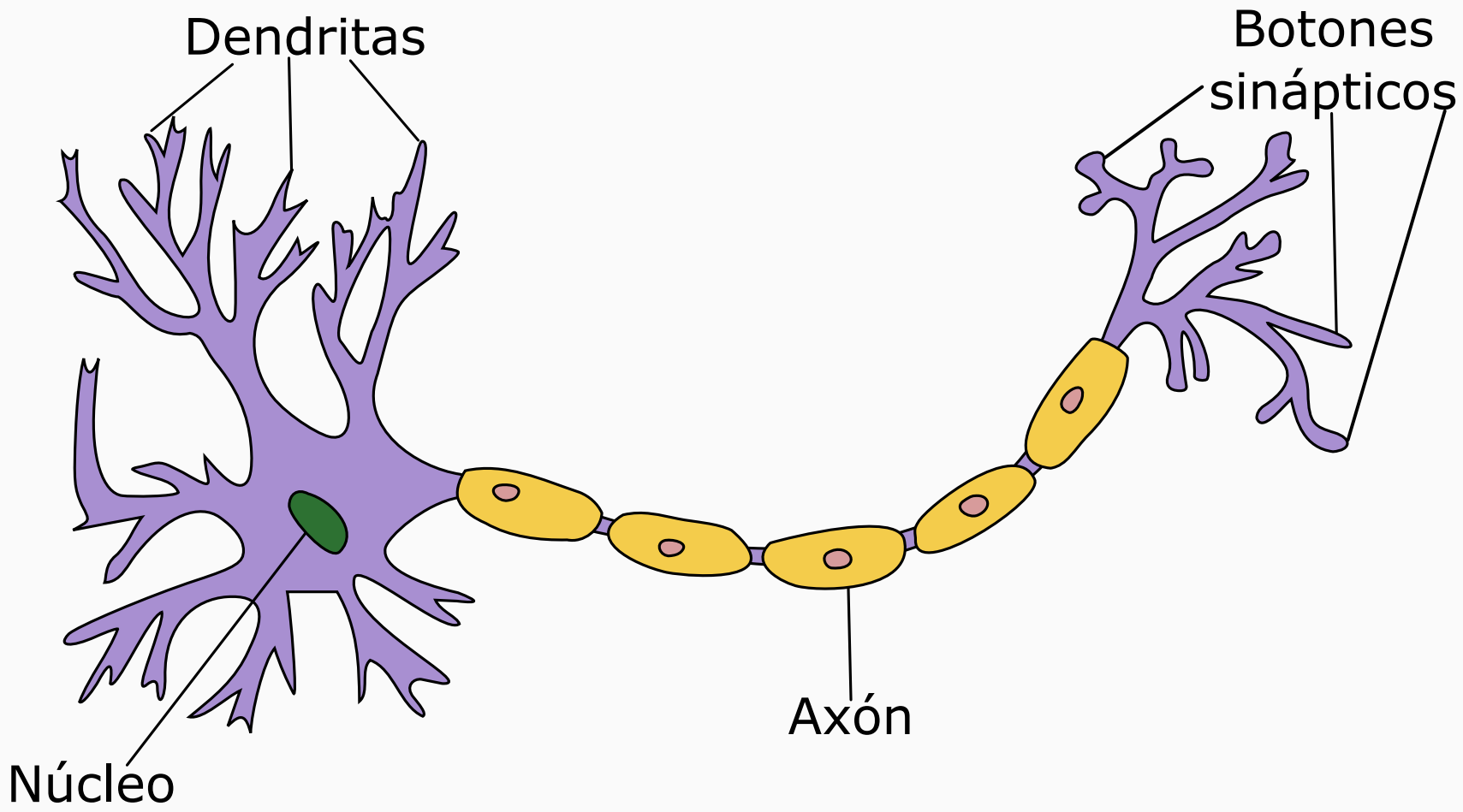
Aprendizaje profundo

Perceptrón multicapa

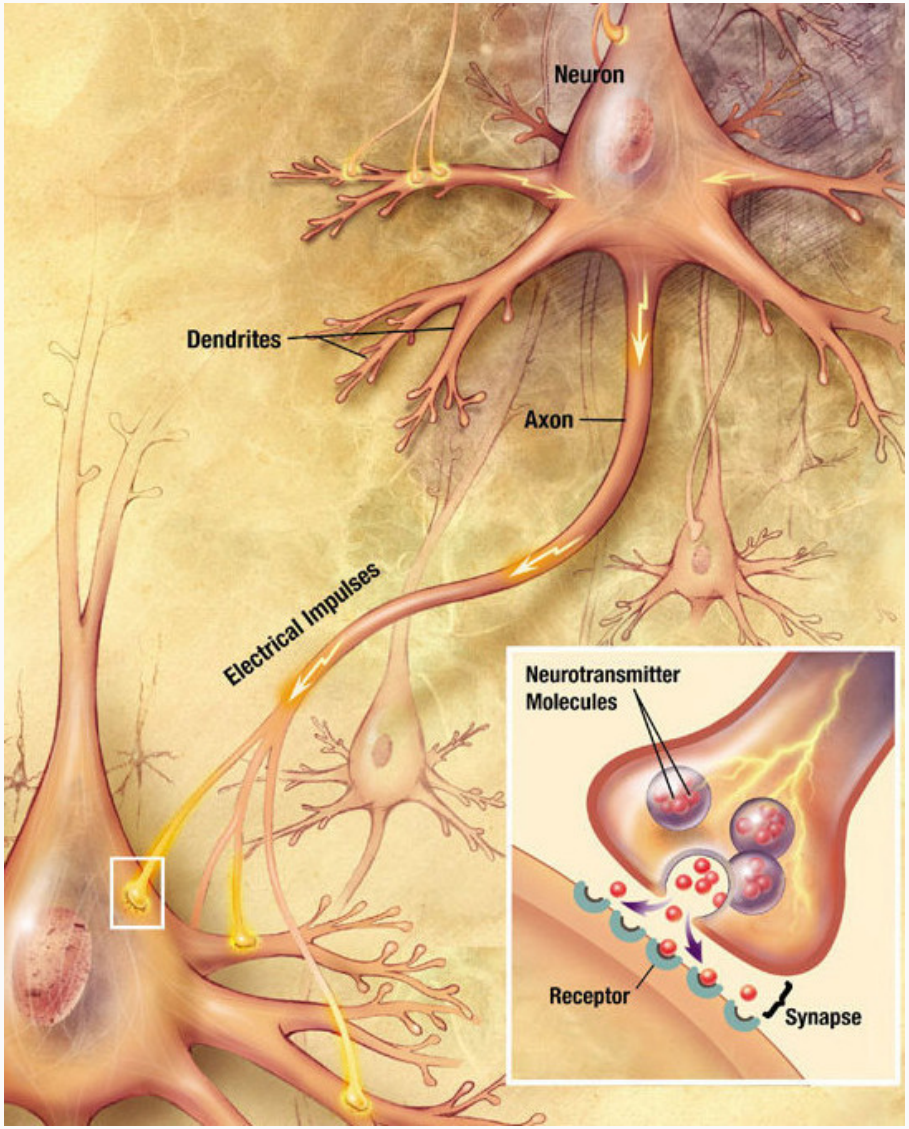
Gibran Fuentes-Pineda

Agosto 2019

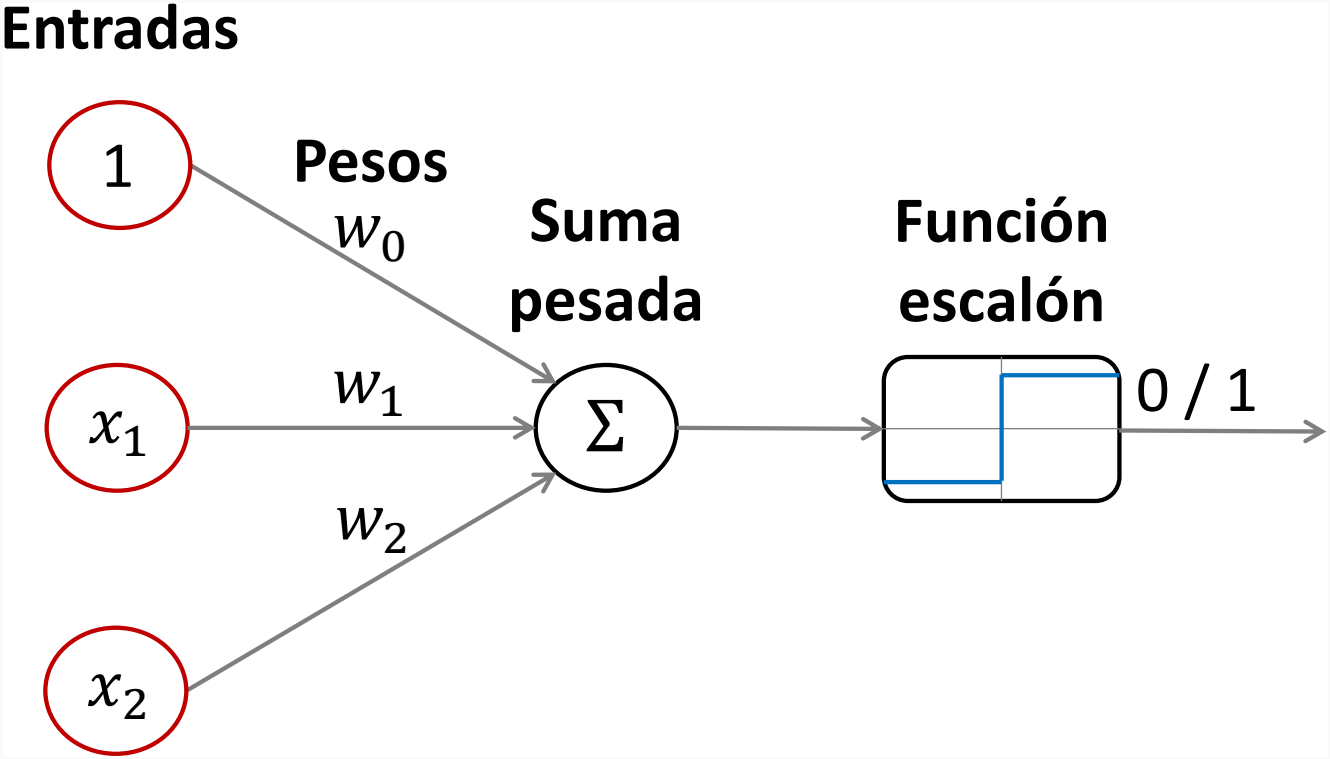
Neurona natural



Comunicación entre neuronas



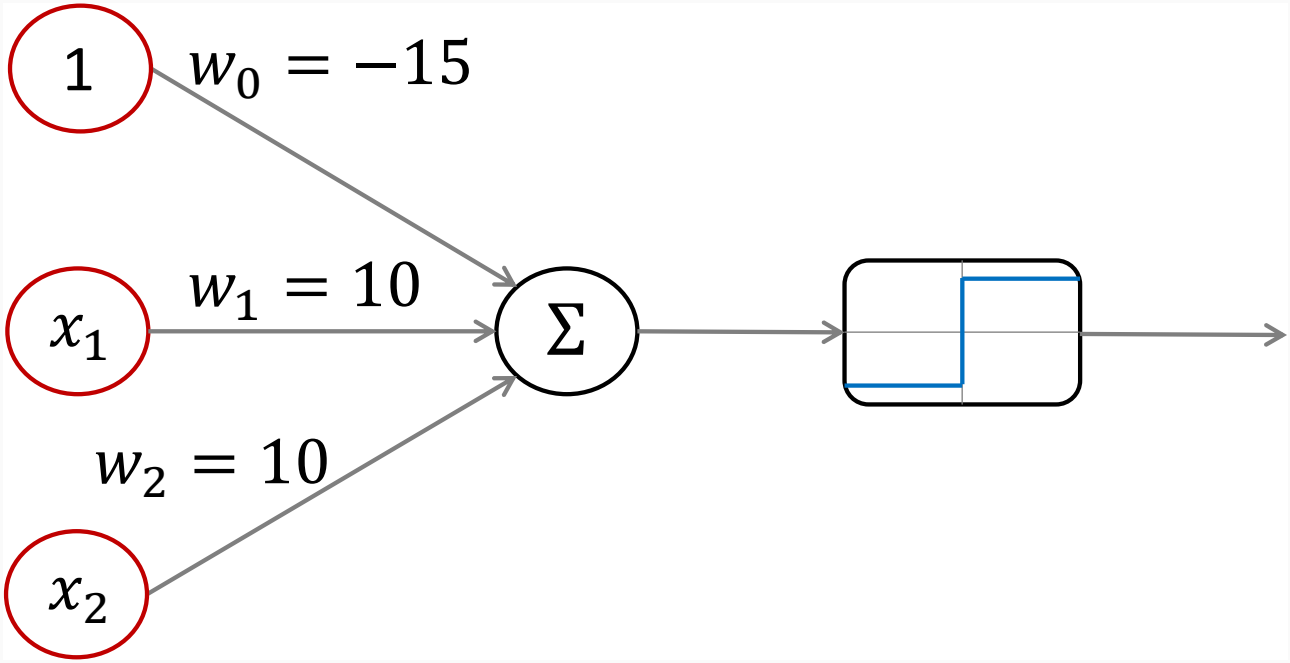
Neurona artificial



- Elementos básicos
  1. Pesos sinápticos
  2. Estímulo cumulativo
  3. Todo o nada (activación)

