

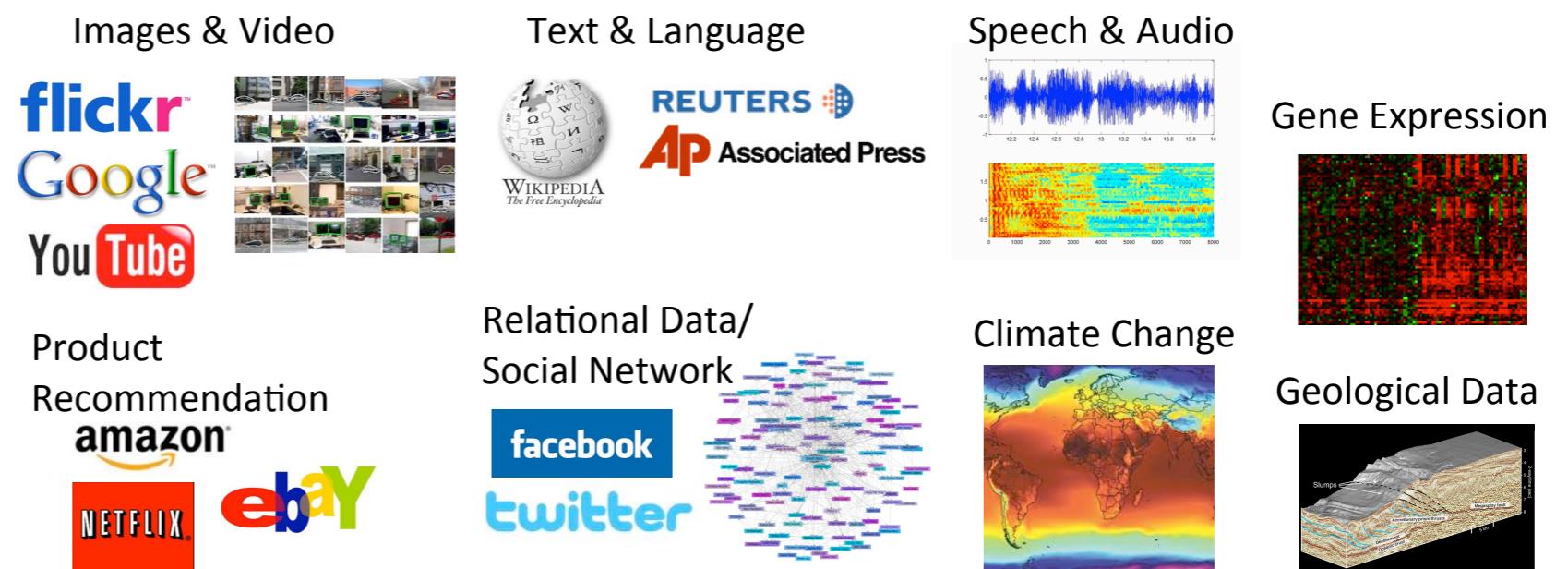
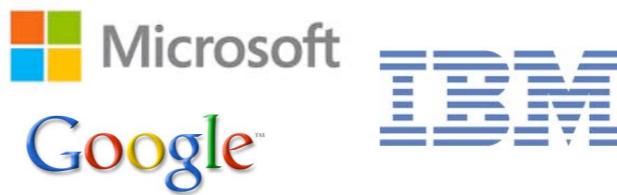
INTRODUCCIÓN A NLP CON APRENDIZAJE PROFUNDO

Pablo Martínez Olmos, pamartin@ing.uc3m.es

Aprendizaje profundo (deep learning)

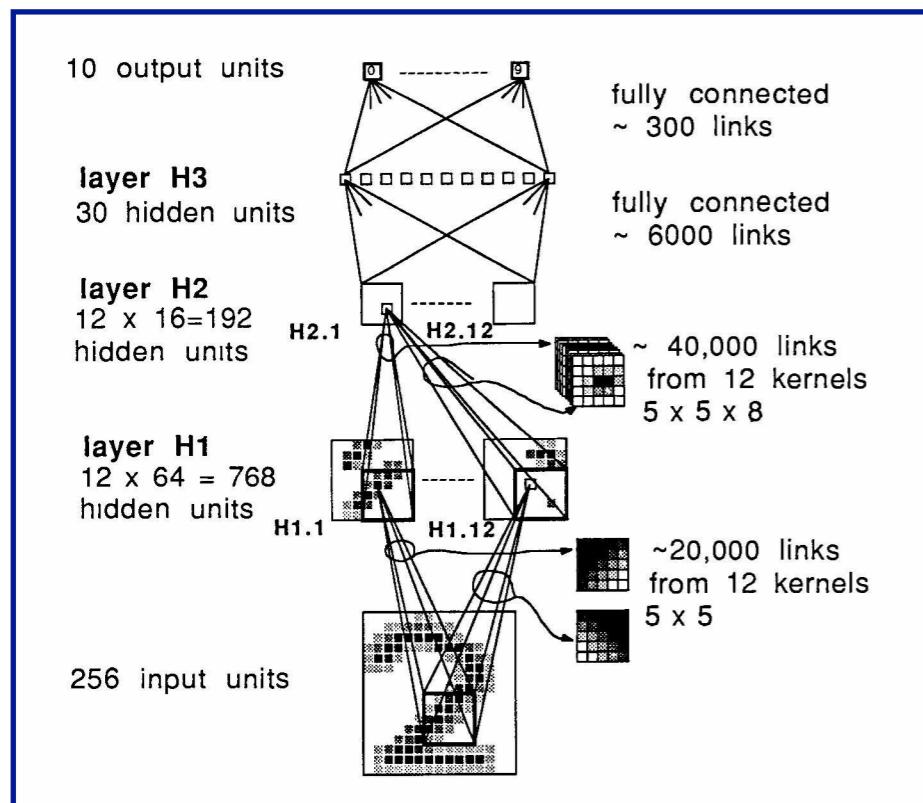
Impacto aprendizaje profundo

- Reconocimiento del habla
- Visión por ordenador
- Sistemas de recomendación
- Procesado de Lenguaje Natural
- Sistemas autónomos
- Bioinformática
- Imagen médica



Conceptos básicos fueron propuestos hace 40 años!

