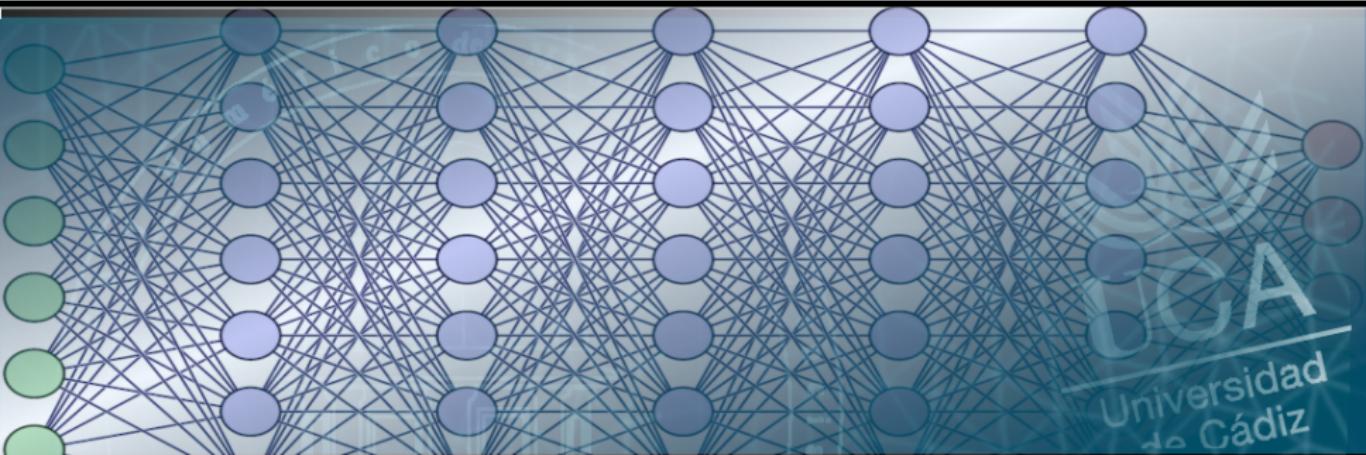


Aplicaciones de la IA al Mundo Real

Álex Pérez Fernández, Rafa Rodríguez Galván

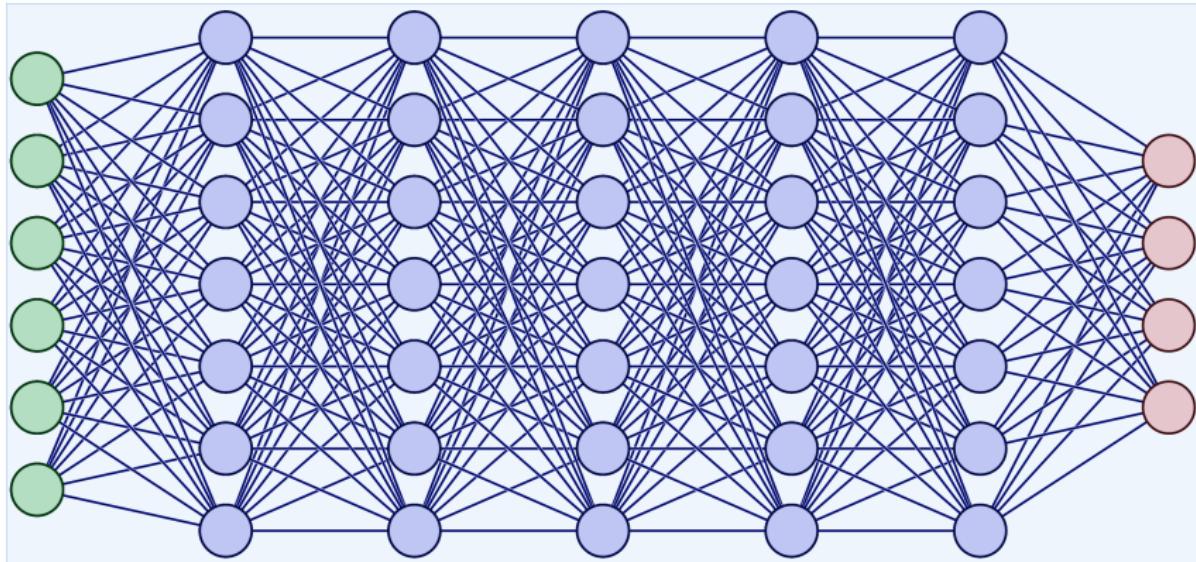
May 14, 2024



Section 1

Redes Neuronales

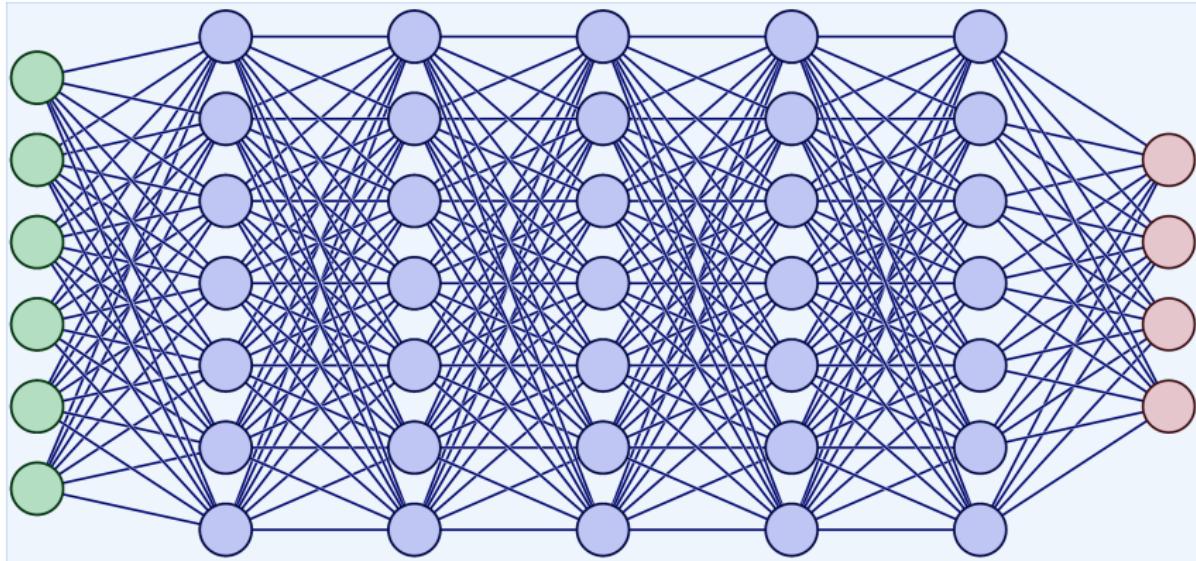
Las Redes Neuronales...



son artefactos matemáticos:

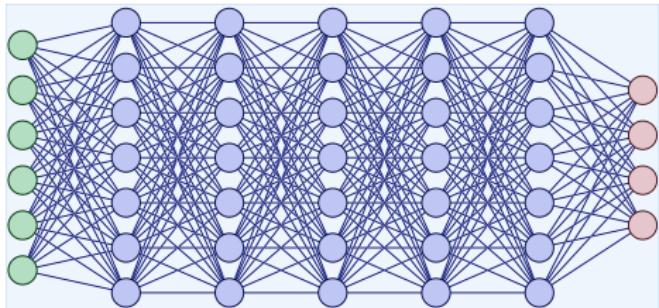
$$x \mapsto f_1(x) \mapsto f_2 \circ f_1(x) \mapsto \dots \mapsto f_L \circ \dots \circ f_2 \circ f_1(x) = y$$

Las Redes Neuronales...



son artefactos matemáticos:

$$x \mapsto f_1(x) \mapsto f_2 \circ f_1(x) \mapsto \dots \mapsto f_L \circ \dots \circ f_2 \circ f_1(x) = y$$



Definición:

una **Red Neuronal** (RN o NN) es una función $f_{NN} : \mathbb{R}^n \rightarrow \mathbb{R}^m$ del tipo:

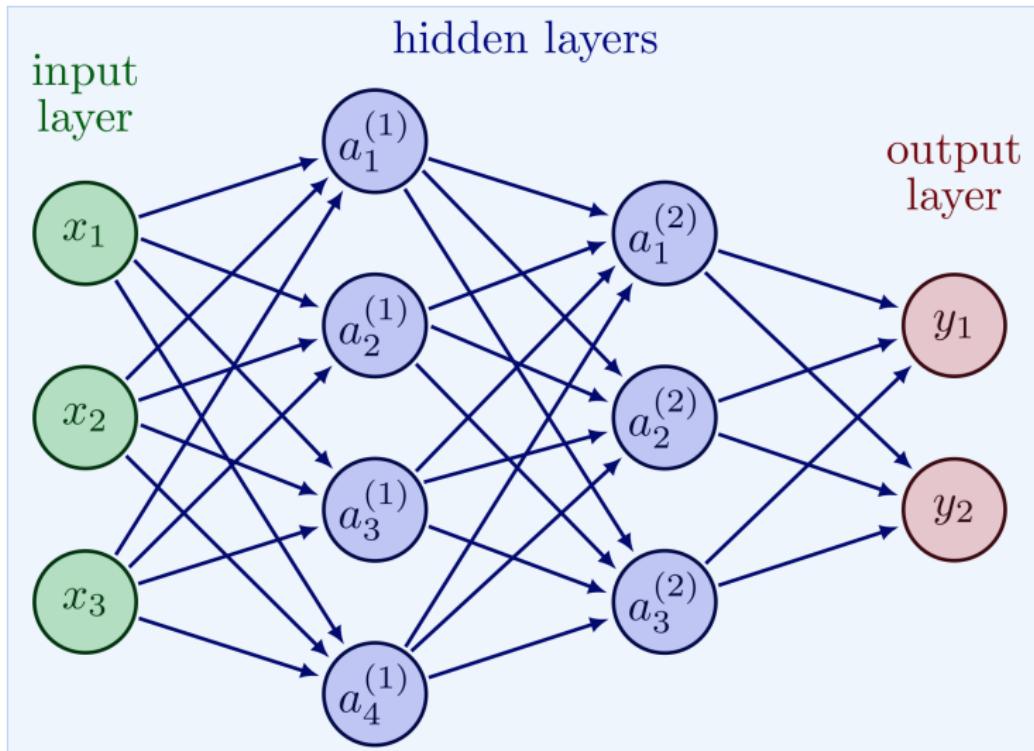
$$\mathbf{y} = f_{NN}(\mathbf{x}) = \mathbf{f}_L \circ \dots \circ \mathbf{f}_2 \circ \mathbf{f}_1(\mathbf{x}).$$

Donde...

- Cada función f_i se llama una **capa** (**entrada** → **oculta** → **salida**)
- Cada capa f_i está compuesta por un n° variable de **neuronas**
- Cada neurona depende un conjunto de **parámetros**, que determinarán a la RN

* La RN de la figura se dice de tipo «*feed forward*» o prealimentada

Un ejemplo



$$f_{NN} : \mathbb{R}^3 \rightarrow \mathbb{R}^2 \text{ con 2 capas ocultas de 4 y 3 neuronas}$$

Neurona o perceptrón simple

Cada neurona j de una capa oculta f_i (o de salida y_i) es una función¹:

$$x \in \mathbb{R}^{N_i} \rightarrow a_j^{(i)}(x) \in \mathbb{R},$$

composición de

- una función afín con parámetros $w = (w_1, \dots, w_{N_i})$ y b
- una función no lineal σ , llamada «función de activación»

$$\begin{aligned} a_j^{(i)}(x) &= \sigma(w_1x_1 + w_2x_2 + \cdots + w_{N_i}x_{N_i} + b) = \\ &= \sigma\left(\sum_{k=1}^{N_i} w_kx_k + b\right) = \sigma(w \cdot x + b) \end{aligned}$$

¹Donde N_i es el número de neuronas de la capa $i - 1$

Con más propiedad...

Para aligerar la notación se omitieron los índices correspondientes a la capa, i , y a la neurona, j . Debería ser:

$$a_j^{(i)}(x) = \sigma \left(\sum_{k=1}^{N_i} w_{j,k}^{(i)} x_k + b_j^{(i)} \right).$$

Así, si $W^{(i)}$ denota a la matriz de valores $w_{j,k}^{(i)}$, y $b^{(i)}$ es el vector $(b_j^{(i)})$, podemos escribir a tod la **capa i** como:

$$f_i(x) = \sigma_i(W^{(i)}x + b^{(i)})$$

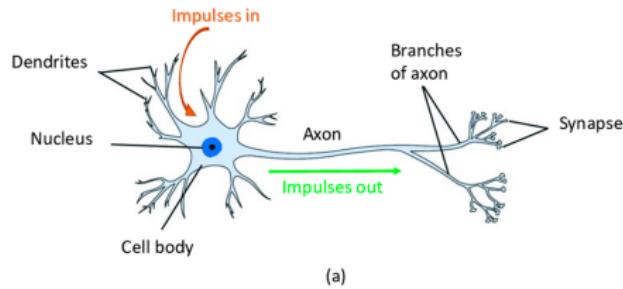
La RN está determinada por los parámetros $W^{(i)}$, los desplazamientos $b^{(i)}$ y las funciones de activación σ_i ;

Section 2

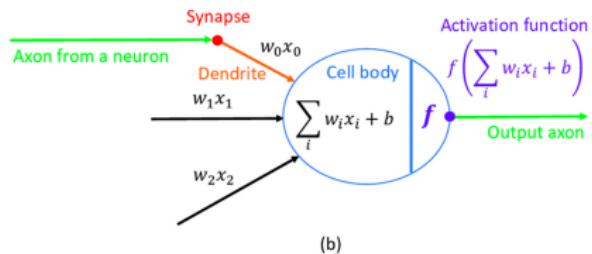
Partes básicas de una red neuronal

Funciones de activación

Es una abstracción que representa la **tasa potencial de acción**.