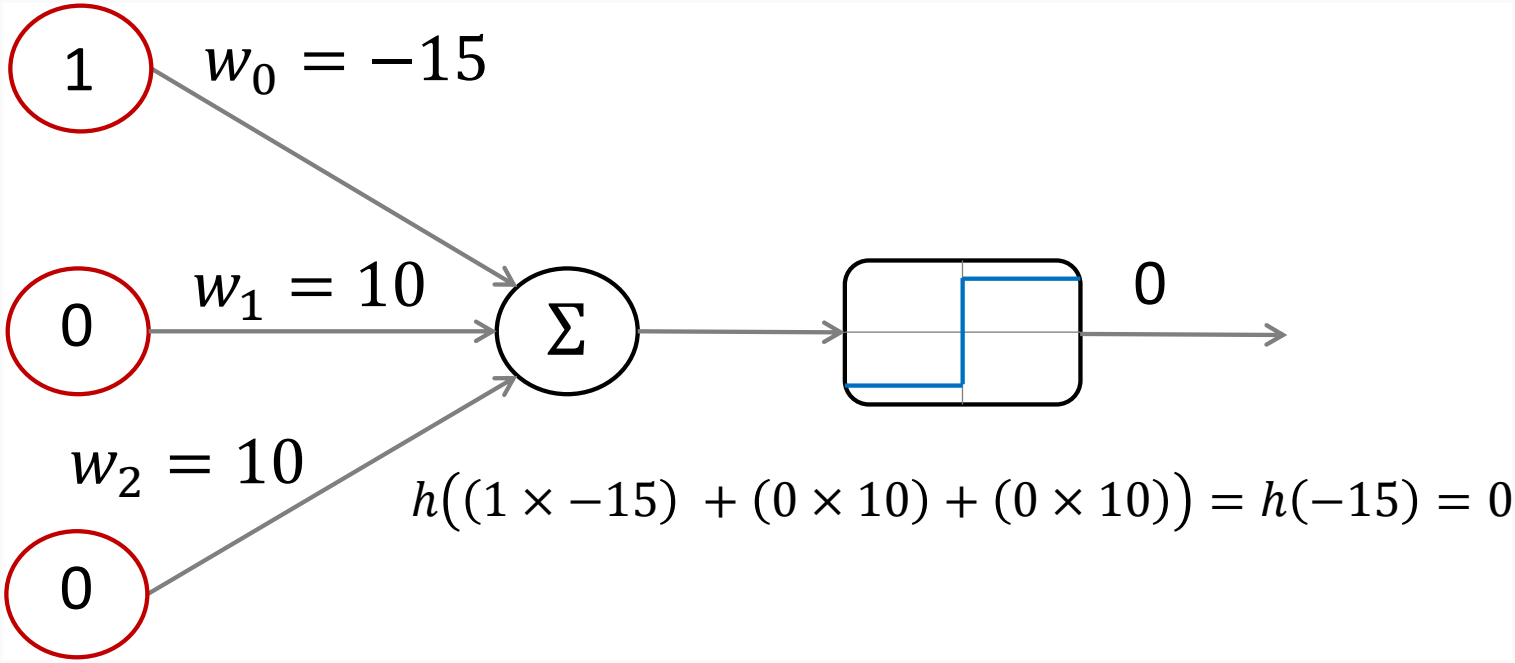
$$\hat{y} = a = \phi(b + \sum_{i=1}^{m} w_i \cdot x_i)$$
$$= \phi(b + \mathbf{w}^\top \cdot \mathbf{x})$$

Compuerta AND ($\wedge$)



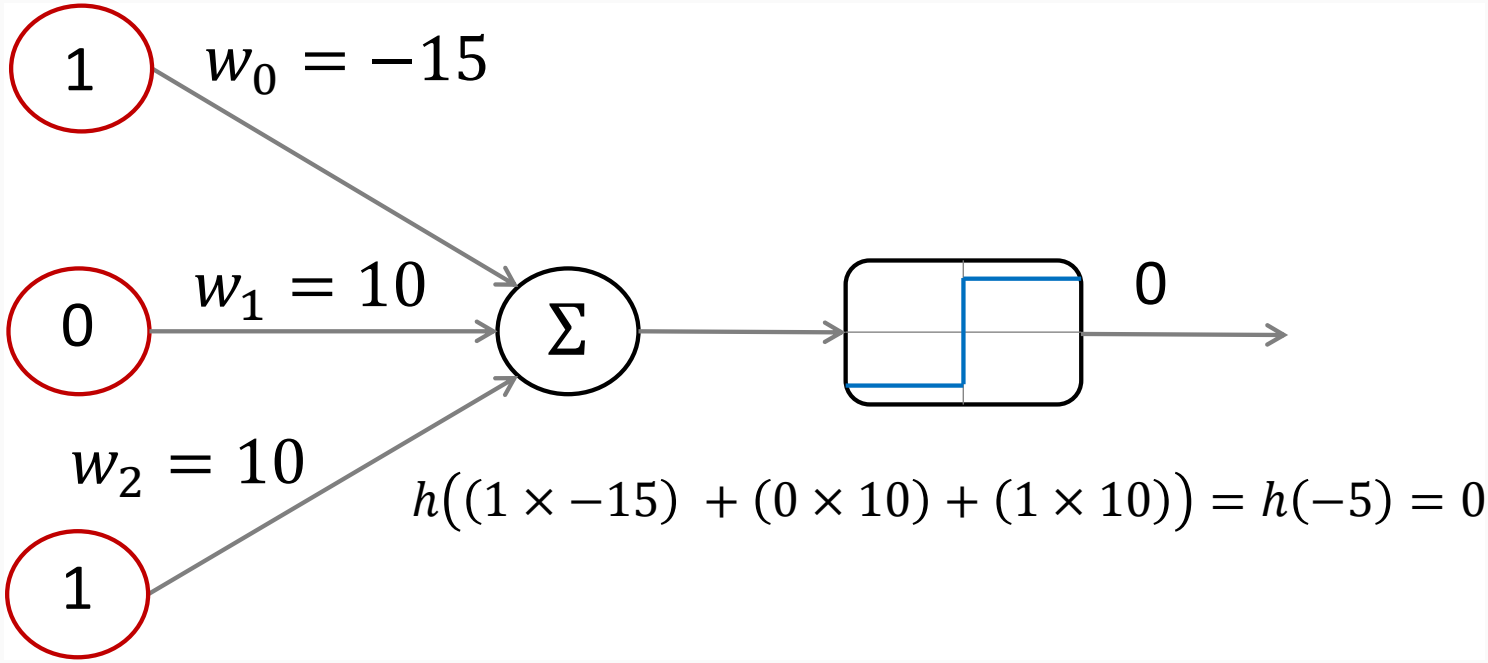
x_1	x_2	AND ($\wedge$)
0	0	0
0	1	0
1	0	0
1	1	1

Compuerta AND ($\wedge$)



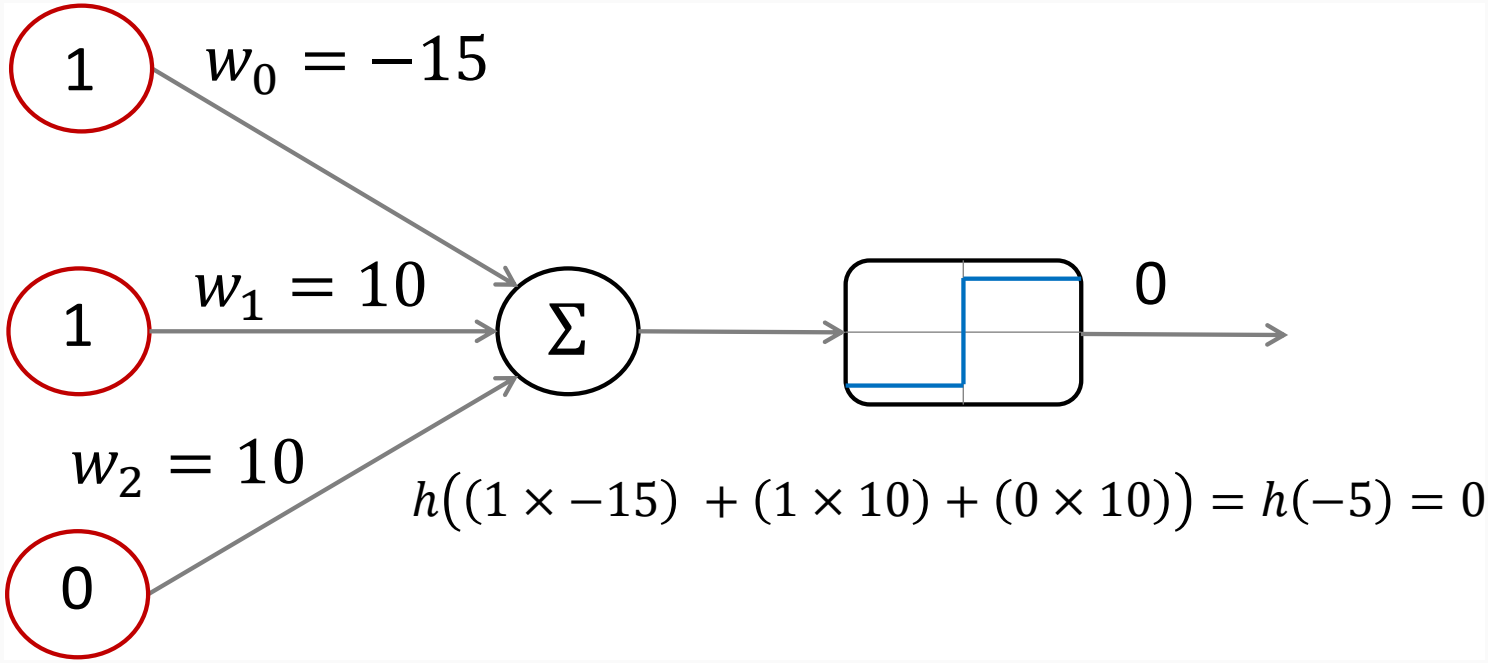
x_1	x_2	AND ($\wedge$)
0	**0**	**0**
0	1	0
1	0	0
1	1	1

Compuerta AND ($\wedge$)



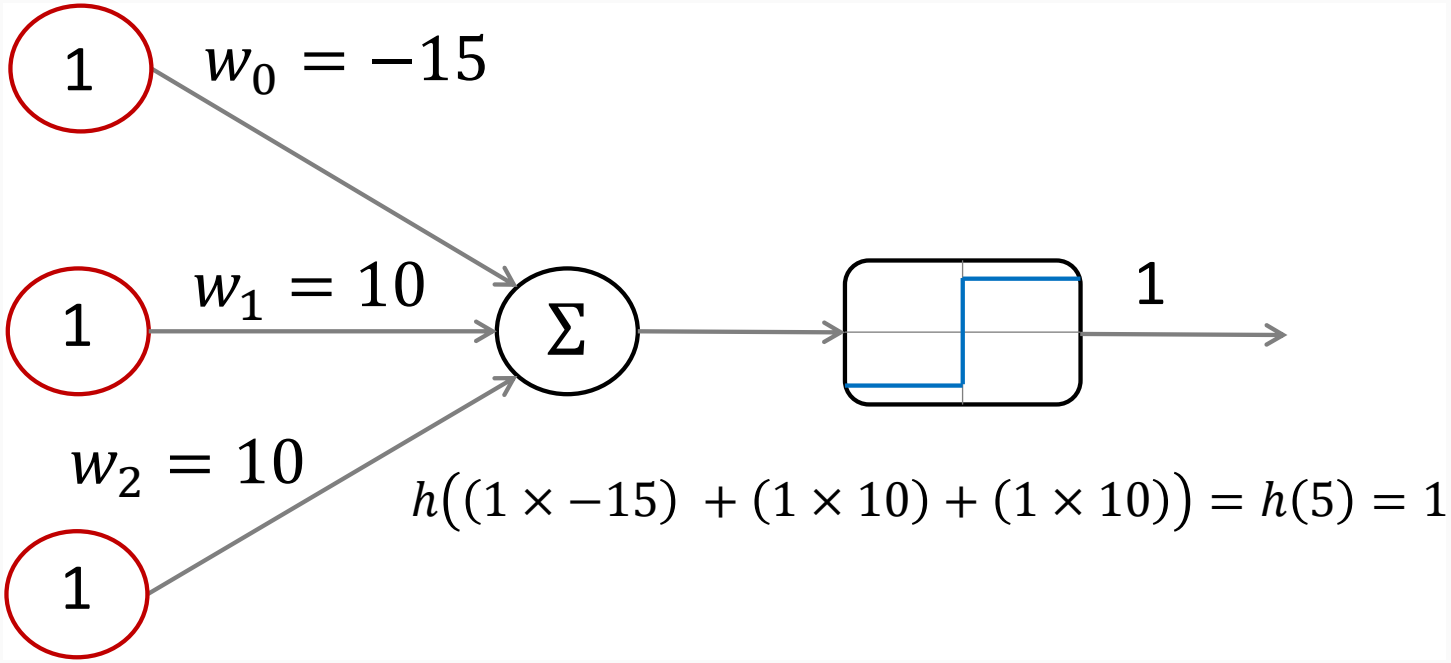
x_1	x_2	AND ($\wedge$)
0	0	0
0	**1**	**0**
1	0	0
1	1	1

Compuerta AND ($\wedge$)



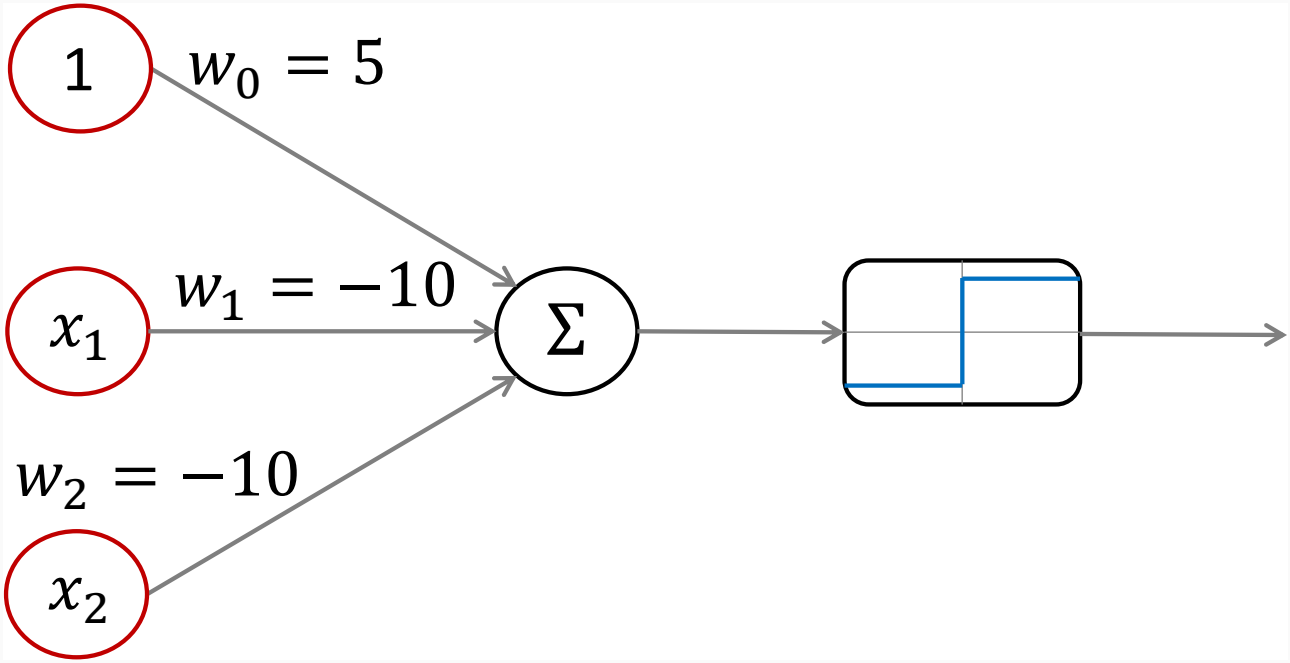
x_1	x_2	AND ($\wedge$)
0	0	0
0	1	0
1	**0**	**0**
1	1	1