A History of Deep Learning ([link](#))



Backpropagation applied to Handwritten Zip Code Recognition
LeCun et al. (1989)

The backward pass starts by computing $\partial E / \partial y$ for each of the output units. Differentiating equation (3) for a particular case, c , and suppressing the index c gives

$$\frac{\partial E}{\partial y_j} = y_j - d_j \quad (4)$$

We can then apply the chain rule to compute $\partial E / \partial x_j$

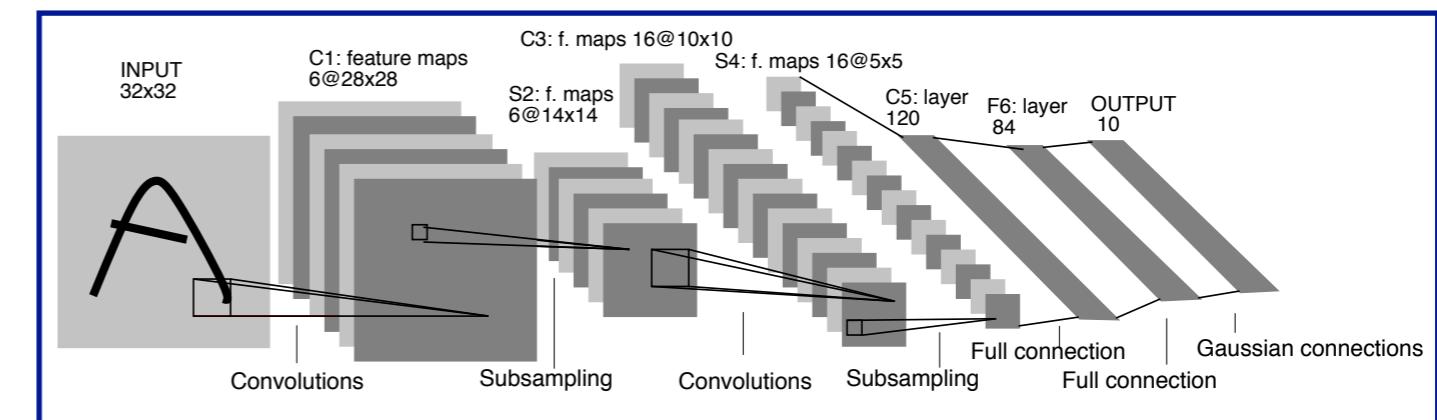
$$\frac{\partial E}{\partial x_j} = \frac{\partial E}{\partial y_j} \cdot \frac{dy_j}{dx_j} \quad (5)$$

Differentiating equation (2) to get the value of dy_j / dx_j and substituting gives

$$\frac{\partial E}{\partial x_j} = \frac{\partial E}{\partial y_j} \cdot y_j(1 - y_j) \quad (5)$$

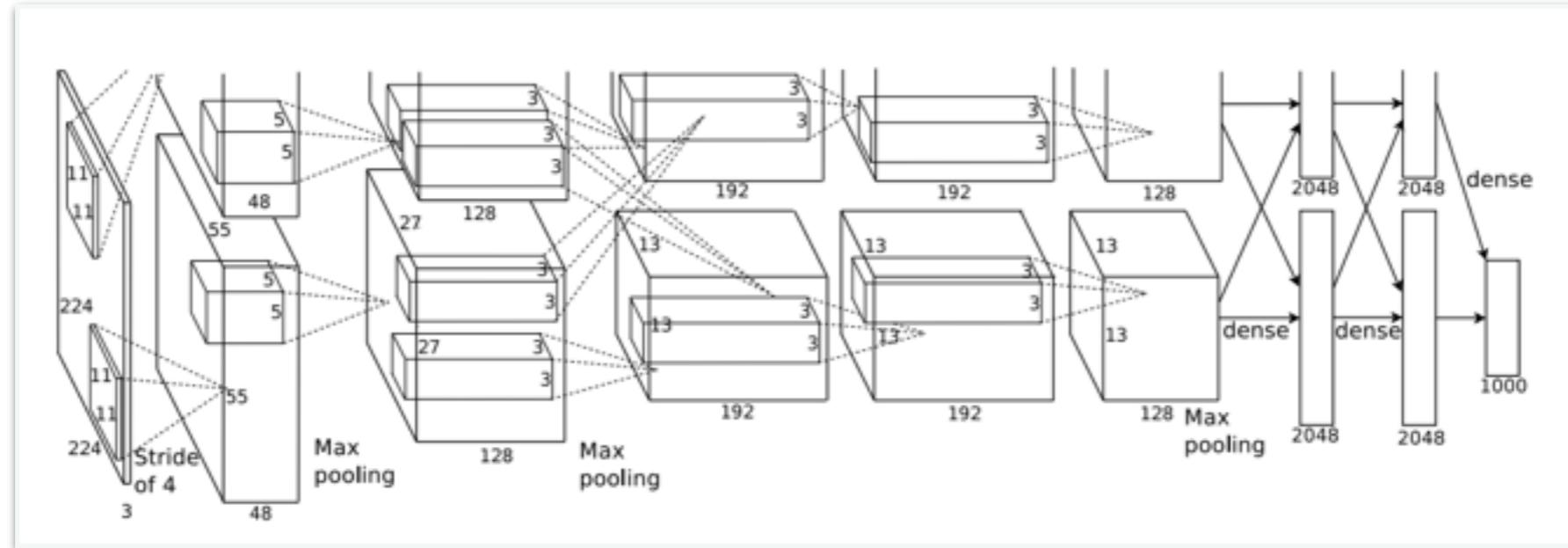
Learning representations
by back-propagating errors

Rumelhart, Hinton, Williams (1986)



Gradient-Based Learning applied to Document Recognition
LeCun, Bengio, Haffner (1998)

¿Por qué ahora?



ImageNet Classification with Deep Convolutional Neural Networks

A Krizhevsky | Sutskever, G Hinton (2012)

- CNN profunda (AlexNet) gana en 2012 la competición ImageNet classification contest
- 60 millones de parámetros!
- La capacidad de cómputo y el acceso a datos son críticos
 - Bases de públicas.
 - SIMD hardware (GPU's). Paralelización masiva.
 - Herramientas de diferenciación automática.

