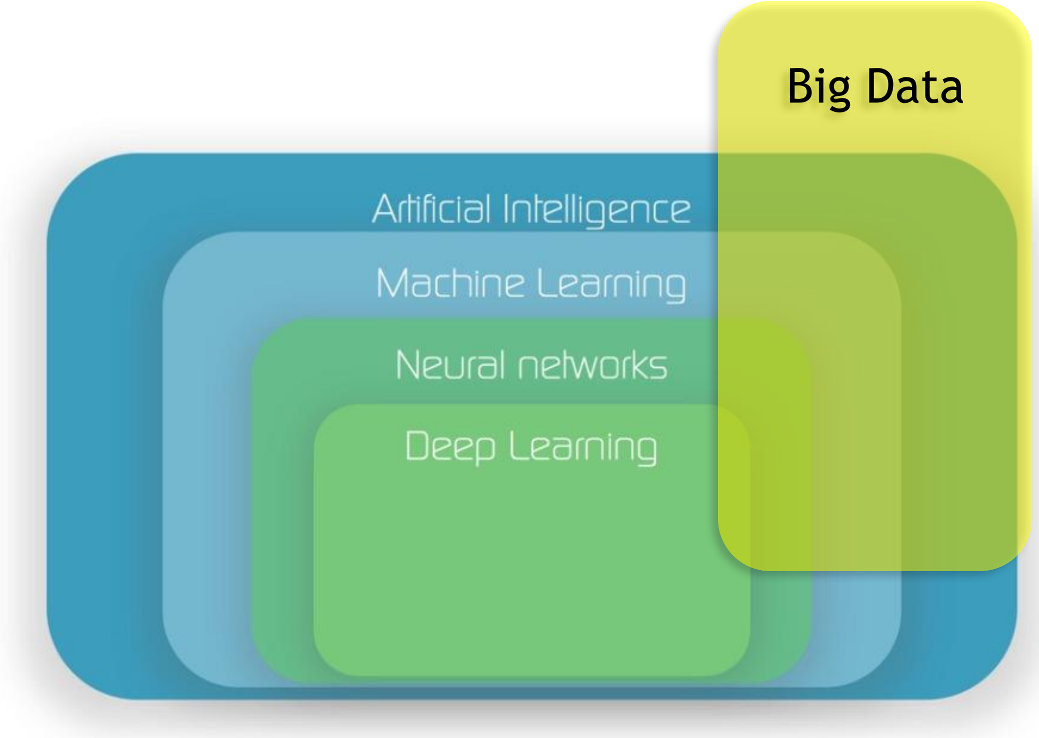
Redes Neuronales

Conceptos básicos

- **Big data**: almacenamiento y procesado de grandes volúmenes de datos
- **Inteligencia artificial**: Resolver problemas mediante máquinas
- **Machine Learning**: Conjunto de algoritmos capaces de identificar y aprender patrones en datos

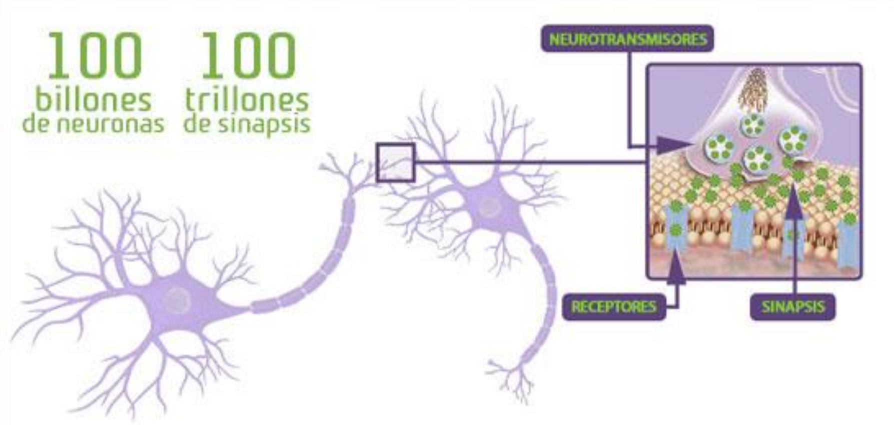


Redes Neuronales

- Algoritmos que simulan la manera en la que los organismos inteligentes procesan la información, a través de **neuronas**

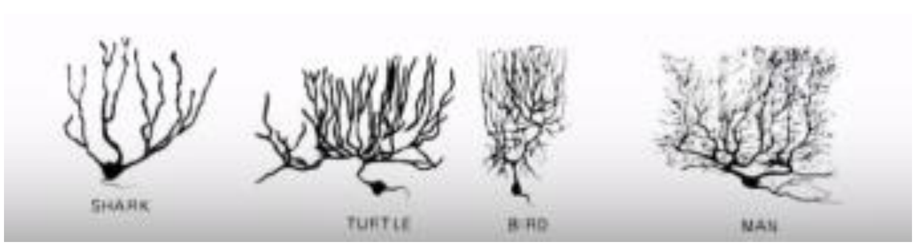
- Una neurona recibe muchas entradas y genera una única salida, que también constituirá una de las muchas entradas de otras neuronas

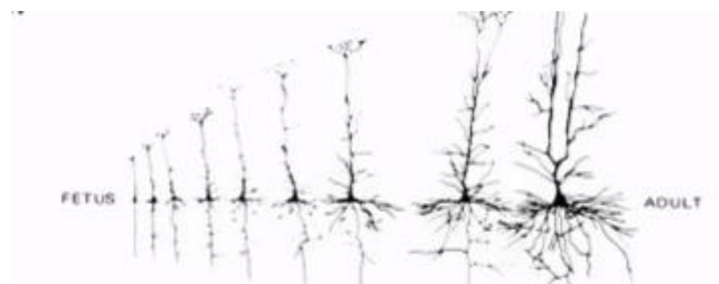
- La comunicación entre neuronas se realiza a través de una conexión llamada **sinapsis**, donde se encuentra la memoria



Redes Neuronales

- Entrenar la red neuronal significa alterar sus sinapsis, de forma que aprenda lo que queremos enseñarle
- Cuando vivimos alguna experiencia, se almacena en nuestro cerebro creando nuevas conexiones neuronales, deshaciendo otras y alterando la ponderación de cada una de esas conexiones
- Permiten realizar regresiones y clasificaciones

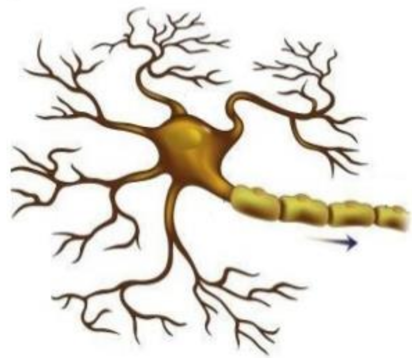




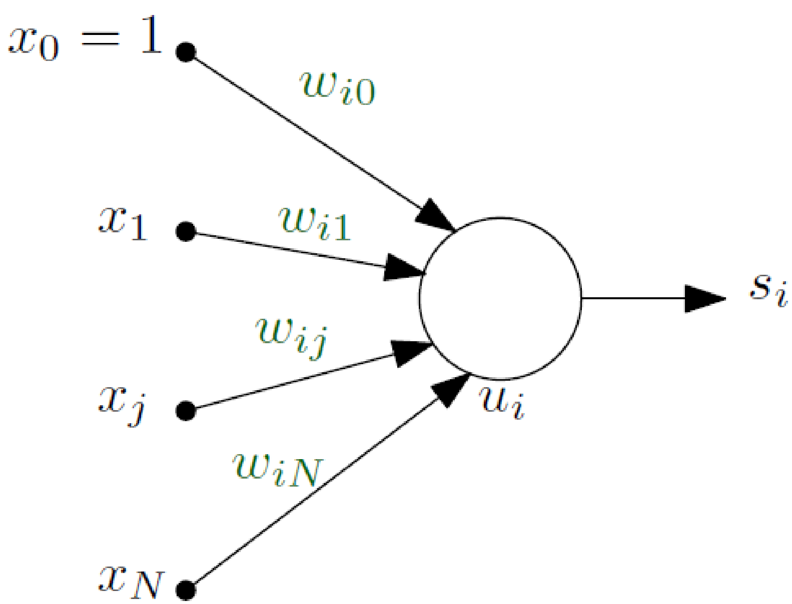
https://www.youtube.com/watch?v=rKF48Th__5c&ab_channel=alexgramma4

Redes Neuronales

- Modelo de neurona:



Neurona i

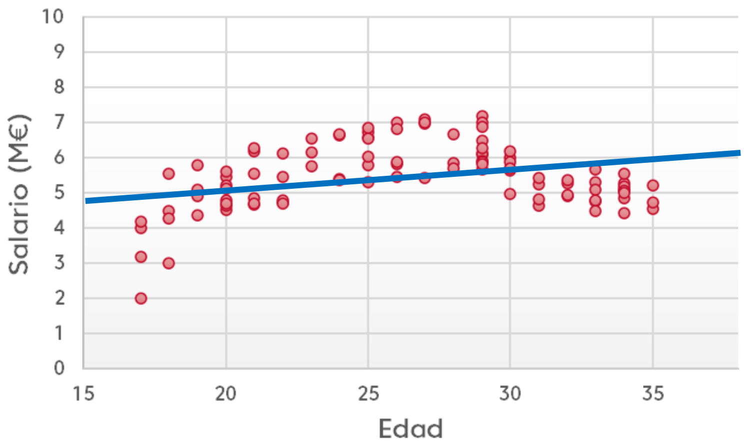


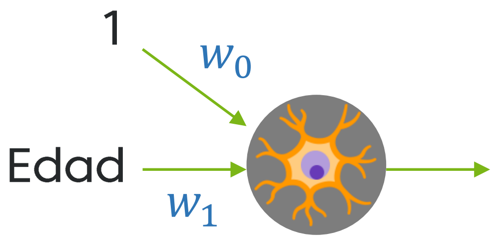
$$u_i = \sum_{j=1}^{N} w_{ij} x_j + w_{i0} = \mathbf{w_i^T x} + w_{i0}$$

$$s_i = s(u_i)$$

Redes Neuronales

- Ejemplo: Queremos predecir el salario de los jugadores en función de su edad, utilizando datos históricos





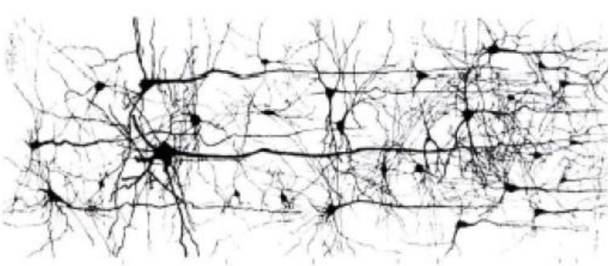
$$Salario = w_0 + w_1 \cdot Edad$$

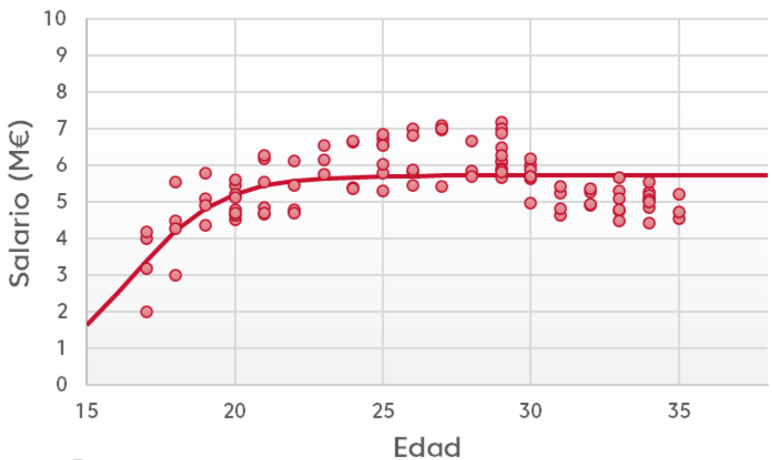
- Se buscan los valores de las sinapsis w_0 y w_1 que minimizan el error