Compuerta AND ($\wedge$)



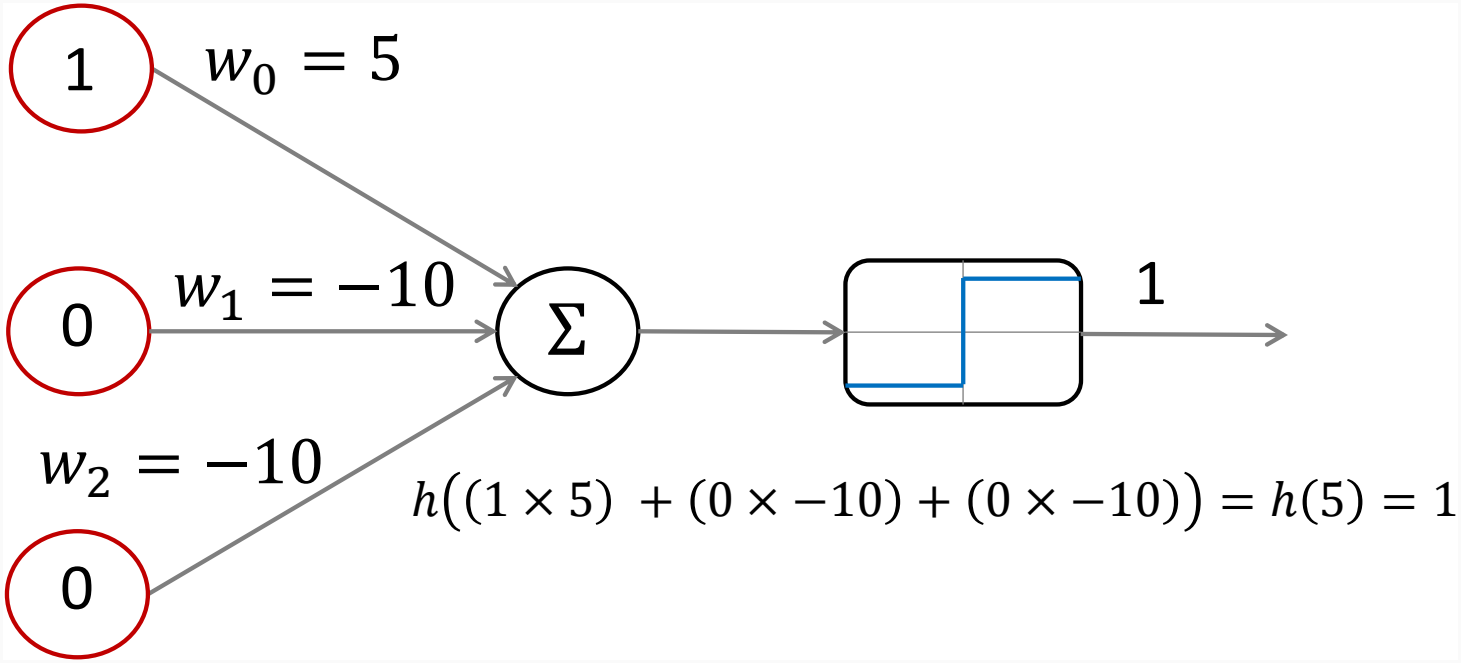
x_1	x_2	AND ($\wedge$)
0	0	0
0	1	0
1	0	0
1	**1**	**1**

Compuerta NOR (↓)



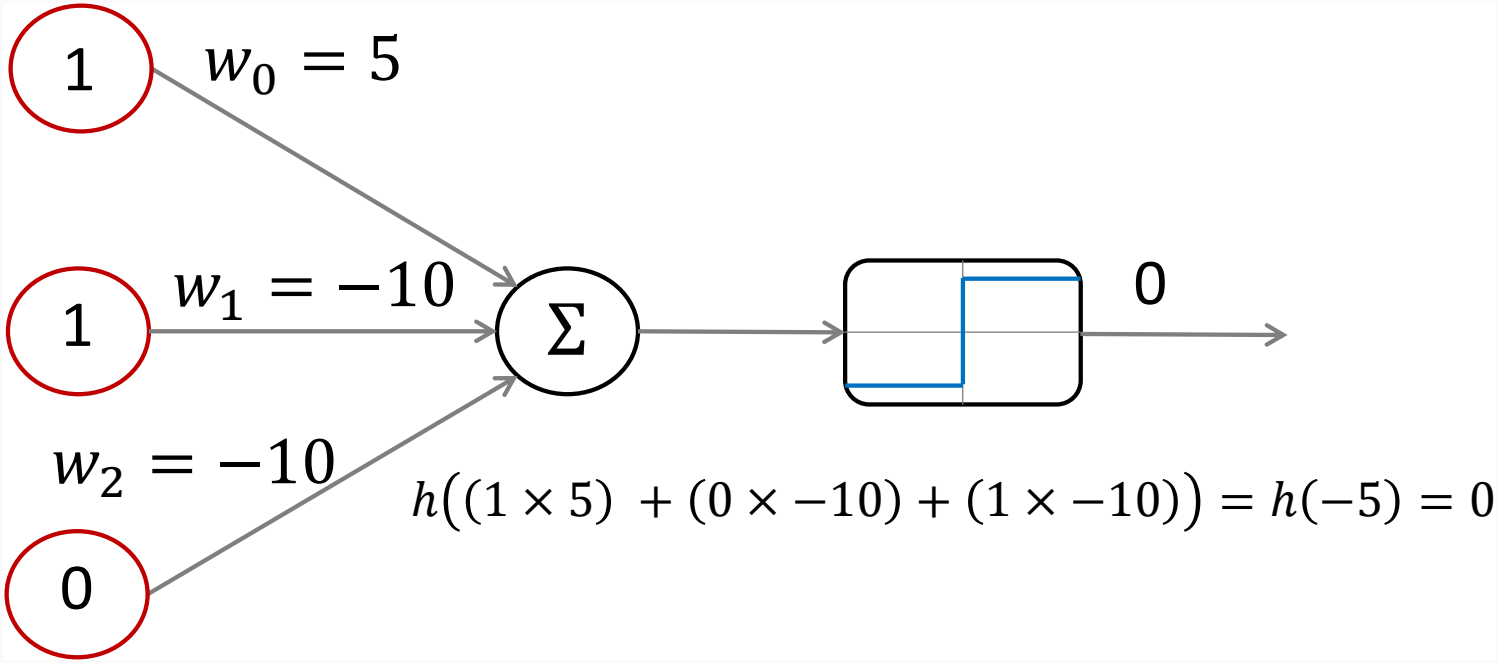
x_1	x_2	NOR (↓)
0	0	1
0	1	0
1	0	0
1	1	0

Compuerta NOR (↓)



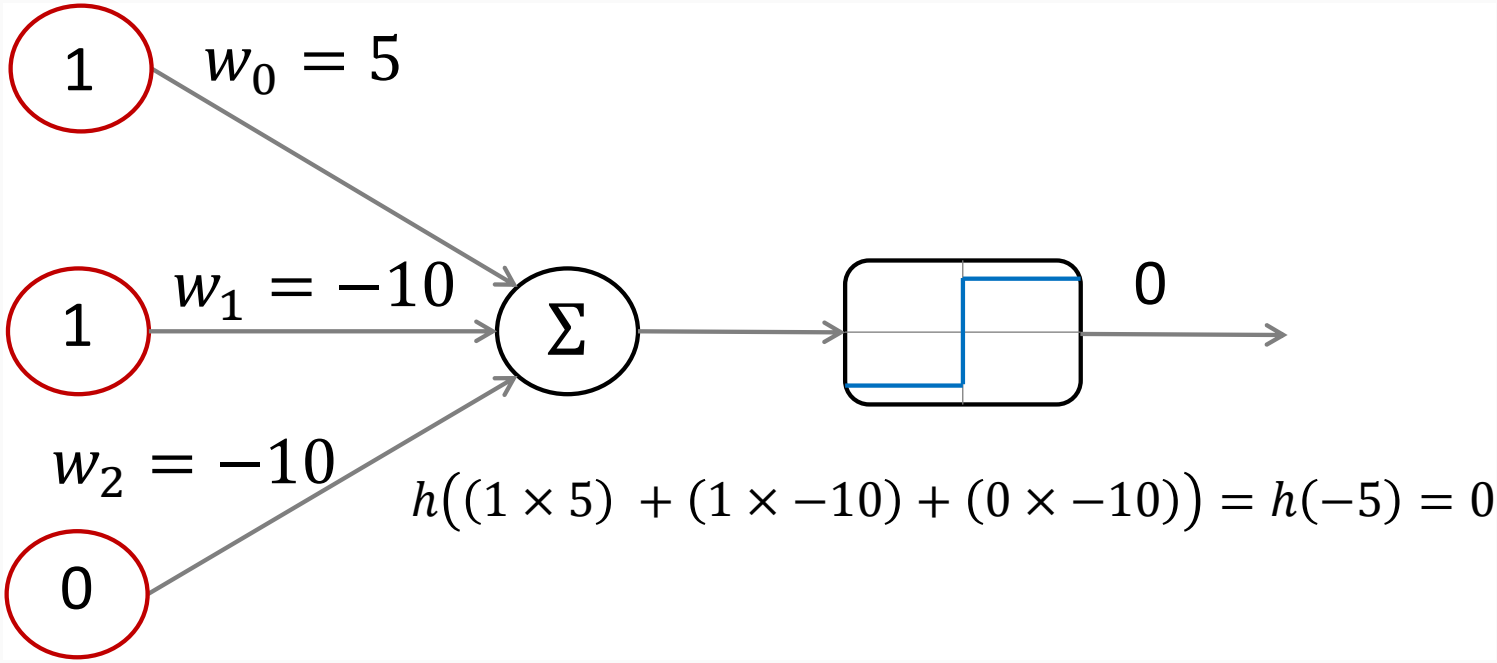
x_1	x_2	NOR (↓)
0	**0**	**1**
0	1	0
1	0	0
1	1	0

Compuerta NOR (↓)



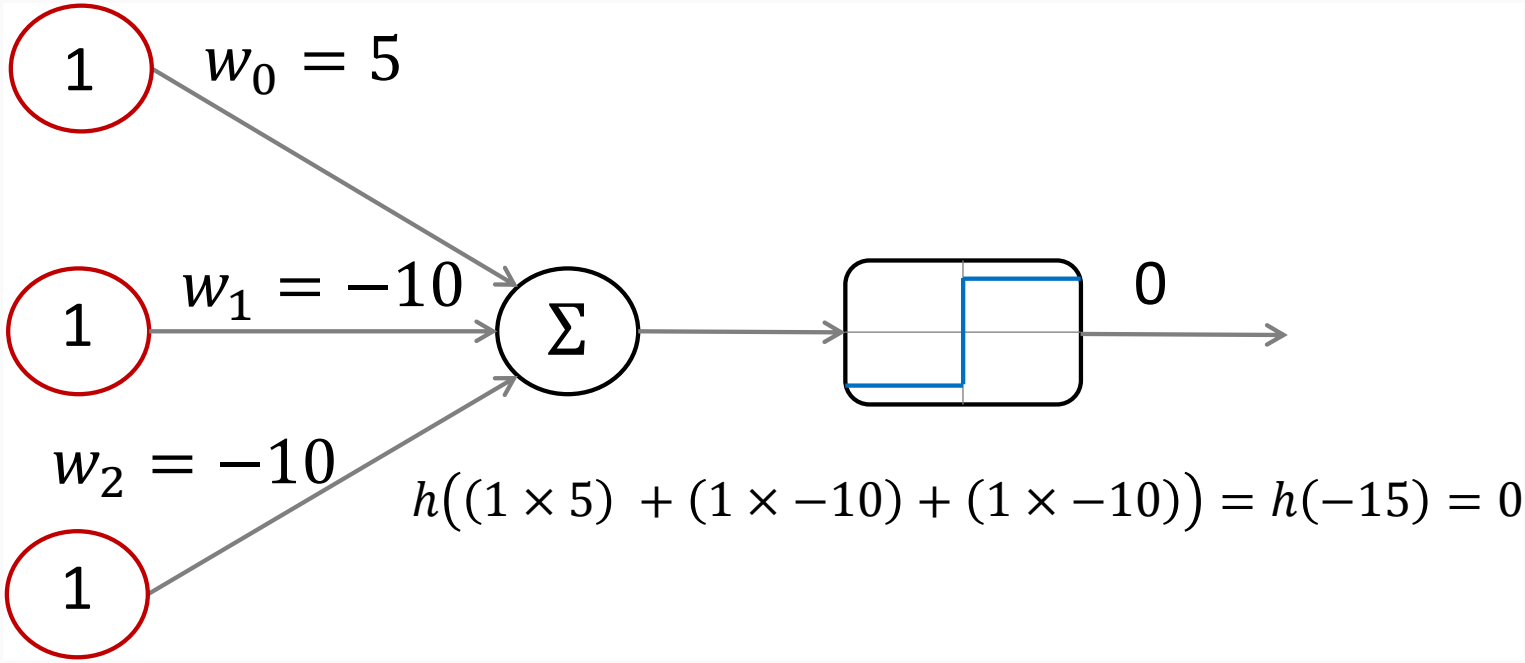
x_1	x_2	NOR (↓)
0	0	1
0	**1**	**0**
1	0	0
1	1	0

Compuerta NOR (↓)



x_1	x_2	NOR (↓)
0	0	1
0	1	0
1	**0**	**0**
1	1	0

Compuerta NOR (↓)



x_1	x_2	NOR (↓)
0	0	1
0	1	0
1	0	0
1	**1**	**0**

Algoritmo de aprendizaje: perceptrón

1. Inicializa pesos y sesgo con zeros o un número aleatorio pequeño
2. Para cada ejemplo en el conjunto de entrenamiento
   2.1 Calcula la salida