IMAGENET



WIKIPEDIA
The Free Encyclopedia

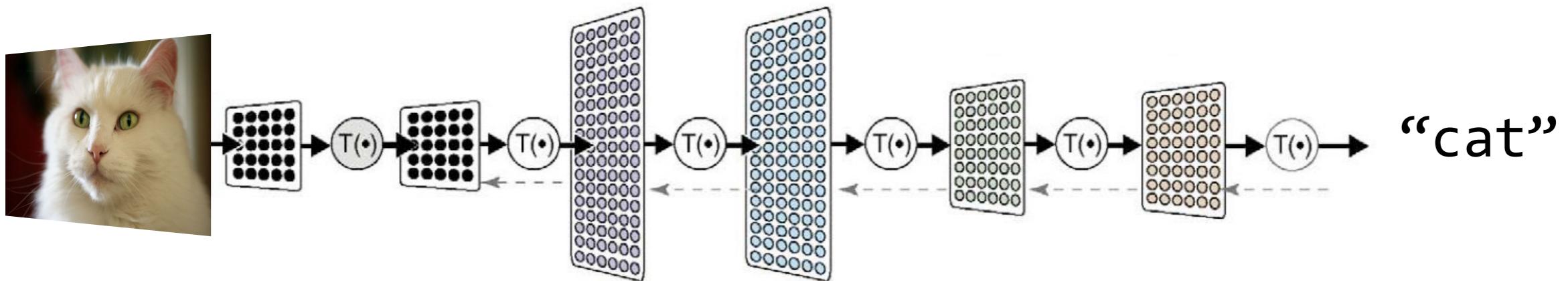


PyTorch

Conceptos básicos

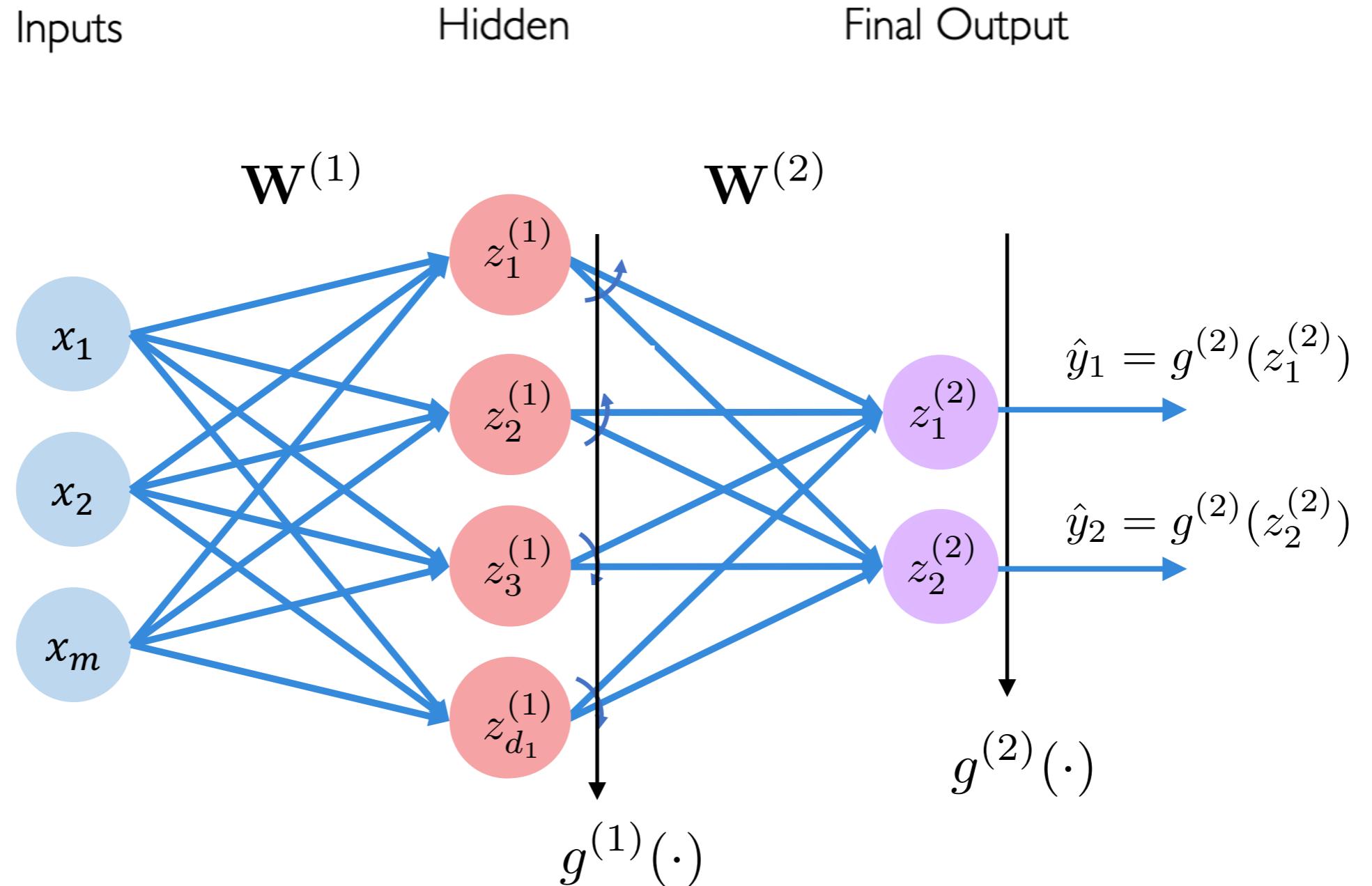
Red neuronal

Composición **jerárquica** de funciones matemáticas relativamente **simples**



Untangling invariant object recognition
J DiCarlo and D Cox (2007)

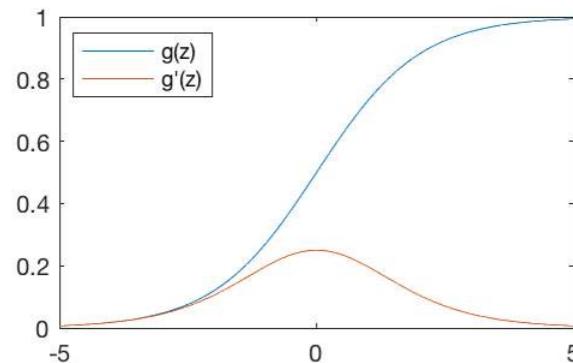
Red neuronal densa de dos capas



Funciones de activación

Introducir no linealidad en la composición ...