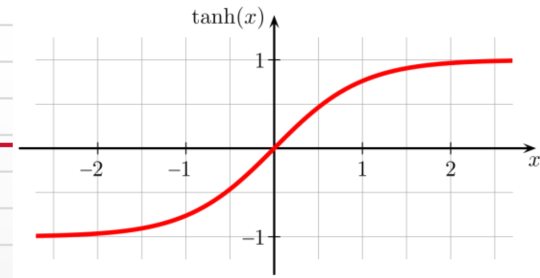
$$Salario = 4.51 + 0.036 \cdot Edad$$

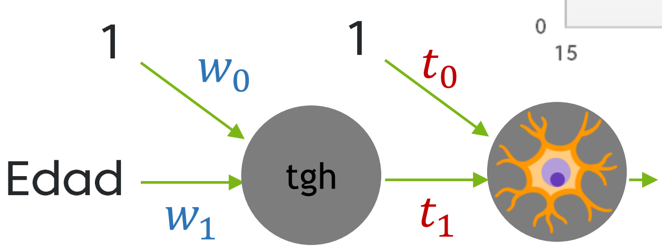
Redes Neuronales

- Podemos mejorar el desempeño incluyendo más capas de neuronas y funciones de activación no lineales (por ejemplo, tangentes hiperbólicas)









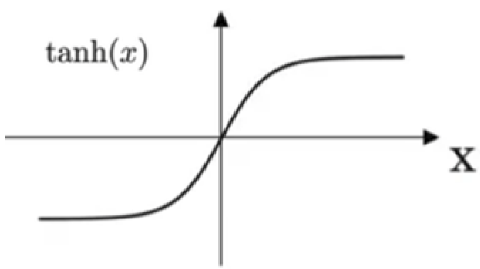
$$Salario = t_0 + t_1 \cdot tgh(w_0 + w_1 \cdot Edad)$$

- Se buscan los valores de las sinapsis w_0 w_1 t_0 y t_1 que minimizan el error

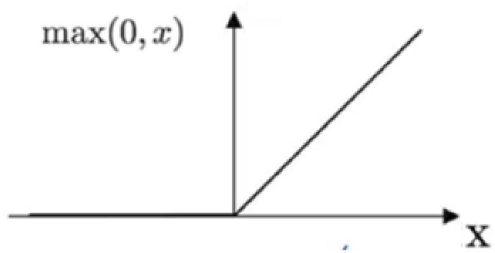
Redes Neuronales

- Función de activación

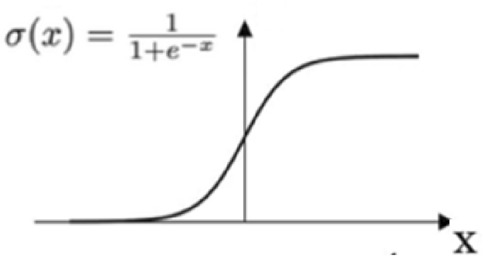
Hyper Tangent Function



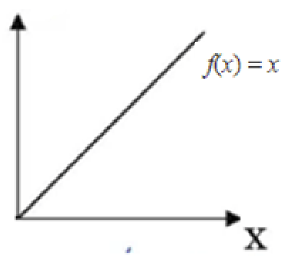
ReLU Function



Sigmoid Function

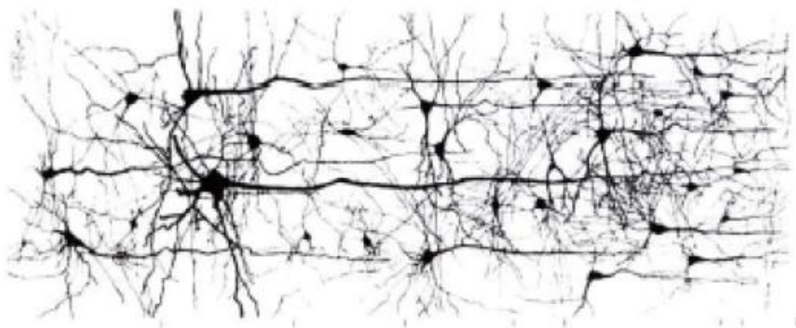


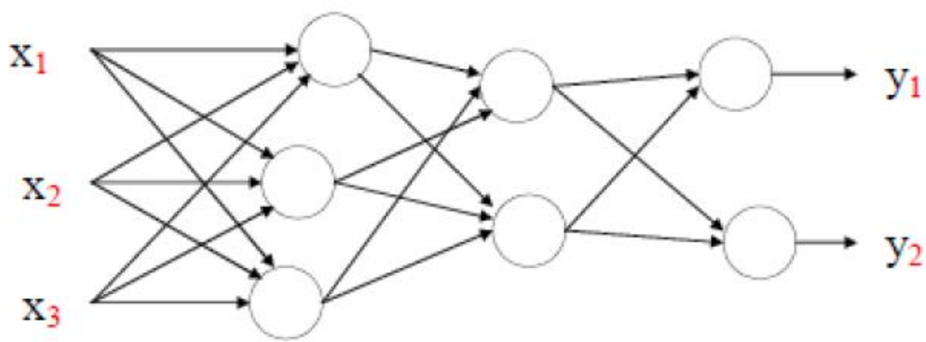
Identity Function



Redes Neuronales

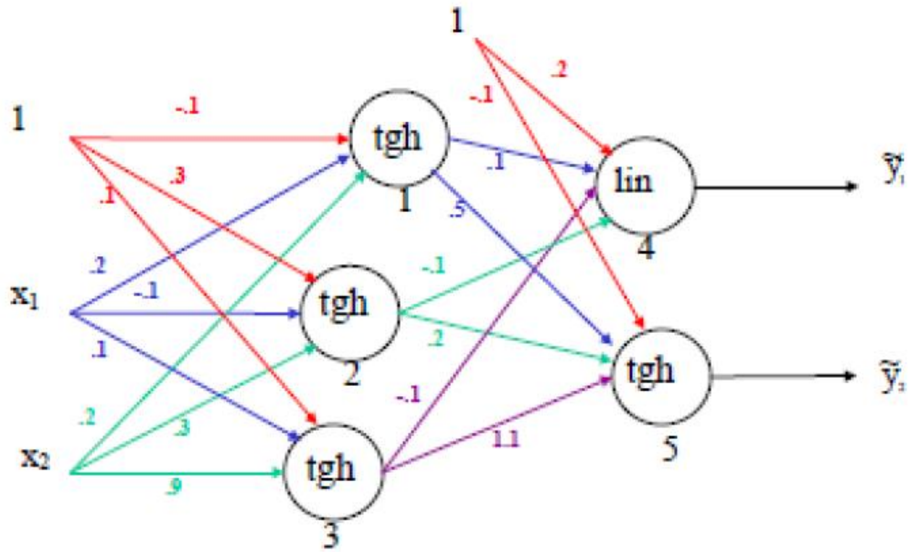
- Arquitectura de la red (*feedforward*)





Redes Neuronales

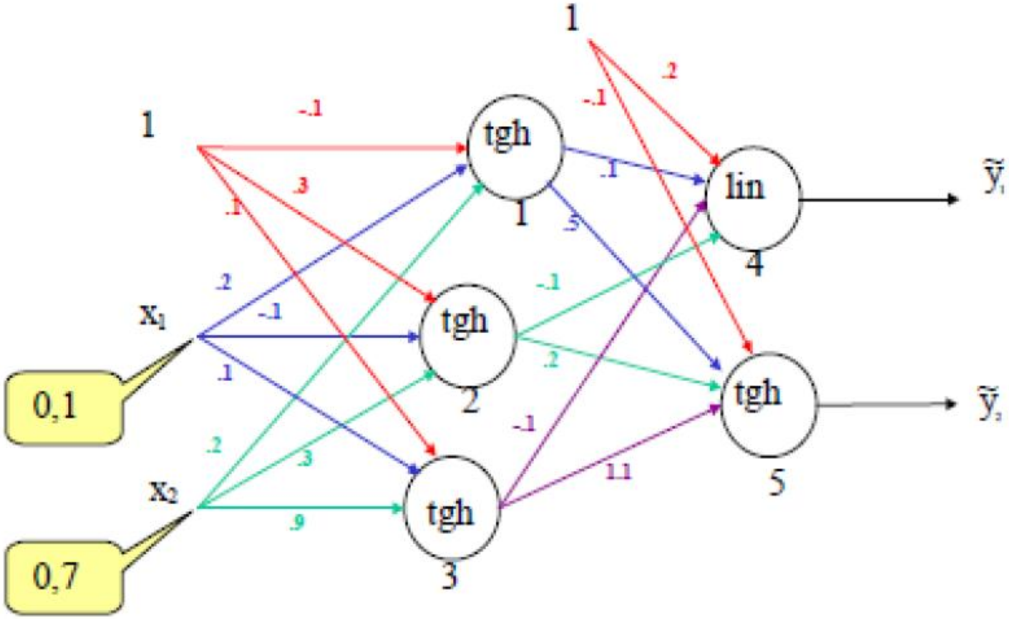
- Ejemplo:



$$\underline{\mathbf{x}} = \begin{bmatrix} 0,1 \\ 0,7 \end{bmatrix} \qquad \tilde{\mathbf{y}} = ?$$

Redes Neuronales

- Ejemplo:

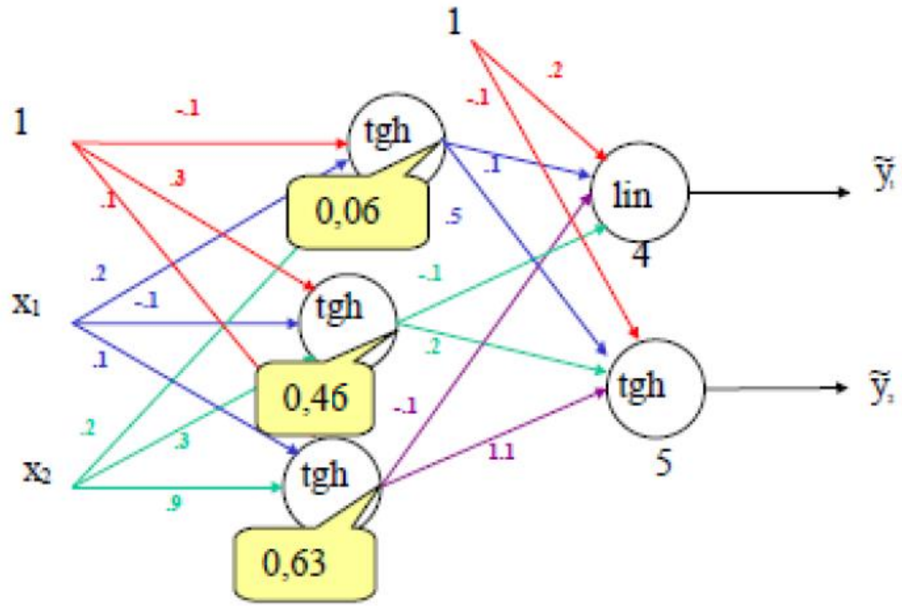


$$u_1 = -0{,}1 + (0{,}2)(0{,}1) + (0{,}2)(0{,}7) = 0{,}06$$

$$v_1 = \text{tgh}(0{,}06) = 0{,}06$$

Redes Neuronales

- Ejemplo:

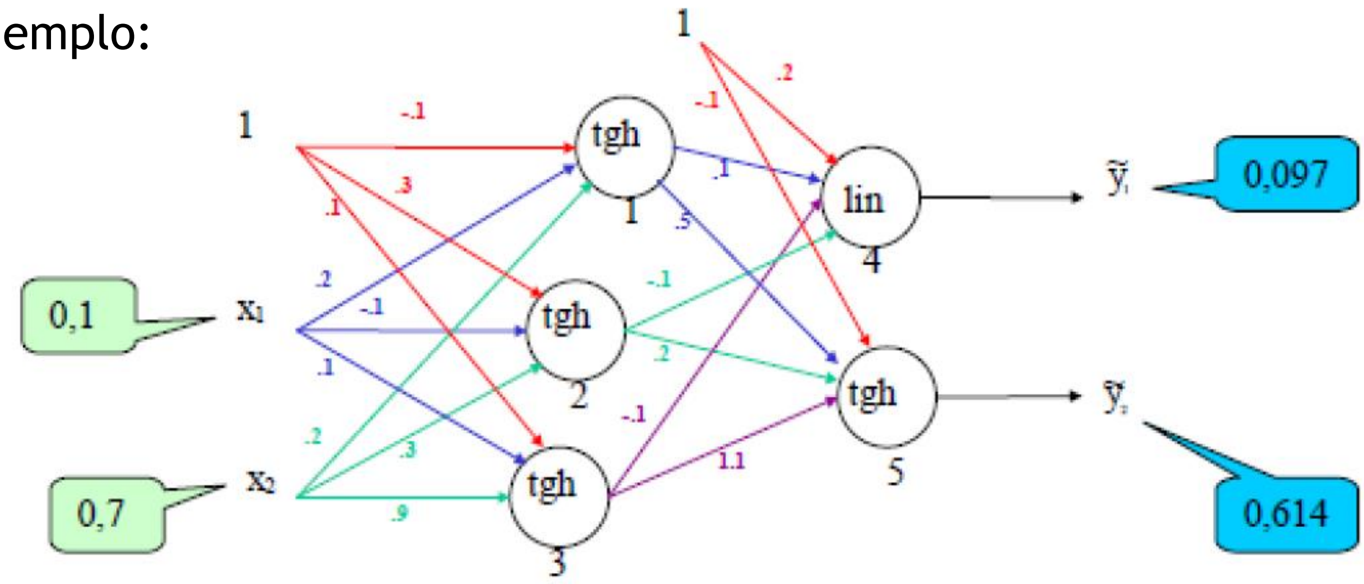


$$u_4 = 0{,}2 + (0{,}1)(0{,}06) + (-0{,}1)(0{,}46) + (-0{,}1)(0{,}63) = 0{,}097$$

$$v_4 = 0{,}097 \quad \text{(linear!)}$$

Redes Neuronales

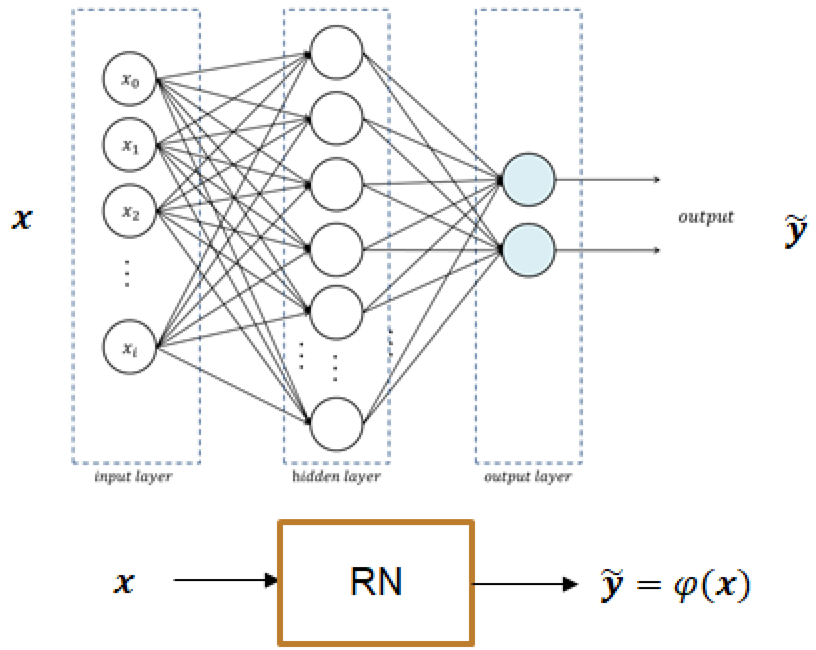
- Ejemplo:



$$\underline{\mathbf{x}} = \begin{bmatrix} 0{,}1 \\ 0{,}7 \end{bmatrix} \qquad \tilde{\mathbf{y}} = \begin{bmatrix} 0{,}097 \\ 0{,}614 \end{bmatrix}$$

Redes Neuronales

- Las redes neuronales *feedforward* permiten aproximar relaciones no lineales entre los datos de entrada y salida



Entrenamiento *backpropagation*

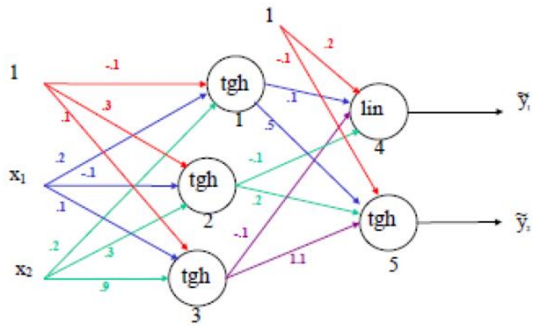
- La red se entrena a partir de ejemplos

Pares entrada – salida (filas)

$\boldsymbol{x}^k$			$\boldsymbol{y}^k$		
x_1	x_2	…	y_1	y_2	…
2	45	…	3	6	…
…	…	…	…	…	…

$$\boldsymbol{x}^1 = \begin{bmatrix} x_1 \\ x_2 \\ \dots \end{bmatrix} = \begin{bmatrix} 2 \\ 45 \\ \dots \end{bmatrix}$$

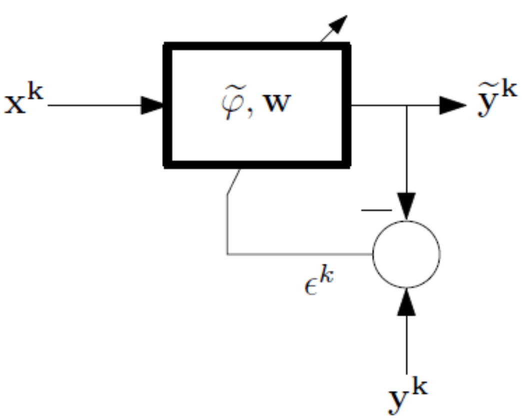
$$\boldsymbol{y}^1 = \begin{bmatrix} y_1 \\ y_2 \\ \dots \end{bmatrix} = \begin{bmatrix} 3 \\ 6 \\ \dots \end{bmatrix}$$



$\epsilon_1 = y_1 - \tilde{y}_1$

$\epsilon_2 = y_2 - \tilde{y}_2$

Aprendizaje: minimización del error en la salida



Error a minimizar:

$$\varepsilon^{2^k} = ||\mathbf{y}^k - \tilde{\mathbf{y}}^k||^2 = \sum_{l=1}^{m} (\mathbf{y}_l^k - \tilde{\mathbf{y}}_l^k)^2 \quad \text{Para todas las neuronas} \qquad \epsilon^{2^k} = \epsilon_1^2 + \epsilon_2^2$$

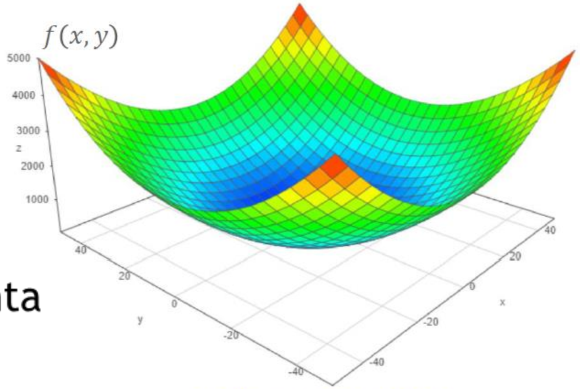
$$F_o = E[\varepsilon^{2^k}] = \frac{1}{P} \sum_{k=1}^{P} \varepsilon^{2^k} = F_o(\mathbf{w}) \geq 0 \quad \text{Para todos los pares}$$

Entrenamiento *backpropagation*

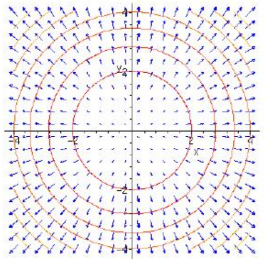
- Objetivo: Minimizar $F_o(\boldsymbol{w}) = E[{\epsilon^2}^k]$
- Se utiliza el algoritmo del **gradiente desdendente**
    - El gradiente es la generalización de derivada para funciones de más de una variable
    - Ejemplo: Sea $f(x,y)$ una función de dos variables. Su gradiente es un vector compuesto por las derivadas parciales:

$$f(x,y) = x^2 + y^2$$

$$\left.\begin{array}{l} \dfrac{\partial f}{\partial x} = 2x \\ \dfrac{\partial f}{\partial y} = 2y \end{array}\right. \Rightarrow \nabla f(x,y) = \begin{bmatrix} 2x \\ 2y \end{bmatrix}$$



    - En cada punto del plano XY, el vector gradiente apunta en la dirección de máximo crecimiento de $f(x,y)$

