

# Aprendizaje Profundo

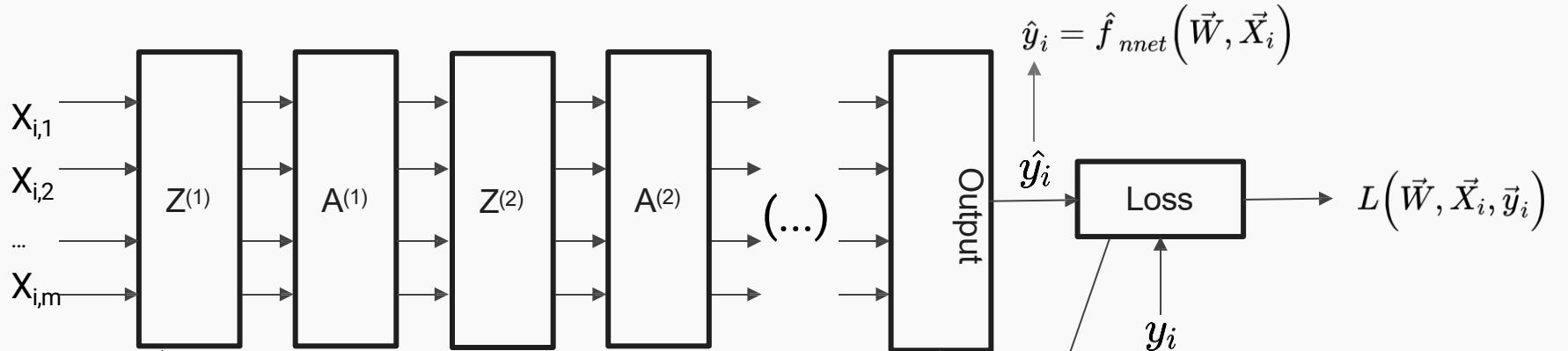
Facultad de Ingeniería  
Universidad de Buenos Aires



Profesores:

Marcos Maillot  
Gerardo Vilcamiza

# Red neuronal feedforward con p layers



Layer:

- Lineal
- Recurrente
- Convolucional
- Attention

Activación:

- Sigmoid
- ReLU
- Tanh

Optimización:

- GD, SGD, Mini-Batch
- AdaGrad, RMSprop, Adam (2014)

- Regresión → MSE
- Clasificación binaria → Binary Cross Entropy
- Clasificación multiclas → Cross Entropy

- Regresión →  $\emptyset$
- Clasificación binaria → Sigmoid
- Clasificación multiclas → Softmax

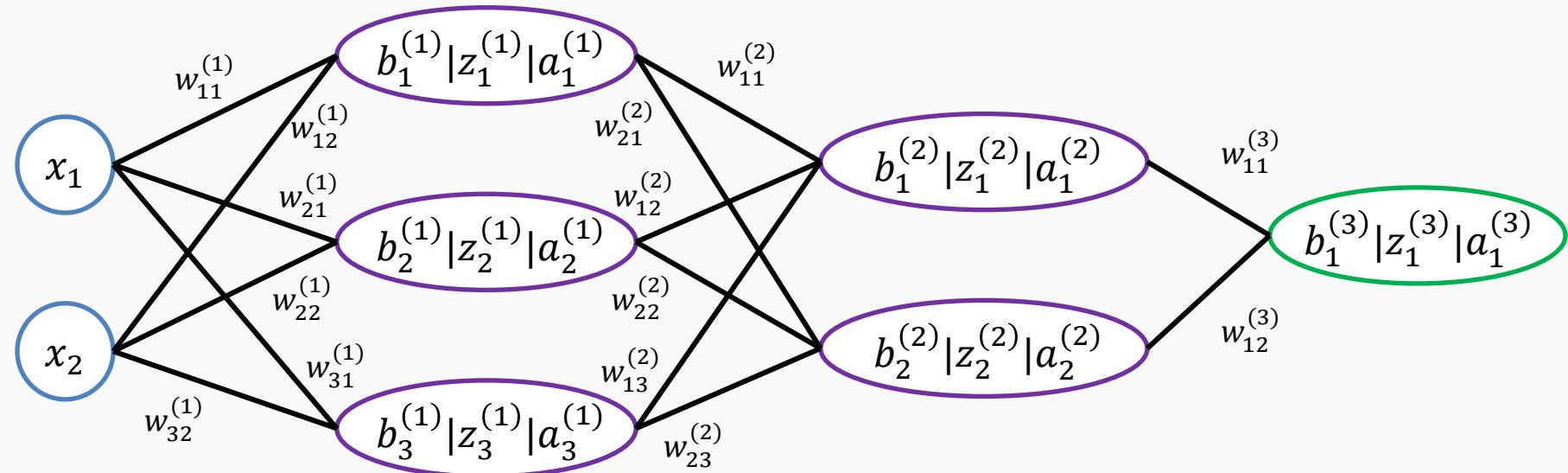
# Backpropagation

# Backpropagation

Capa de entrada

Capas ocultas

Capa de salida



Layer 0

Layer 1

Layer 2

Layer 3

# Backpropagation

## Regla de la cadena

$$\frac{\partial C}{\partial W^{(L)}} = \frac{\partial C}{\partial a^{(L)}} \cdot \frac{\partial a^{(L)}}{\partial z^{(L)}} \cdot \frac{\partial z^{(L)}}{\partial w^{(L)}} = (a_j^{(L)} - y_j) \cdot a^{(L)}(z^{(L)}) \cdot (1 - a^{(L)}(z^{(L)})) \cdot a_i^{(L-1)}$$