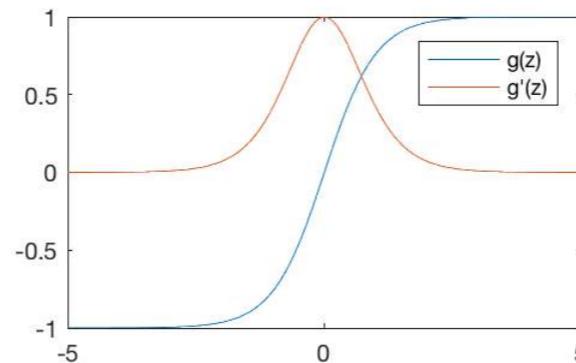
Sigmoid Function



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

$$g'(z) = g(z)(1 - g(z))$$

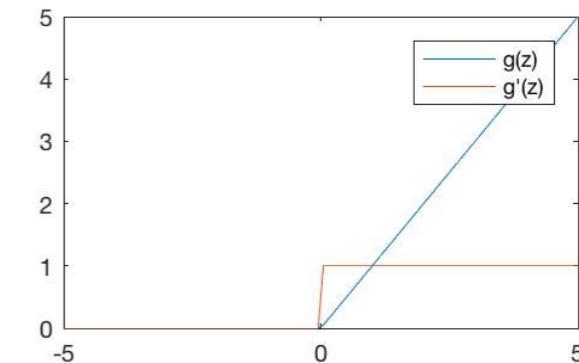
Hyperbolic Tangent



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

$$g'(z) = 1 - g(z)^2$$

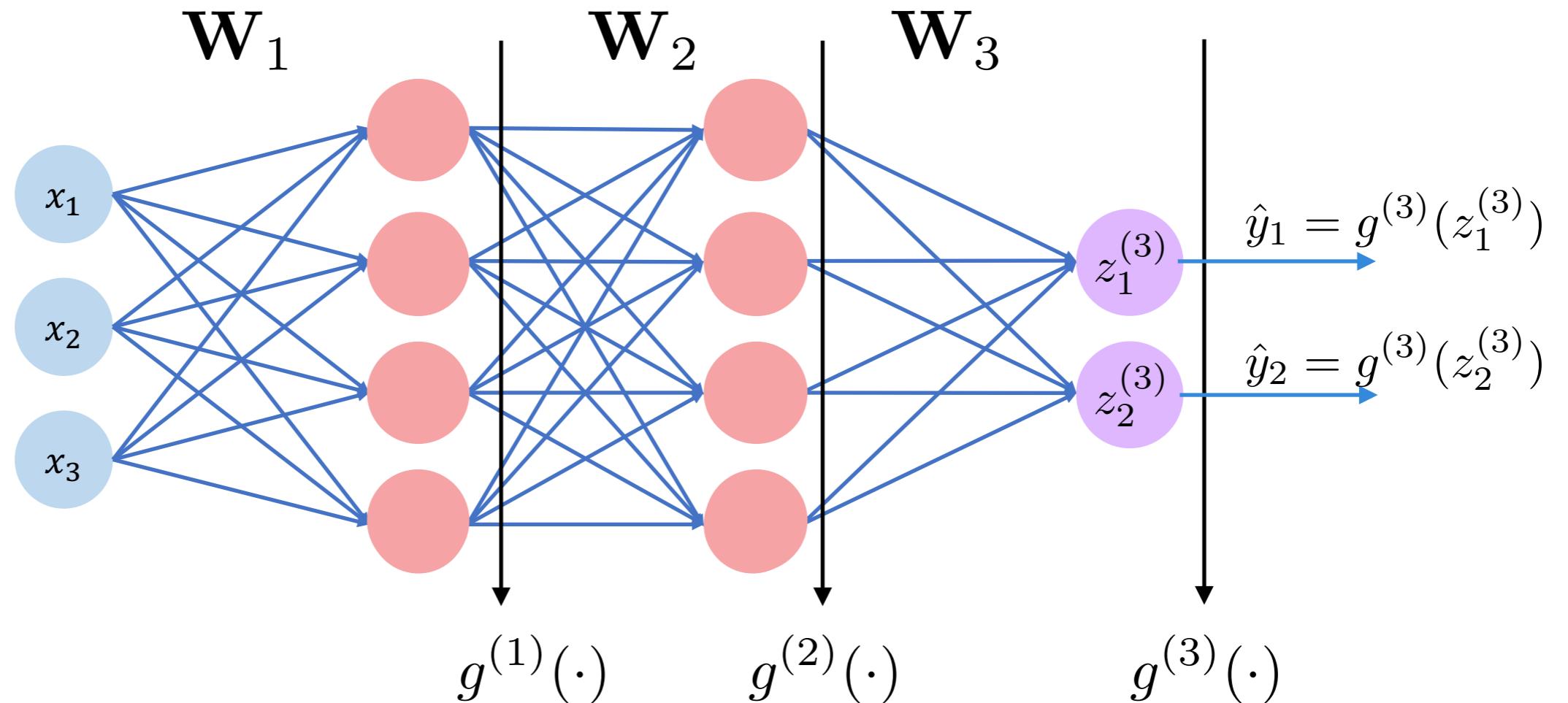
Rectified Linear Unit (ReLU)



$$g(z) = \max(0, z)$$

$$g'(z) = \begin{cases} 1, & z > 0 \\ 0, & \text{otherwise} \end{cases}$$

Red neuronal de tres capas

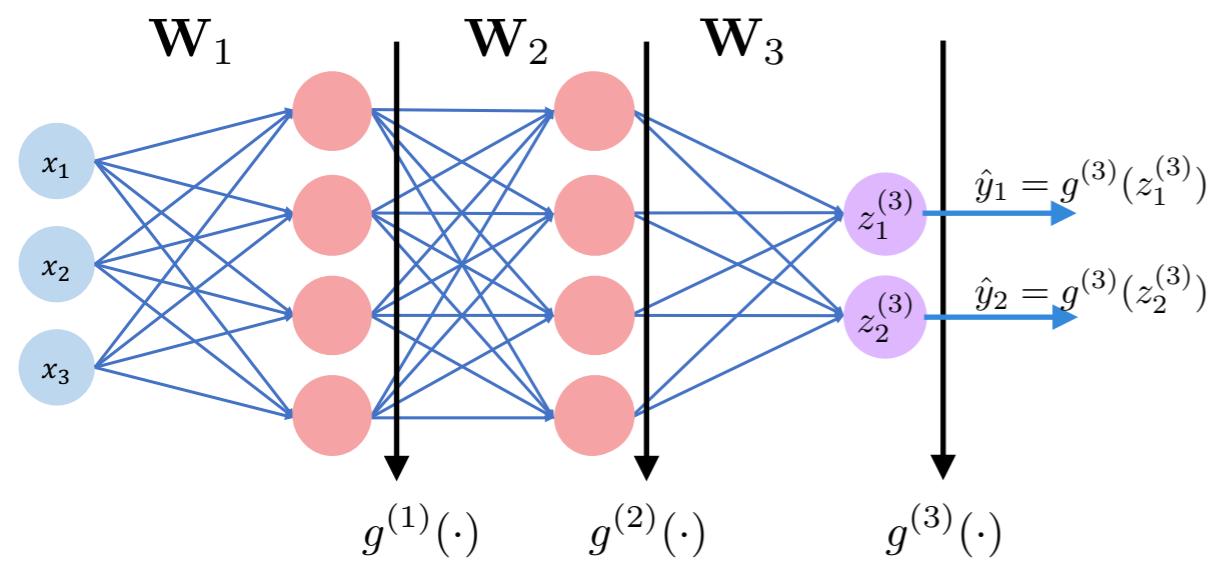


© MIT 6.S191: Introduction to Deep Learning

introtodeeplearning.com

$$\hat{\mathbf{y}} = g^{(3)} \left(\mathbf{W}^{(3)} g^{(2)} \left(\mathbf{W}^{(2)} g^{(1)} \left(\mathbf{W}^{(1)} \mathbf{x} + \mathbf{w}_0^{(1)} \right) + \mathbf{w}_0^{(2)} \right) + \mathbf{w}_0^{(3)} \right)$$

