TALLER DE DEEP LEARNING

Lectura 2:
Conceptos básicos de redes neuronales

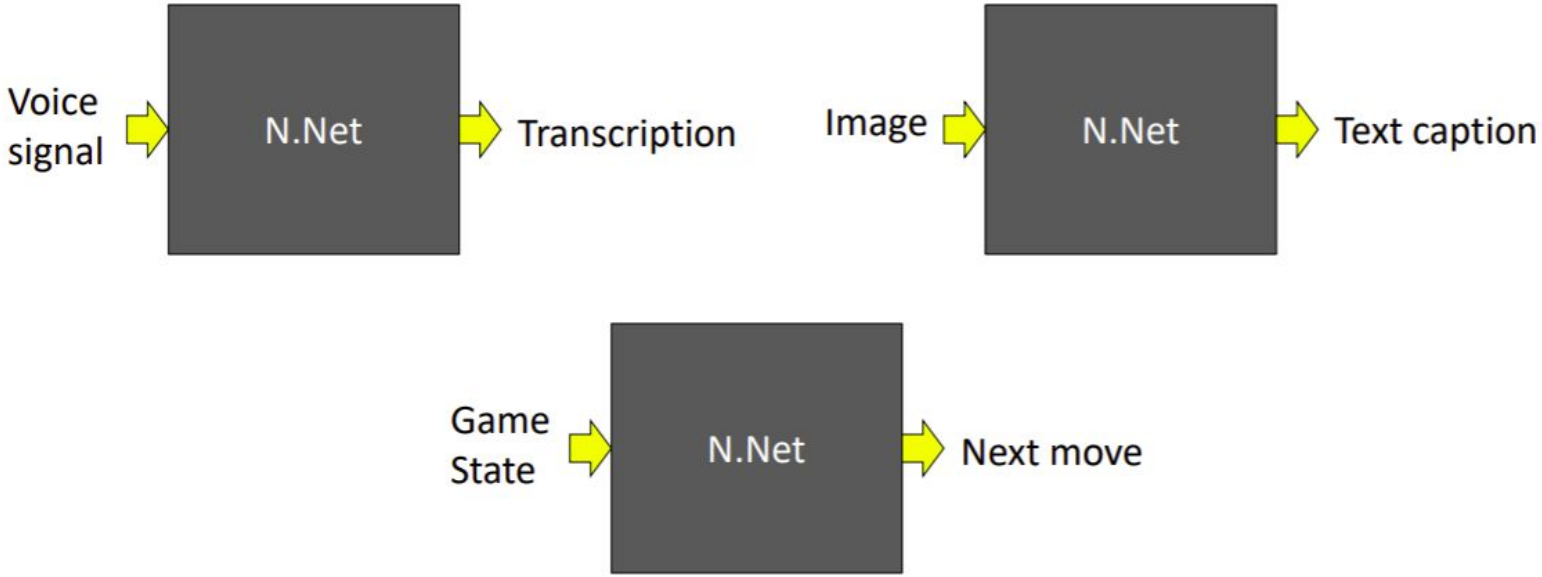
Dennis Núñez Fernández

Neural nets and the employment market

This guy didn't know about neural networks (a.k.a deep learning)

This guy learned about neural networks (a.k.a deep learning)

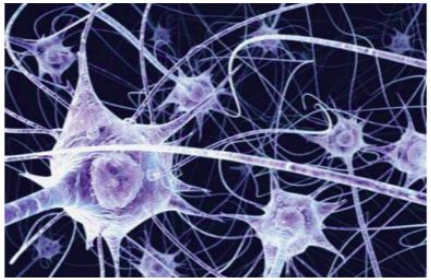
Redes Neuronales



Connectionism lives on..

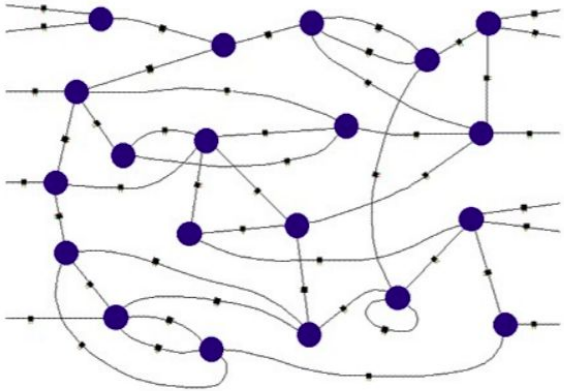
- The human brain is a connectionist machine
  - Bain, A. (1873). *Mind and body. The theories of their relation.* London: Henry King.
  - Ferrier, D. (1876). *The Functions of the Brain.* London: Smith, Elder and Co

- Neurons connect to other neurons. The processing/capacity of the brain is a function of these connections



- Connectionist machines emulate this structure

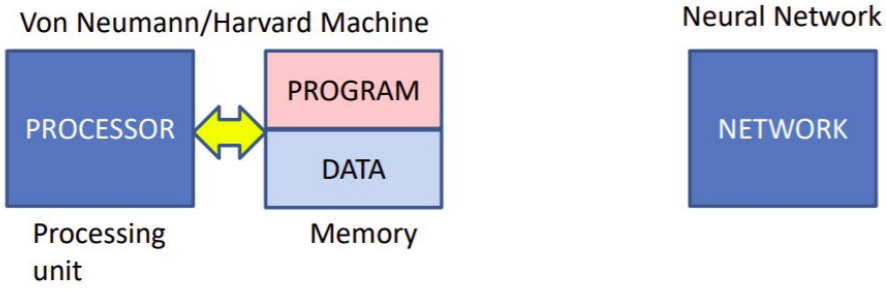
Connectionist Machines



- Network of processing elements
- **All world knowledge is stored in the *connections* between the elements**

Connectionist Machines

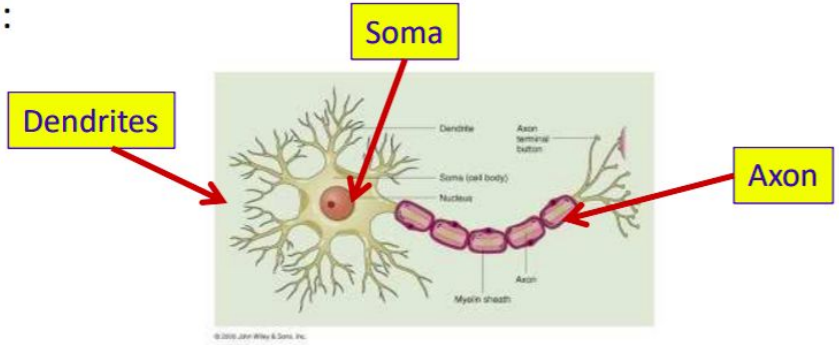
- Neural networks are *connectionist* machines
  - As opposed to Von Neumann Machines



- The machine has many non-linear processing units
  - The program is the connections between these units
    - Connections may also define memory

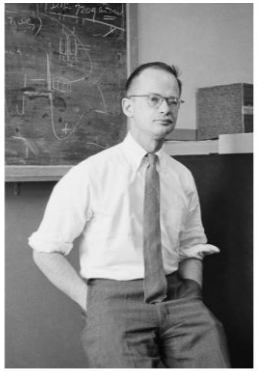
Modelling the brain

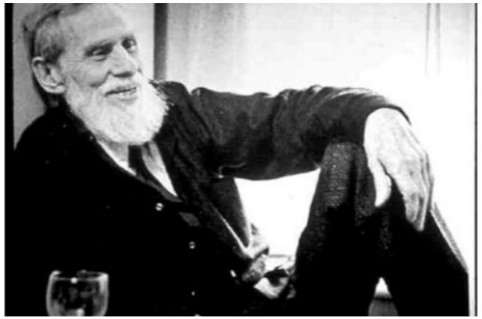
- What are the units?
- A neuron:



- Signals come in through the dendrites into the Soma
- A signal goes out via the axon to other neurons
  - Only one axon per neuron

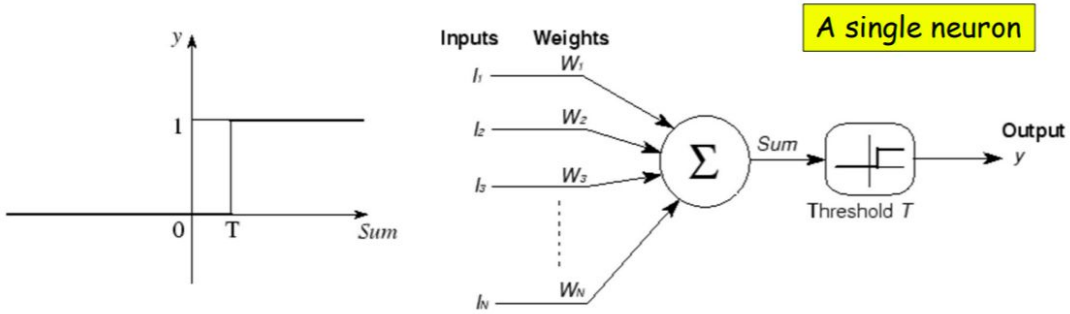
McCullough and Pitts





- The Doctor and the Hobo..
  - Warren McCulloch: Neurophysician
  - Walter Pitts: Homeless wannabe logician who arrived at his door