$$\frac{\partial C}{\partial b^{(L)}} = \frac{\partial C}{\partial a^{(L)}} \cdot \frac{\partial a^{(L)}}{\partial z^{(L)}} \cdot \frac{\partial z^{(L)}}{\partial b^{(L)}} = (a_j^{(L)} - y_j) \cdot a^{(L)}(z^{(L)}) \cdot (1 - a^{(L)}(z^{(L)})) \cdot 1$$

## Error imputado a la neurona

$$\delta^{(L)} = \frac{\partial C}{\partial z^{(L)}} = \frac{\partial C}{\partial a^{(L)}} \cdot \frac{\partial a^{(L)}}{\partial z^{(L)}} = (a_j^{(L)} - y_j) \cdot a^{(L)}(z^{(L)}) \cdot (1 - a^{(L)}(z^{(L)}))$$

# Backpropagation

## Propagación del error hacia capas anteriores

$$\delta^{(L-1)} = \frac{\partial C}{\partial z^{(L-1)}} = \frac{\partial C}{\partial z_j^{(L)}} \cdot \frac{\partial z_j^{(L)}}{\partial a^{(L-1)}} \cdot \frac{\partial a^{(L-1)}}{\partial z^{(L-1)}}$$

$$\frac{\partial z_j^{(L)}}{\partial a_i^{(L-1)}} = w_{ji}^{(L)} = W^{(L)}$$

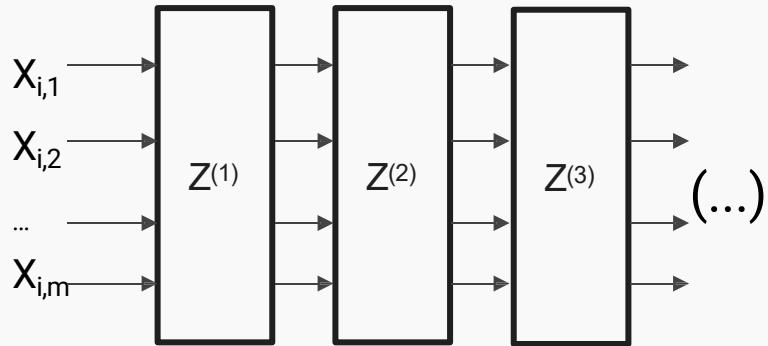
$$\frac{\partial C}{\partial z_j^{(L)}} = \delta_j^{(L)}$$

$$\delta_i^{(L-1)} = \delta_j^{(L)} \cdot W^{(L)} \cdot \frac{\partial a^{(L-1)}}{\partial z^{(L-1)}}$$

# Funciones de activación

# Funciones de activación

- No usar función de activación



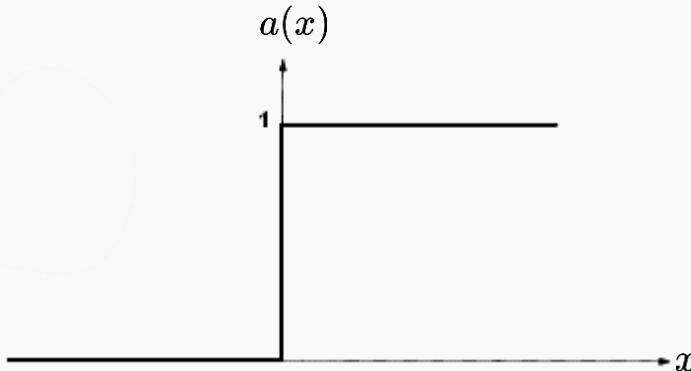
$$\begin{aligned}\vec{Z}_i^{(1)} &= \vec{W}^{(1)} \cdot \vec{X}_i + \vec{b}^{(1)} \\ \vec{Z}_i^{(2)} &= \vec{W}^{(2)} \cdot \vec{Z}_i^{(1)} + \vec{b}^{(2)} = \vec{W}^{(2)} \cdot \left( \vec{W}^{(1)} \cdot \vec{X}_i + \vec{b}^{(1)} \right) + \vec{b}^{(2)} \\ &= \left( \vec{W}^{(2)} \cdot \vec{W}^{(1)} \right) \cdot \vec{X}_i + \left( \vec{W}^{(2)} \cdot \vec{b}^{(1)} + \vec{b}^{(2)} \right) \\ &\downarrow \\ &\vec{W}' \\ &\downarrow \\ &\vec{b}' \\ \hat{y}_i &= \vec{W}' \cdot \vec{X}_i + \vec{b}' \quad \text{Regresión Lineal}\end{aligned}$$

¡Debo utilizar una función de activación no lineal!