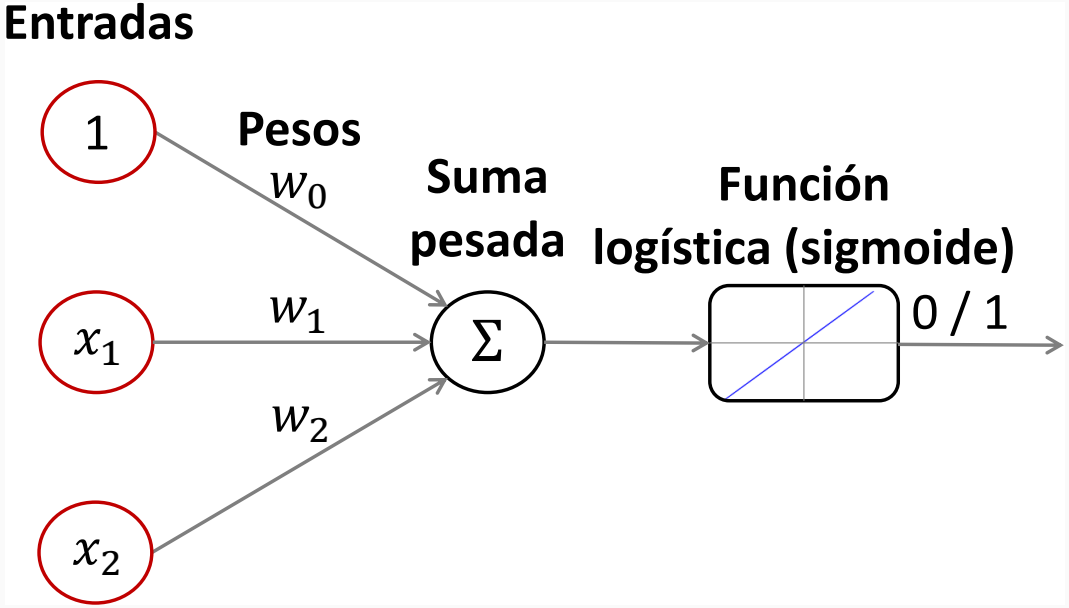
$$\hat{y}^{(i)} = H(\mathbf{w}(t)^\top \mathbf{x}^{(i)} + b)$$

   2.2 Actualiza cada peso $w_j, j = 1, \ldots, d$ y el sesgo b

$$\begin{aligned} w_j[t+1] &= w_j[t] + (y^{(i)} - \hat{y}^{(i)}) \cdot x_j^{(i)} \\ b[t+1] &= b[t] + (y^{(i)} - \hat{y}^{(i)}) \end{aligned}$$

3. Realiza hasta que converja (e.g. error sea menor a umbral)

Neurona con función de activación lineal o identidad



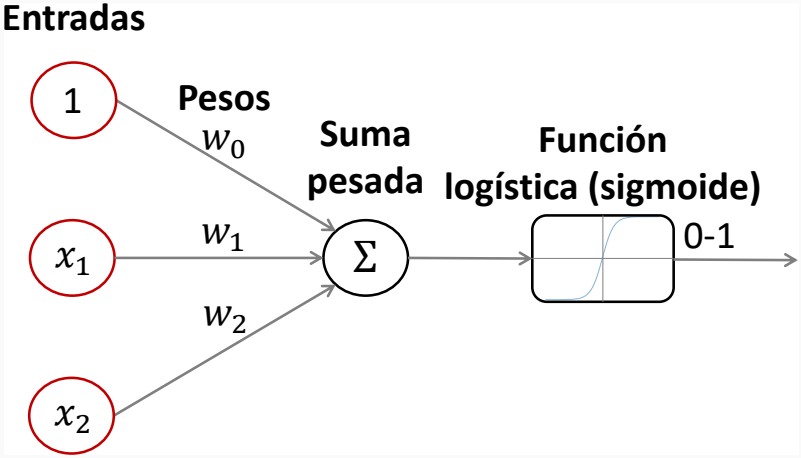
$$lineal(z) = z$$

$$\frac{dlineal(z)}{dz} = 1$$

- Función de pérdida: error cuadráticos medio (ECM)

$$ECM(\mathbf{y}, \hat{\mathbf{y}}) = \frac{1}{2}\sum_{i=1}^{N}(y^{(i)} - \hat{y}^{(i)})^2$$

Neurona con función de activación sigmoide o logística



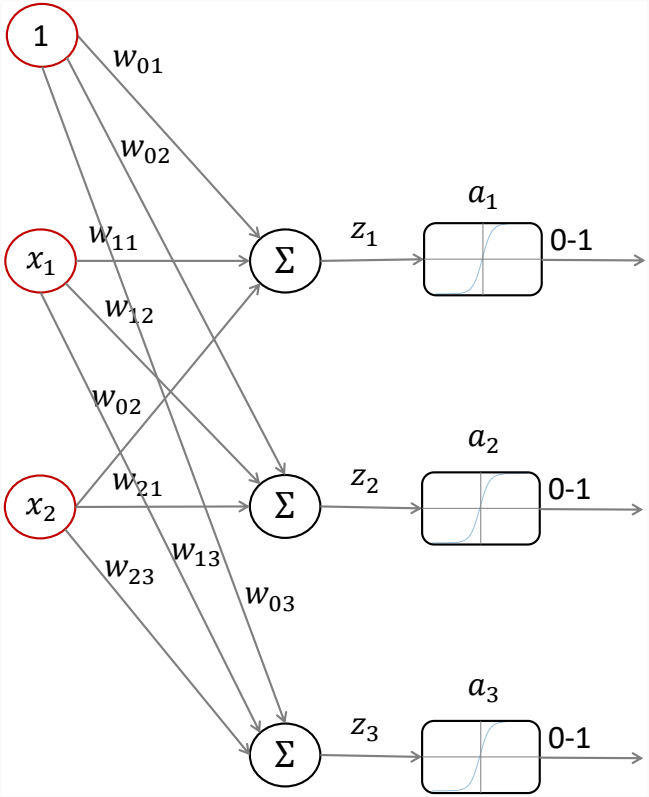
$$sigm(z) = \frac{1}{1 + exp(-z)}$$

$$\frac{dsigm(z)}{dz} = sigm(z)(1 - sigm(z))$$

- Función de pérdida: entropía cruzada binaria (ECB)

$$ECB(\mathbf{y}, \hat{\mathbf{y}}) = -\sum_{i=1}^{N} \left[y^{(i)} \log \hat{y}^{(i)} + (1 - y^{(i)}) \log(1 - \hat{y}^{(i)}) \right]$$

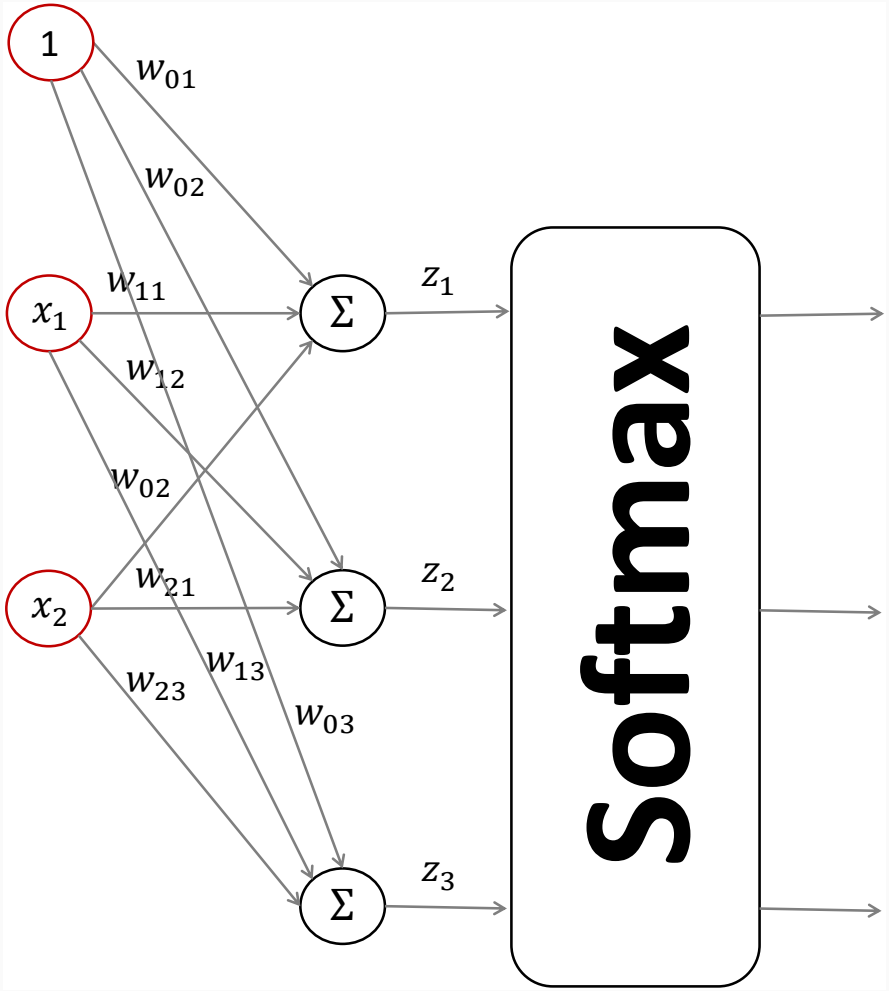
Clasificación multietiqueta



- Función de pérdida: ECB de cada categoría

$$ECB(\mathbf{y}_k, \hat{\mathbf{y}}_k) = -\sum_{i=1}^{N} \left[y_k^{(i)} \log \hat{y}_k^{(i)} + (1 - y_k^{(i)}) \log(1 - \hat{y}_k^{(i)}) \right]$$

Clasificación softmax (1)



Clasificación softmax (2)

- Neuronas de la capa de salida tienen una función de activación *softmax* compartida, dada por

$$softmax(\mathbf{z})_i = \frac{e^{z_i}}{\sum_{j=1}^{K} e^{z_j}}, i = 1, \ldots, K$$

- Función de pérdida de clasificación multiclase: entropía cruzada categórica

$$E(\mathbf{Y}, \hat{\mathbf{Y}}) = -\sum_{i=1}^{N}\sum_{k=1}^{K}\left[1\{y^{(i)} = k\} \log \frac{e^{z_k}}{\sum_j e^{z_j}}\right]$$

Entrenando el modelo: algoritmo del gradiente descendente

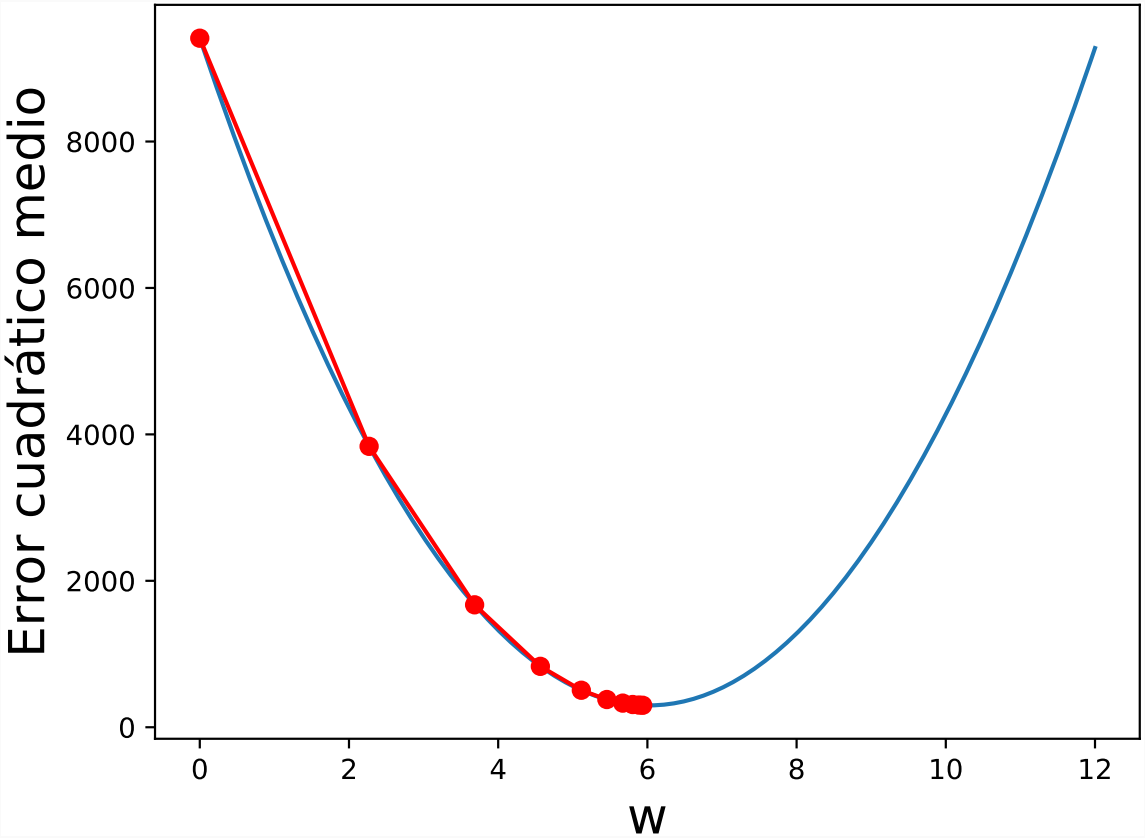
- Mueve sucesivamente pesos y sesgo ($\boldsymbol{\theta} = \{\mathbf{w}, \mathbf{b}\}$) hacia donde más descienda la pérdida

$$\boldsymbol{\theta}^{(k+1)} = \boldsymbol{\theta}^{(k)} - \alpha \mathbf{g}(\boldsymbol{\theta}^{(k)})$$