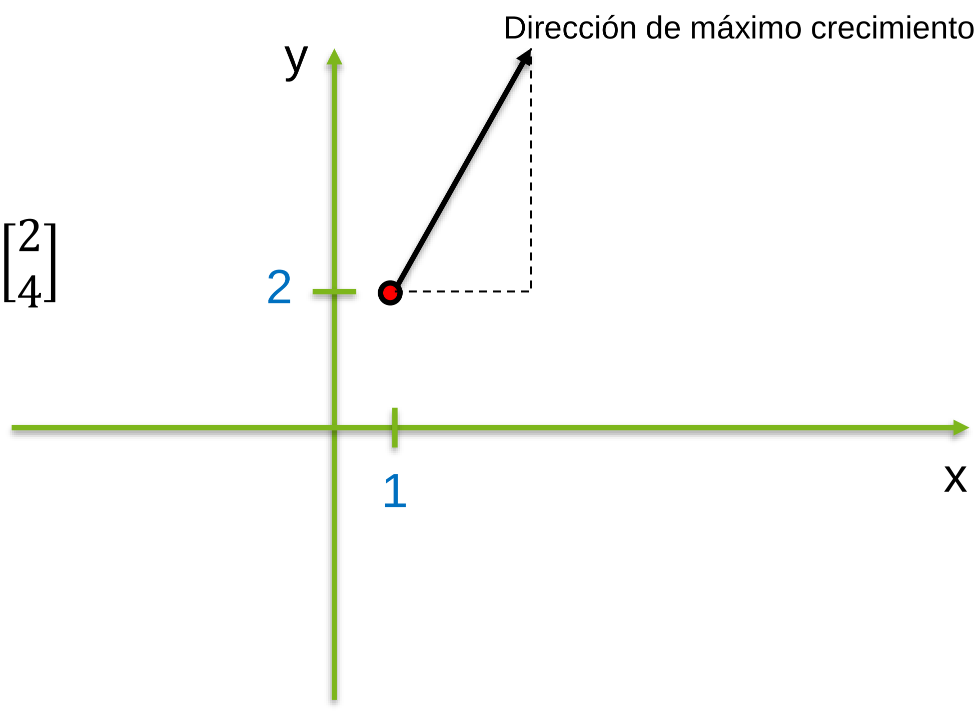


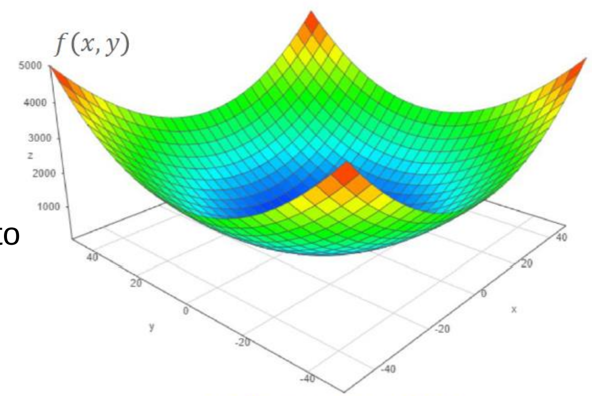
Entrenamiento *backpropagation*

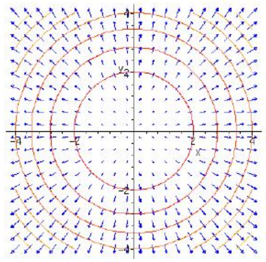
$$f(x, y) = x^2 + y^2$$

$$\left.\begin{aligned}\frac{\partial f}{\partial x} &= 2x \\ \frac{\partial f}{\partial y} &= 2y\end{aligned}\right. \Rightarrow \nabla f(x, y) = \begin{bmatrix} 2x \\ 2y \end{bmatrix}$$

$$\nabla f(1,2) = \begin{bmatrix} 2 \\ 4 \end{bmatrix}$$

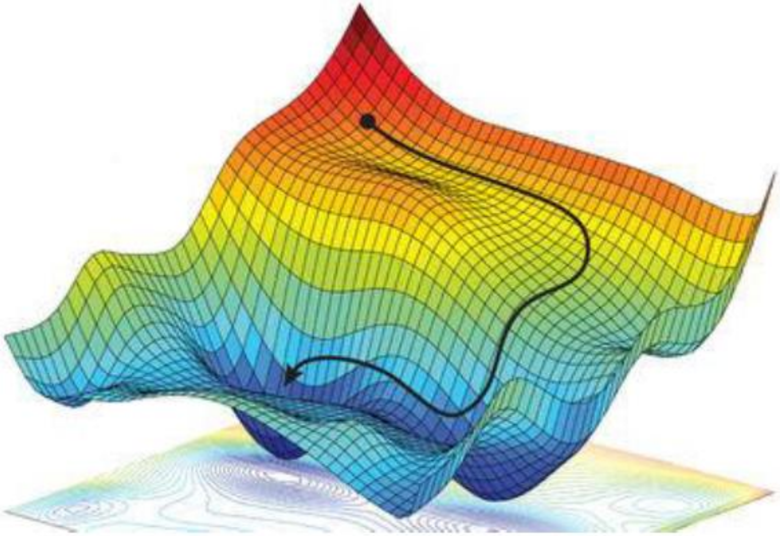






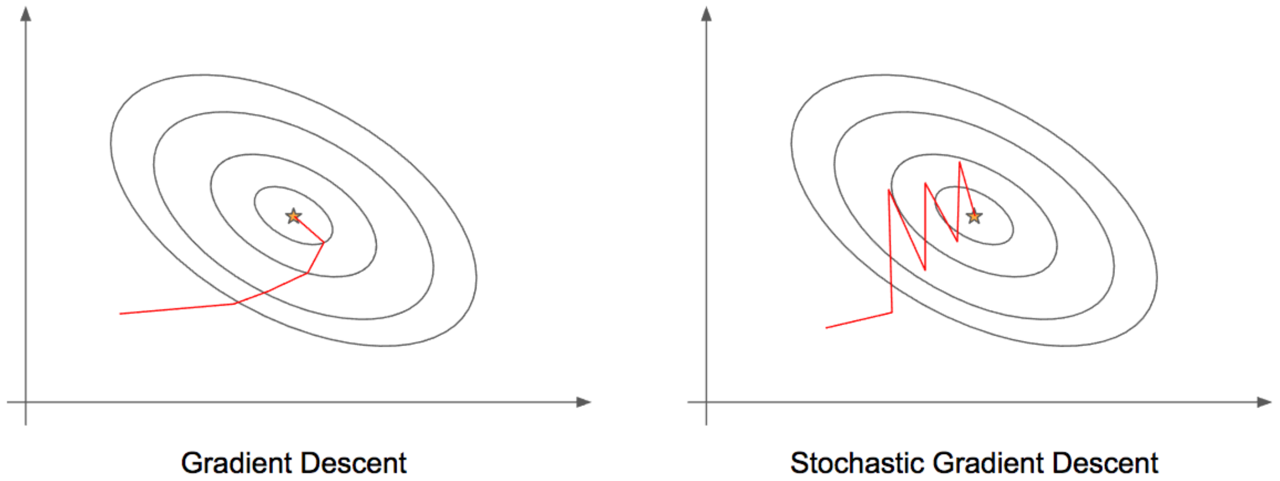
Entrenamiento *backpropagation*

- Objetivo: Minimizar $F_o(\boldsymbol{w}) = E[\epsilon^{2^k}]$
- Se utiliza el algoritmo del **gradiente desdendente**



Entrenamiento batch

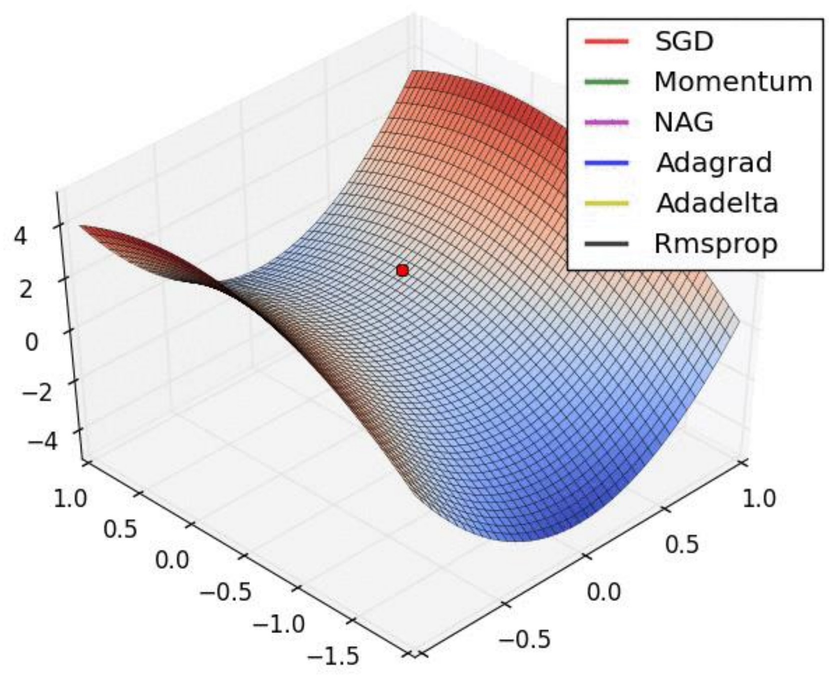
- Para acelerar el entrenamiento, suelen utilizarse subconjuntos de datos (*batches*) durante el proceso del gradiente descendente.
- Existen dos variaciones del algoritmo del gradiente descendente:
    - **Stochastic Gradient Descent.** Batch Size = 1
    - **Mini-Batch Gradient Descent.** 1 < Batch Size < Size of Training Set. El tamaño de los batches suele ser de 32, 64 o 128 muestras



Entrenamiento batch

Existen múltiples algoritmos de optimización, cada una con diferentes maneras de llegar al punto mínimo.

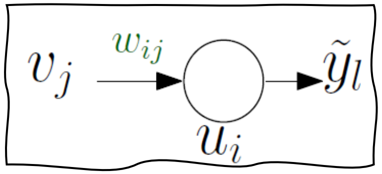
- Gradient Descent
- Stochastic Gradient Descent
- Mini-Batch Gradient Descent
- Momentum
- Nesterov Accelerated Gradient
- Adagrad
- AdaDelta
- Adam
- (…)



Entrenamiento *backpropagation*

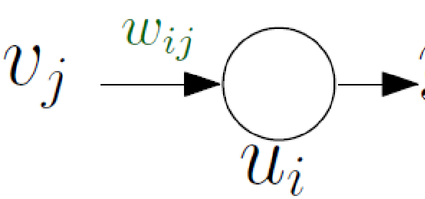
α: *learning rate*

- Objetivo: Minimizar $F_o(\boldsymbol{w}) = E[\epsilon^{2^k}]$
- Se utiliza el algoritmo del **gradiente desdendente:** $\mathbf{w} \rightarrow \mathbf{w} + \Delta\mathbf{w}$; $\Delta w_{ij} = -\alpha \frac{\partial F_o}{\partial w_{ij}}$