# Funciones de activación

## Escalón



$$a(x) = \begin{cases} 1, & x > 0 \\ 0, & x \leq 0 \end{cases} \longrightarrow a'(x) = \begin{cases} 0, & x > 0 \\ 0, & x < 0 \\ ? & x = 0 \end{cases}$$

$$\partial L / \partial W_{1,1}^{(1)} = \partial L / \partial \hat{y}_i \cdot \partial \hat{y}_i / \partial a_{i,1}^{(1)} \cdot \boxed{\partial a_{i,1}^{(1)} / \partial Z_{i,1}^{(1)}} \cdot \partial Z_{i,1}^{(1)} / \partial W_{1,1}^{(1)}$$

$$\partial L / \partial W_{1,1}^{(1)} = 0$$

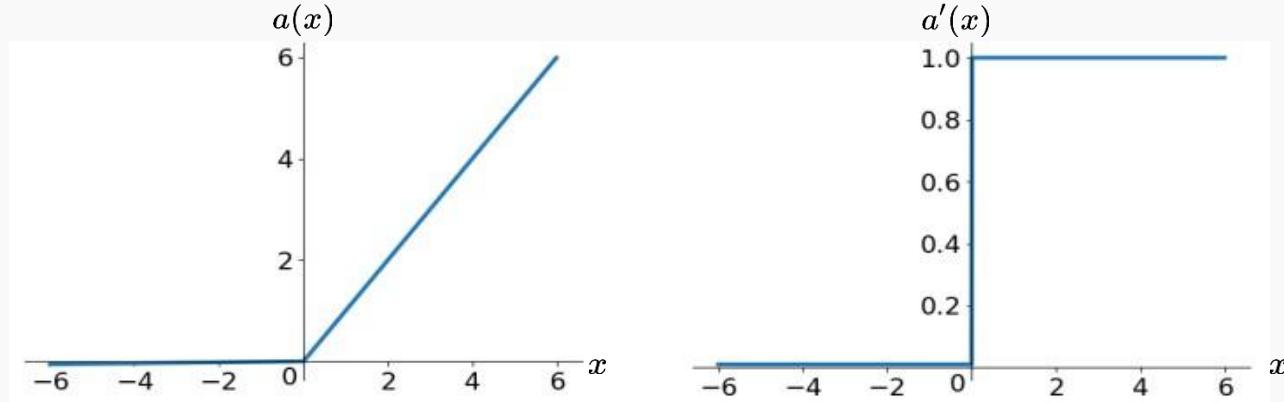
$$W_{1,1}^{(1)} \leftarrow W_{1,1}^{(1)} - \alpha \cdot \cancel{\partial L / \partial W_{1,1}^{(1)}}$$

Me cancela el Backpropagation y el modelo deja de aprender

# Funciones de activación

## ReLU

- **Paso Forward y Backward extremadamente simples de calcular (entrenamiento muy rápido)**
- **Los gradientes negativos se anulan (problema de las neuronas muertas)**



$$a(x) = \begin{cases} x, & x > 0 \\ 0, & x \leq 0 \end{cases}$$
$$a(x) = \max(0, x)$$

$$a'(x) = \begin{cases} 1, & x > 0 \\ 0, & x \leq 0 \end{cases}$$

¿Cuándo usarla?

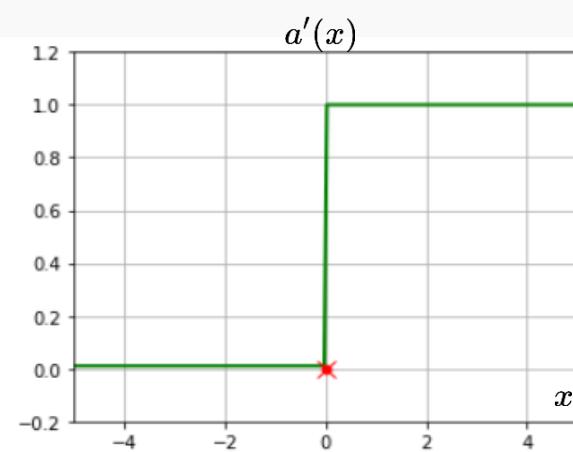
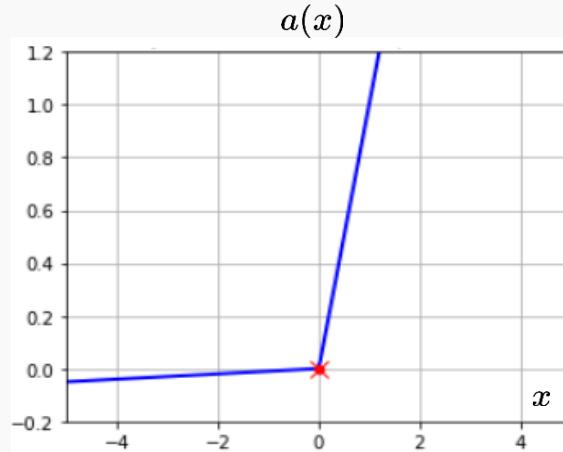
- Para capas ocultas de redes profundas en la mayoría de tareas (buen punto de partida).
- Default seguro cuando no hay razón específica para otra.