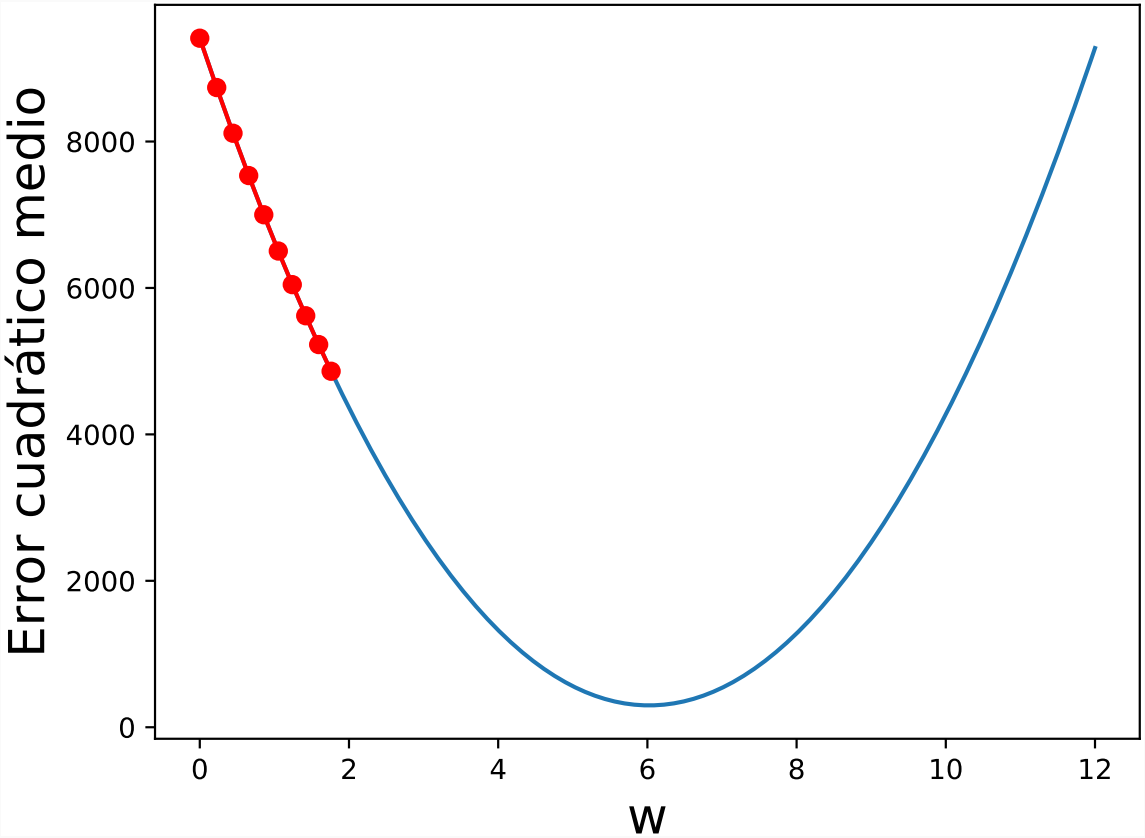
$$\mathbf{g}(\boldsymbol{\theta}) = \frac{d}{d\boldsymbol{\theta}} E = \mathbf{X}^T(\hat{\mathbf{y}} - \mathbf{y})$$

- Gradiente descendente estocástico: calcula gradiente con lote pequeño de datos

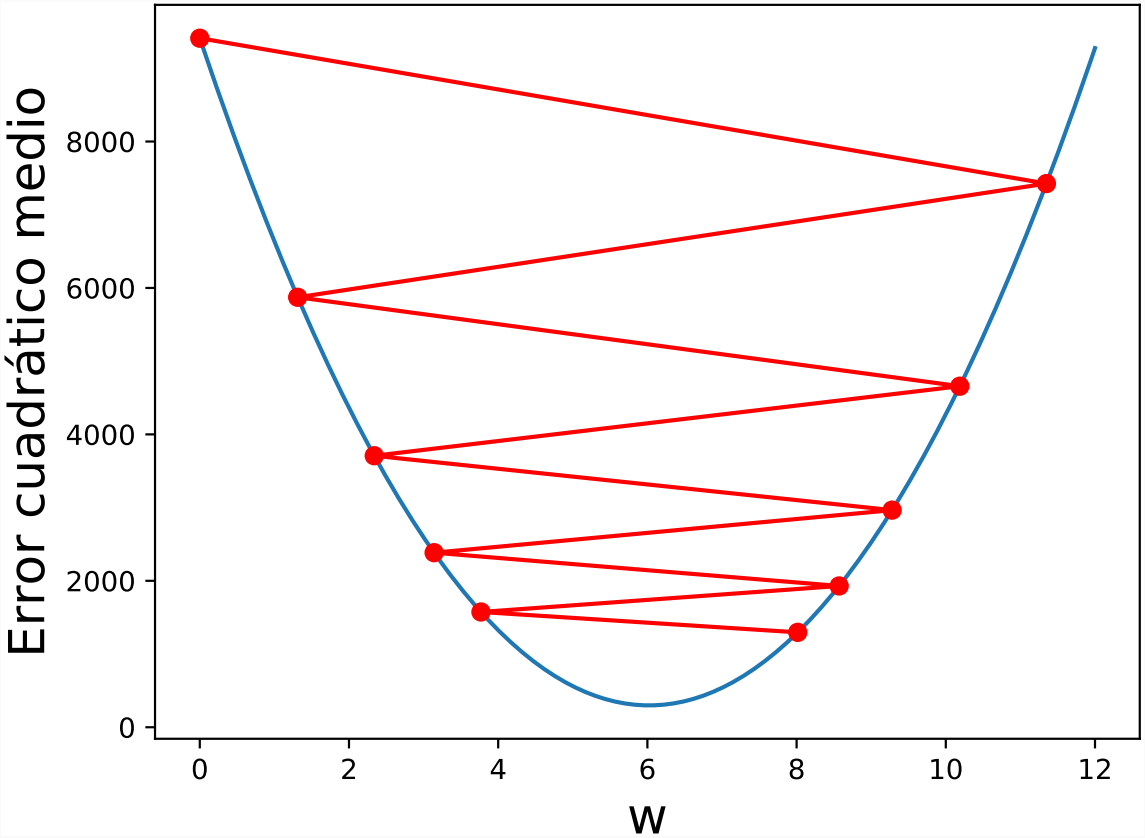
Entrenando el modelo: algoritmo del gradiente descendente



Sensibilidad a coeficiente de aprendizaje α



Sensibilidad a coeficiente de aprendizaje α



Problemas no lineales

- ¿Cómo modelamos una computer XOR?

x_1	x_2	XOR
0	0	0
0	1	1
1	0	1
1	1	0

Problemas no lineales

- ¿Cómo modelamos una computer XOR?