The McCulloch and Pitts model

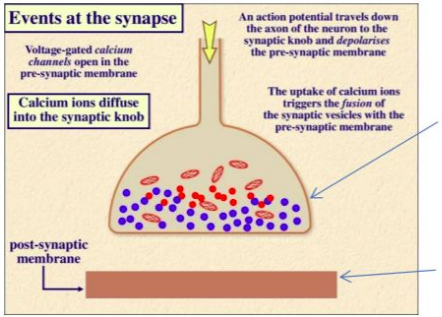


- A mathematical model of a neuron
  - McCulloch, W.S. & Pitts, W.H. (1943). A Logical Calculus of the Ideas Immanent in Nervous Activity, Bulletin of Mathematical Biophysics, 5:115-137, 1943
    - Pitts was only 20 years old at this time
  - Threshold Logic

Donald Hebb

- “Organization of behavior”, 1949
- A learning mechanism:
  - Neurons that fire together wire together

Hebbian Learning



- If neuron x_i repeatedly triggers neuron y, the synaptic knob connecting x_i to y gets larger
- In a mathematical model:

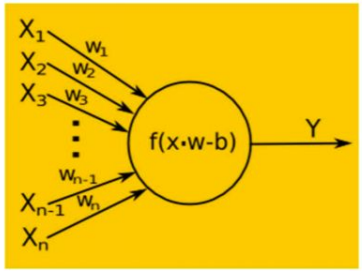
$$w_i = w_i + \eta x_i y$$

  - Weight of i^{th} neuron's input to output neuron y
- This simple formula is actually the basis of many learning algorithms in ML

A better model

- Frank Rosenblatt
  - Psychologist, Logician
  - Inventor of the solution to everything, aka the Perceptron (1958)

Simplified mathematical model

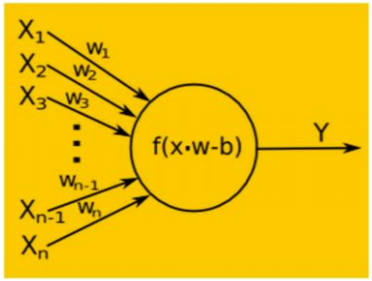


- Number of inputs combine linearly
  - Threshold logic: Fire if combined input exceeds threshold

$$Y = \begin{cases} 1 \;\; if \;\; \sum_i w_i x_i + b > 0 \\ 0 \qquad\qquad else \end{cases}$$

His "Simple" Perceptron

- Originally assumed could represent *any* Boolean circuit and perform any logic
  - "*the embryo of an electronic computer that [the Navy] expects will be able to walk, talk, see, write, reproduce itself and be conscious of its existence*," New York Times (8 July) 1958
  - "*Frankenstein Monster Designed by Navy That Thinks*," Tulsa, Oklahoma Times 1958



Also provided a learning algorithm

$$\mathbf{w} = \mathbf{w} + \eta(d(\mathbf{x}) - y(\mathbf{x}))\mathbf{x}$$

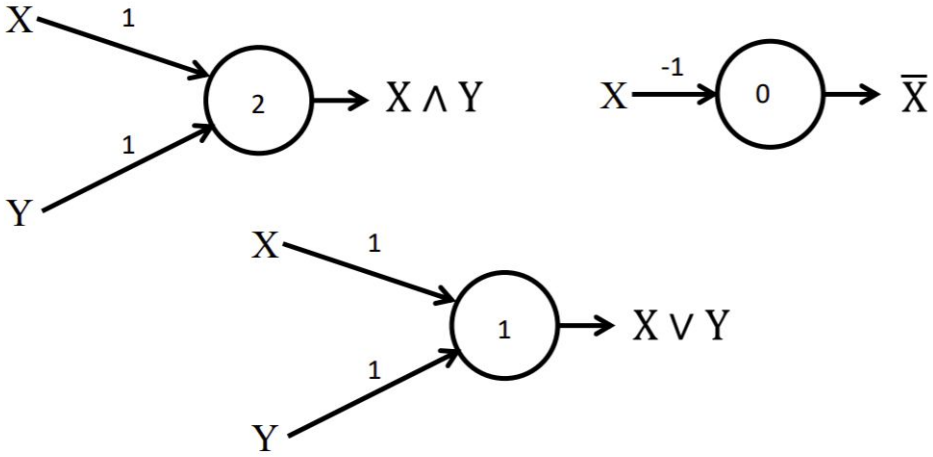
Sequential Learning:

$d(x)$ is the desired output in response to input x

$y(x)$ is the actual output in response to x

- Boolean tasks
- Update the weights whenever the perceptron output is wrong
- Proved convergence

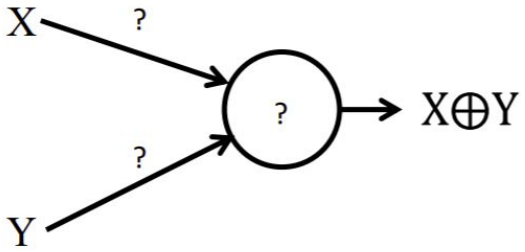
Perceptron



- Easily shown to mimic any Boolean gate
- But…

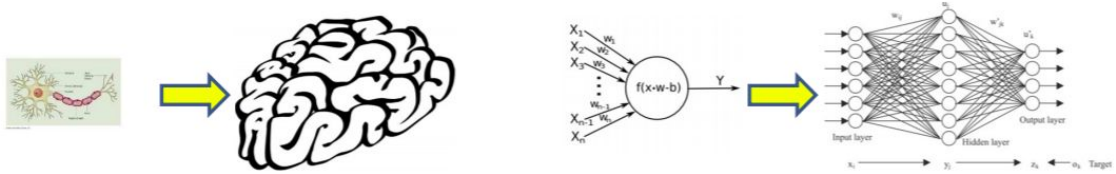
Perceptron

No solution for XOR!
Not universal!



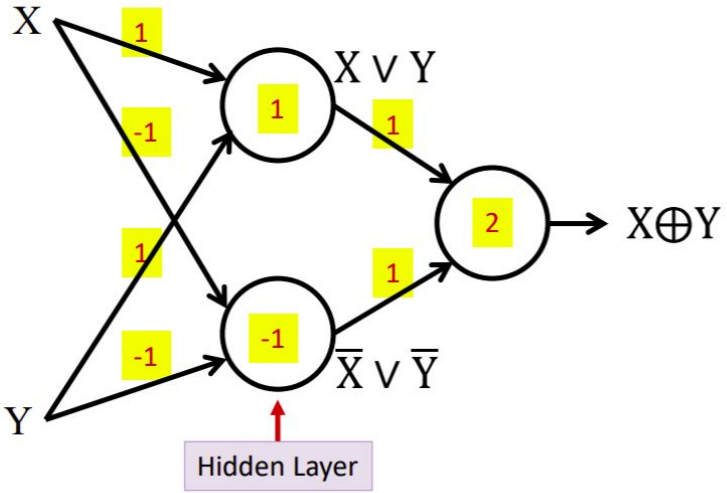
- Minsky and Papert, 1968

A single neuron is not enough



- Individual elements are weak computational elements
  - Marvin Minsky and Seymour Papert, 1969, *Perceptrons: An Introduction to Computational Geometry*
- *Networked* elements are required

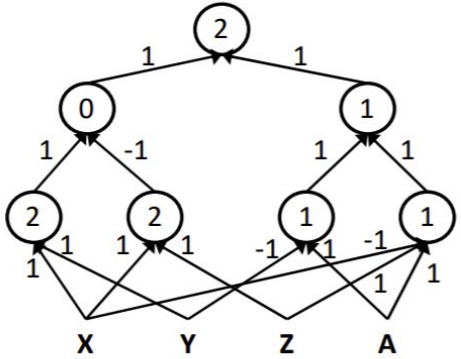
Multi-layer Perceptron!



- XOR
  - The first layer is a "hidden" layer
  - Also originally suggested by Minsky and Papert, 1968

A more generic model

$$((A\&\bar{X}\&Z)|(A\&\bar{Y}))\&((X\ \&\ Y)|\overline{(X\&Z)})$$

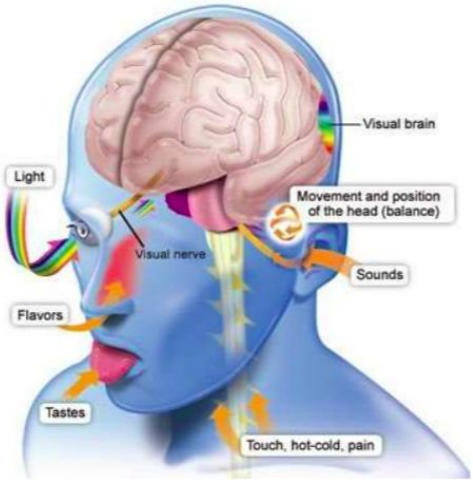


- A “multi-layer” perceptron
- Can compose arbitrarily complicated Boolean functions!
  - More on this in the next part

Story so far

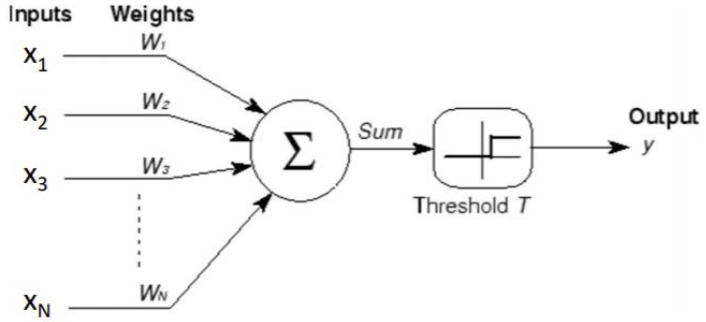
- Neural networks began as computational models of the brain
- Neural network models are *connectionist machines*
  - The comprise networks of neural units
- McCullough and Pitt model: Neurons as Boolean threshold units
  - Models the brain as performing propositional logic
  - But no learning rule
- Hebb's learning rule: Neurons that fire together wire together
  - Unstable
- Rosenblatt's perceptron : A variant of the McCulloch and Pitt neuron with a provably convergent learning rule
  - But individual perceptrons are limited in their capacity (Minsky and Papert)
- Multi-layer perceptrons can model arbitrarily complex Boolean functions