Entrenamiento supervisado



Conjunto de parámetros

$$\mathbf{W} = \left\{ \mathbf{W}^{(i)}, \mathbf{w}_0^{(i)} \right\}_{i=1}^3$$

Función de coste

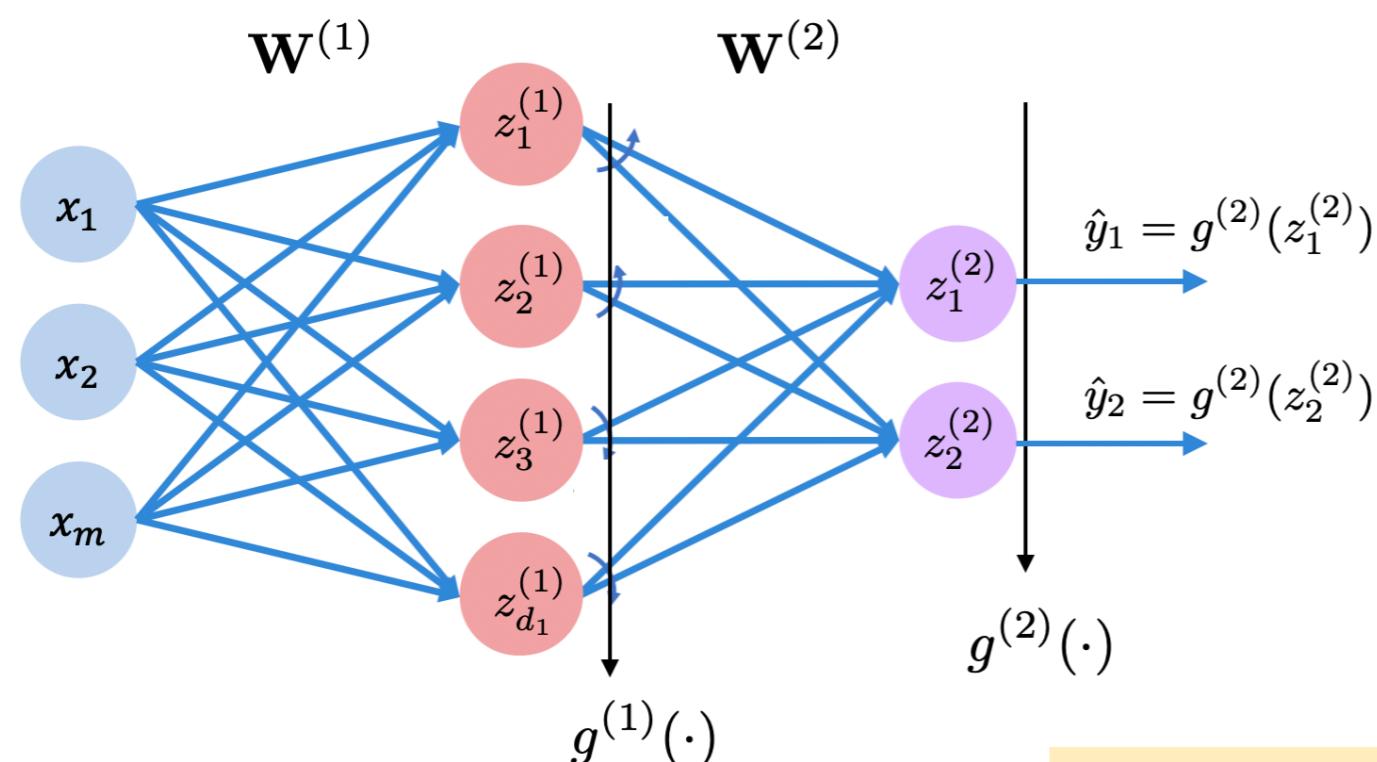
$$J(\mathbf{W}) = \frac{1}{N} \sum_{n=1}^N \mathcal{L} \left(\hat{\mathbf{y}}^{(n)}, \mathbf{y}^{(n)} \right)$$

Entrenamiento

$$\mathbf{W}^* = \arg \min_{\mathbf{W}} J(\mathbf{W})$$

Clasificación

$$\mathcal{D} \doteq (\boldsymbol{x}^{(i)}, y^{(i)})_{i=1}^N \quad \boldsymbol{x}^{(i)} \in \mathbb{R}^m \quad y^{(i)} \in \{0, 1, \dots, K\}$$