# Funciones de activación

## Leaky ReLU

- Permite gradientes pequeños en la zona negativa (evita neuronas muertas).
- Introduce un hiperparámetro ( $\alpha$ ) más que se debe configurar.



$$a(x) = \begin{cases} x, & x > 0 \\ \beta \cdot x, & x \leq 0 \end{cases}$$

$$a(x) = \max(\beta \cdot x, x)$$

Valor típico  $\beta = 0.01$

$$a'(x) = \begin{cases} 1, & x > 0 \\ \beta, & x \leq 0 \end{cases}$$

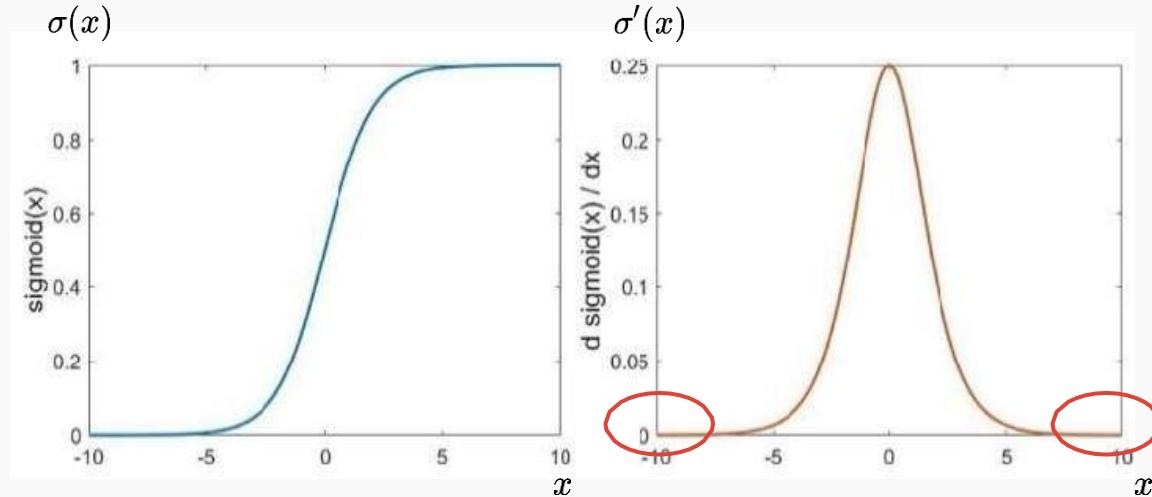
### ¿Cuándo usarla?

- En redes donde observas que muchas neuronas quedan inactivas con ReLU.
- En tareas donde los datos pueden tener un rango más amplio o valores negativos significativos (por ejemplo, series temporales o audio).

# Funciones de activación

## Sigmoid

- Interpretable como probabilidad.
- Normaliza la salida entre 0 y 1.
- Gradiente suave y efecto gradual
- Saturación: gradientes muy pequeños en extremos (vanishing gradient)
- Lento entrenamiento cuando se usa en capas ocultas.



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

$$\sigma'(x) = \sigma(x) \cdot (1 - \sigma(x))$$

¿Cuándo usarla?

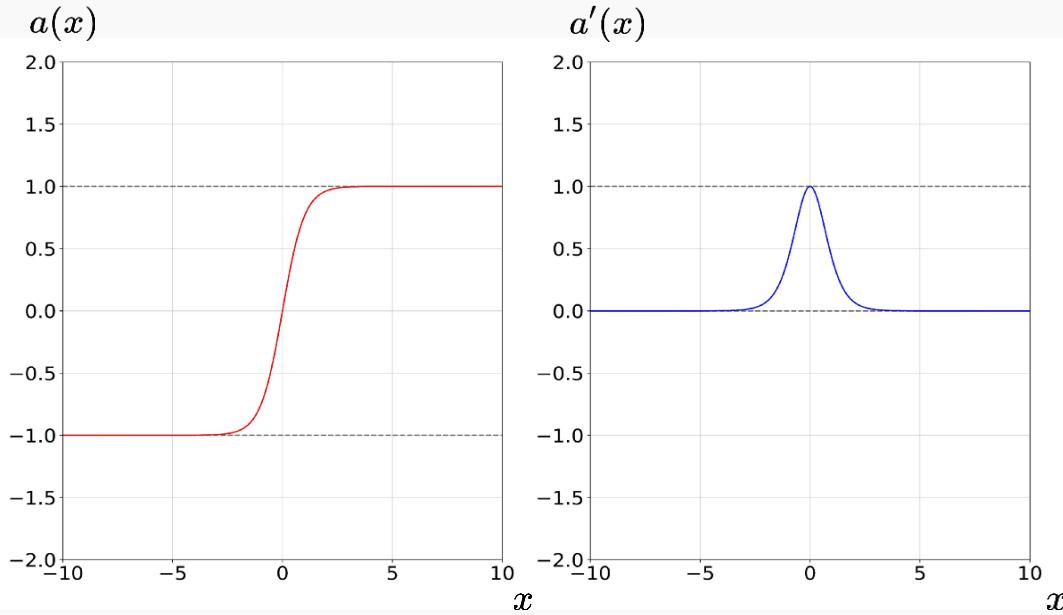
- En la capa de salida para modelos de clasificación binaria (solo dos clases).
- En redes pequeñas donde la estabilidad del gradiente no sea crítica.