But our brain is not Boolean



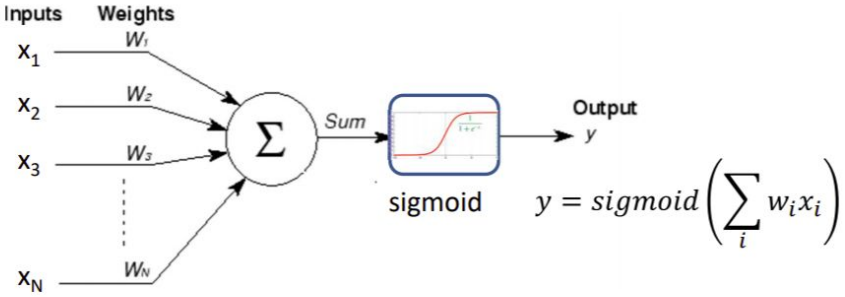
- We have real inputs
- We make non-Boolean inferences/predictions

The perceptron with *real* inputs



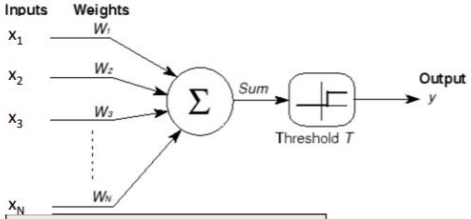
- $x_1 \ldots x_N$ are real valued
- $W_1 \ldots W_N$ are real valued
- Unit "fires" if weighted input exceeds a threshold

The perceptron with *real* inputs and a real *output*

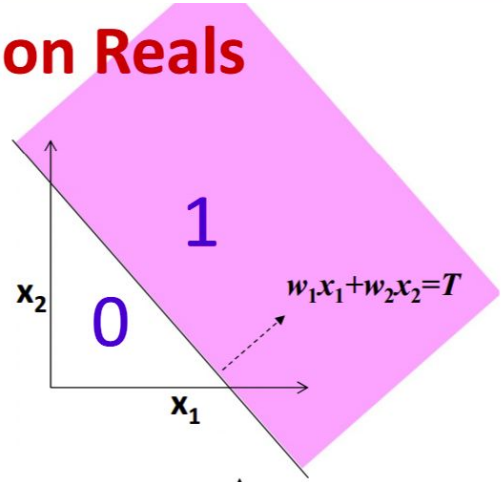


- $x_1 \ldots x_N$ are real valued
- $W_1 \ldots W_N$ are real valued
- The output y can also be real valued
  - Sometimes viewed as the "probability" of firing
  - *Is useful to continue assuming Boolean outputs though*

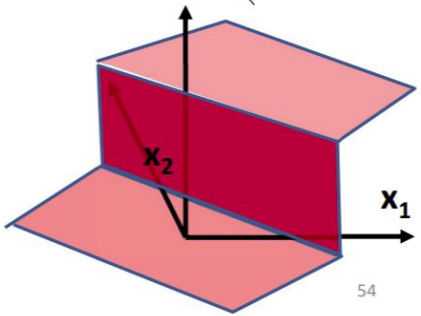
A Perceptron on Reals



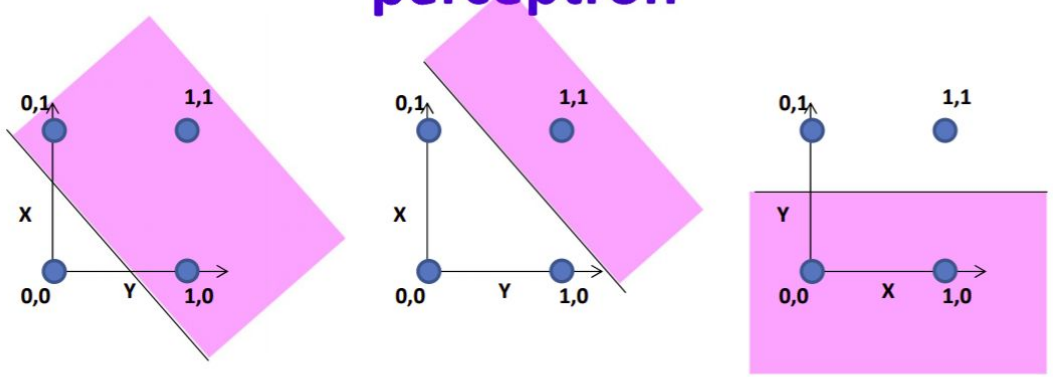
$$y = \begin{cases} 1 \text{ if } \sum_i w_i x_i \geq T \\ 0 \text{ else} \end{cases}$$



- A perceptron operates on *real*-valued vectors
  - **This is a *linear classifier***

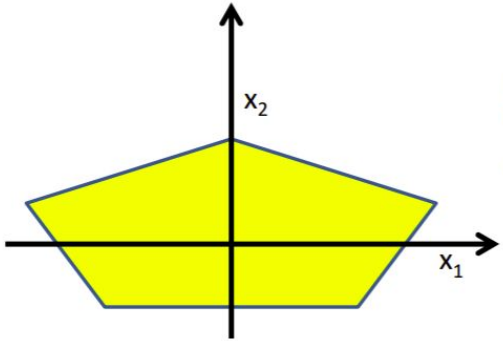


Boolean functions with a real perceptron



- Boolean perceptrons are also linear classifiers
  - Purple regions have output 1 in the figures
  - What are these functions
  - Why can we not compose an XOR?

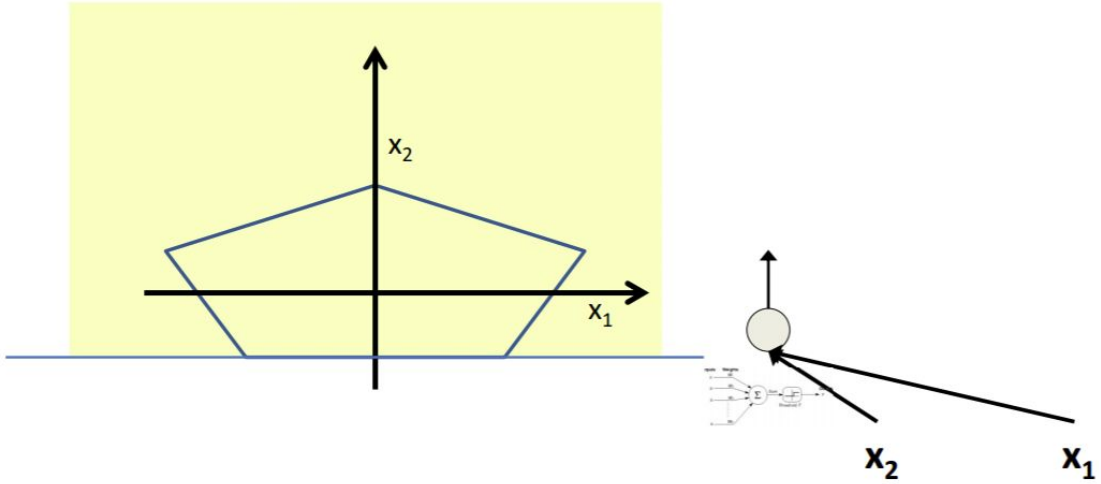
Composing complicated "decision" boundaries



Can now be composed into "networks" to compute arbitrary classification "boundaries"

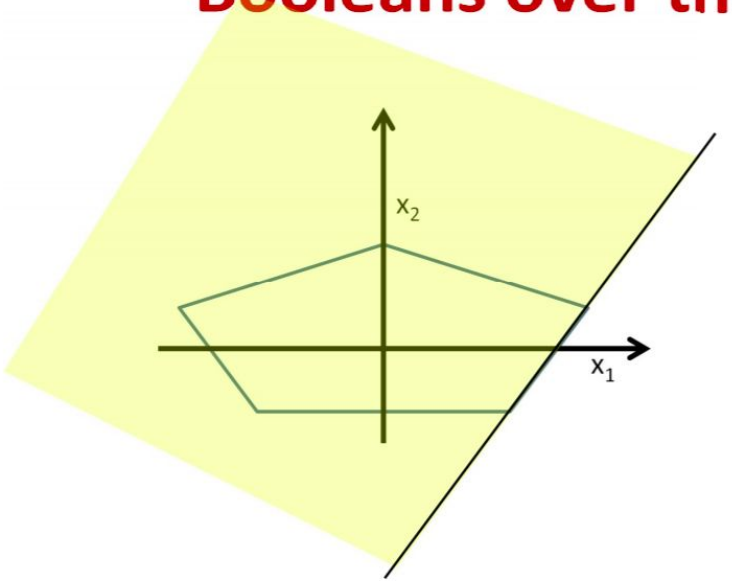
- Build a network of units with a single output that fires if the input is in the coloured area