Softmax a la salida

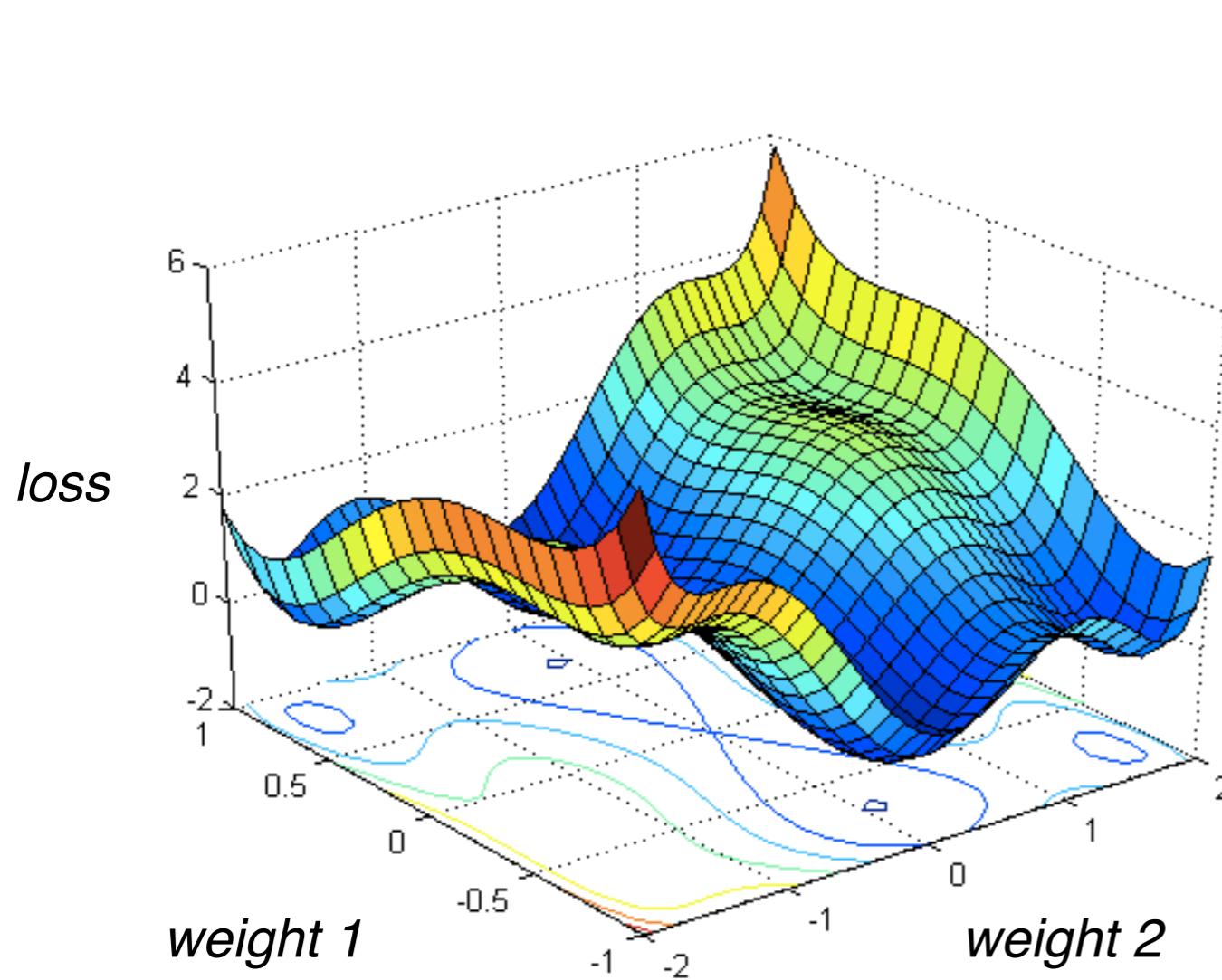
$$\begin{aligned}\hat{y}_k &= g^{(2)}(z_k^{(2)}) = P(y = k | \boldsymbol{x}) \\ &= \frac{e^{z_k^{(2)}}}{\sum_{j=1}^K e^{z_j^{(2)}}}\end{aligned}$$

Función de coste

$$\mathcal{L}(\hat{\mathbf{y}}, y) = - \sum_{k=1}^K \mathbb{I}[y == k] \log P(y = k | \boldsymbol{x})$$

Problema no convexo!! Múltiples mínimos locales

$$\mathbf{W}^* = \arg \min_{\mathbf{W}} J(\mathbf{W})$$



$$J(\mathbf{W}) = \frac{1}{N} \sum_{n=1}^N \mathcal{L} \left(\hat{\mathbf{y}}^{(n)}, \mathbf{y}^{(n)} \right)$$

Asumimos que no alcanzamos el óptimo global

Descenso por gradiente

1. Inicialización aleatoria de parámetros $\sim \mathcal{N}(0, \sigma^2)$

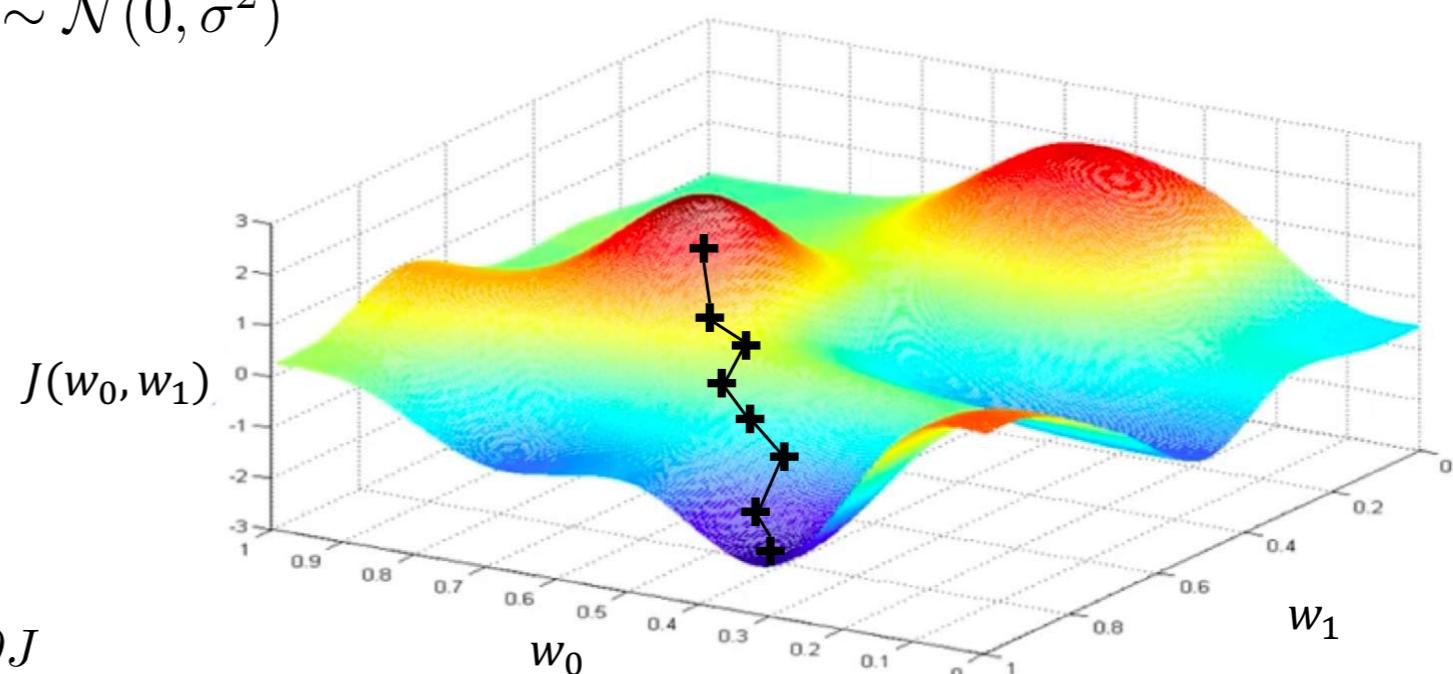
2. Hasta convergencia:

3. Cálculo de gradiente

$$\frac{\partial J}{\partial \mathbf{W}}$$

4. Actualización

$$\mathbf{W} \leftarrow \mathbf{W} - \eta \frac{\partial J}{\partial \mathbf{W}}$$



Backpropagation: Cálculo eficiente de los gradientes!