# Funciones de activación

## Tanh

- Mantiene una salida acotada entre (-1, 1).
- Centrada en cero (más rápida que la sigmoide para capas ocultas).
- Saturación: gradientes muy pequeños en extremos (vanishing gradient)



$$\tanh(x) = 2 \cdot \sigma(x) - 1$$

$$1 - (\tanh(x))^2$$

¿Cuándo usarla?

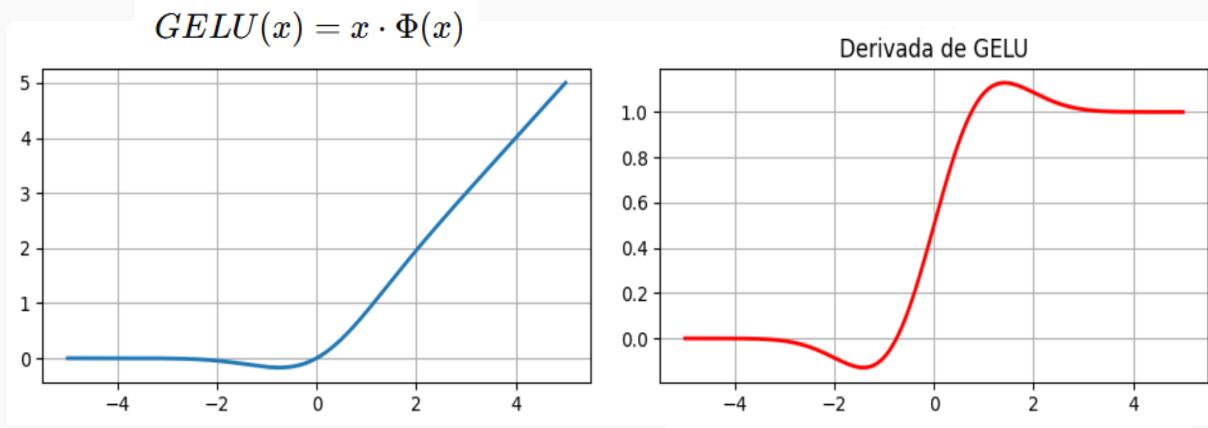
- En capas ocultas de redes más pequeñas o RNNs cuando los datos tienen valores tanto positivos como negativos.
- En la capa de salida de modelos de regresión, cuando se desea valores entre -1 y 1 (rango simétrico).



# Funciones de activación

## GELU

- Mezcla lo mejor de ReLU y funciones suaves.
- Más “probabilística”: permite pequeñas activaciones negativas con peso proporcional a su valor.
- Mejora la estabilidad en Transformers y modelos modernos.
- Computacionalmente más costosa y lenta.



$$GELU(x) = \frac{x}{2} \left[ 1 + \text{erf}\left(\frac{x}{\sqrt{2}}\right) \right]$$

$$\text{erf}(x) = \frac{2}{\sqrt{\pi}} \int_0^x e^{-t^2} dt$$

$$GELU'(x) = \Phi(x) + x \cdot \phi(x)$$

$$\phi(x) = \frac{1}{\sqrt{2\pi}} e^{-x^2/2}$$

$$GELU(x) \approx 0.5x \left[ 1 + \tanh\left(\sqrt{\frac{2}{\pi}}(x + 0.044715x^3)\right) \right]$$

¿Cuándo usarla?

- En modelos modernos tipo Transformer, Vision Transformers, BERT, T5, etc.
- Cuando deseas suavidad y robustez numérica sin perder no linealidad.