(a)



(b)

2

Dada una entrada se procesa (**parte lineal**) y se valora si la neurona dispara o no **perceptrón**. Para comportamientos más complejos podemos devolver una probabilidad si es baja menos probable es que dispare y viceversa, de ahí proviene la **función de activación sigmoidal**.

²Ranking to Learn and Learning to Rank: On the Role of Ranking in Pattern Recognition Applications

¿Por qué necesitamos funciones de activación?

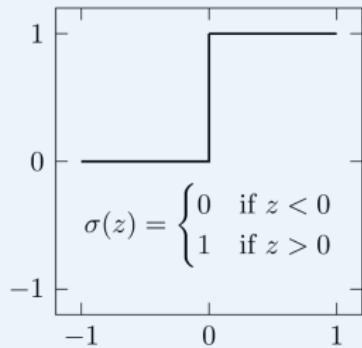
Supongamos que no tenemos funciones de activación, luego la red neural es composición de aplicaciones lineales, esto implica que **la red es una función lineal**.

$$f_{NN}(\textcolor{green}{x}) = \textcolor{blue}{w_1x_1 + w_2x_2 + \cdots + w_{N_i}x_{N_i} + b}$$

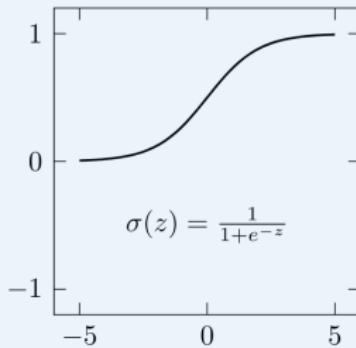
Por lo tanto, no importa cuantas capas tenga el modelo, puesto que estamos realizando una transformación lineal en las entradas. Esto es un problema, puesto que no podemos **modelar problemas no lineales**.

Representación de funciones de activación

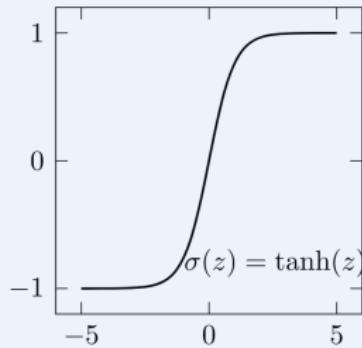
Perceptron



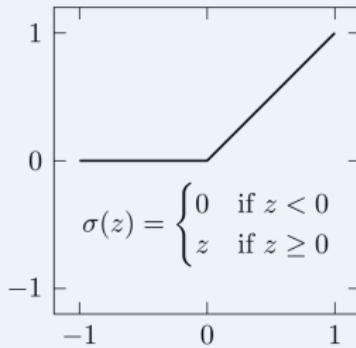
Sigmoid



Tanh



ReLU



Funciones de perdida

Una función de perdida compara la salida de la red neuronal con la salida esperada, y nos dice que tan **bien o mal lo esta haciendo la red neuronal**.

Cuando entrenamos, nuestro objetivo es **minimizar la perdida entre la salida esperada y la salida de la red neuronal**.

Ejemplos:

- Problema de regresión

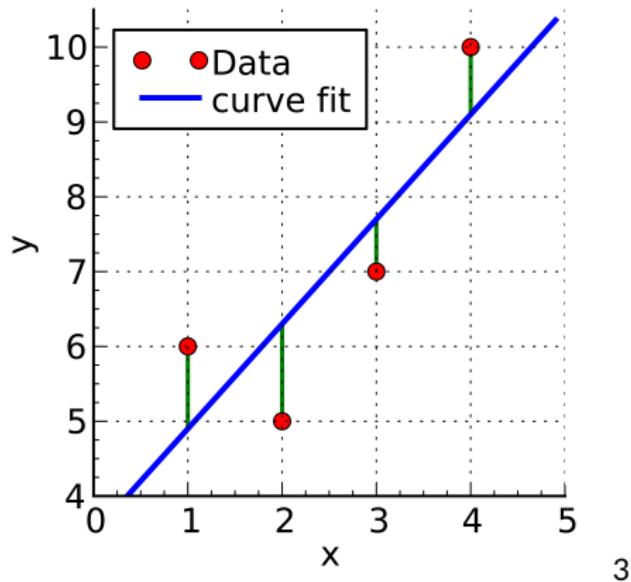
$$y_{pred} = \begin{pmatrix} 250 & 300 \\ 300 & 400 \end{pmatrix} \quad y_{real} = \begin{pmatrix} 100 & 150 \\ 400 & 200 \end{pmatrix}$$