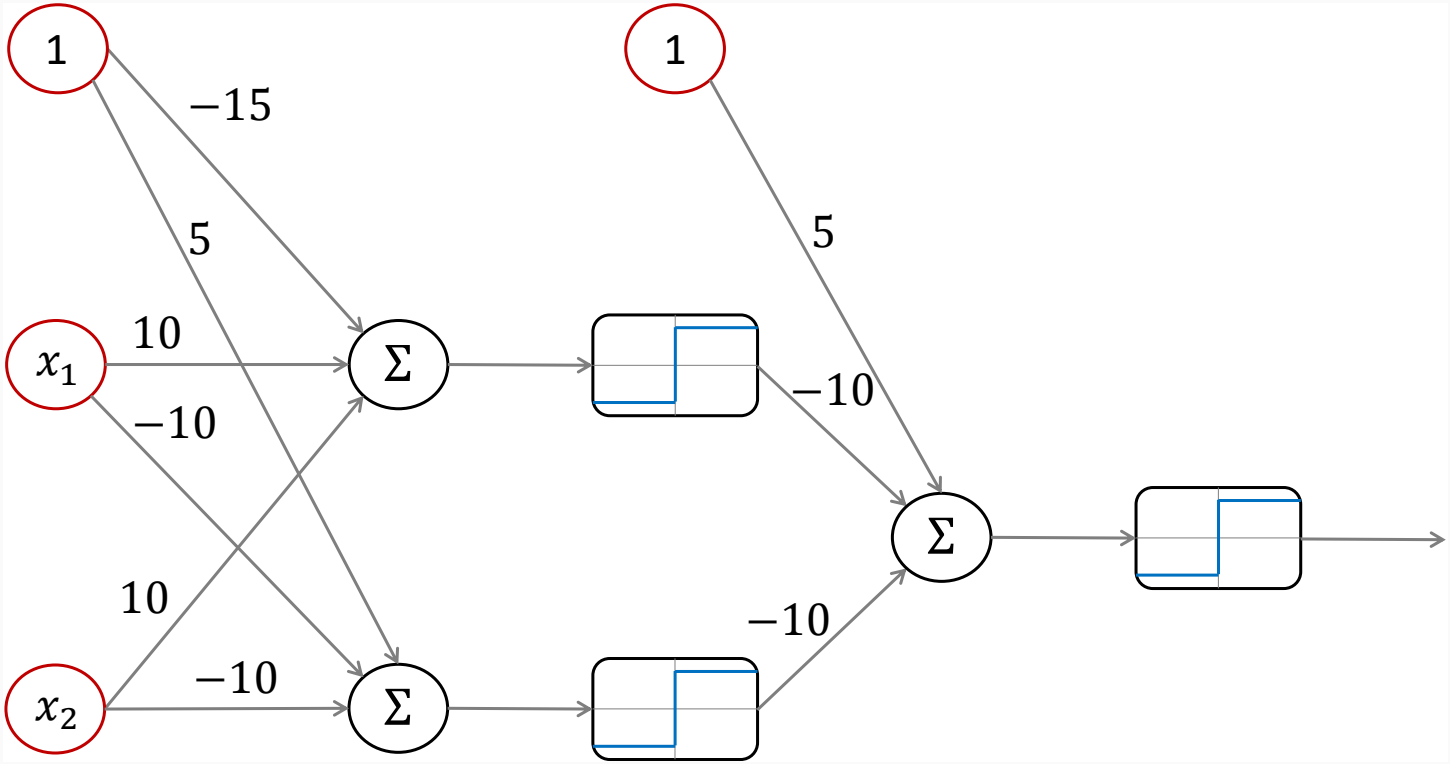
x_1	x_2	XOR
0	0	0
0	1	1
1	0	1
1	1	0

- Minsky y Papert demostraron que era imposible aprender la XOR con perceptrones

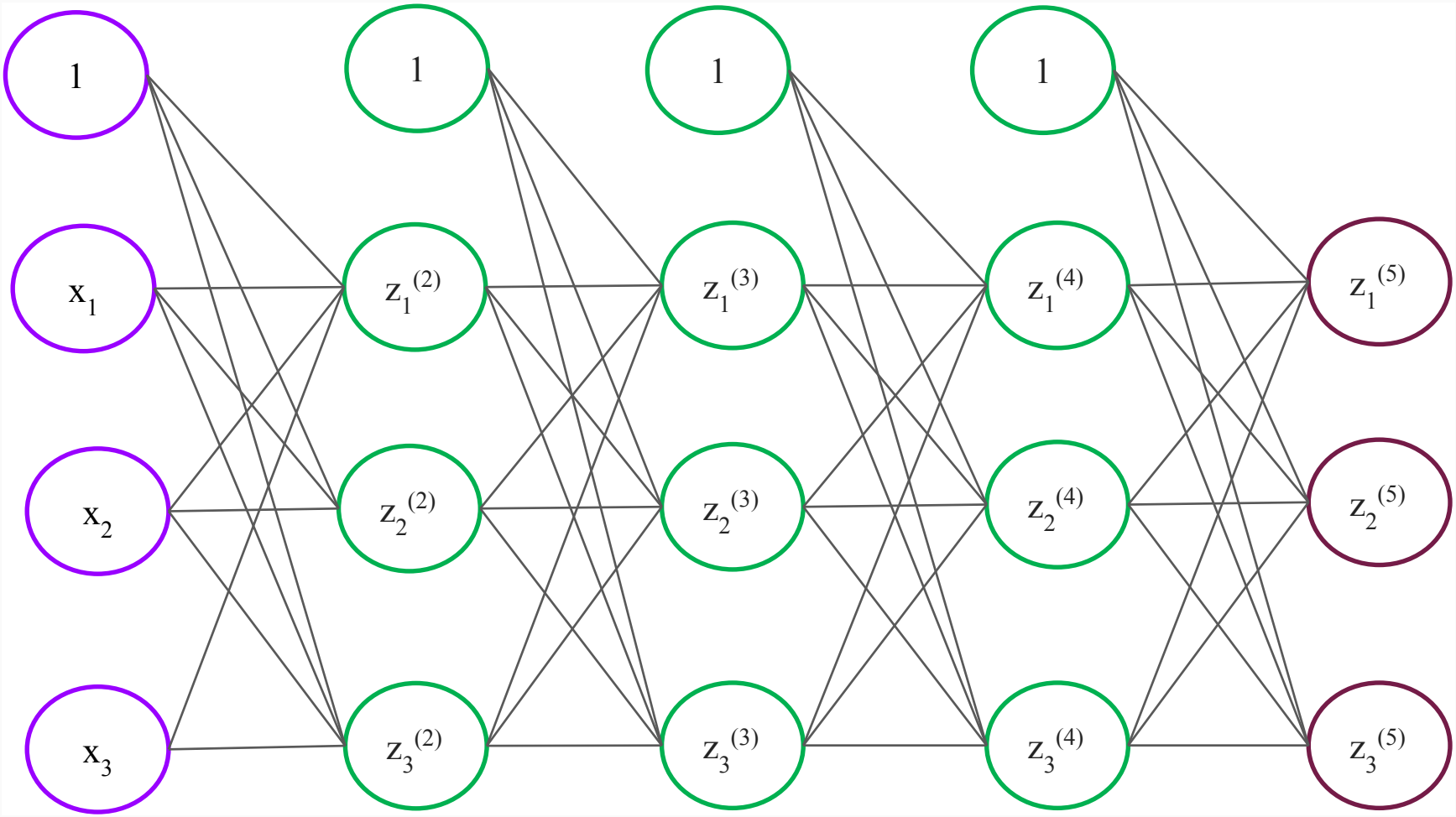
Múltiples capas

- Podemos usar la fórmula

$$x_1 \oplus x_2 = (x_1 \vee x_2) \wedge \neg(x_1 \wedge x_2)$$



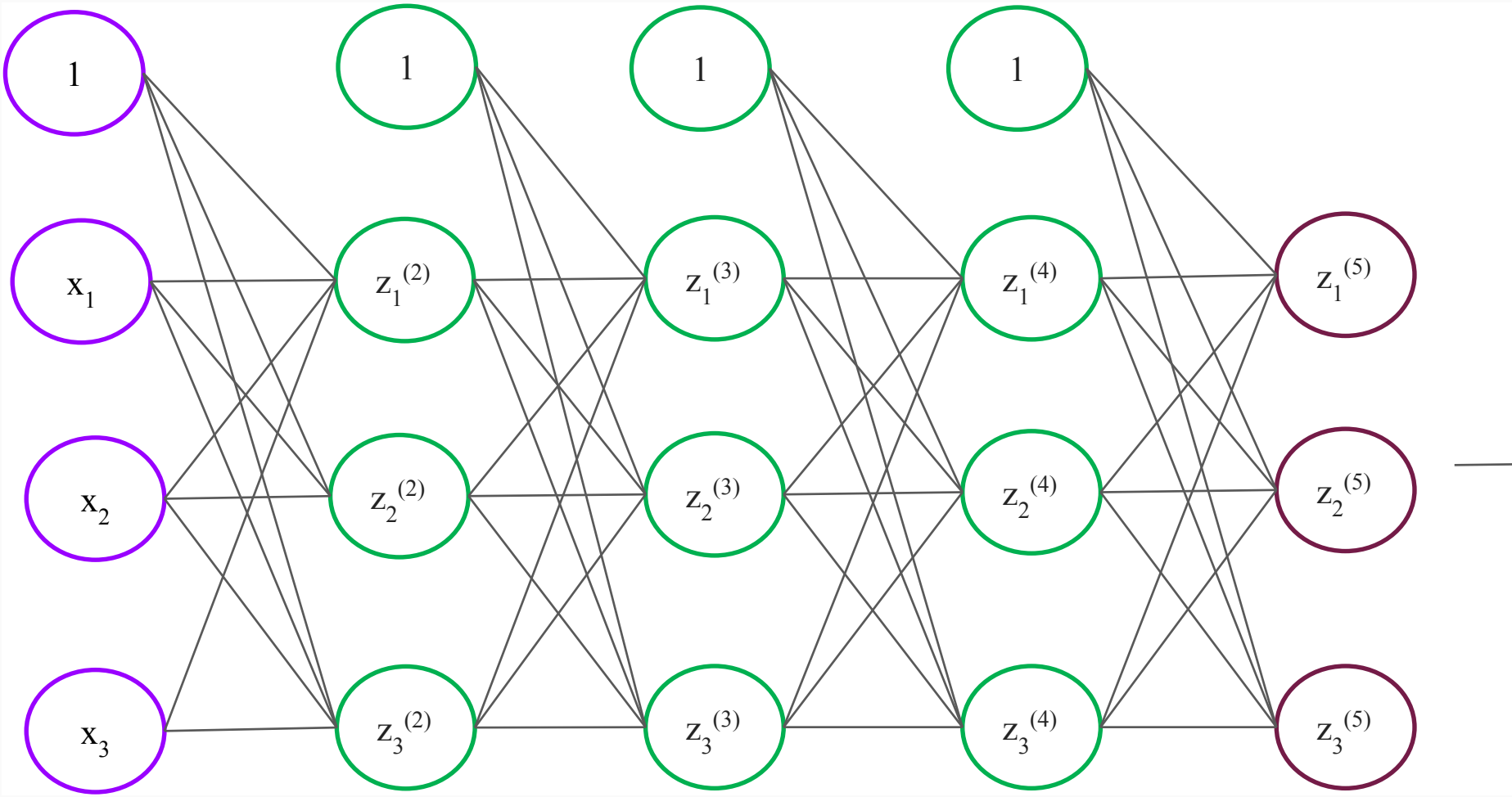
Red neuronal densa



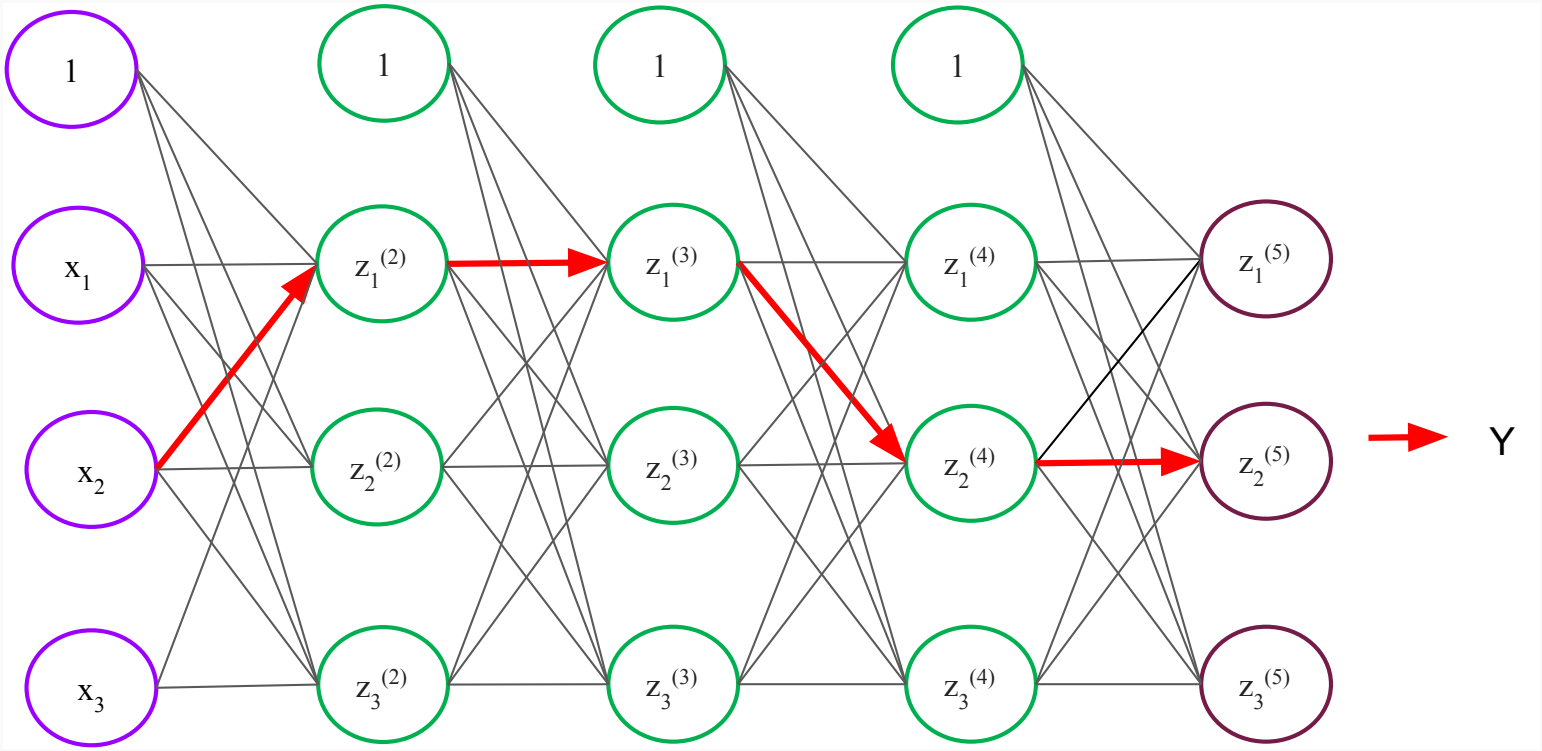
Pasos en la retro-propagación de errores

1. Propagamos hacia adelante las entradas para generar salidas
2. Calculamos error usando las salidas generadas y las salidas deseadas
3. Retro-propagamos error de salida hacia atrás

