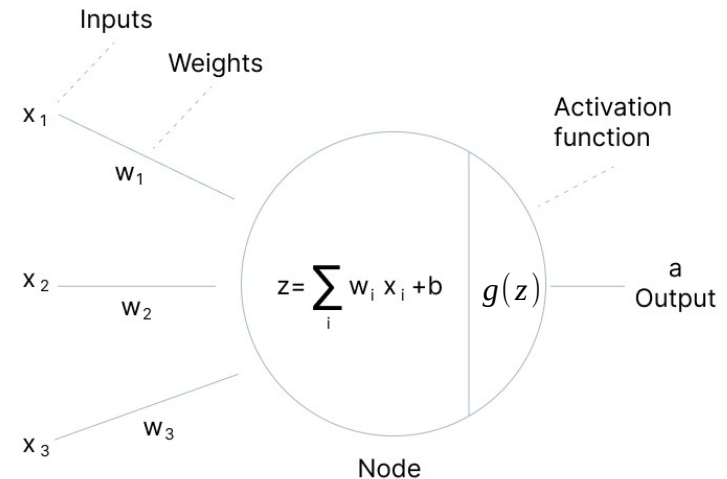


# Exercici 1: Simple Neural Network

Python

# Node behaviour in Forward propagation

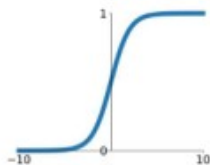
- Multipliquem inputs per pesos
- Els sumem tots
- Apliquem funcio d'activacio  
 $a = g(z) = \text{sigmoid}(z)$
- 'b' es el bias, un valor intern



# Popular 'activation functions'

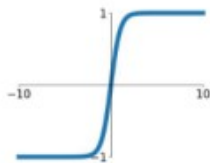
**Sigmoid**

$$\sigma(x) = \frac{1}{1+e^{-x}}$$



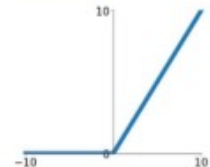
**tanh**

$$\tanh(x)$$



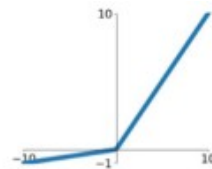
**ReLU**

$$\max(0, x)$$



**Leaky ReLU**

$$\max(0.1x, x)$$

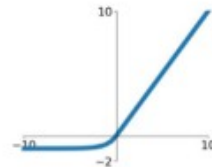


**Maxout**

$$\max(w_1^T x + b_1, w_2^T x + b_2)$$

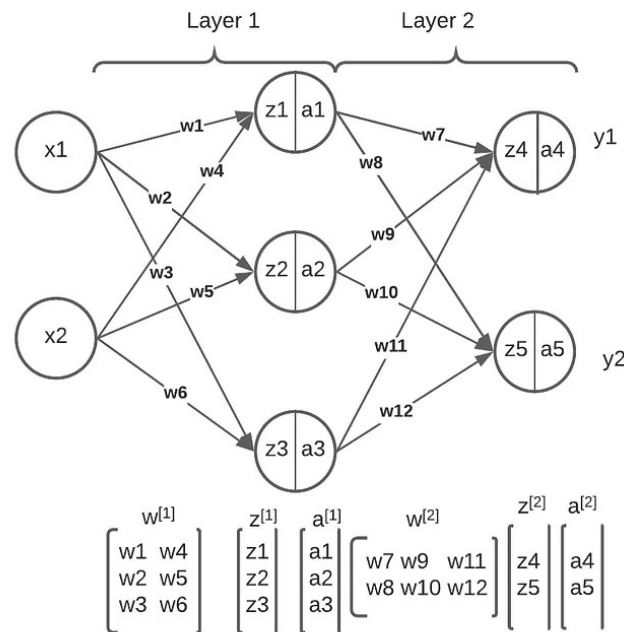
**ELU**

$$\begin{cases} x & x \geq 0 \\ \alpha(e^x - 1) & x < 0 \end{cases}$$

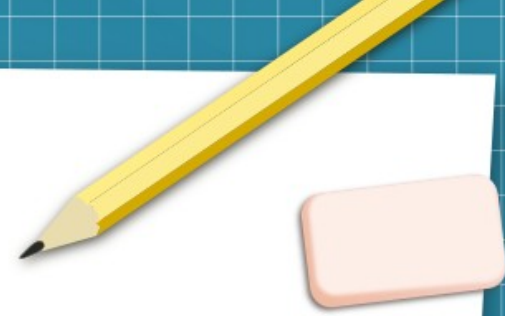


# Network Scheme

- 2 layers
- 2 inputs
- 3 nodes hidden layer
- 2 node output layer
- Aquesta xarxa es un classificador