© MIT 6.S191: Introduction to Deep Learning
introtodeeplearning.com

Optimización con mini-batches

$$J(\mathbf{W}) = \frac{1}{N} \sum_{n=1}^N \mathcal{L} \left(\hat{\mathbf{y}}^{(n)}, \mathbf{y}^{(n)} \right) = \sum_{n=1}^N \mathcal{L}_n \quad \longrightarrow \quad \frac{\partial J(\mathbf{W})}{\partial w_{ji}^{(m)}} = \sum_{n=1}^N \frac{\partial \mathcal{L}_n}{\partial w_{ji}^{(m)}}$$

Alto coste en grandes volúmenes de datos

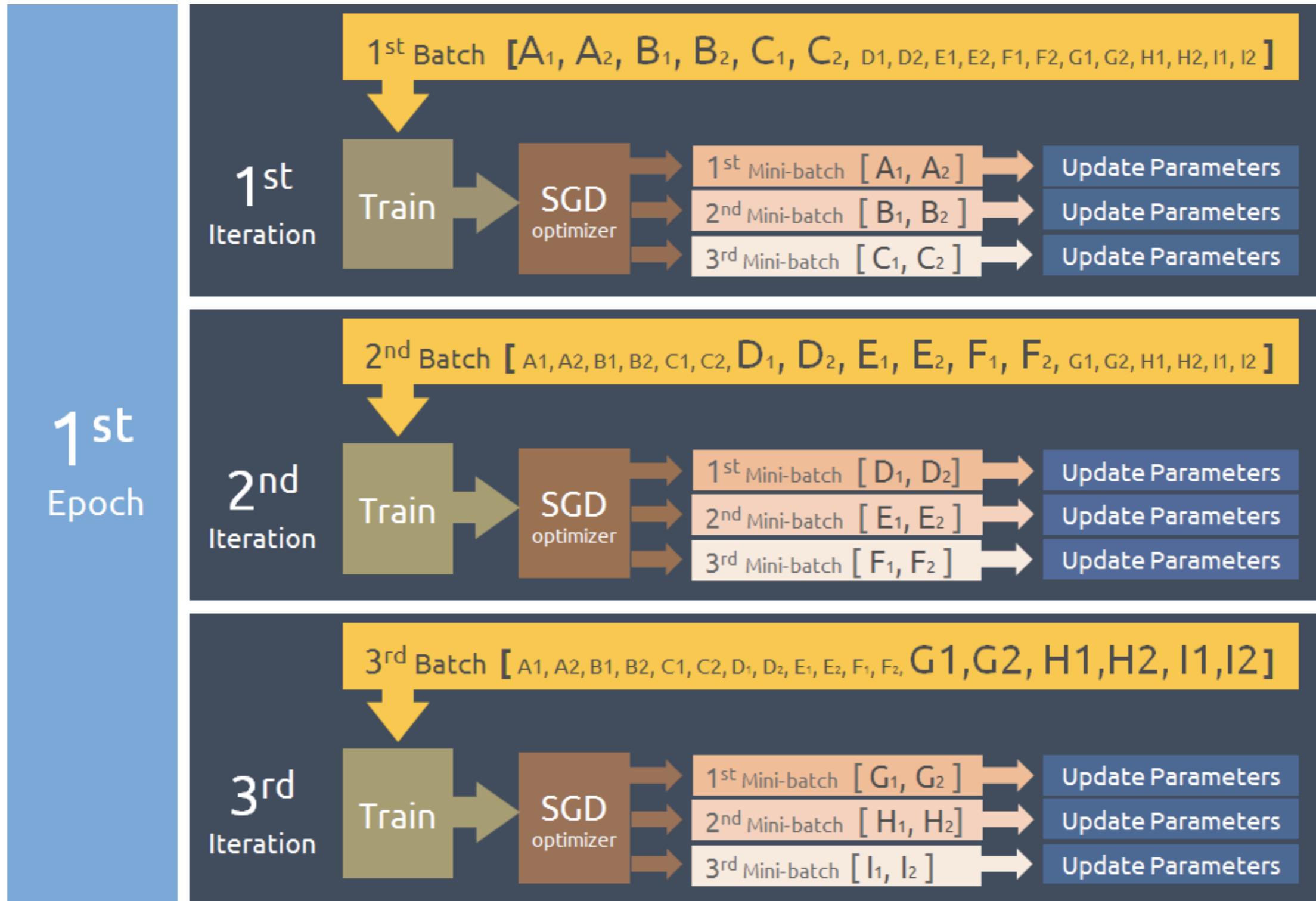
Stochastic Gradient Descent (SGD)

Seleccionamos un grupo (mini-batch) de datos \mathcal{B}

$$\frac{\partial J(\mathbf{W})}{\partial w_{ji}^{(m)}} \approx \sum_{n \in \mathcal{B}} \frac{\partial \mathcal{L}_n}{\partial w_{ji}^{(m)}}$$

Paralelizable!