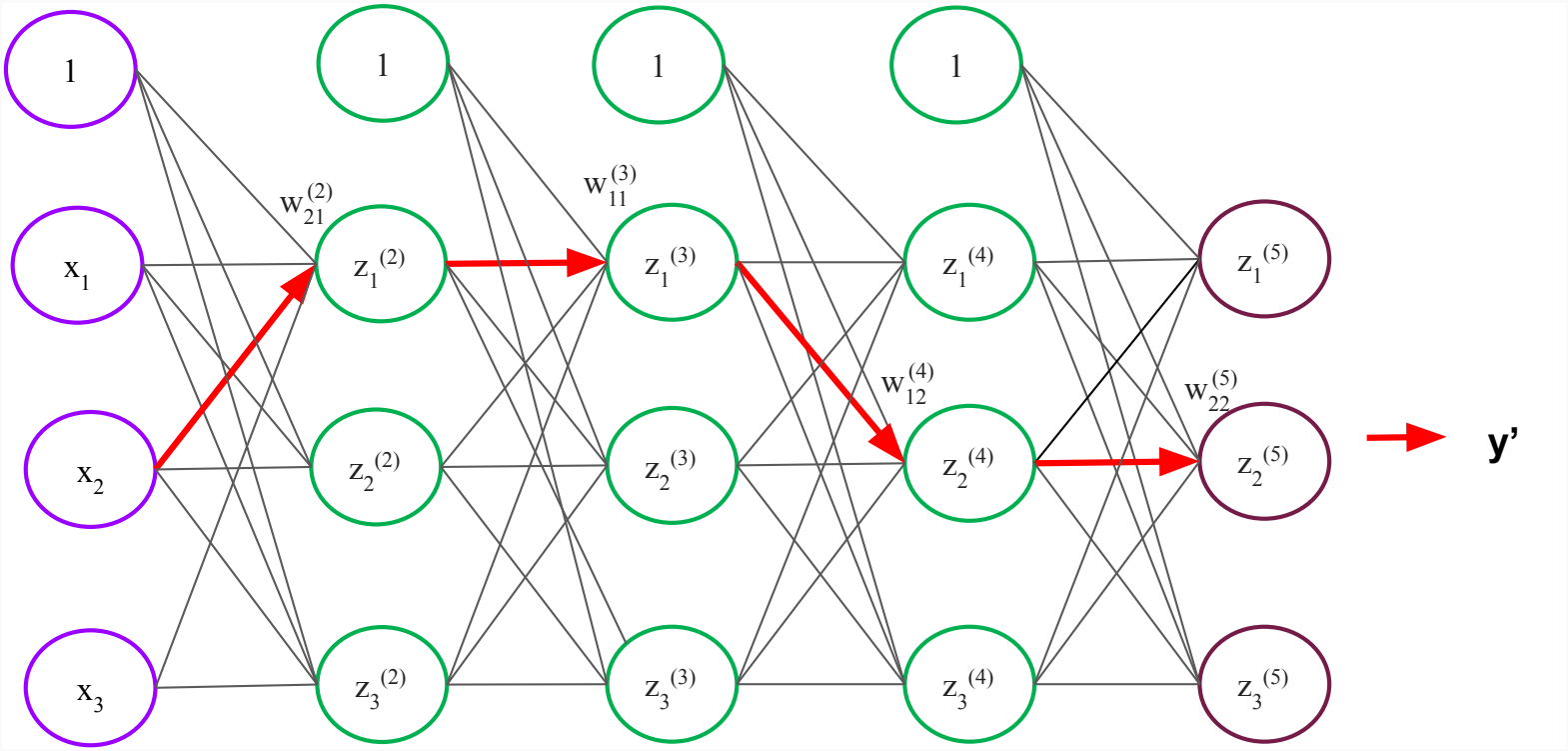
Propagación hacia adelante



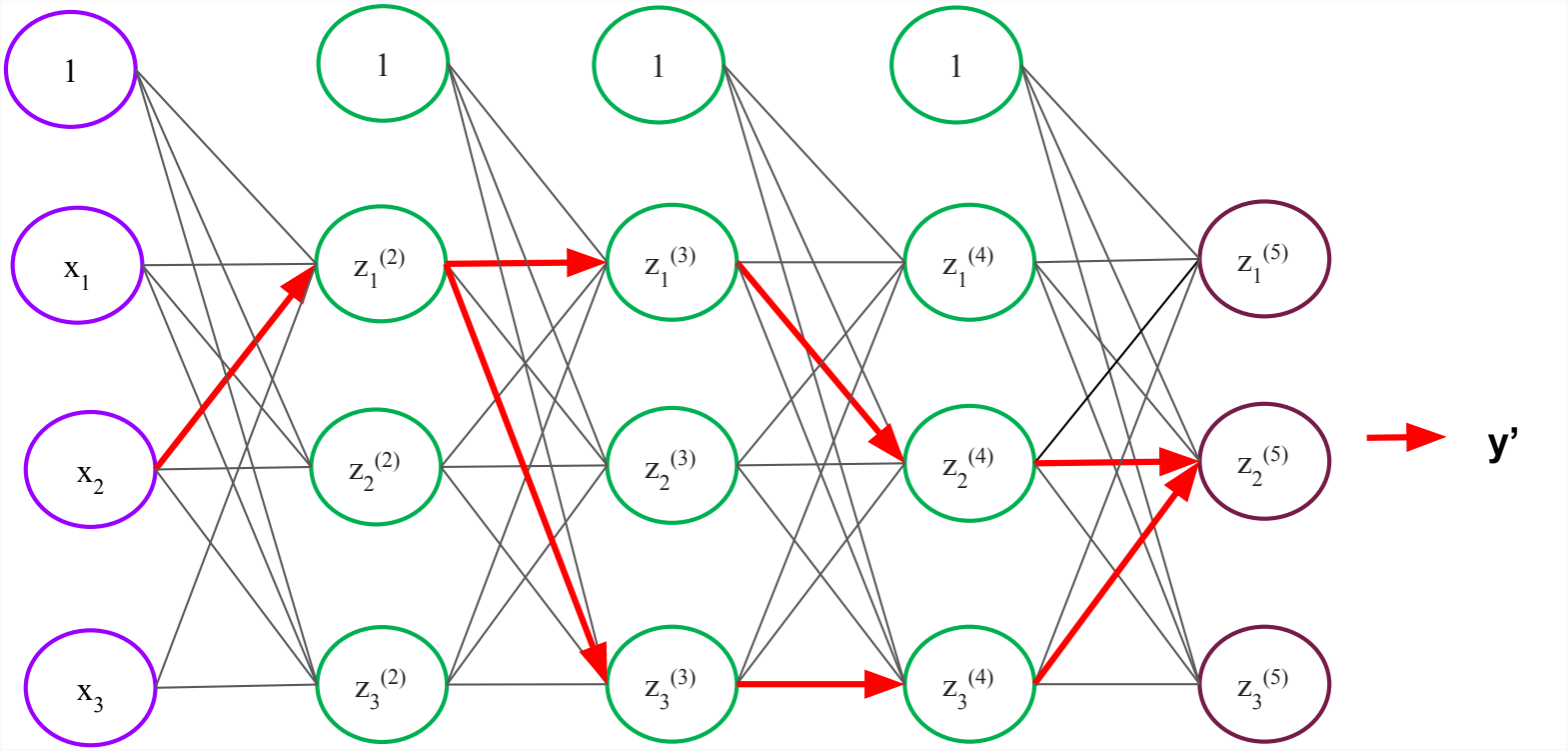
Entrenando por gradiente descendente



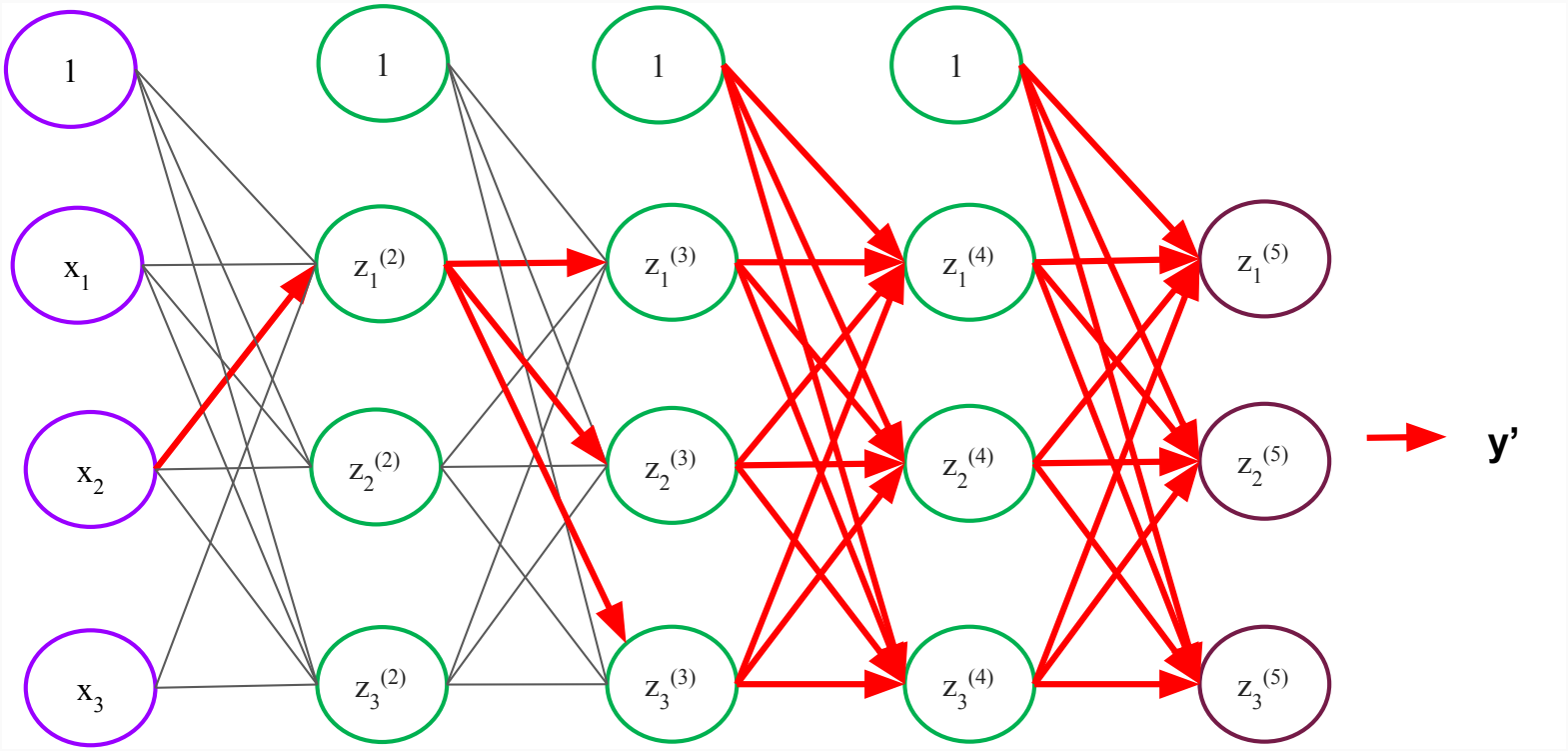
Entrenando por gradiente descendente



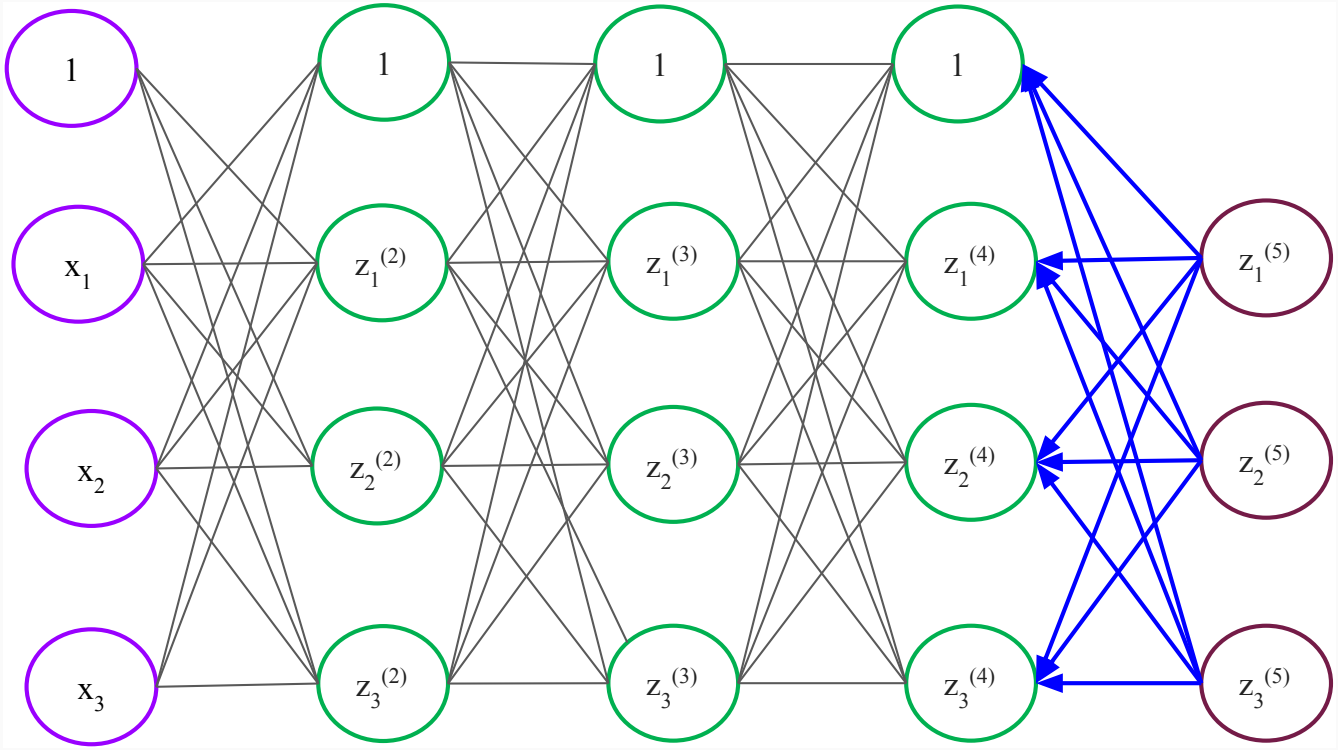
Entrenando por gradiente descendente



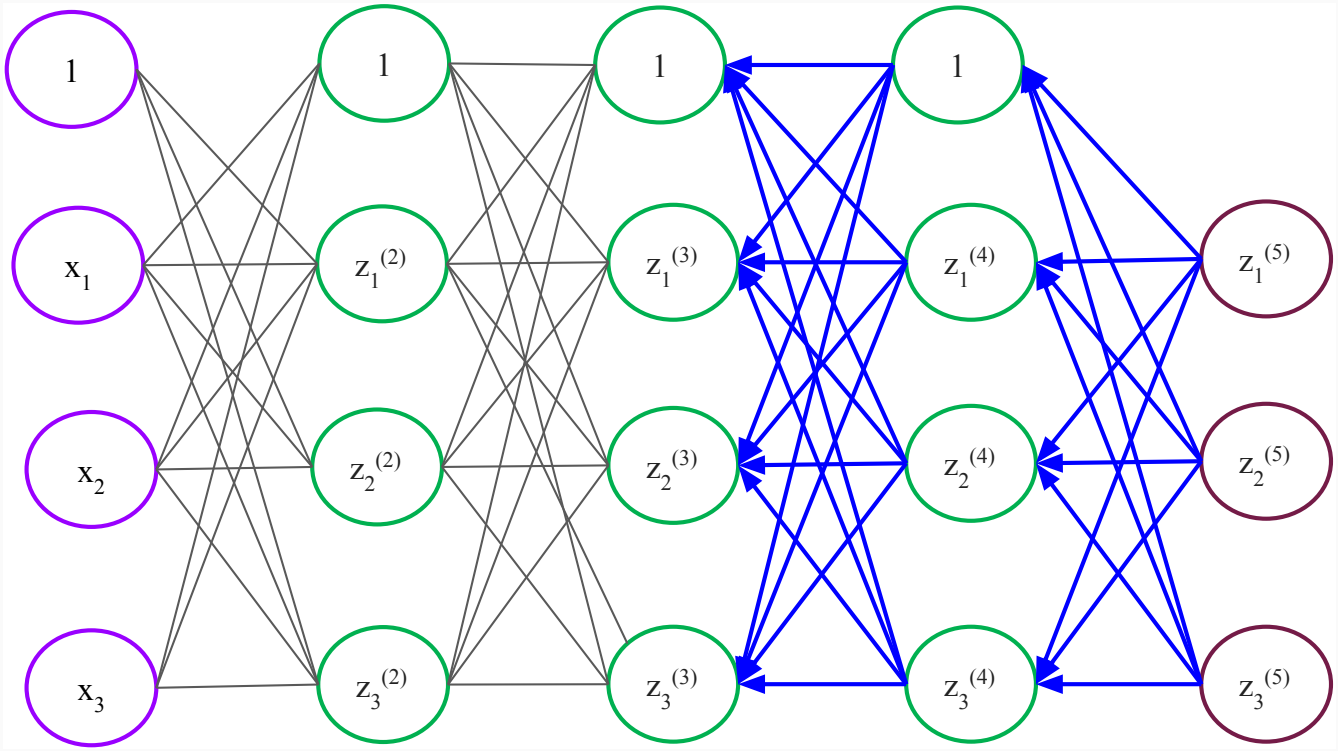
Entrenando por gradiente descendente



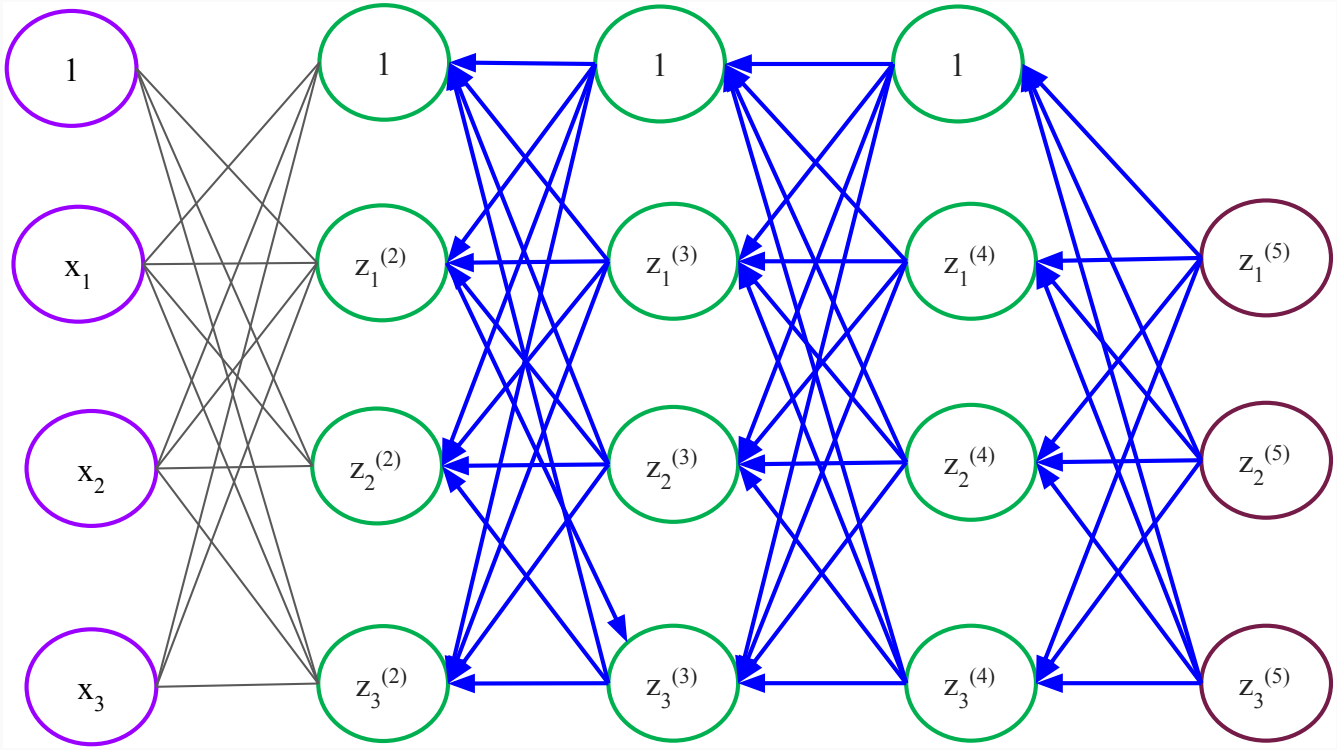
Algoritmo de retropropagación



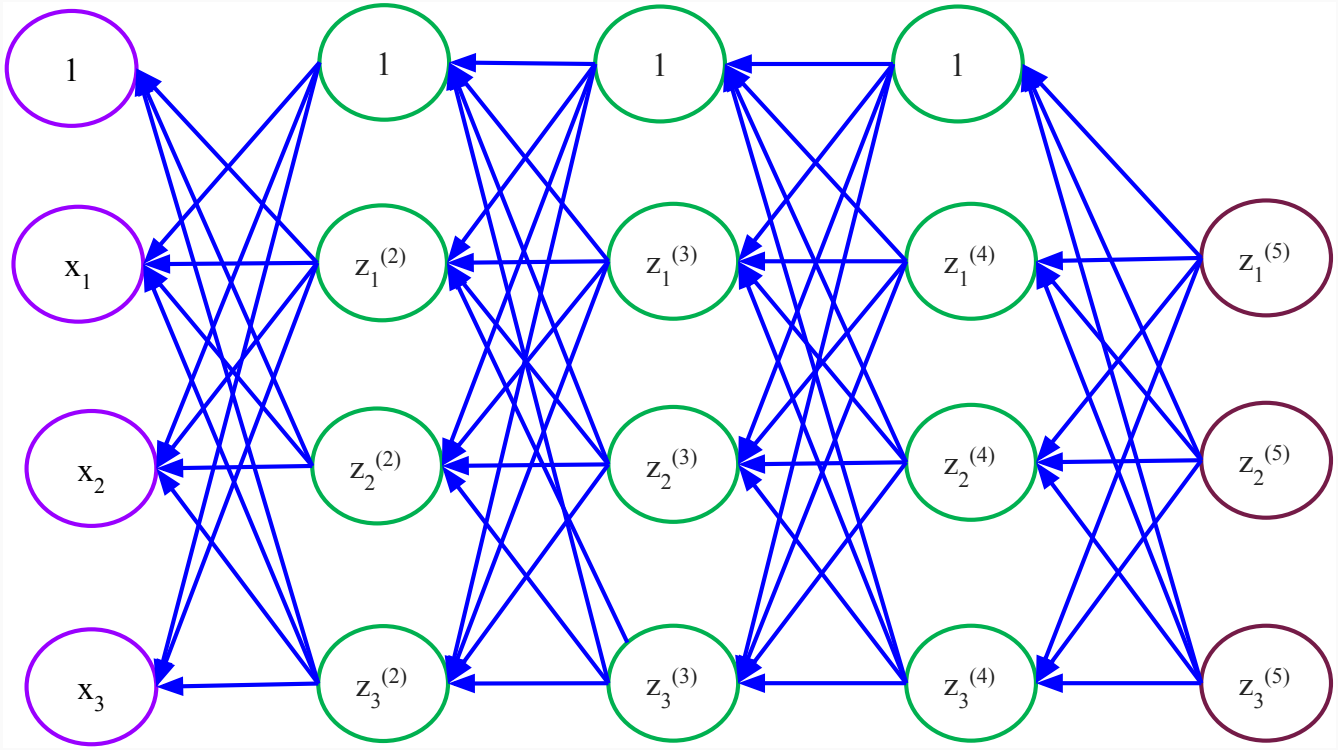
Algoritmo de retropropagación



Algoritmo de retropropagación



Algoritmo de retropropagación



Propagación hacia adelante

- Propagamos hacia adelante las entradas para generar salidas

$$
\begin{aligned}
a^{(1)} &= x \\
z^{(2)} &= W^{(1)} \cdot a^{(1)} \\
a^{(2)} &= \phi(z^{(2)}) \\
z^{(3)} &= W^{(2)} \cdot a^{(2)} \\
a^{(3)} &= \phi(z^{(3)})
\end{aligned}
$$