# Exercici 1: Part 1



- Crea una classe xarxa neuronal
- Crea una instància de la xarxa neuronal amb l'estructura de la diapositiva anterior que en rebre  $[0, 1]$  doni de resultat  $[0.974, 0.974]$  per totes 3 sortides.
- Perquè això passi només caldrà definir tots els pesos com a 1 per defecte.
- Hem d'afegir 1 bias de 1 a totes les neurones

$$\begin{bmatrix} z_1 \\ z_2 \\ z_3 \end{bmatrix} = \begin{bmatrix} w_1 & w_4 \\ w_2 & w_5 \\ w_3 & w_5 \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \end{bmatrix} + \begin{bmatrix} b_1 \\ b_2 \\ b_3 \end{bmatrix}$$

Matrix multiplication of layer 1 operation

$$\begin{aligned} z^{[1]} &= w^{[1]}x + b^{[1]} \\ z^{[2]} &= w^{[2]}a^{[1]} + b^{[2]} \end{aligned}$$

Vectorized expression of layer 1 and 2 operations



# Backward Propagation

- La funció de cost  $J(w)$  evalua com de bo es el resultat.
- Després repartim responsabilitats propagant l'error entre els nodes

$$J(a, y) = -[y \log a + (1 - y) \log(1 - a)]$$

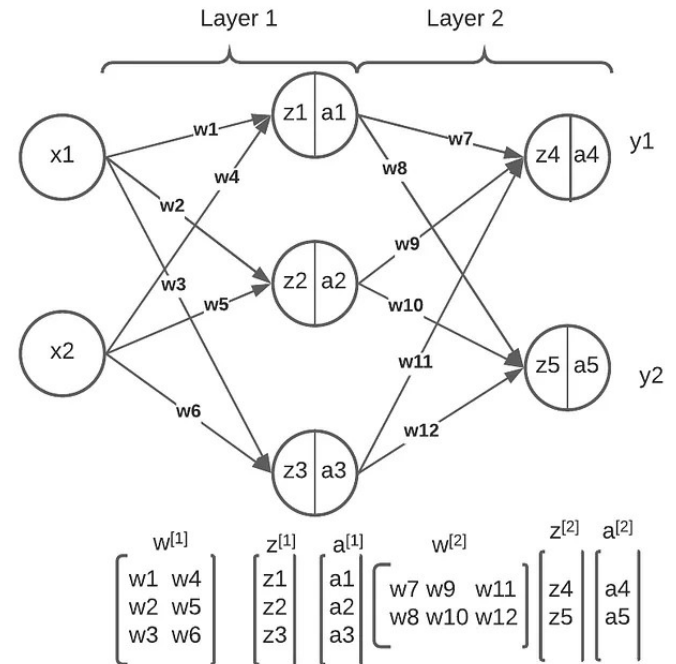
$$\frac{\partial J(a, y)}{\partial a} = \frac{a - y}{a(1 - a)}$$

Loss function and gradient

$$g(z) = \frac{1}{1 + e^{-z}}$$

$$\frac{\partial g(z)}{\partial z} = g(z)(1 - g(z))$$

Activation function and gradient



# Backward Propagation

- $\alpha$  = Learning rate <1. p.e. 0.25
- Calculeu deltes layer 2: son els errors corresponents a un node

$$\delta a^{[2]} = J'(a^{[2]}, y) \rightarrow \delta z^{[2]} = \frac{a^{[2]} - y}{a^{[2]}(1 - a^{[2]})} \odot a^{[2]}(1 - a^{[2]}) = a^{[2]} - y = -E$$

$$\delta z^{[2]} = \delta a^{[2]} \odot g'(z^{[2]})$$

- Actualitzem pesos layer 2

$$\delta w^{[2]} = \delta z^{[2]} \cdot a^{[1]T} \rightarrow w = w - \alpha \delta w$$

$$\delta b^{[2]} = \delta z^{[2]} \rightarrow b = b - \alpha \delta b$$

- Calculeu deltes layer 1

$$\delta a^{[1]} = w^{[2]T} \cdot \delta z^{[2]} \rightarrow \delta z^{[1]} = (w^{[2]T} \cdot \delta z^{[2]}) \odot g'(z^{[1]}) = (w^{[2]T} \cdot \delta z^{[2]}) \odot (a^{[1]}(1 - a^{[1]}))$$