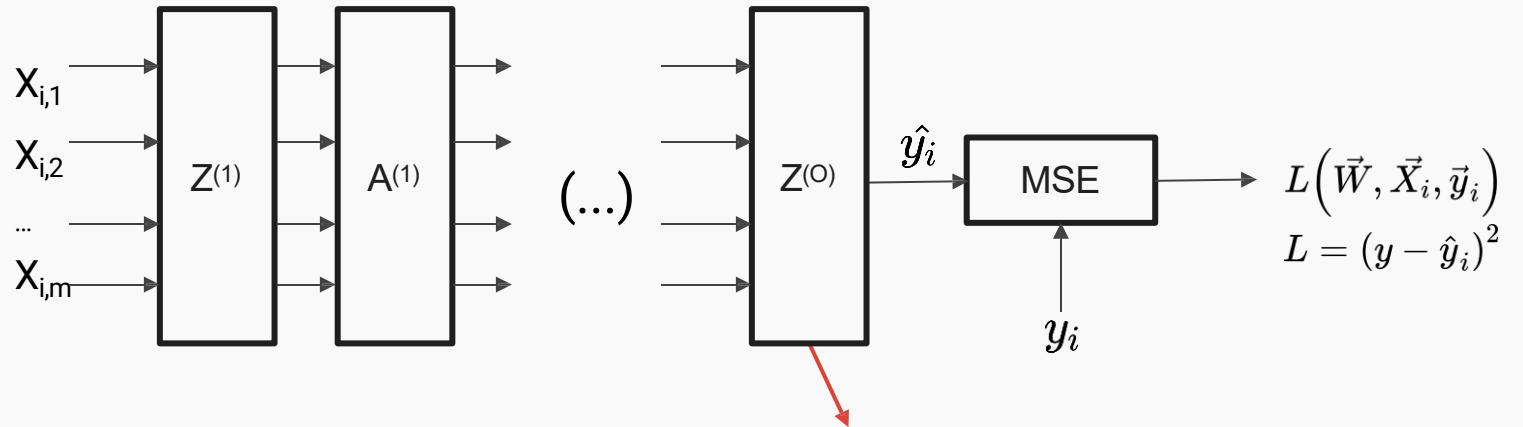


# Funciones de salida

## Loss function

# Funciones de salida + Loss function

## - Regresión



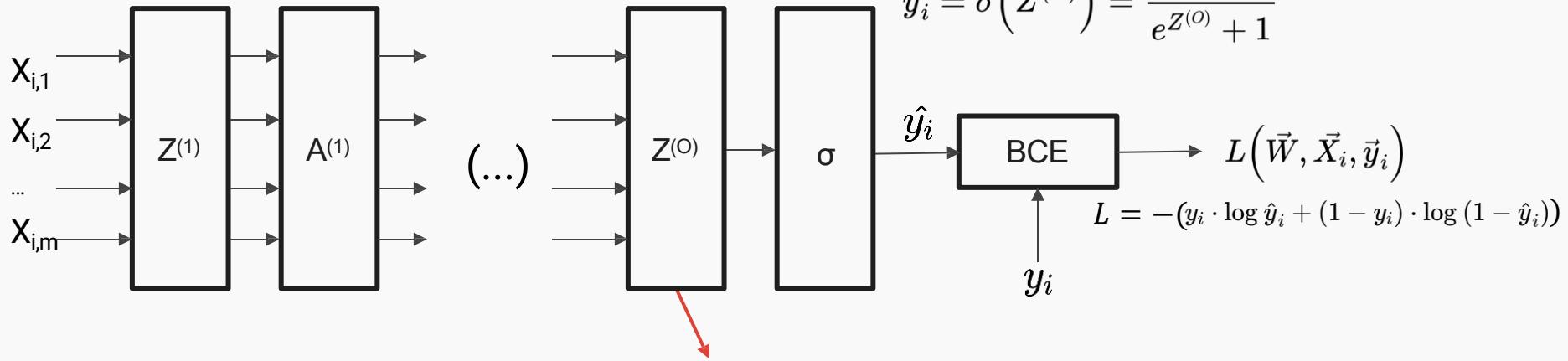
$$\begin{aligned}\hat{y}_i &\in R \\ y_i &\in R \\ L &\in R\end{aligned}$$