- Problema de clasificación

$$y_{pred} = \begin{bmatrix} 0.12 \\ 0.48 \\ 0.4 \end{bmatrix} \quad y_{real} = \begin{bmatrix} 0 \\ 1 \\ 0 \end{bmatrix}$$

Podemos pensarla como un residuo en estadística, que se encarga de medir la distancia entre los **valores actuales de y** y los de la **línea de regresión** (valores predecidos).

El objetivo es minimizar la distancia neta.



3

³Wikipedia - Linear regression

Ejemplos de funciones de perdida

Principalmente se agrupan en:

- **Perdida de regresión:** Dado un valor de entrada, el modelo predice el valor de salida.

Ejemplos: Mean Squared Error, Mean Absolute Error...

$$MSE = \frac{1}{n} \sum_{i=1}^n (y^{(i)} - y_{pred}^{(i)})^2$$

$$MAE = \frac{1}{n} \sum_{i=1}^n |y^{(i)} - y_{pred}^{(i)}|$$

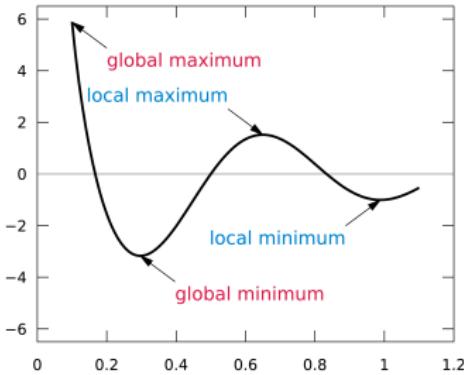
- **Perdida de clasificación:** Dado un valor de entrada, el modelo devuelve un vector de probabilidades de que la entrada pertenezca a una serie de categorías preestablecidas.

Ejemplos: Binary Cross Entropy, Categorical Cross Entropy...

Optimizadores

Son algoritmos que se usan para **minimizar la función de perdida** o maximizar la función de ganancia. Ejemplos: **Adam, RMSprop, SGD...**

Cada vez que predecimos un valor mediante la red y comparamos con el valor real, **el optimizador ajusta los pesos de la red para minimizar la perdida.**



Queremos obtener un mínimo global en la función de perdida.

Gradient Descent⁴

El algoritmo de descenso de gradiente es un **algoritmo de optimización que se utiliza para minimizar una función objetivo.**

Para implementarlo:

$$a_{n+1} = a_n - \delta \nabla F(a_n)$$

La idea es que el gradiente muestra la dirección de descenso entonces al movernos en esa dirección vamos a llegar a un mínimo, el problema es que puede no ser un mínimo global.

⁴Simulador de descenso de gradiente - Elaboración Propia

¿Cómo se calculan las derivadas?

Se emplea una técnica llamada **Backpropagation** que se encarga de calcular las derivadas de la función de perdida con respecto a los pesos de la red.

Ejemplo: $f(x, y) = \left(\frac{x}{y} + 5\right)^2 + x \quad \frac{\partial f}{x}(6, 2)$

Construimos el grafo de computación:

$w_0 = x$
$w_1 = y$
$w_2 = w_0/w_1 + 5$
$w_3 = w_2^2$
$w_4 = w_3 + w_0$

¿Cómo se calculan las derivadas?

Se emplea una técnica llamada **Backpropagation** que se encarga de calcular las derivadas de la función de perdida con respecto a los pesos de la red.

Ejemplo: $f(x, y) = \left(\frac{x}{y} + 5\right)^2 + x \quad \frac{\partial f}{\partial x}(6, 2)$

Hacemos un pasada evaluando cada nodo.

$w_0 = x$	$w_0 = 6$
$w_1 = y$	$w_1 = 2$
$w_2 = w_0 / w_1 + 5$	$w_2 = 8$
$w_3 = w_2^2$	$w_3 = 64$
$w_4 = w_3 + w_0$	$w_4 = 70$

¿Cómo se calculan las derivadas?

Se emplea una técnica llamada **Backpropagation** que se encarga de calcular las derivadas de la función de perdida con respecto a los pesos de la red.