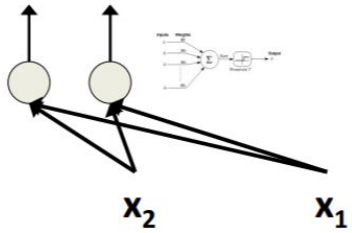
Booleans over the reals



- The network must fire if the input is in the coloured area

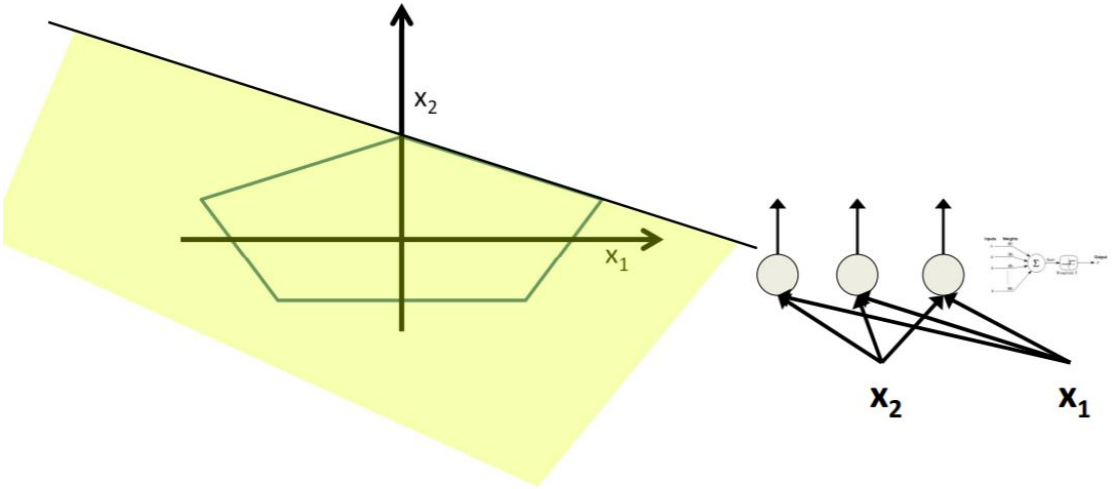
Booleans over the reals





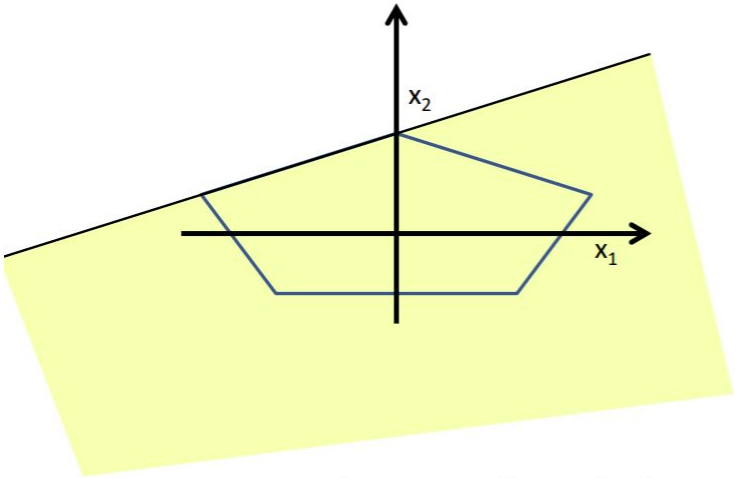
- The network must fire if the input is in the coloured area