Entrenamiento *backpropagation*

- En resumen:



$$\tilde{y}_l = s(u) = s(v \cdot w)$$

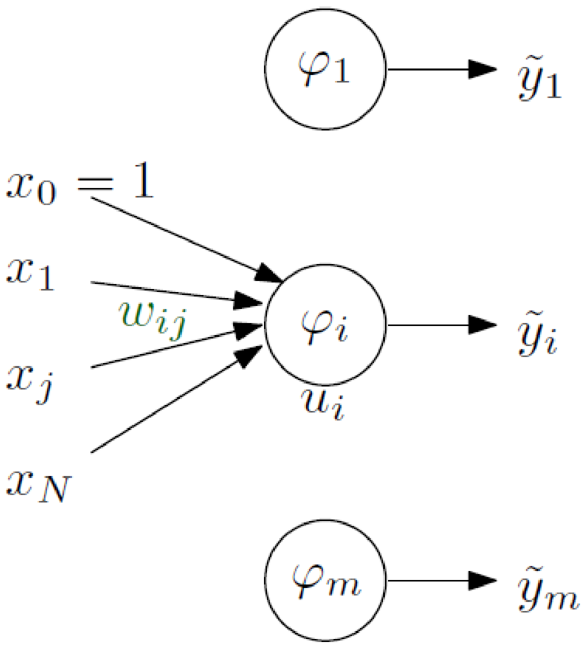
$$\tilde{y} = tgh(u) \Rightarrow \frac{\partial \tilde{y}_l}{\partial u_i} = 1 - \tilde{y}^2$$

$$\boxed{\Delta w_{ij}^p = 2\alpha v_j \delta_i}$$

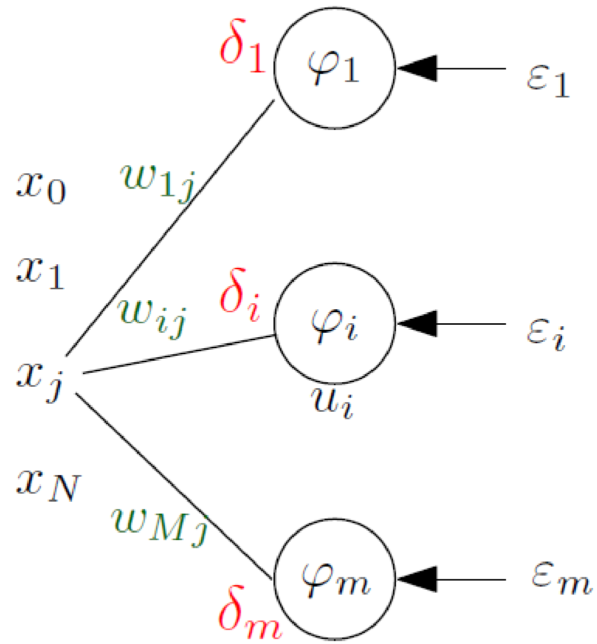
$$\delta_i = \varepsilon_i \frac{\partial \tilde{y}_l}{\partial u_i}$$

$$v_j = \frac{\partial u_i}{\partial w_{ij}}$$

Entrenamiento *backpropagation*



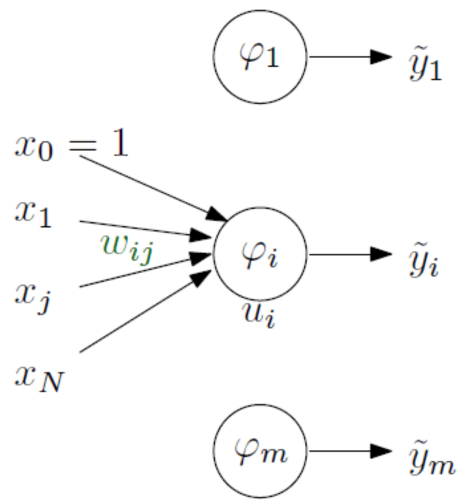
Red original



Red asociada y retropropagación del error

Redes de una capa

- Para cada par (x,y)



$$u_i = \sum_{j=0}^{N} w_{ij} x_j$$

$$\tilde{y}_i = \varphi_i(u_i) = \begin{cases} u_i & \text{(neurona lineal)} \\ tgh(u_i) & \text{(neurona tgh)} \end{cases}$$

- Error retropropagado:

$$\delta_i = \begin{cases} \varepsilon_i \cdot 1 & \text{(neurona lineal)} \\ \varepsilon_i \cdot (1 - \tilde{y}_i^2) & \text{(neurona tgh)} \end{cases}$$

- Incremento en cada sinapsis debido al par *p*: $\Delta w_{ij}^{p} = 2\alpha x_j \delta_i$

Redes de una capa (ejemplo numérico)

- Vamos a implementar una red neuronal con una capa y dos neuronas, la primera de tipo lineal y la segunda de tipo tangente hiperbólica

- Se presenta el par de entrada-salida (**x**,**y**), donde **x** = $[0.1 \;\; 0.7]^T$ e **y** = $[0.2 \;\; 1]^T$

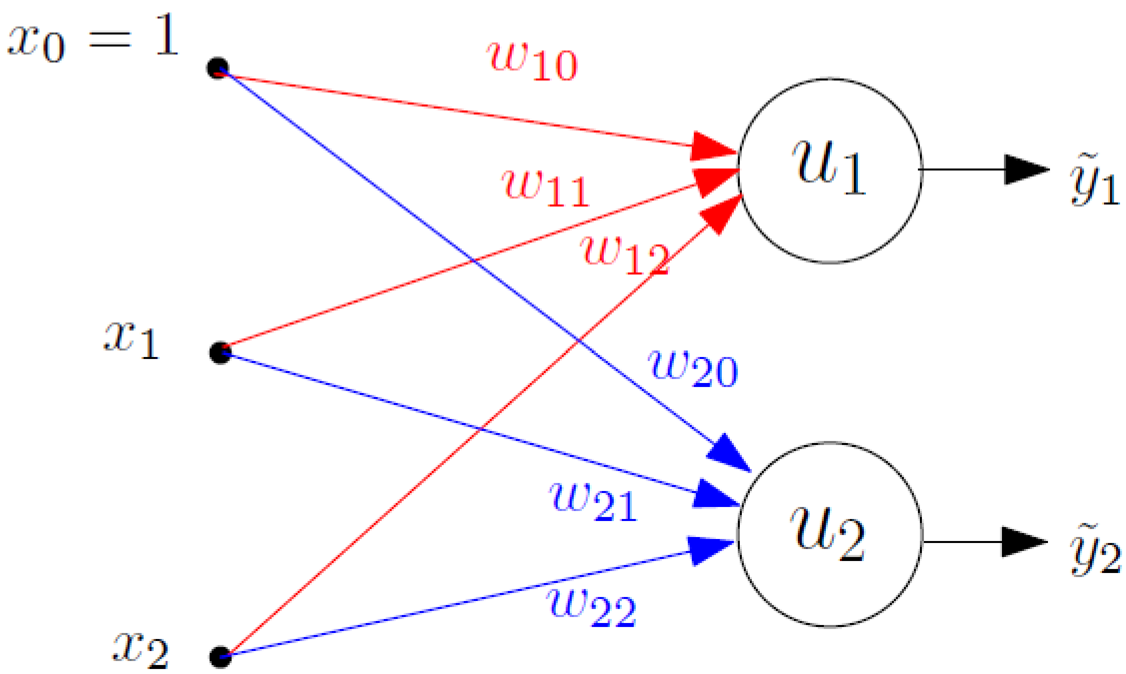
- $\alpha = 0.1$ y El valor inicial de las sinapsis es:

$$
\begin{aligned}
w_{10} &= 0.1 \\
w_{11} &= -0.2 \\
w_{12} &= 0.3 \\
w_{20} &= 0.5 \\
w_{21} &= 0.4 \\
w_{22} &= -0.6
\end{aligned}
$$

- Vamos a calcular los nuevos valores de las sinapsis tras el primer paso del entrenamiento

Redes de una capa (ejemplo numérico)

1) Dibuja la red neuronal



Redes de una capa (ejemplo numérico)

2) Calcula los valores u_1 y u_2

$$u_1 = w_{10} \cdot 1 + w_{11} \cdot x_1 + w_{12} \cdot x_2 = 0.1 \cdot 1 + (-0.2) \cdot 0.1 + 0.3 \cdot 0.7 = 0.29$$
$$u_2 = w_{20} \cdot 1 + w_{21} \cdot x_1 + w_{22} \cdot x_2 = 0.5 \cdot 1 + 0.4 \cdot 0.1 + (-0.6) \cdot 0.7 = 0.12$$

3) Calcula los valores $\hat{y}_1$ e $\hat{y}_2$

$$\tilde{y}_1 = u_1 = 0.29$$
$$\tilde{y}_2 = tgh(u_2) = tgh(0.12) \approx 0.12$$

4) Calcula los errores ε_1 y ε_2

$$\varepsilon_1 = y_1 - \tilde{y}_1 = 0.2 - 0.29 = -0.09$$
$$\varepsilon_2 = y_2 - \tilde{y}_2 = 1 - 0.12 = 0.88$$

Redes de una capa (ejemplo numérico)

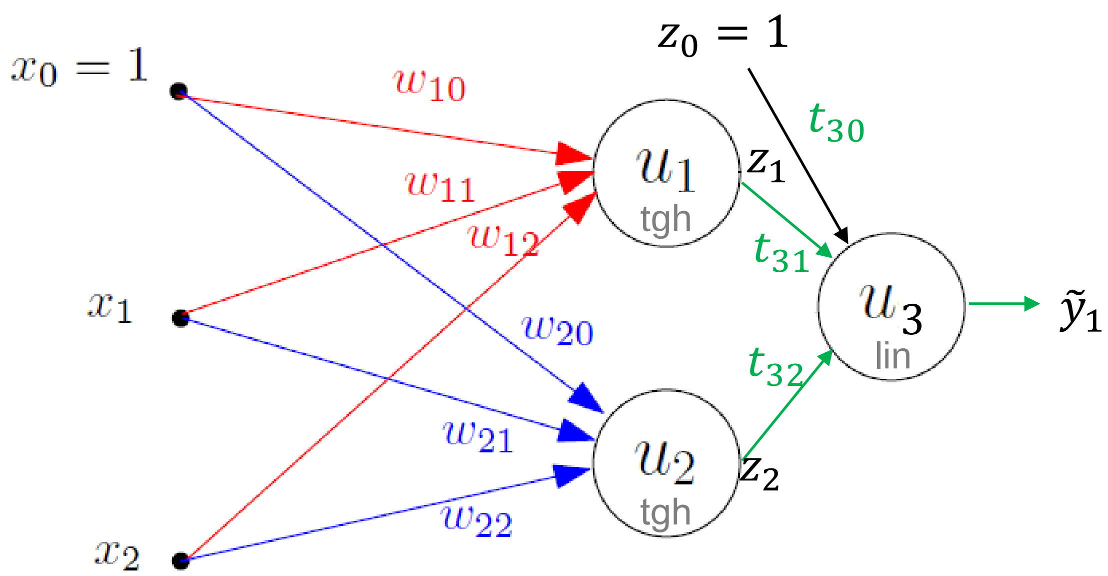
5) Calcula los errores retropropagados δ_1 y δ_2

$$\delta_1 = \varepsilon_1 = -0.09$$
$$\delta_2 = \varepsilon_2(1 - \tilde{y}_2^2) = 0.88 \cdot (1 - 0.12^2) = 0.87$$

6) Calcula el nuevo valor de las sinapsis $w_{ij} \rightarrow w_{ij} + \Delta w_{ij}$

$$w_{10} \rightarrow w_{10} + 2\alpha x_0 \delta_1 = 0.1 + 2 \cdot 0.1 \cdot 1 \cdot (-0.09) = 0.1 - 0.018 = 0.082$$
$$w_{11} \rightarrow w_{11} + 2\alpha x_1 \delta_1 = -0.2 + 2 \cdot 0.1 \cdot 0.1 \cdot (-0.09) = -0.2 - 0.0018 = -0.2018$$
$$w_{12} \rightarrow w_{12} + 2\alpha x_2 \delta_1 = 0.3 + 2 \cdot 0.1 \cdot 0.7 \cdot (-0.09) = 0.3 - 0.0126 = 0.2874$$
$$w_{20} \rightarrow w_{20} + 2\alpha x_0 \delta_2 = 0.5 + 2 \cdot 0.1 \cdot 1 \cdot (0.87) = 0.5 + 0.174 = 0.674$$
$$w_{21} \rightarrow w_{21} + 2\alpha x_1 \delta_2 = 0.4 + 2 \cdot 0.1 \cdot 0.1 \cdot (0.87) = 0.4 + 0.0174 = 0.4174$$
$$w_{22} \rightarrow w_{22} + 2\alpha x_2 \delta_2 = -0.6 + 2 \cdot 0.1 \cdot 0.7 \cdot (0.87) = -0.6 + 0.1218 = -0.4784$$