Ejemplo: $f(x, y) = \left(\frac{x}{y} + 5\right)^2 + x$ $\frac{\partial f}{\partial x}(6, 2)$

Empleamos la regla de la cadena en reverso.

$w_0 = x$	$w_0 = 6$	$\frac{\partial z}{\partial w_0} = \frac{\partial z}{\partial w_2} \frac{\partial w_2}{\partial w_0} + \frac{\partial z}{\partial w_4} \frac{\partial w_4}{\partial w_0} = (2 \cdot w_2) \cdot \frac{1}{w_1} + 1$
$w_1 = y$	$w_1 = 2$	$\frac{\partial z}{\partial w_1} = \frac{\partial z}{\partial w_2} \frac{\partial w_2}{\partial w_1} = 2 \cdot w_2 \cdot \left(-\frac{w_0}{w_1^2}\right)$
$w_2 = w_0/w_1 + 5$	$w_2 = 8$	$\frac{\partial z}{\partial w_2} = \frac{\partial z}{\partial w_3} \frac{\partial w_3}{\partial w_2} = 2 \cdot w_2$
$w_3 = w_2^2$	$w_3 = 64$	$\frac{\partial z}{\partial w_3} = \frac{\partial z}{\partial w_4} \frac{\partial w_4}{\partial w_3} = 1$
$w_4 = w_3 + w_0$	$w_4 = 70$	$\frac{\partial z}{\partial w_4} = 1 \text{ (seed)}$

¿Cómo se calculan las derivadas?

Se emplea una técnica llamada **Backpropagation** que se encarga de calcular las derivadas de la función de perdida con respecto a los pesos de la red.

Ejemplo: $f(x, y) = \left(\frac{x}{y} + 5\right)^2 + x$ $\frac{\partial f}{x}(6, 2)$

Sustituyendo obtenemos $\frac{\partial f}{x}(6, 2) = 9$ $\frac{\partial f}{y}(6, 2) = -24$

$w_0 = x$	$w_0 = 6$	$\frac{\partial z}{\partial w_0} = 9$
$w_1 = y$	$w_1 = 2$	$\frac{\partial z}{\partial w_1} = -24$
$w_2 = w_0/w_1 + 5$	$w_2 = 8$	$\frac{\partial z}{\partial w_2} = 16$
$w_3 = w_2^2$	$w_3 = 64$	$\frac{\partial z}{\partial w_3} = 1$
$w_4 = w_3 + w_0$	$w_4 = 70$	$\frac{\partial z}{\partial w_4} = 1$ (seed)

Hiperparámetros

Un hiperparámetro es un **valor constante que se establece previo al entrenamiento**.

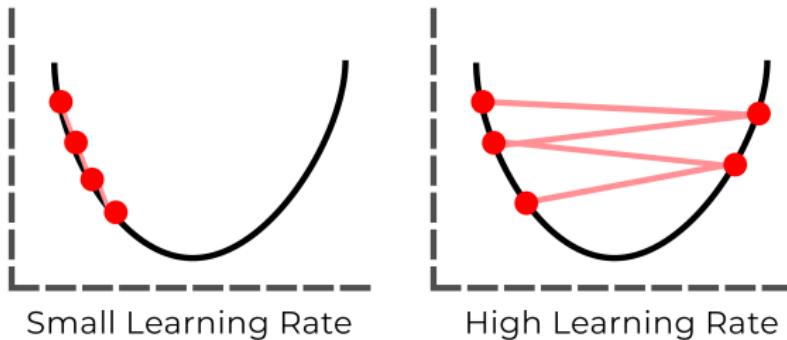
Ejemplos: **Tasa de aprendizaje, número de capas, funciones de activación, funciones de perdida, optimizadores...**

Si un modelo no se comporta bien, entonces tenemos que modificar los hiperparámetros, ya sea para conseguir que la red aprenda más lento (tasa de aprendizaje), hacerlo más complejo (añadir más capas)...