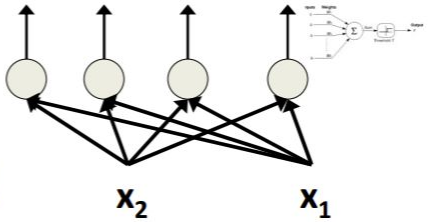
Booleans over the reals



- The network must fire if the input is in the coloured area

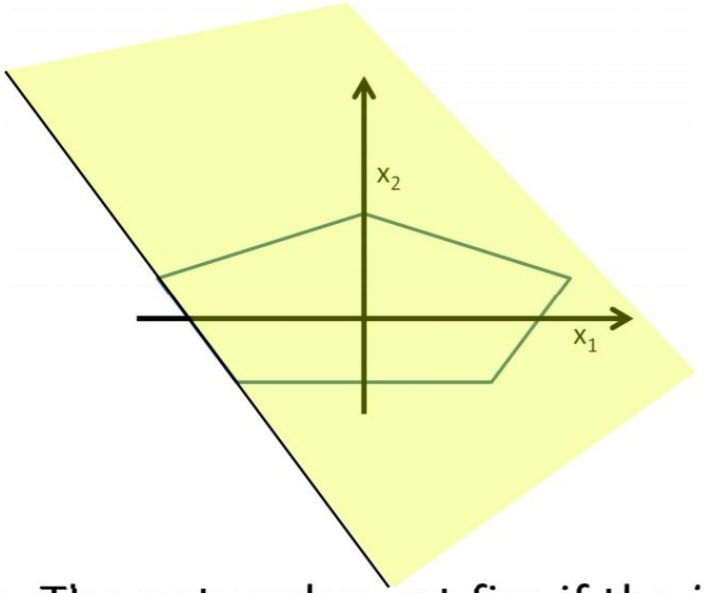
Booleans over the reals

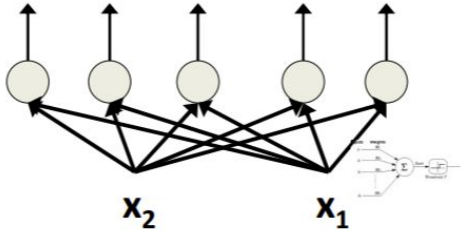




- The network must fire if the input is in the coloured area

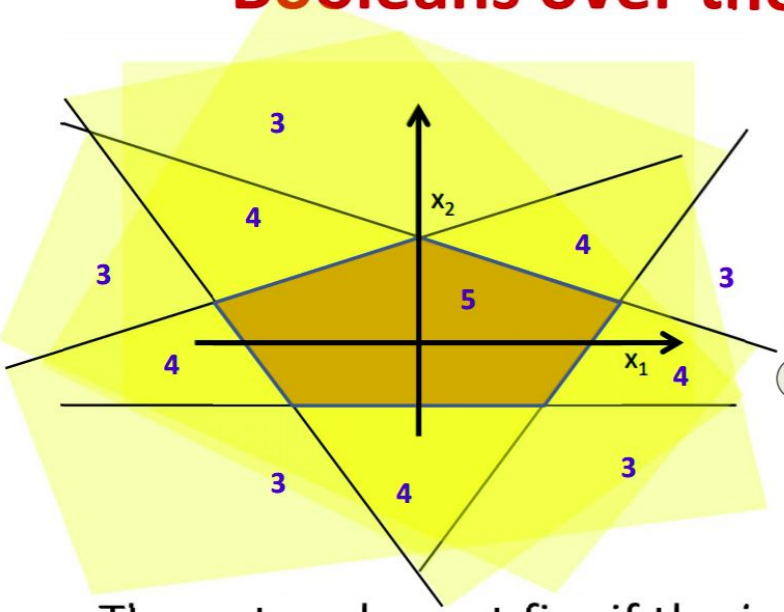
Booleans over the reals

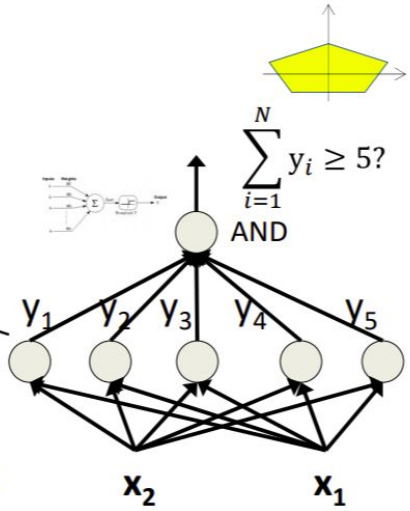




- The network must fire if the input is in the coloured area

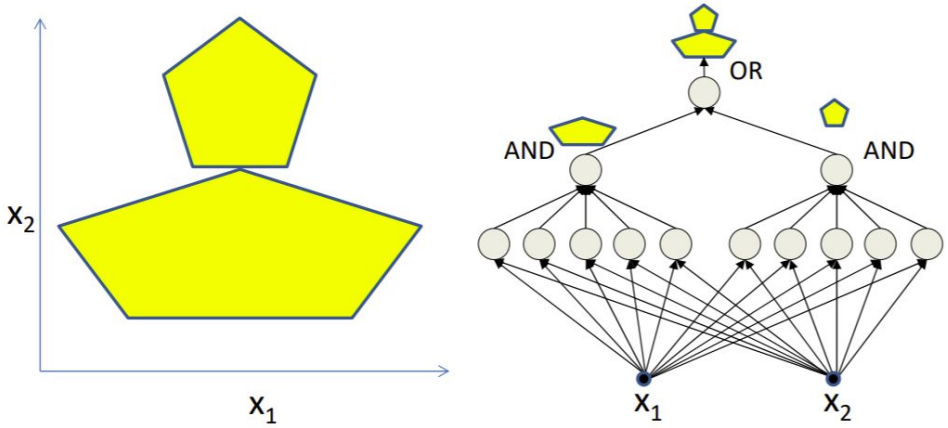
Booleans over the reals





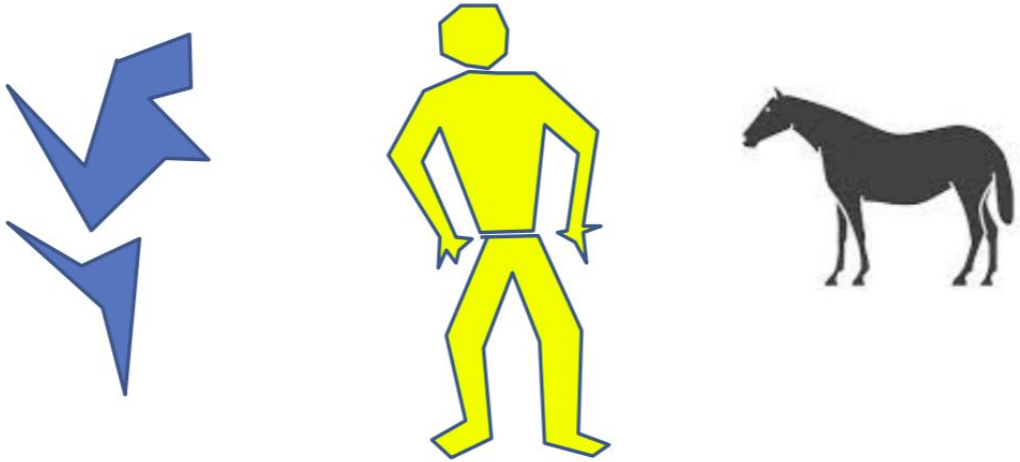
- The network must fire if the input is in the coloured area

More complex decision boundaries



- Network to fire if the input is in the yellow area
  - "OR" two polygons
  - A third layer is required

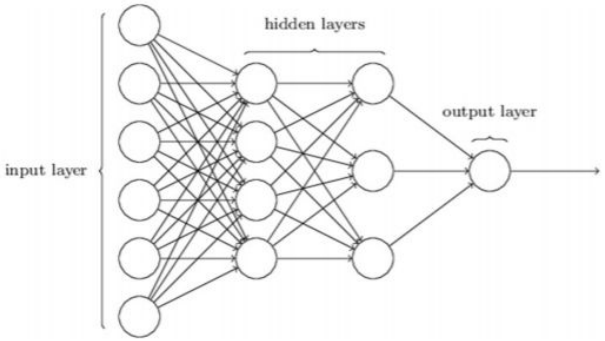
Complex decision boundaries

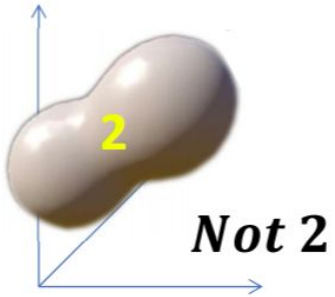


- Can compose very complex decision boundaries
  - How complex exactly? More on this in the next part

Complex decision boundaries

784 dimensions
(MNIST)

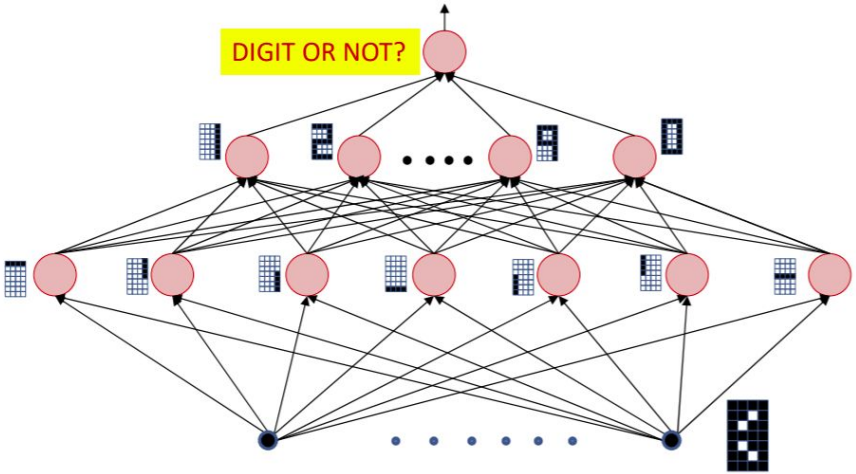




784 dimensions

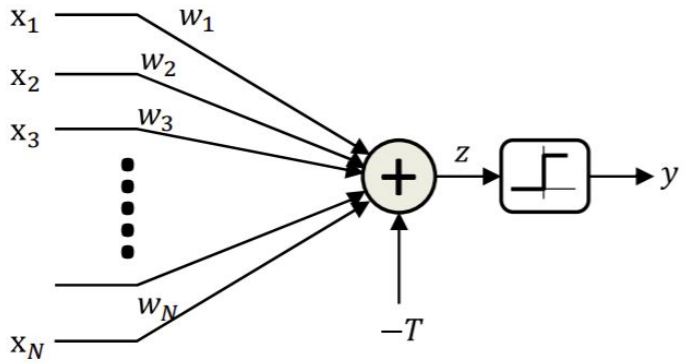
- Classification problems: finding decision boundaries in high-dimensional space

The MLP as a cascade of feature detectors



- The network is a cascade of feature detectors
  - Higher level neurons compose complex templates from features represented by lower-level neurons

Recap: The perceptron

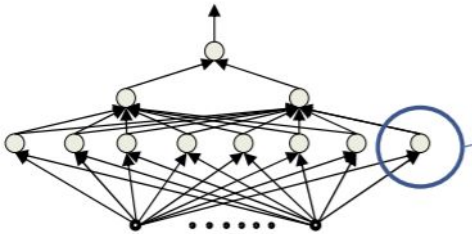


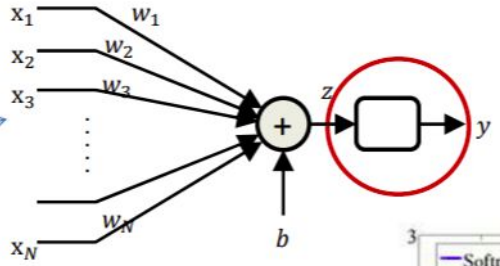
$$z = \sum_i w_i x_i - T$$

$$y = \begin{cases} 1 \; if \; z \geq 0 \\ 0 \; else \end{cases}$$

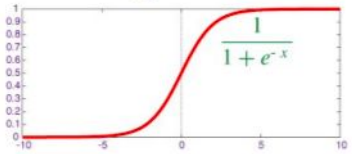
- A threshold unit
  - "Fires" if the weighted sum of inputs and the "bias" T is positive

Other "activations"

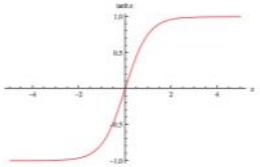


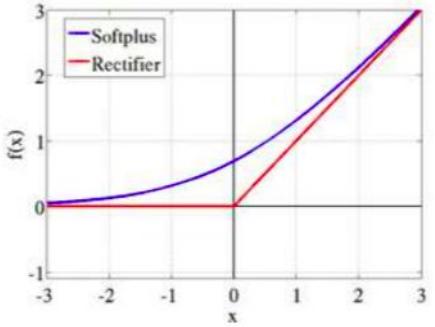


sigmoid

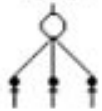
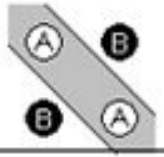
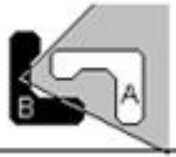


tanh

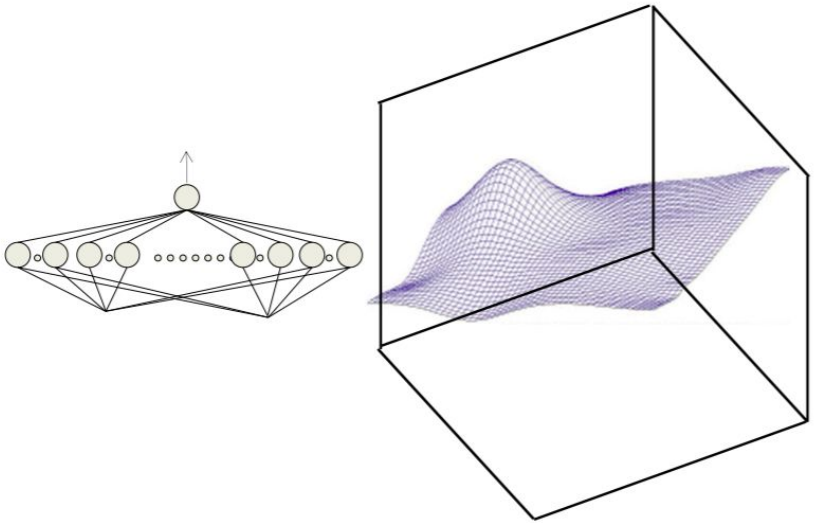




Redes Neuronales

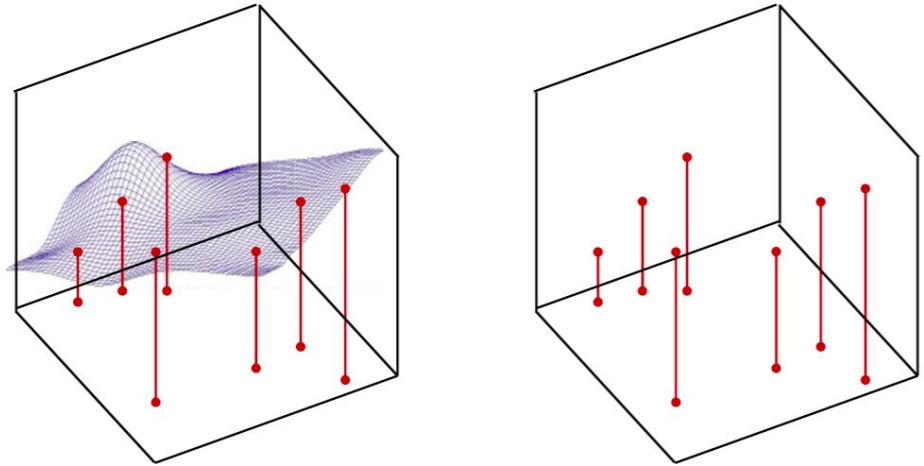
Estructura	Regiones de Desición	Problema de la XOR	Clases con Regiones Mezcladas	Formas de Regiones más Generales
1 Capa 	Medio Plano Limitado por un Hiperplano			
2 Capas 	Regiones Cerradas o Convexas			
3 Capas 	Complejidad Arbitraria Limitada por el Número de Neuronas			