Epoch / Batch / Mini-batch / Iteration

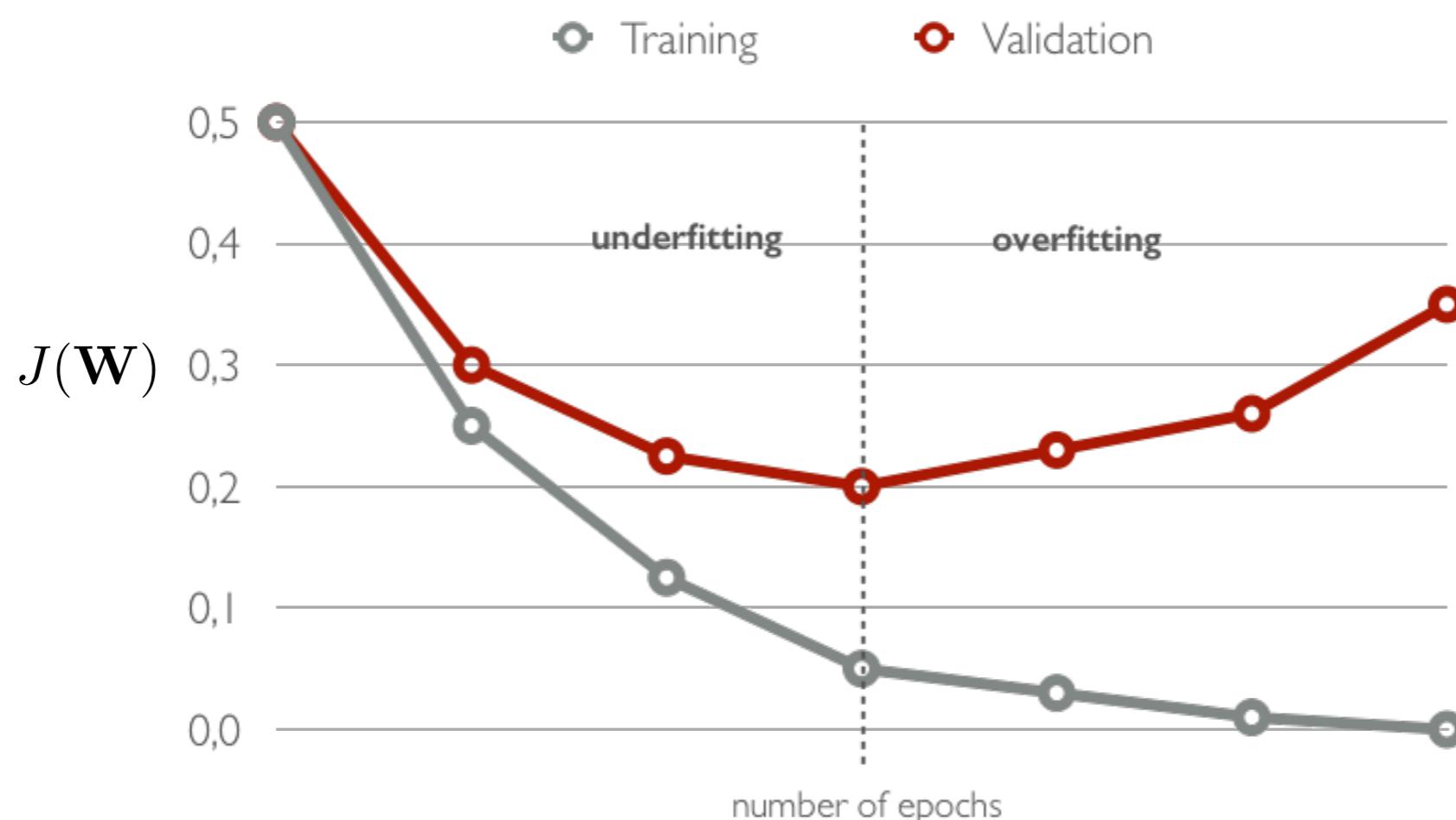


Selección del modelo

- Entrenar usando un **conjunto de entrenamiento** $\mathcal{D}_{\text{train}}$
- Ajuste hiperparámetros usando **conjunto de validación** \mathcal{D}_{val}
 - ✓ Cuántas capas, funciones de activación, cuántas iteraciones ...
- Estimar **generalización** usando conjunto de test $\mathcal{D}_{\text{test}}$

Regularización: parada prematura (early stopping)

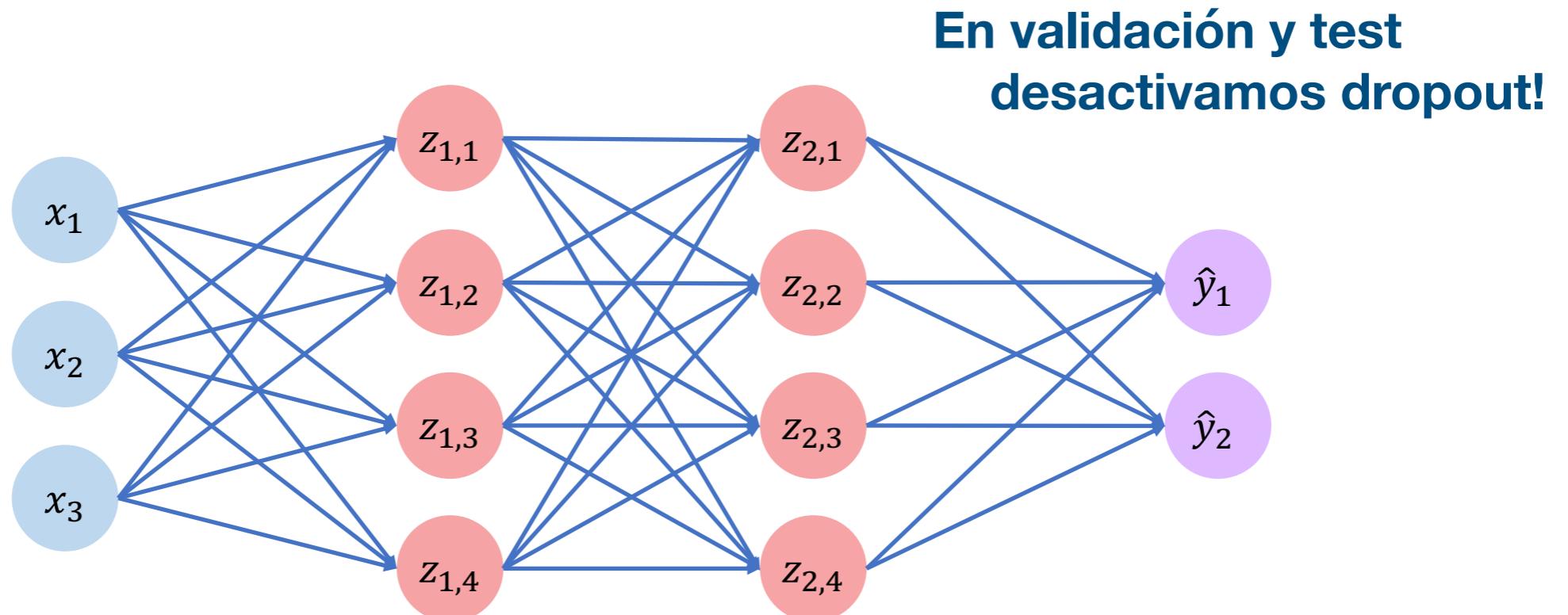
Detener entrenamiento cuando el error de validación sube



Regularización: Dropout

Durante el entrenamiento, cancelar (poner a cero) dimensiones de forma aleatoria

Esto limita la capacidad de la red para memorizar los datos de entrenamiento.



Redes neuronales con Pytorch

- Librería de código abierto para Python
- Fundamentalmente desarrollada por el grupo de IA de Facebook
- Módulo **Autograd** realiza el cálculo de los gradientes de forma automática
 - ✓ Usuario indica cómo ir de los datos a la función de coste
 - ✓ Pytorch proporciona gradientes para implementar SGD
- **Documentación:** <https://pytorch.org>