Redes de dos capas (ejemplo)

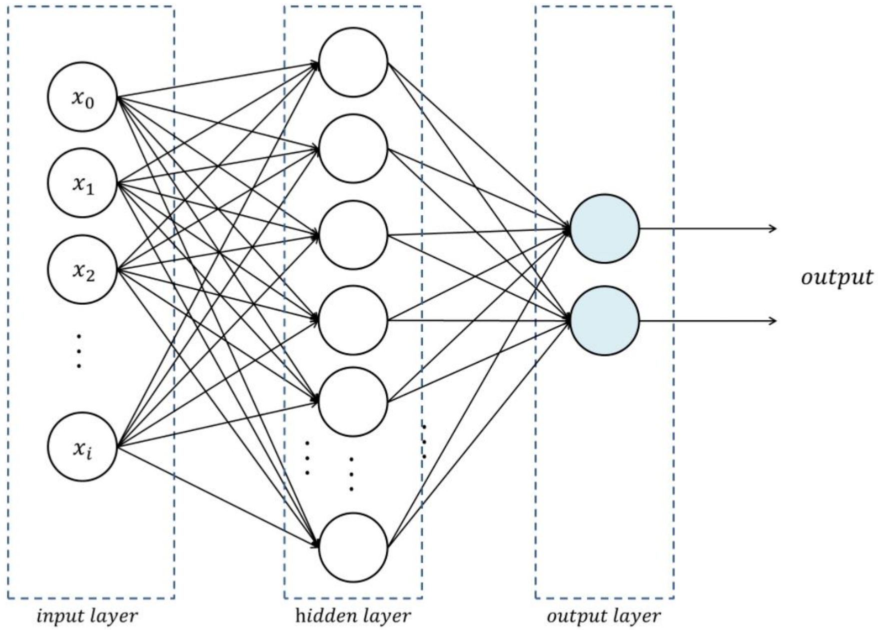


$$\delta_3 = \epsilon_1$$
$$t_{31} = t_{31} + 2\alpha z_1 \delta_3$$
$$t_{32} = t_{32} + 2\alpha z_2 \delta_3$$
$$t_{30} = t_{30} + 2\alpha z_0 \delta_3$$

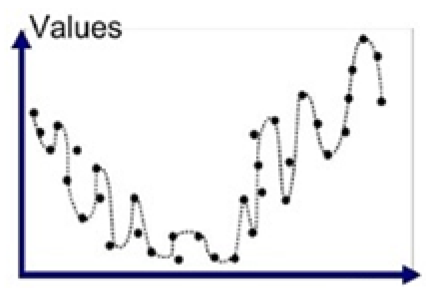
$$\gamma_1 = (1 - z_1^2)\, \delta_3 t_{31}$$
$$\gamma_2 = (1 - z_2^2)\, \delta_3 t_{32}$$

$$w_{10} = w_{10} + 2\alpha x_0 \gamma_1$$
$$w_{11} = w_{11} + 2\alpha x_1 \gamma_1$$
$$w_{12} = w_{12} + 2\alpha x_2 \gamma_1$$
$$w_{20} = w_{20} + 2\alpha x_0 \gamma_2$$
$$w_{21} = w_{21} + 2\alpha x_1 \gamma_2$$
$$w_{22} = w_{22} + 2\alpha x_2 \gamma_2$$

Dimensionando las Redes Neuronales



- Red sub-dimensionada
  - No tiene capacidad de representar el mapeo $x \rightarrow y$
  - No aprende, el error se mantiene elevado
- Red sobredimensionada
  - Lenta
  - Tendencia al overfitting



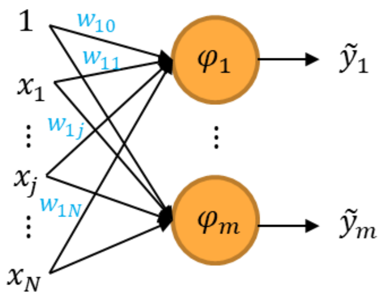
¿Cuántas capas de neuronas?

¿Cuántas neuronas por capa?

¿Qué tipo de neurona?

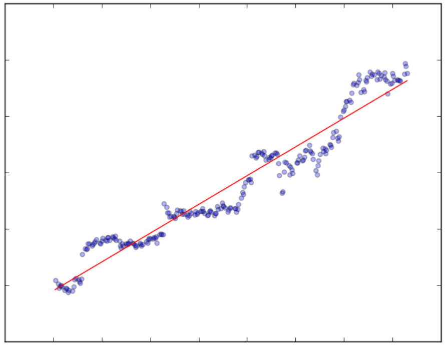
Dimensionando las Redes Neuronales

Redes de una capa



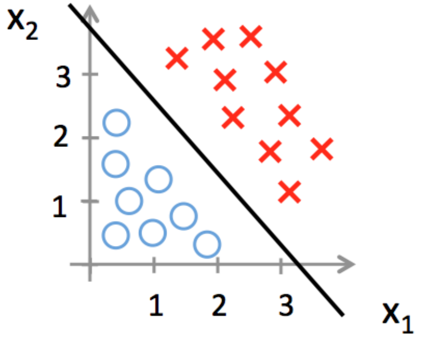
- Si las neuronas son lineales, la red es un regresor lineal

$$\tilde{y}_1 = \sum x_j w_{1j} + w_{1o} = w_{10} + w_{11}x_1 + \cdots + w_{1j}x_j + \cdots + w_{1N}x_N$$



- Si las neuronas son de tipo tgh(), la salida es un clasificador lineal

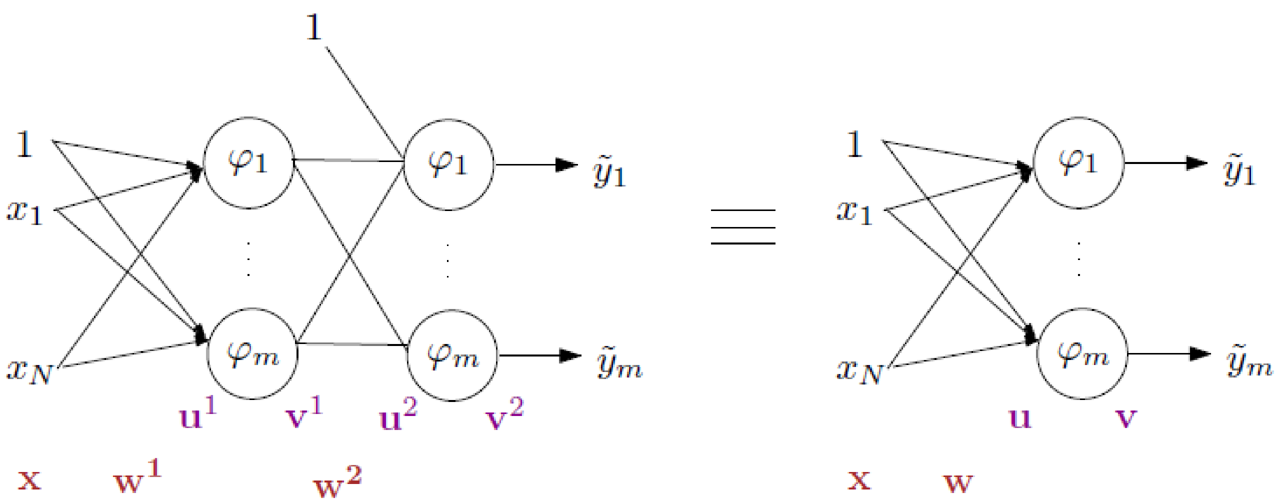
$$\tilde{y}_1 = \mathrm{tgh}(\sum x_j w_{1j} + w_{1o})$$



Dimensionando las Redes Neuronales

Redes de dos capas: Multilayer perceptron (MLP)

- Si todas las neuronas son lineales, la red es idéntica a una red con una sola capa



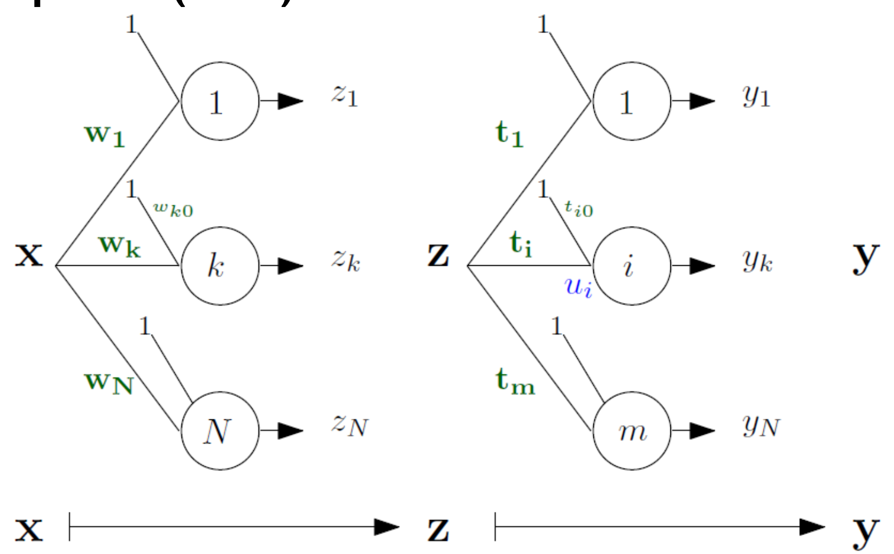
$$\mathbf{u}^2 = \mathbf{w}^2\mathbf{v}^1 = \mathbf{w}^2\mathbf{w}^1\mathbf{x}$$

$$\mathbf{u} = \mathbf{w}\mathbf{x} = \mathbf{w}^2\mathbf{w}^1\mathbf{x}$$

Dimensionando las Redes Neuronales

Redes de dos capas: Multilayer perceptron (MLP)

- La capa intermedia debe tener neuronas de tipo tgh()
- Teorema de aproximación universal: Si una funcion es L^2 ($\int |f|^2 < \infty$) en un dominio, entonces una serie de tangentes hiperbólicas es un aproximador universal para la función en el dominio.
- Una red neuronal con dos capas, la primera con neuronas de tipo tgh() y la segunda con neuronas lineales es un aproximador universal para todas las funciones de interés práctico