Learning the network



- The neural network can approximate *any* function
- But only if the function is known *a priori*

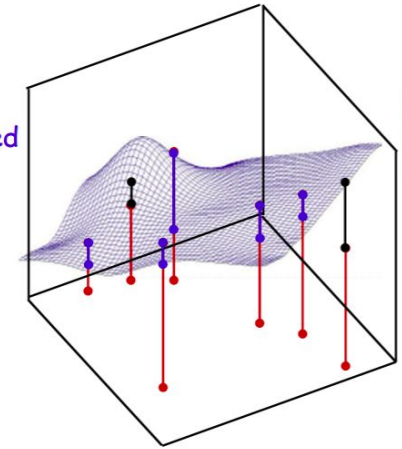
Learning the network



- In reality, we will only get a few *snapshots* of the function to learn it from
- We must learn the entire function from these “training” snapshots

General approach to training

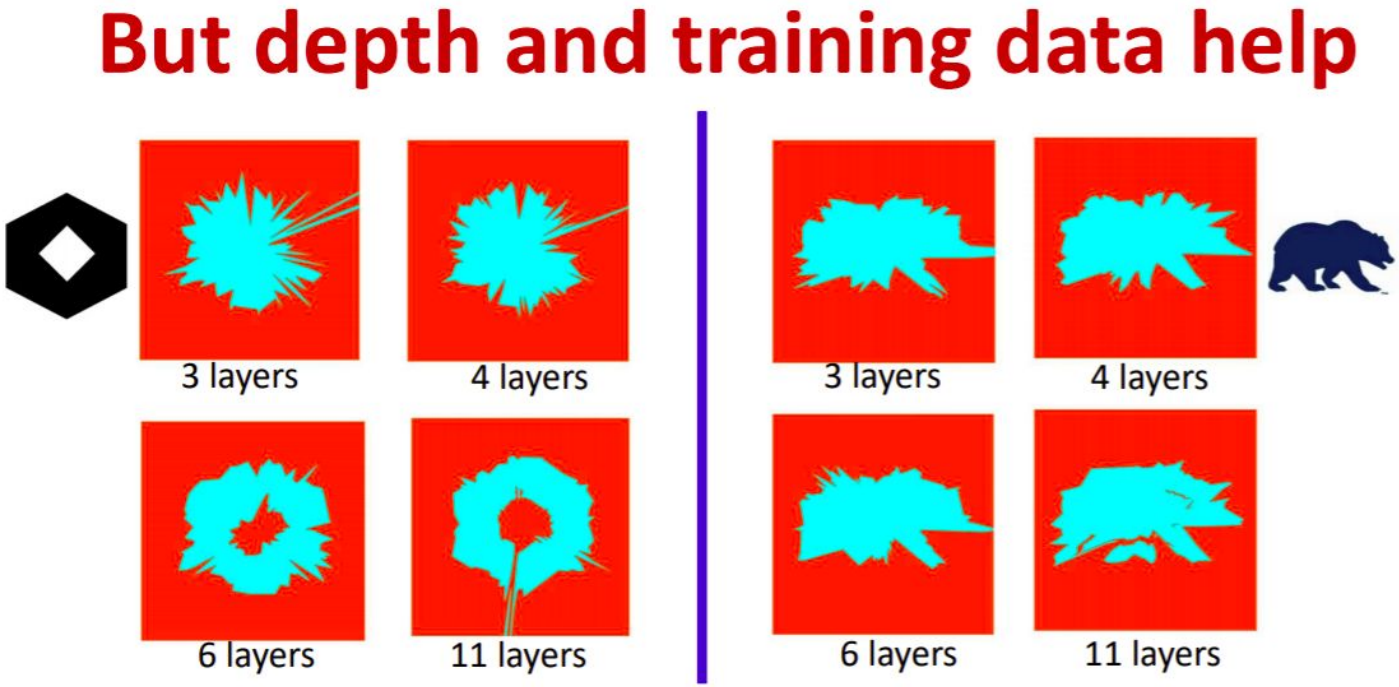
Blue lines: error when function is *below* desired output



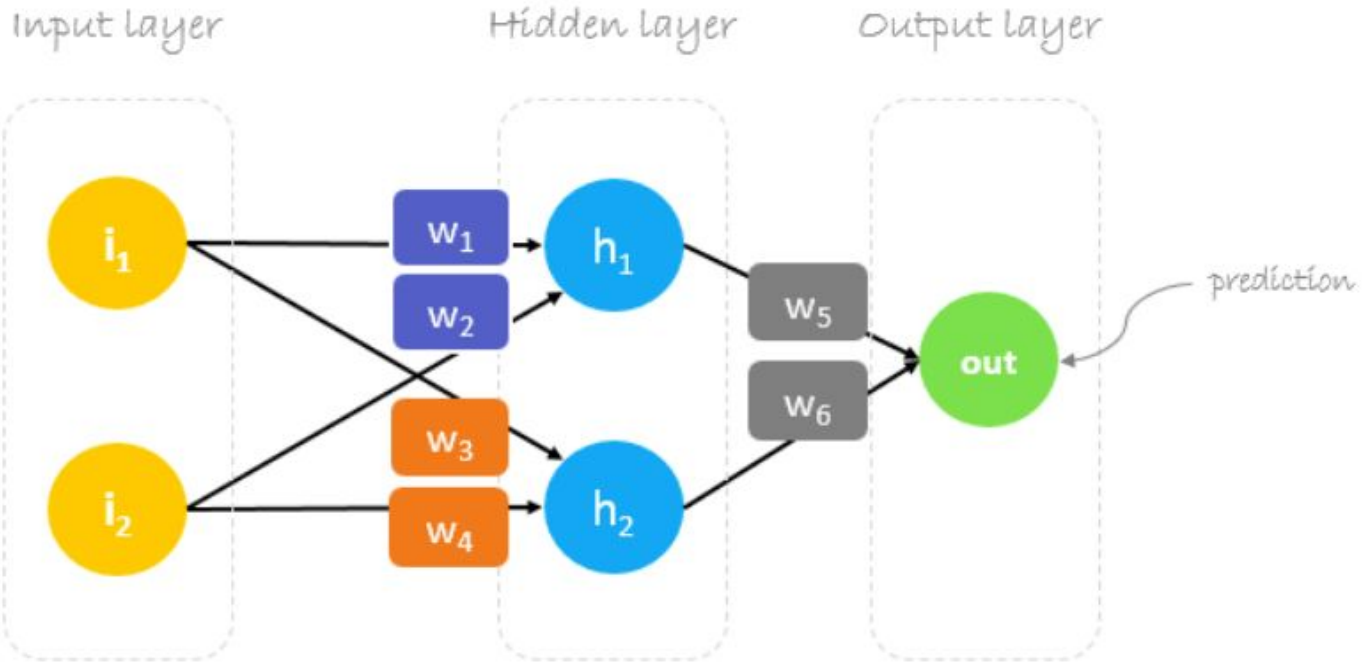
Black lines: error when function is *above* desired output

$$E = \sum_i (y_i - f(\mathbf{x}_i, \mathbf{W}))^2$$

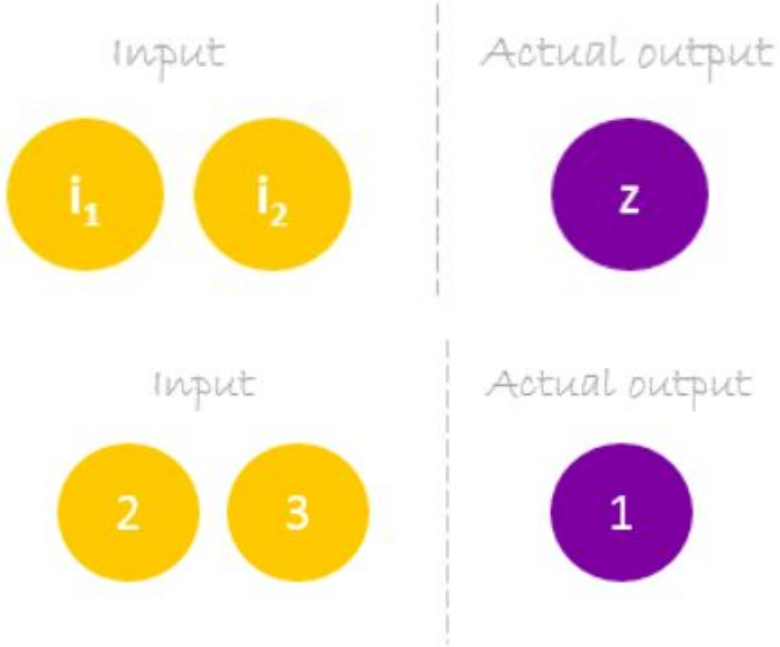
- Define an *error* between the ***actual*** network output for any parameter value and the *desired* output
  - E.g. error defined as the *sum* of the squared error over individual training instances