Tasa de aprendizaje

La tasa de aprendizaje define el **tamaño de los pasos correctivos para que el modelo ajuste los errores en cada observación.**

- Tasa de aprendizaje alta →
Tiempo de entrenamiento menor // Precisión menor
- Tasa de aprendizaje baja →
Tiempo de entrenamiento mayor // Precisión mayor

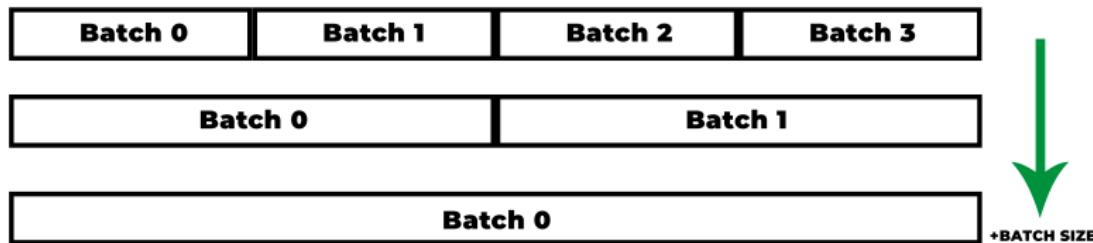


Batch Size

Número de ejemplos de entrenamiento que se proporcionan a la red antes de que el optimizador actualice los pesos.

Un buen tamaño de lote es generalmente **32**.

Otros válidos son: 32, 64, 128, 256...



Numeros de epochs

Es el **número de veces que se entrena la red** con el conjunto de datos de entrenamiento.

**BATCH SIZE =
2000**

**BATCH SIZE =
1000**

**BATCH SIZE =
500**

**ITERATIONS PER
EPOCH: 1**

**ITERATIONS PER
EPOCH: 2**

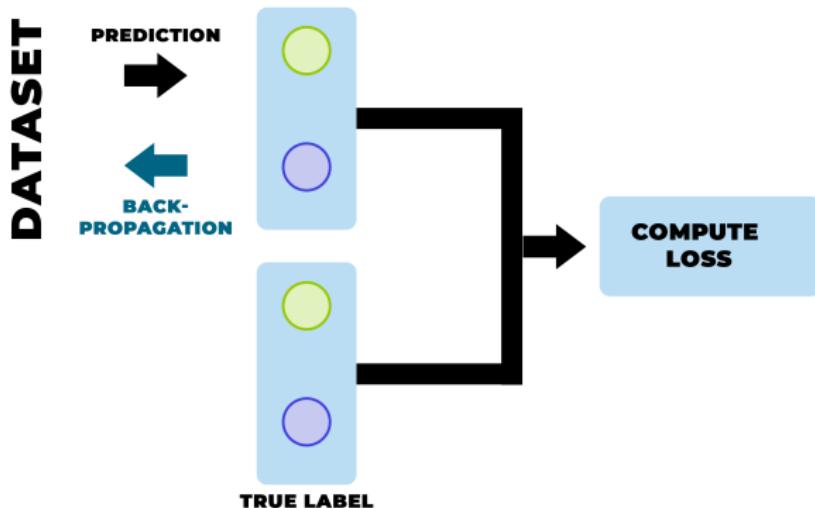
**ITERATIONS PER
EPOCH: 4**

En resumen

Mientras que $EPOCH < n_epoch$

Tomamos una porción de datos $\#BATCH_SIZE$ y la pasamos por la red, calculamos la perdida (**función de perdida**) y actualizamos los pesos (**optimizadores**).

Fin Mientras



Frameworks

Principalmente hay dos frameworks que se utilizan:

- **TensorFlow** es una biblioteca de código abierto para aprendizaje automático a través de un rango de tareas, y desarrollado por **Google** para satisfacer sus necesidades de sistemas capaces de construir y entrenar redes.
- **PyTorch** es una biblioteca de aprendizaje automático de código abierto basada en la biblioteca de Torch, utilizado para aplicaciones como visión artificial y procesamiento de lenguajes naturales, principalmente desarrollado por el Laboratorio de Investigación de Inteligencia Artificial de **Facebook** (FAIR).



Section 3

Ejemplos prácticos

Ejemplos prácticos

La idea de esta sección es ir mostrando una serie de ejemplos prácticos, para que se vea como se implementa una red neuronal en la práctica.

Para ello tenemos que responder:

- ¿Es una red neuronal adecuada para nuestro problema?
- ¿Cuál va a ser nuestros **datos de entrenamiento**?
- ¿Cuál es nuestra **arquitectura**?
- ¿Qué hiperparámetros escoger?