1 sola salida = 1 sola neurona

$$MSE = \frac{1}{n} \sum_{i=1}^n (y_i - \hat{y}_i)^2$$

# Funciones de salida + Loss function

## - Clasificación binaria



1 sola salida = 1 sola neurona

$$Z_i^{(o)} \in R$$

$\hat{y}_i \in (0, 1) \rightarrow$  probabilidad de clasificar  $y_i = 1$

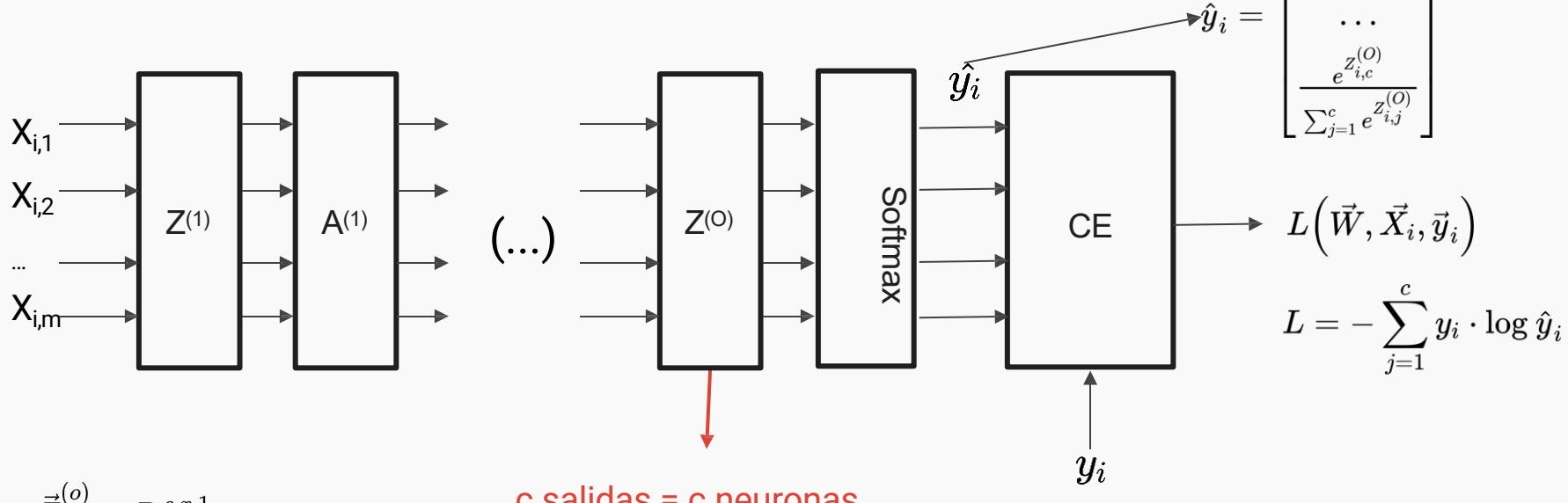
$$y_i \in \{0, 1\}$$

$$L \in R$$



# Funciones de salida + Loss function

## - Clasificación multiclas



$$\vec{Z}_i^{(o)} \in R^{c \times 1},$$

**c salidas = c neuronas**

$\vec{y}_i \in R^{c \times 1} / \hat{y}_{i,j} \in [0, 1] \forall j \in \{1, \dots, c\}$  → probabilidad de clasificar  $y_i$  en cada clase

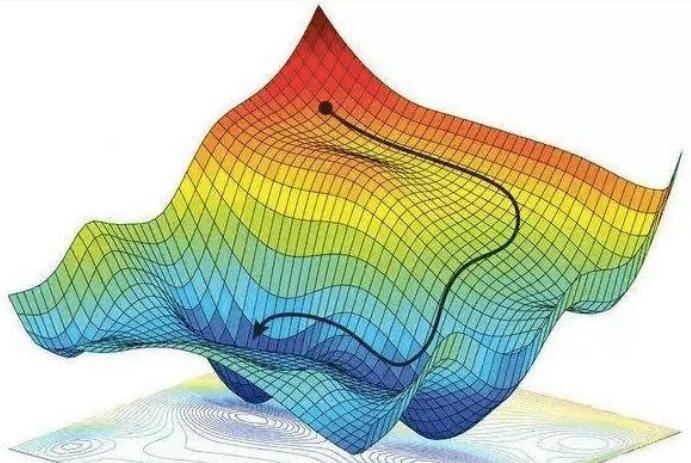
$y_i \in R^{c \times 1}$  → one hot encoding

$L \in R$

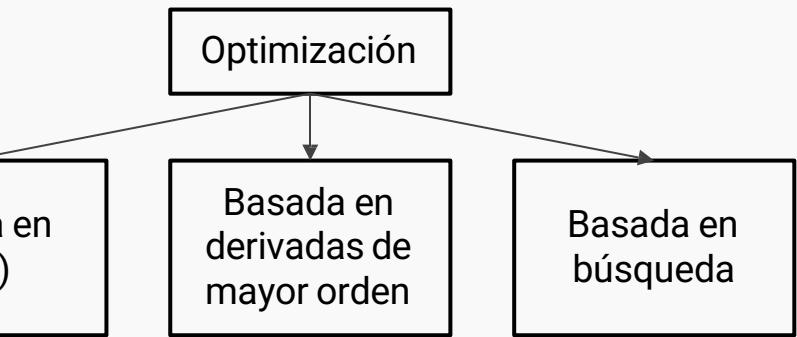


# Optimización

# Optimización



Objetivo: encontrar el mínimo



Basada en  
 $\nabla_w(L)$

Basada en derivadas de mayor orden

Basada en búsqueda

- GD
- SGD
- Mini-Batch

- Newton
- BFGS
- L-BFGS

- Hill Climbing
- Genéticos

AdaGrad

RMSProp

Adam (2014)



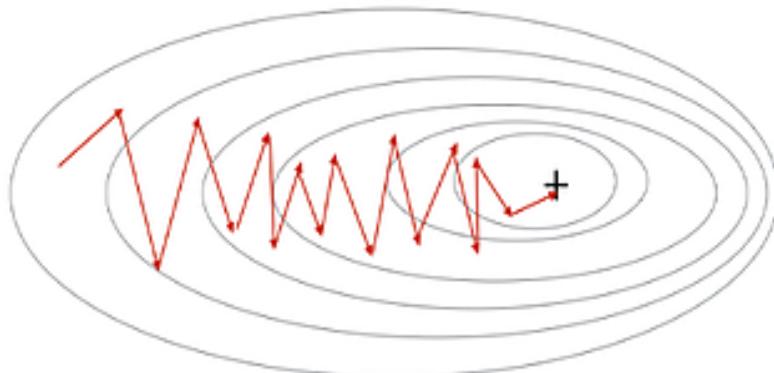
# Optimización

$$\text{MSE} = \frac{1}{n} \sum_{i=1}^n (y - \hat{y}_i)^2 = \frac{1}{n} \sum_{i=1}^n l_i(\vec{W}, \vec{X}_i, y_i)$$