Cálculo de error

- Calculamos error usando las salidas generadas y las salidas deseadas. Por ejemplo:

$$E \triangleq \sum \frac{1}{2}(y - \hat{y})^2$$

Retro-propagación de errores (1)

- Calculamos el gradiente del error con respecto a $W^(2)$ de la siguiente forma

$$\begin{aligned}
\frac{\partial E}{\partial W^{(2)}} &= \frac{\partial \sum \frac{1}{2}(y-\hat{y})^2}{\partial W^{(2)}} \\
&= \frac{\sum \partial \frac{1}{2}(y-\hat{y})^2}{\partial W^{(2)}} \\
\frac{\partial \frac{1}{2}(y-\hat{y})^2}{\partial W^{(2)}} &= (y-\hat{y}) \cdot \left(-\frac{\partial \hat{y}}{\partial W^{(2)}} \right) \\
&= (y-\hat{y}) \cdot \left(-\frac{\partial \hat{y}}{\partial z^{(3)}} \cdot \frac{\partial z^{(3)}}{\partial W^{(2)}} \right) \\
&= \underbrace{-(y-\hat{y}) \cdot \frac{\partial \hat{y}}{\partial z^{(3)}}}_{\delta^{(3)}} \cdot a^{(2)}
\end{aligned}$$

Retro-propagación de errores (2)

- Calculamos el gradiente del error con respecto a $W^(1)$ de la siguiente forma

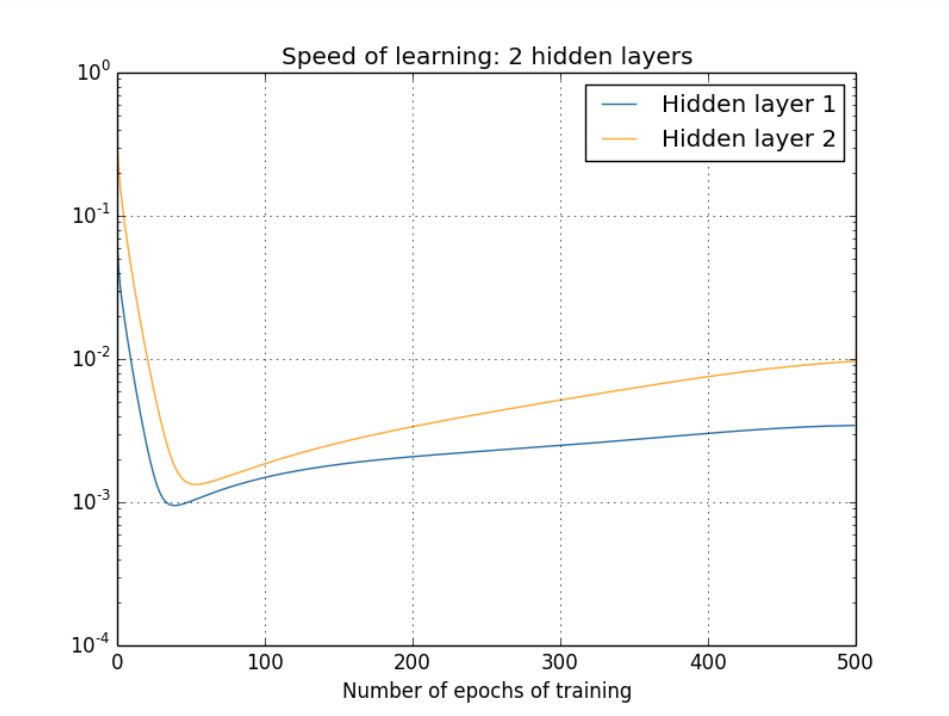
$$\begin{aligned}
\frac{\partial E}{\partial W^{(1)}} &= \frac{\partial \sum \frac{1}{2}(y-\hat{y})^2}{\partial W^{(1)}} \\
&= \frac{\sum \partial \frac{1}{2}(y-\hat{y})^2}{\partial W^{(1)}} \\
\frac{\partial \frac{1}{2}(y-\hat{y})^2}{\partial W^{(1)}} &= (y-\hat{y})\left(-\frac{\partial \hat{y}}{\partial W^{(1)}}\right) \\
&= (y-\hat{y})\left(-\frac{\partial \hat{y}}{\partial z^{(3)}} \cdot \frac{\partial z^{(3)}}{\partial W^{(1)}}\right) \\
&= \delta^{(3)} \cdot \frac{\partial z^{(3)}}{\partial W(1)}
\end{aligned}$$

Retro-propagación de errores (3)

$$
\begin{aligned}
&= \delta^{(3)} \cdot \left(\frac{\partial z^{(3)}}{\partial a^{(2)}} \cdot \frac{\partial a^{(2)}}{\partial W^{(1)}} \right) \\
&= \delta^{(3)} \cdot W^{(2)} \cdot \left(\frac{\partial a^{(2)}}{\partial W^{(1)}} \right) \\
&= \delta^{(3)} \cdot W^{(2)} \cdot \left(\frac{\partial a^{(2)}}{\partial z^{(2)}} \cdot \frac{\partial z^{(2)}}{\partial W^{(1)}} \right) \\
&= \delta^{(3)} \cdot W^{(2)} \cdot \frac{\partial a^{(2)}}{\partial z^{(2)}} \cdot x
\end{aligned}
$$

El problema del desvanecimiento del gradiente

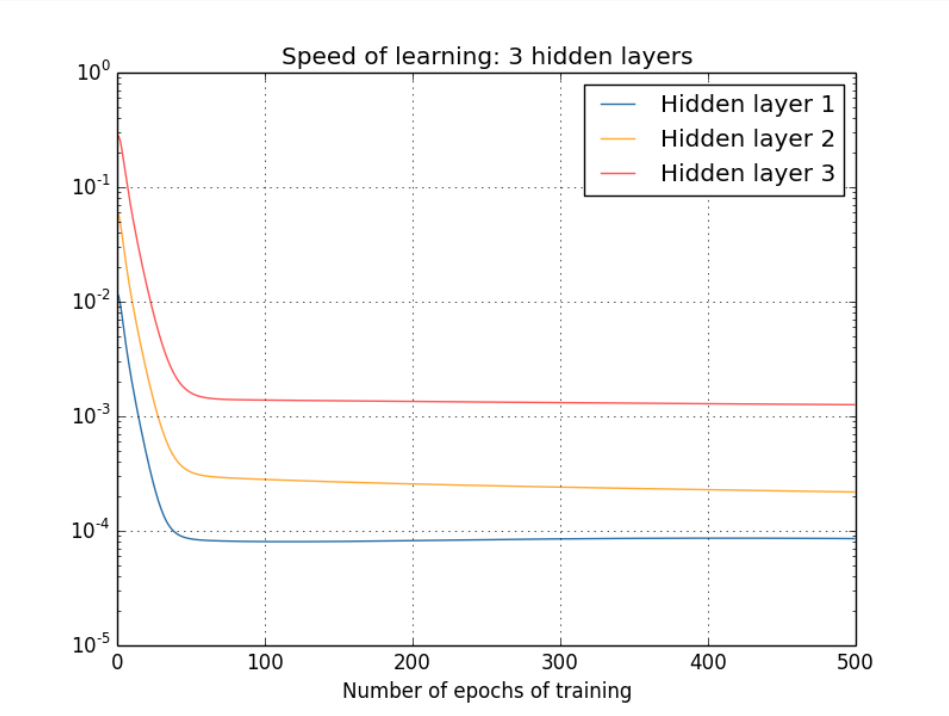
- Gradientes de primeras capas se vuelven muy pequeños si la red es muy profunda
- Muy lento actualizar pesos de estas capas



Tomado de `http://neuralnetworksanddeeplearning.com/chap5.html`

El problema del desvanecimiento del gradiente

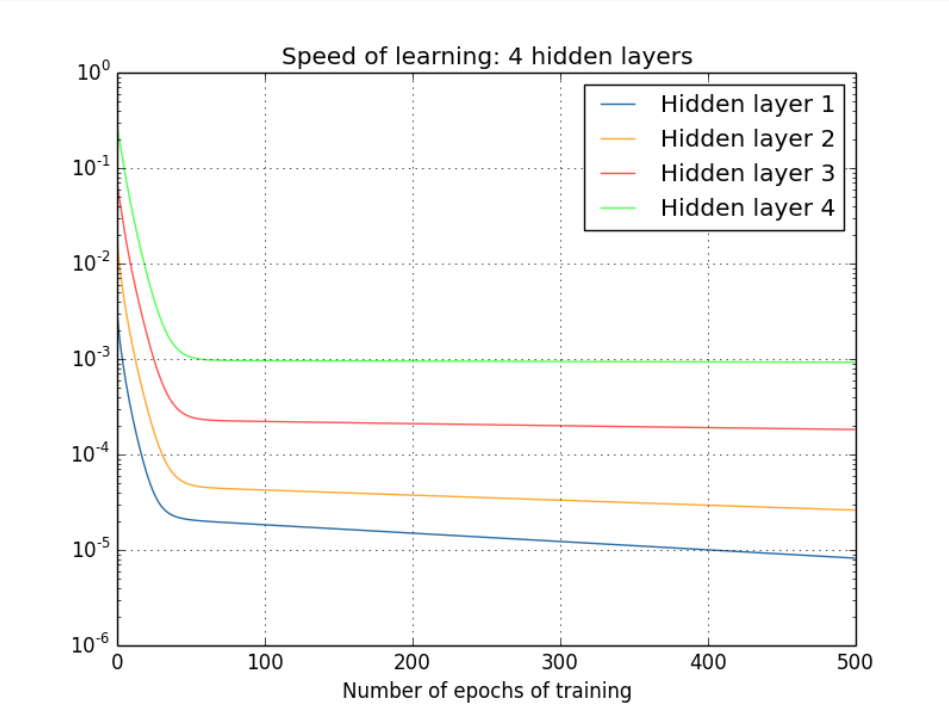
- Gradientes de primeras capas se vuelven muy pequeños si la red es muy profunda
- Muy lento actualizar pesos de estas capas



Tomado de `http://neuralnetworksanddeeplearning.com/chap5.html`

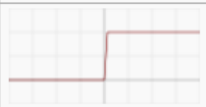
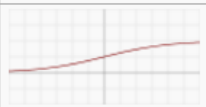
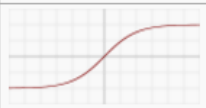
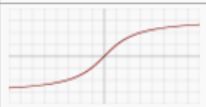
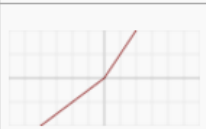
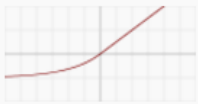
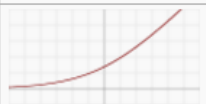
El problema del desvanecimiento del gradiente

- Gradientes de primeras capas se vuelven muy pequeños si la red es muy profunda
- Muy lento actualizar pesos de estas capas



Tomado de `http://neuralnetworksanddeeplearning.com/chap5.html`

Funciones de activación

Name	Plot	Equation	Derivative
Identity		$f(x) = x$	$f'(x) = 1$
Binary step		$f(x) = \begin{cases} 0 & \text{for } x < 0 \\ 1 & \text{for } x \geq 0 \end{cases}$	$f'(x) = \begin{cases} 0 & \text{for } x \neq 0 \\ ? & \text{for } x = 0 \end{cases}$
Logistic (a.k.a Soft step)		$f(x) = \frac{1}{1+e^{-x}}$	$f'(x) = f(x)(1-f(x))$
TanH		$f(x) = \tanh(x) = \frac{2}{1+e^{-2x}} - 1$	$f'(x) = 1 - f(x)^2$
ArcTan		$f(x) = \tan^{-1}(x)$	$f'(x) = \frac{1}{x^2+1}$
Rectified Linear Unit (ReLU)		$f(x) = \begin{cases} 0 & \text{for } x < 0 \\ x & \text{for } x \geq 0 \end{cases}$	$f'(x) = \begin{cases} 0 & \text{for } x < 0 \\ 1 & \text{for } x \geq 0 \end{cases}$
Parameteric Rectified Linear Unit (PReLU) [2]		$f(x) = \begin{cases} \alpha x & \text{for } x < 0 \\ x & \text{for } x \geq 0 \end{cases}$	$f'(x) = \begin{cases} \alpha & \text{for } x < 0 \\ 1 & \text{for } x \geq 0 \end{cases}$
Exponential Linear Unit (ELU) [3]		$f(x) = \begin{cases} \alpha(e^x - 1) & \text{for } x < 0 \\ x & \text{for } x \geq 0 \end{cases}$	$f'(x) = \begin{cases} f(x) + \alpha & \text{for } x < 0 \\ 1 & \text{for } x \geq 0 \end{cases}$
SoftPlus		$f(x) = \log_e(1+e^x)$	$f'(x) = \frac{1}{1+e^{-x}}$

Tomado de Wikipedia (Activation Function)