But depth and training data help
3 layers
4 layers
6 layers
11 layers
3 layers
4 layers
6 layers
11 layers

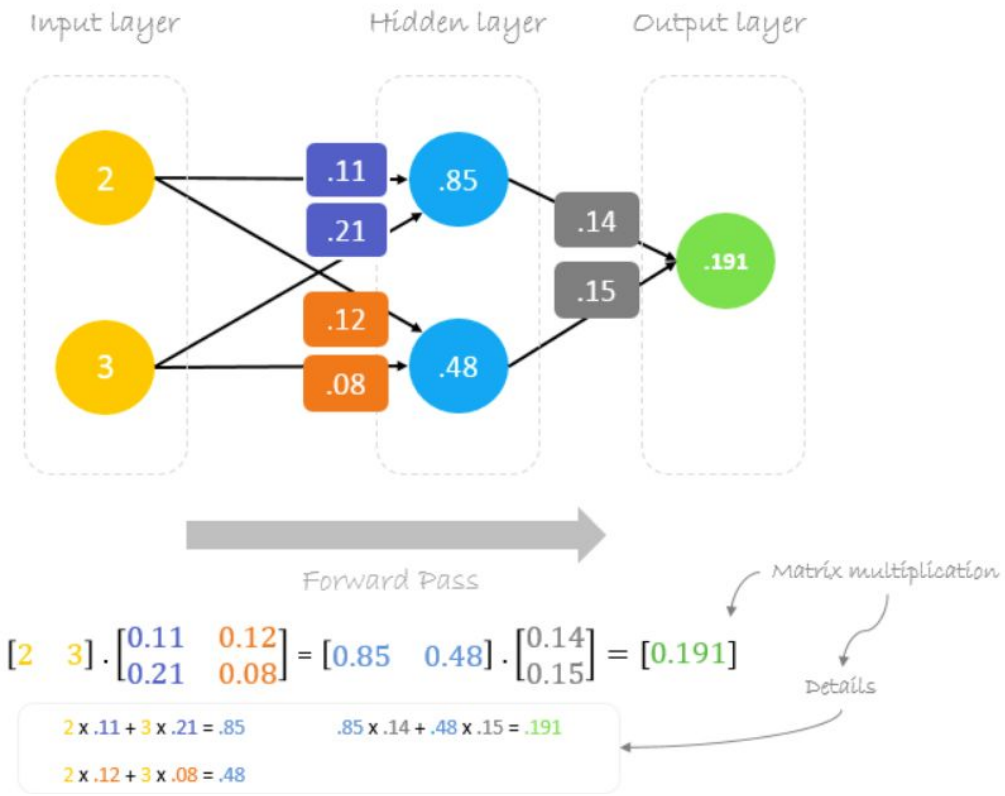
Redes Neuronales



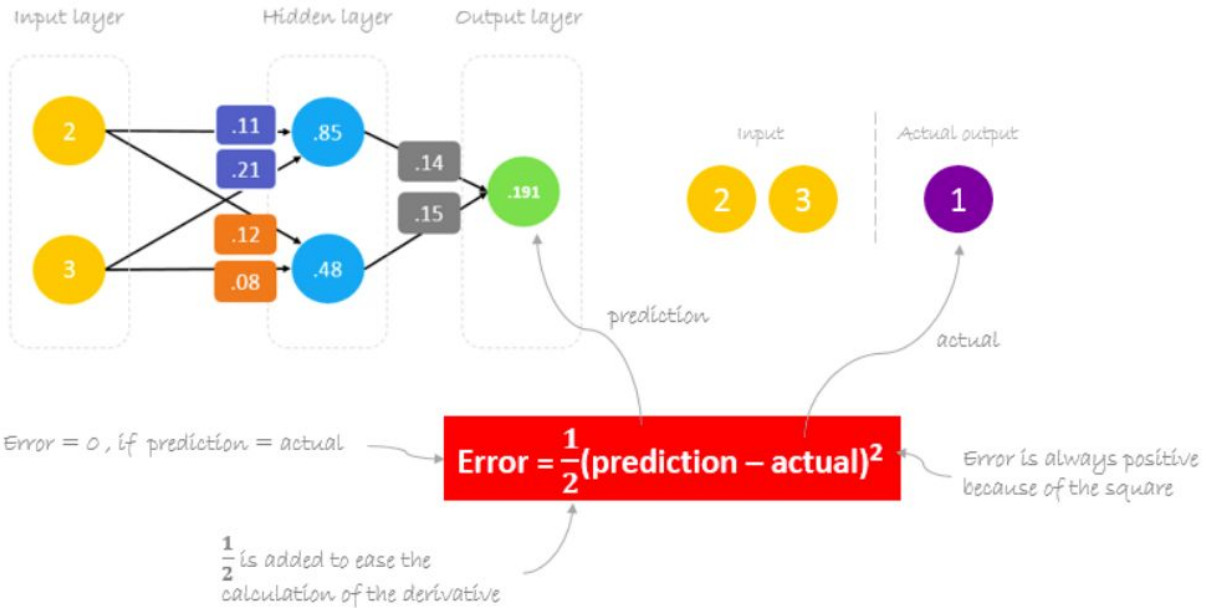
Redes Neuronales



Redes Neuronales



Redes Neuronales



Redes Neuronales

$$\text{prediction} = \text{out}$$

$$\text{prediction} = (h_1)\, w_5 + (h_2)\, w_6$$

$$h_1 = i_1 w_1 + i_2 w_2$$

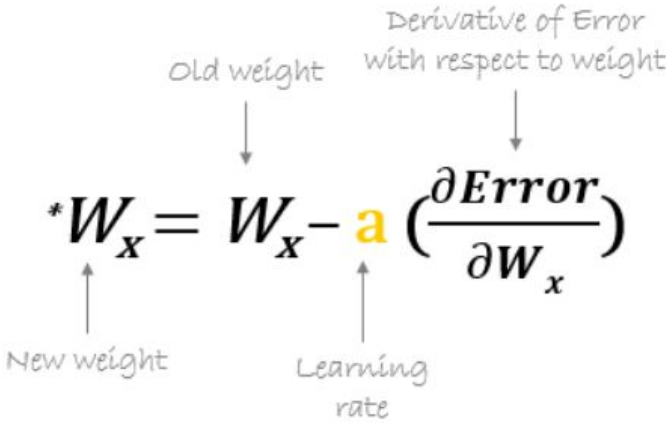
$$h_2 = i_1 w_3 + i_2 w_4$$

$$\text{prediction} = (i_1 w_1 + i_2 w_2)\, w_5 + (i_1 w_3 + i_2 w_4)\, w_6$$

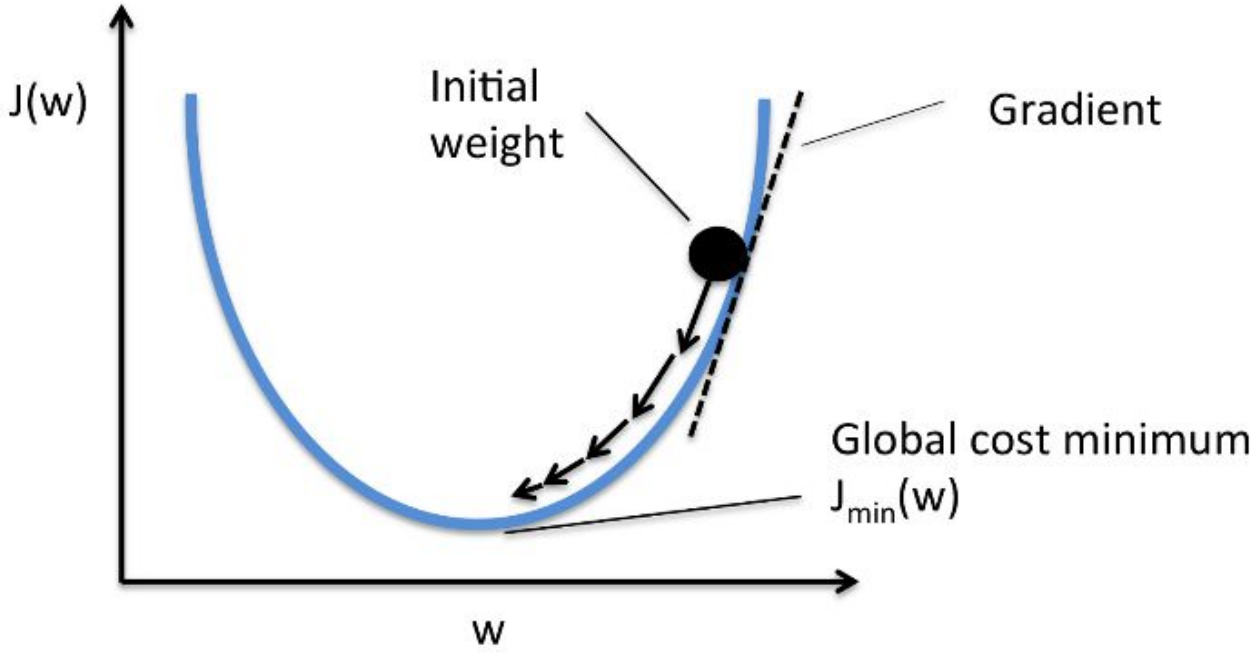
to change ***prediction*** value,
we need to change ***weights***

Redes Neuronales

> **Gradient descent** is an iterative optimization algorithm for finding the minimum of a function; in our case we want to minimize th error function. To find a local minimum of a function using gradient descent, one takes steps proportional to the negative of the gradient of the function at the current point.