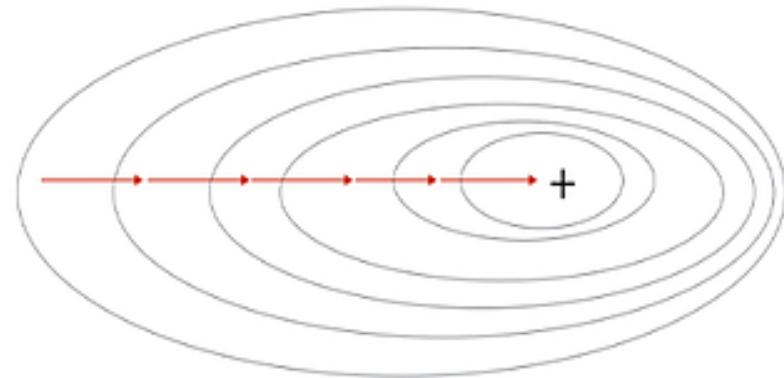
	GD	SGD	Mini-Batch
Cálculo del gradiente	$\nabla_{\vec{W}} = \vec{\nabla} \left( \frac{1}{n} \sum_{i=1}^n l_i(\vec{W}, \vec{X}_i, y_i) \right)$	$\nabla_{\vec{W}} = \vec{\nabla} \left( l_i(\vec{W}, \vec{X}_i, y_i) \right)$	$\nabla_{\vec{W}} = \vec{\nabla} \left( \frac{1}{b} \sum_{i=1}^b l_i(\vec{W}, \vec{X}_i, y_i) \right)$
# actualizaciones de W	n_epoch	n_epoch x n	n_epoch x (n / b)
Cantidad de memoria	Mucha memoria $O(n)$	Muy poca memoria $O(1)$	Intermedio $O(b)$
Velocidad cálculos	Muy rápido $O(n\_epoch)$	Muy lento $O(n\_epoch \times n)$	Intermedio $O(n\_epoch \times (n / b))$

# Optimizadores

Stochastic Gradient Descent



Gradient Descent



# Optimización

- **Mini-Batch**

for epoch in n\_epoch:

    for b in batches:  $\longrightarrow$  # batches: n/b

        \* Forward:  $\vec{y}_b = \hat{f}_{nnet}(\vec{X}_b)$   $\longrightarrow \vec{X}_b \in R^{b \times m}$

        \* Error  $\rightarrow L(\vec{y}_b, \hat{y}_b)$

        \* Backward  $\rightarrow \vec{\nabla}_{\vec{W}} = \frac{1}{b} \sum_{i=1}^b l_i(y_i, \hat{y}_i)$

        \*  $\vec{W} \leftarrow \vec{W} - \vec{\alpha} \cdot \vec{\nabla}_{\vec{W}}$

- ¿Cuántas veces actualizo W por epoch? n/b veces

- ¿Cuántas veces actualizo W en total? n\_epoch\*n/b

- ¿Hiperparámetros?  
n\_epoch, b,  $\alpha$

# Optimización

- Mini-Batch + Momento de primer orden

for epoch in n\_epoch:

    for b in batches:

        \* Forward:  $\vec{\hat{y}}_b = \hat{f}_{nnet}(\vec{X}_b)$

        \* Error  $\rightarrow L(\vec{y}_b, \vec{\hat{y}}_b)$

        \* Backward  $\rightarrow \vec{\nabla}_{\vec{W}} = \frac{1}{b} \sum_{i=1}^b l_i(y_i, \hat{y}_i)$

        \*  $\vec{v} \leftarrow \xi \cdot \vec{v} + \alpha \cdot \vec{\nabla}_{\vec{W}}$    $\begin{cases} \xi = 0 & \Rightarrow \vec{V} = \alpha \cdot \vec{\nabla}_{\vec{W}} \rightarrow \text{Mini-Batch} \\ \xi > 0 & \rightarrow \text{Mini-Batch con momento de 1er orden} \end{cases}$

        \*  $\vec{W} \leftarrow \vec{W} - \vec{v}$

- ¿Hiperparámetros? n\_epoch, b,  $\xi$ ,  $\alpha$



# Optimización

- Adam (2014)

for epoch in n\_epoch:

    for b in batches:

\* Forward:  $\vec{y}_b = \hat{f}_{nnet}(\vec{X}_b)$

\* Error  $\rightarrow L(\vec{y}_b, \hat{y}_b)$

\* Backward  $\rightarrow \vec{\nabla}_{\vec{W}} = \frac{1}{b} \sum_{i=1}^b l_i(y_i, \hat{y}_i)$

\*  $\vec{v} \leftarrow p_1 \cdot \vec{v} + (1 - p_1) \cdot \vec{\nabla}_{\vec{W}}$  → Momento 1<sup>er</sup> orden

\*  $\vec{r} \leftarrow p_2 \cdot \vec{r} + (1 - p_2) \cdot \vec{\nabla}_{\vec{W}} \odot \vec{\nabla}_{\vec{W}}$  → Momento 2<sup>do</sup> orden

\*  $\vec{\Delta} \leftarrow -\frac{\alpha}{\sqrt{\vec{r}}} \cdot \vec{v}$

\*  $\vec{W} \leftarrow \vec{W} + \vec{\Delta}$

- ¿Hiperparámetros? n\_epoch, b, p<sub>1</sub>, p<sub>2</sub>, α

↓ Hadamard product  
(element-wise product)