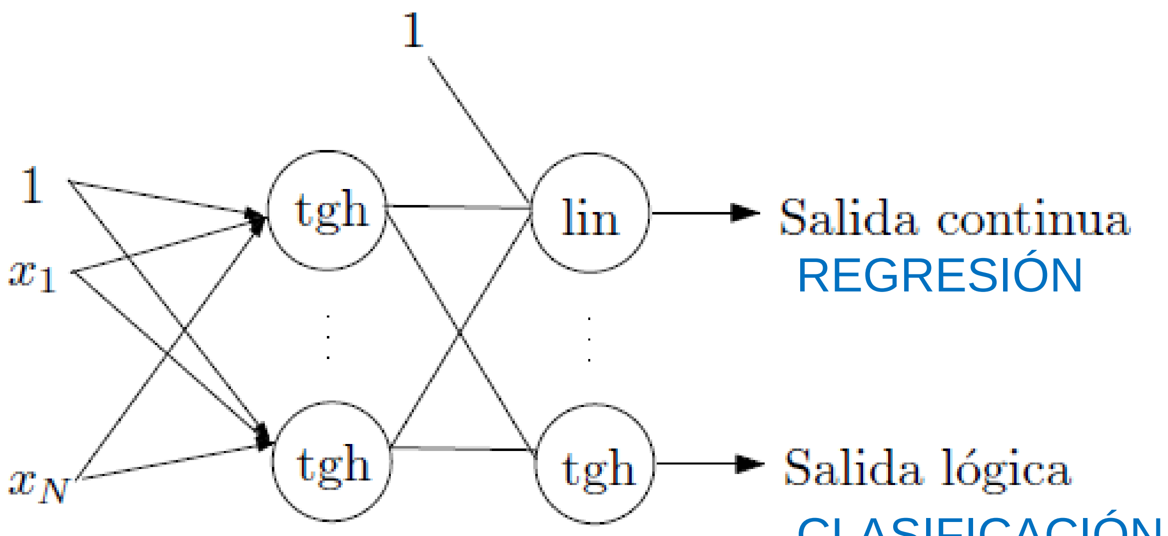
$$z_k = tgh(\mathbf{x}^T \mathbf{w_k} + w_{k0})$$

$$u_i = \mathbf{z^T t_i} + t_{i0} = t_{i0} + \sum_k t_{ik} tgh(\mathbf{x^T w_k} + w_{k0})$$

$$\tilde{y}_i = \begin{cases} u_i & \text{Neurona de salida lineal} \\ tgh(u_i) & \text{Neurona de salida tgh (usada en clasificadores)} \end{cases}$$

Dimensionando las Redes Neuronales

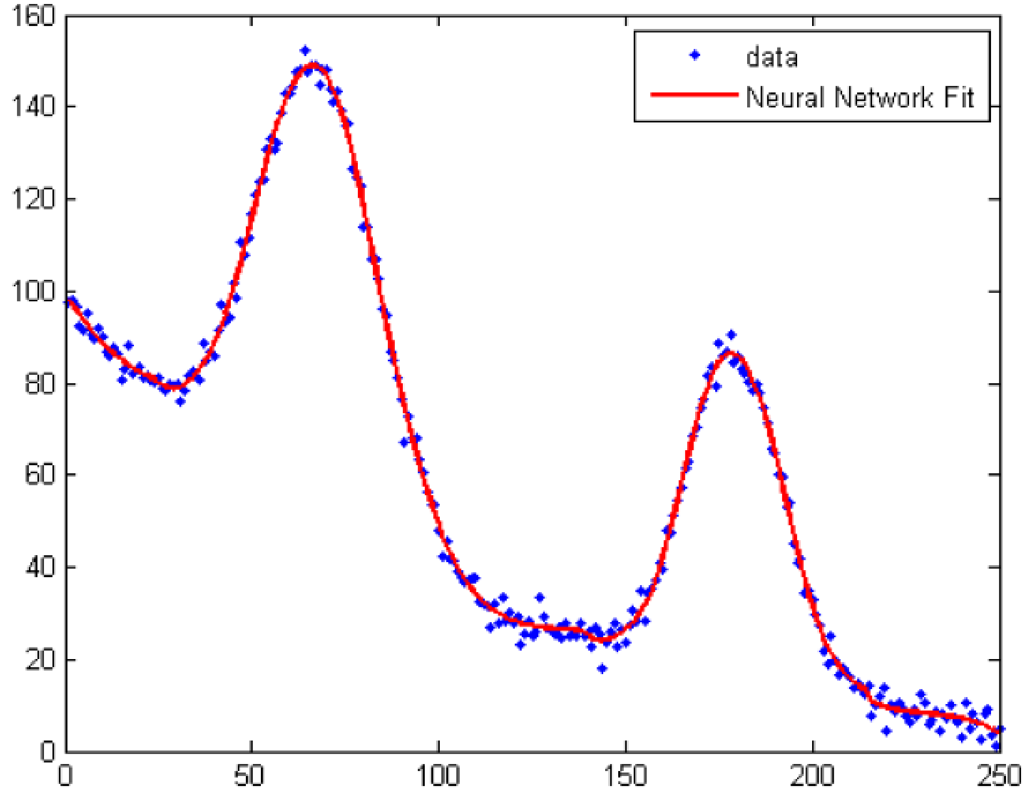
Redes de dos capas: Multilayer perceptron (MLP)



Dimensionando las Redes Neuronales

Redes de dos capas (regresión)

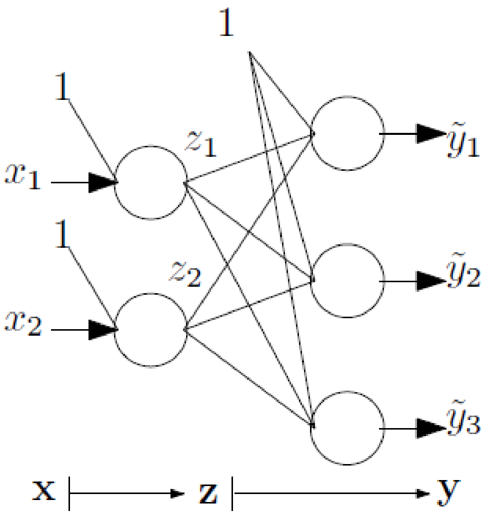
$$\tilde{y}_i = t_{i0} + \sum_k t_{ik} tgh(\mathbf{x}^\mathbf{T}\mathbf{w_k} + w_{k0})$$

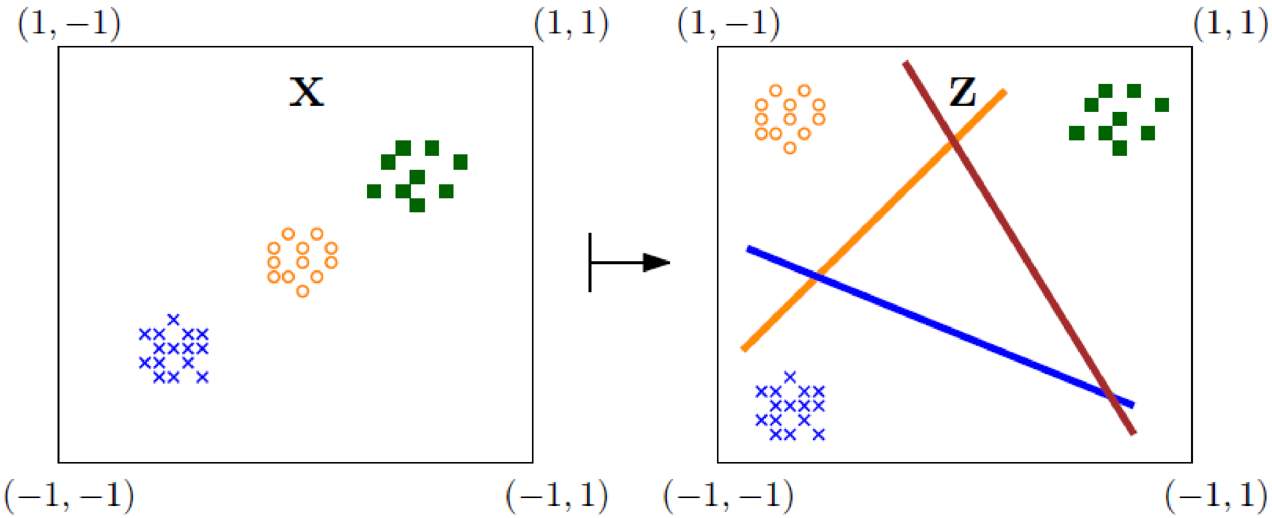


Dimensionando las Redes Neuronales

Redes de dos capas (clasificación)

$$\tilde{y}_i = tgh(t_{i0} + \sum_k t_{ik} tgh(\mathbf{x}^{\mathbf{T}}\mathbf{w_k} + w_{k0}))$$



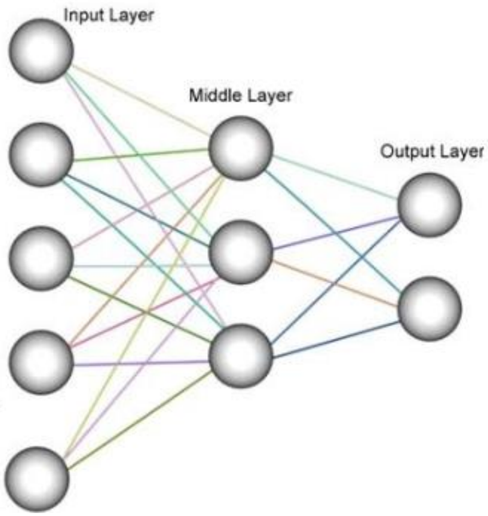


La capa de salida separa clases que sí son linealmente separables en el dominio **z**

La capa intermedia mapea clases no linealmente separables de **x** en clases linealmente separables en **z**

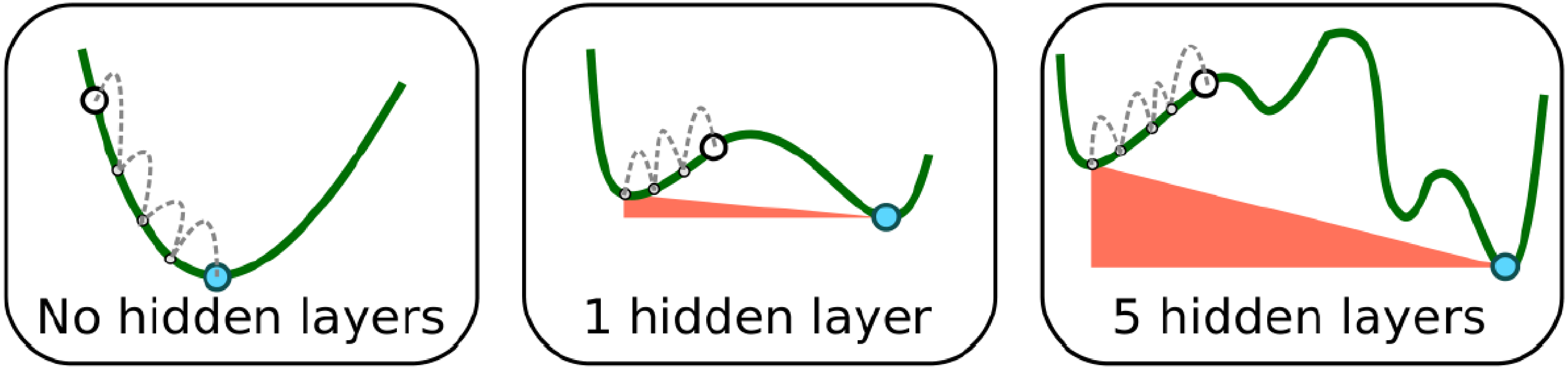
Dimensionando las Redes Neuronales

- Usar una o dos capas
- En la capa de salida:
  - Regresión (salida continua): neurona lineal ($y = u$)
  - Clasificación (salida categórica): neurona tgh ($y = tgh(u)$)
- En la capa intermedia usar siempre neuronas de tipo tgh
- Redes de una capa: si el error es alto, utilizar dos capas
- Redes de dos capas:
  - Número de neuronas en la capa intermedia: m_1
  - Regla general: $n > m_1 > m$
  - Si el error es grande, aumentar m_1
- Redes de tres o más capas no son necesarias