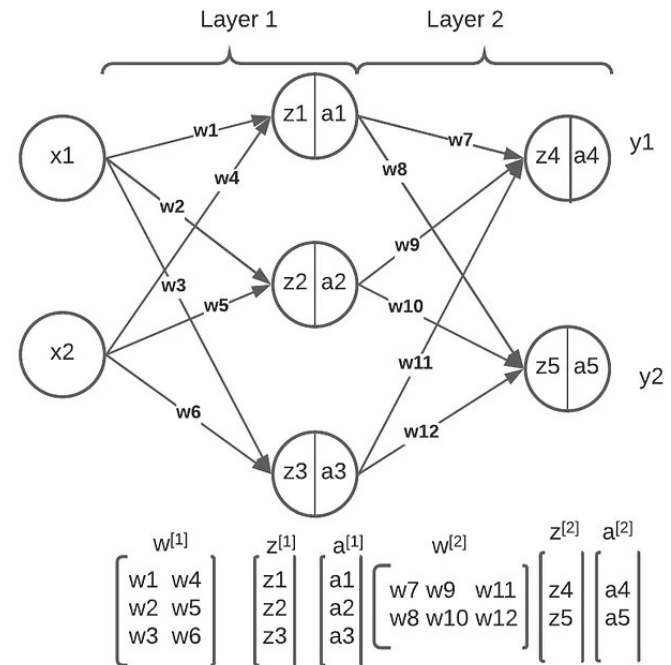
$$\delta z^{[1]} = \delta a^{[1]} \odot g'(z^{[1]})$$

- Actualitzem pesos layer 1

$$\delta w^{[1]} = \delta z^{[1]} \cdot x^T \rightarrow w = w - \alpha \delta w$$

$$\delta b^{[1]} = \delta z^{[1]} \rightarrow b = b - \alpha \delta b$$

- Iterem...



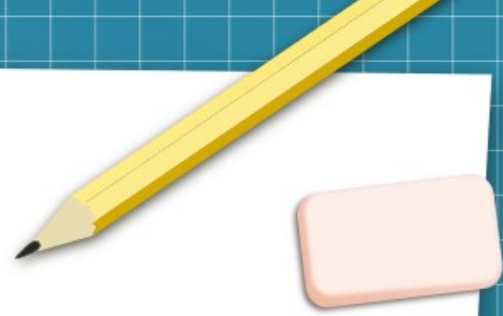
# Backward Propagation - notes



- Vigileu amb els mínims locals
- És recomanable fer servir Numpy
  - `numpy.dot`
  - `numpy.outer`
  - `numpy.multiply`
  - `numpy.matmul`
  - `Numpy.exp`
- L'ordre de magnitud de les iteracions està als milers (~5000 epochs)



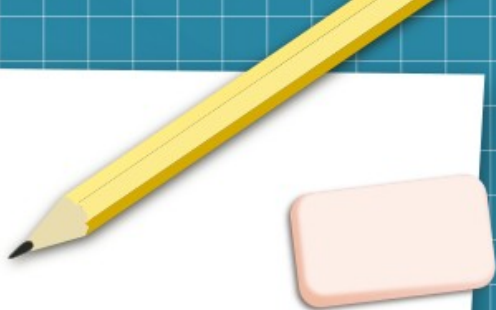
## Exercici 1: Part 2



- Afegiu la metodologia d'aprenentatge 'Back propagation' a la vostra xarxa i entreneula per que, per  $[0,1]$  doni de resultat  $[0, 1]$  per les 2 sortides.

# Exercici 1: Part 3

- Entreneu la xarxa per que una sortida es comporti com una porta lògica XOR i l'altre com una AND.



X1	X2	XOR	AND
0	0	0	0
0	1	1	0
1	0	1	0
1	1	0	1



This work is licensed under a Creative Commons  
Attribution-ShareAlike 3.0 Unported License.  
It makes use of the works of Mateus Machado Luna.  
Credit for to **Y. Lin** for diagrams and math demonstration