Redes Neuronales



Redes Neuronales

$$^{*}W_6 = W_6 - a\left(\frac{\partial Error}{\partial W_6}\right)$$

Redes Neuronales

$$\frac{\partial Error}{\partial W_6} = \frac{\partial Error}{\partial prediction} * \frac{\partial prediction}{\partial W_6}$$

chain rule

$$Error = \frac{1}{2}(prediction - actual)^2$$

$$prediction = (i_1 w_1 + i_2 w_2) w_5 + (i_1 w_3 + i_2 w_4) w_6$$

$$\frac{\partial Error}{\partial W_6} = \frac{\frac{1}{2}(predictoin - actula)^2}{\partial prediciton} * \frac{\partial (i_1 w_1 + i_2 w_2) w_5 + (i_1 w_3 + i_2 w_4) w_6}{\partial W_6}$$

$$\frac{\partial Error}{\partial W_6} = 2 * \frac{1}{2}(predictoin - actula) \frac{\partial (predictoin - actula)}{\partial prediciton} * (i_1 w_3 + i_2 w_4)$$

$h_2 = i_1 w_3 + i_2 w_4$

$$\frac{\partial Error}{\partial W_6} = (predictoin - actula) * (h_2)$$

Δ = prediction – actual ← delta

$$\frac{\partial Error}{\partial W_6} = \Delta h_2$$

$$^{*}W_6 = W_6 - a\,\Delta h_2$$

$$^{*}W_5 = W_5 - a\,\Delta h_1$$

Redes Neuronales

$$\frac{\partial Error}{\partial W_1} = \frac{\partial Error}{\partial prediction} * \frac{\partial prediction}{\partial h_1} * \frac{\partial h_1}{\partial W_1}$$ ← chain rule

$$\frac{\partial Error}{\partial W_1} = \frac{\partial \frac{1}{2}(predictoin - actula)^2}{\partial prediciton} * \frac{\partial (h_1)\, w_5 + (h_2)\, w_6}{\partial h_1} * \frac{\partial\, i_1 w_1 + i_2 w_2}{\partial w_1}$$

$$\frac{\partial Error}{\partial W_1} = 2 * \frac{1}{2}(\mathbf{predictoin - actula}) \frac{\partial (predictoin - actula)}{\partial prediciton} * (w_5) * (i_1)$$

$$\frac{\partial Error}{\partial W_1} = (\mathbf{predictoin - actula}) * (w_5 i_1)$$

$$\frac{\partial Error}{\partial W_1} = \Delta\, w_5 i_1$$

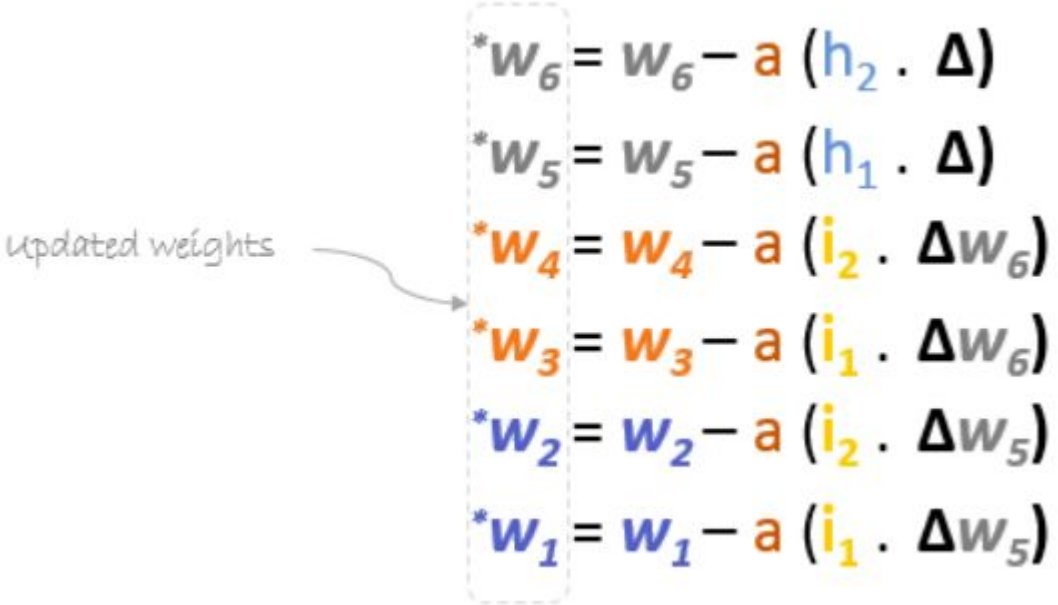
$Error = \frac{1}{2}(prediction - actual)^2$

$prediction = (h_1)\, w_5 + (h_2)\, w_6$

$h_1 = i_1 w_1 + i_2 w_2$

$\Delta = prediction - actual$ ← delta

Redes Neuronales



$$\begin{bmatrix} w_5 \\ w_6 \end{bmatrix} = \begin{bmatrix} w_5 \\ w_6 \end{bmatrix} - a\,\Delta \begin{bmatrix} h_1 \\ h_2 \end{bmatrix} = \begin{bmatrix} w_5 \\ w_6 \end{bmatrix} - \begin{bmatrix} a h_1 \Delta \\ a h_2 \Delta \end{bmatrix}$$

$$\begin{bmatrix} w_1 & w_3 \\ w_2 & w_4 \end{bmatrix} = \begin{bmatrix} w_1 & w_3 \\ w_2 & w_4 \end{bmatrix} - a\,\Delta \begin{bmatrix} i_1 \\ i_2 \end{bmatrix} . [w_5 \quad w_6] = \begin{bmatrix} w_1 & w_3 \\ w_2 & w_4 \end{bmatrix} - \begin{bmatrix} a\, i_1 \Delta w_5 & a\, i_1 \Delta w_6 \\ a\, i_2 \Delta w_5 & a\, i_2 \Delta w_6 \end{bmatrix}$$

Redes Neuronales

Learning rate: is a hyperparameter which means that we need to manually guess its value.

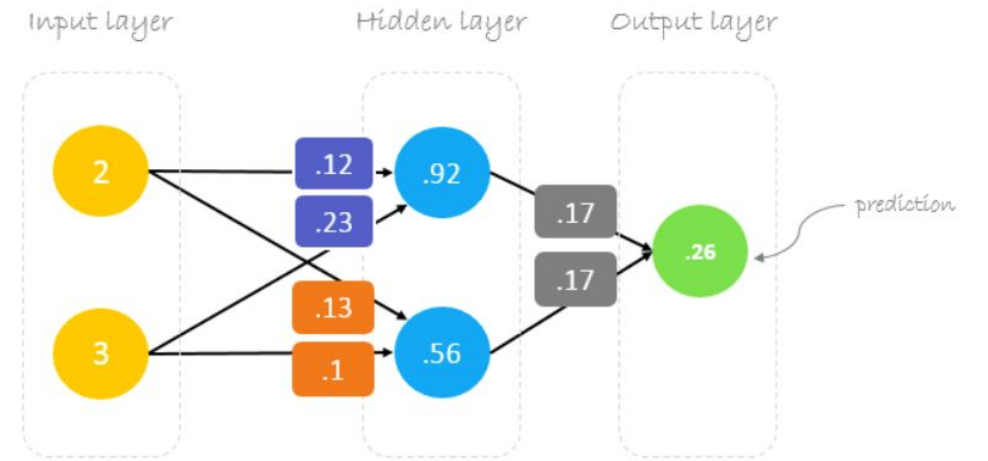
$\Delta = 0.191 - 1 = -0.809$ ← **Delta** = prediction - actual

$a = 0.05$ ← **Learning rate,** we smartly guess this number

$$\begin{bmatrix} w_5 \\ w_6 \end{bmatrix} = \begin{bmatrix} 0.14 \\ 0.15 \end{bmatrix} - 0.05(-0.809)\begin{bmatrix} 0.85 \\ 0.48 \end{bmatrix} = \begin{bmatrix} 0.14 \\ 0.15 \end{bmatrix} - \begin{bmatrix} -0.034 \\ -0.019 \end{bmatrix} = \begin{bmatrix} 0.17 \\ 0.17 \end{bmatrix}$$

$$\begin{bmatrix} w_1 & w_3 \\ w_2 & w_4 \end{bmatrix} = \begin{bmatrix} .11 & .12 \\ .21 & .08 \end{bmatrix} - 0.05(-0.809)\begin{bmatrix} 2 \\ 3 \end{bmatrix} . [0.14 \quad 0.15] = \begin{bmatrix} .11 & .12 \\ .21 & .08 \end{bmatrix} - \begin{bmatrix} -0.011 & -0.012 \\ -0.017 & -0.018 \end{bmatrix} = \begin{bmatrix} .12 & .13 \\ .23 & .10 \end{bmatrix}$$

Redes Neuronales



Redes Neuronales