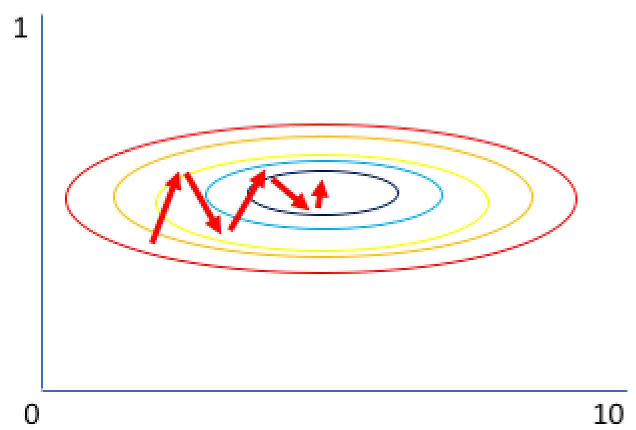
Dimensionando las Redes Neuronales

- El algoritmo del gradiente descendente puede finalizar en un mínimo local de la función de error
- Cuando introducimos más no-linearidad en la red (más capas), se multiplican los mínimos locales

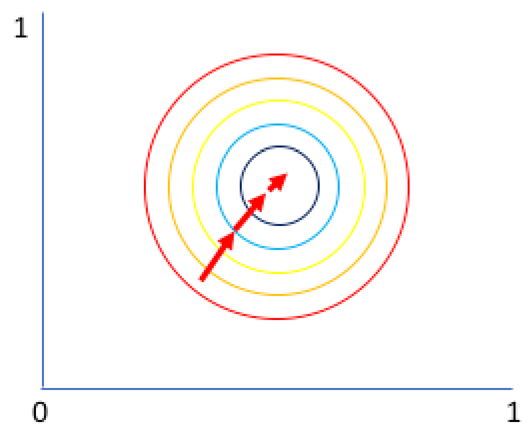


Normalización de variables

- Fundamental para el buen condicionamiento del proceso numérico de optimización
- Queremos valores en el intervalo (-1,+1)



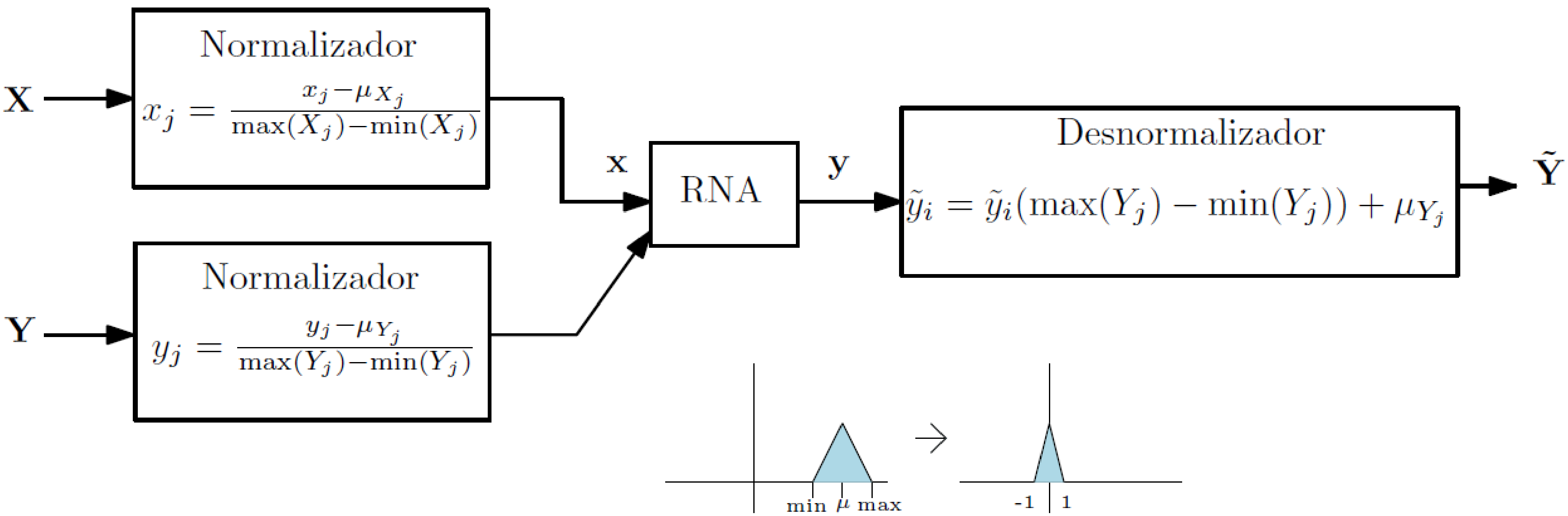
Gradient of larger parameter dominates the update



Both parameters can be updated in equal proportions

Normalización de variables

- Fundamental para el buen condicionamiento del proceso numérico de optimización
- Queremos valores en el intervalo (-1,+1)



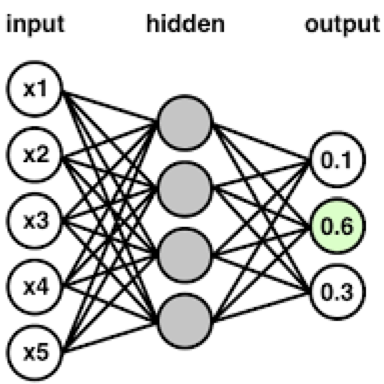
Normalización de variables

- Para variables categóricas, usar el método one-hot encoding

Color
Red
Red
Yellow
Green
Yellow

Red	Yellow	Green
1	0	0
1	0	0
0	1	0
0	0	1

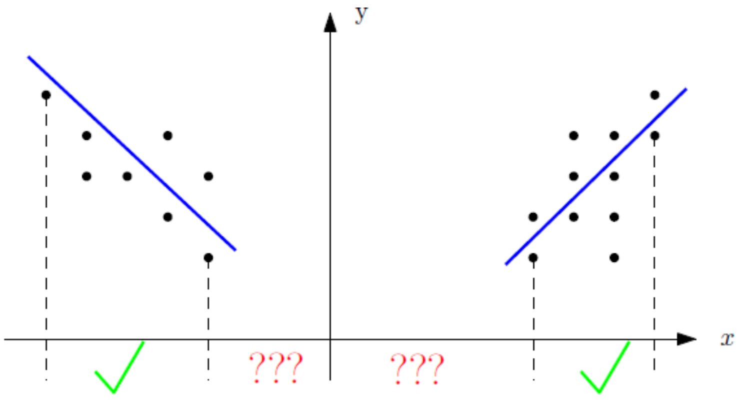
- En clasificadores, las neuronas de salida representan cada una de las categorías



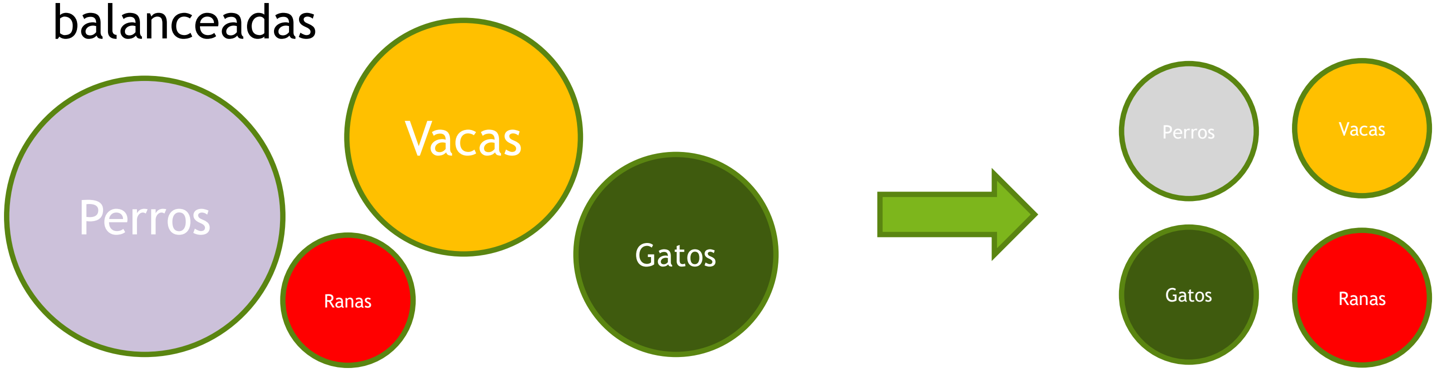
Clase A
Clase B
Clase C

Pares entrada-salida

- El número de pares entrada-salida debe caracterizar bien el dominio de operación



- En problemas de clasificación, es recomendable que las clases estén balanceadas

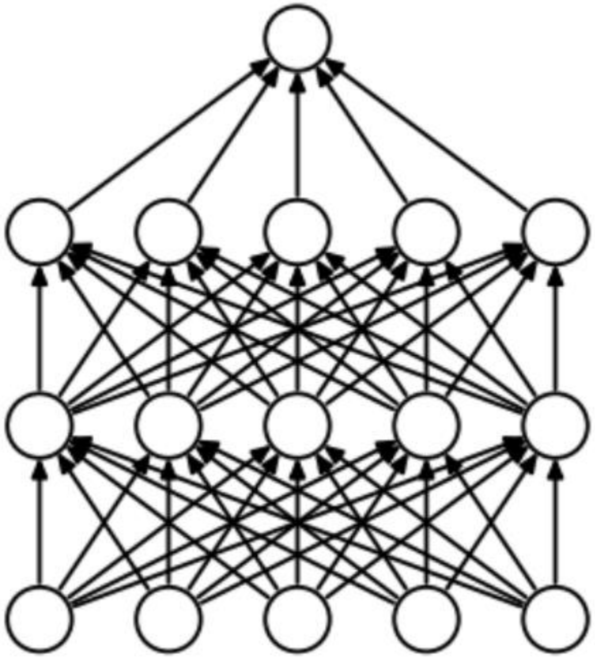


Entrenamiento en Redes Neuronales

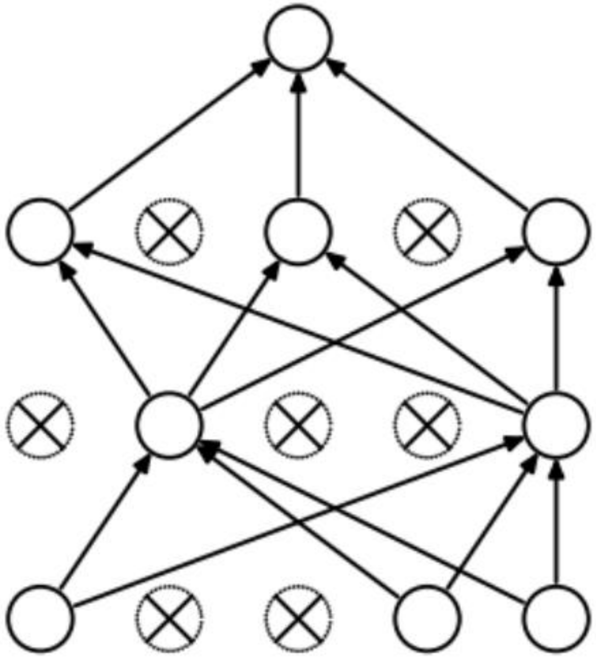
- El conjunto de pares entrada-salida se divide en:
  - Entrenamiento (60%)
  - Validación (20%)
  - Test (20%)

Regularización

- Para evitar el overfitting, a las redes neuronales se les puede aplicar el método *dropout*, que consiste en desactivar neuronas aleatoriamente basándose en una probabilidad



(a) Standard Neural Net



(b) After applying dropout.

Demo

[Demo]