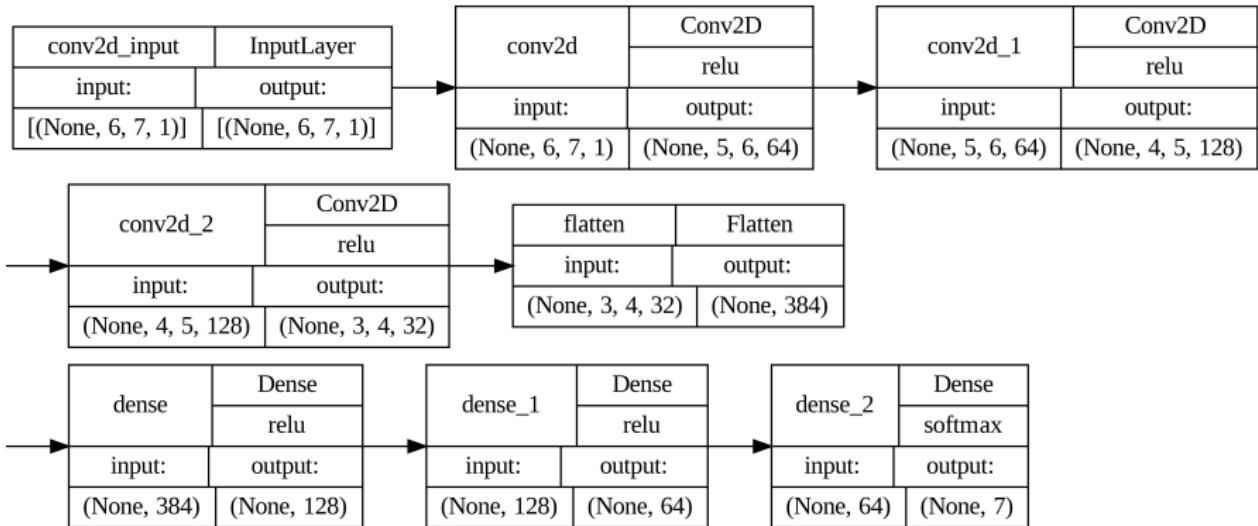
Datos de entrenamiento

Mediante el **algoritmo Minimax**, se genera una tabla de entrenamiento que recoge para una posición del tablero, la mejor jugada que se puede hacer.

Estado Tablero	Actua	Mejor acción
$\begin{bmatrix} 1 & 0 & 0 & 0 & 0 & 0 & 0 \\ 1 & 2 & 0 & 0 & 0 & 0 & 0 \\ 2 & 1 & 0 & 0 & 0 & 0 & 0 \\ 2 & 2 & 0 & 0 & 0 & 0 & 0 \\ 1 & 2 & 0 & 2 & 0 & 1 & 0 \\ 2 & 2 & 1 & 1 & 0 & 1 & 1 \end{bmatrix}$	2	4

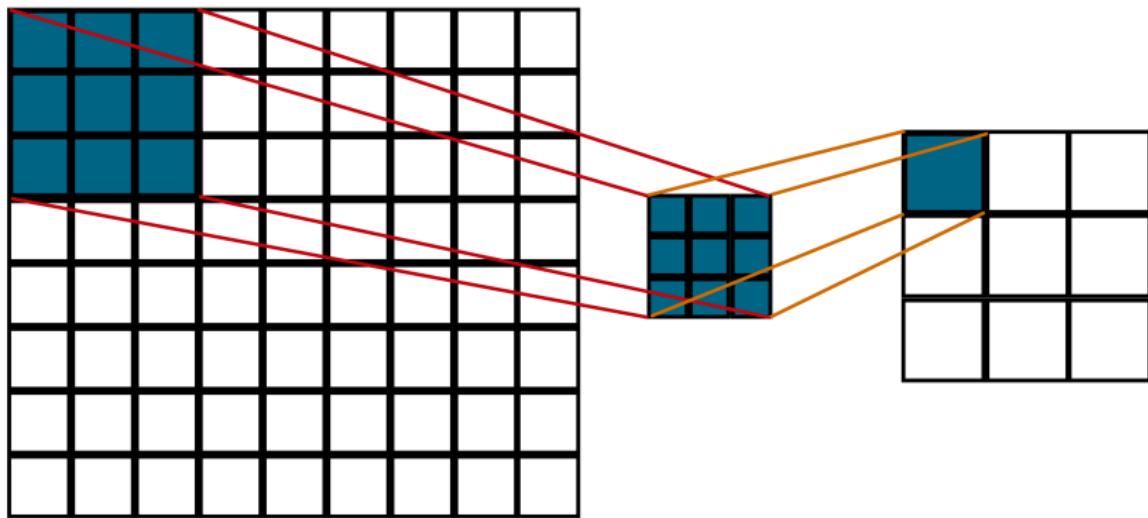
Arquitectura

Input (Imagen) → Convolución → Densa → Salida (7)



Convolución

Son muy importantes cuando tenemos que procesar imágenes, puesto que ayudan a **simplificar la imagen**, para luego pasarlo por una capa densa y obtener un buen resultado.



Hiperparámetros

$train_size = 90000$

$val_size = 5000$

$test_size = 5000$

$BATCH_SIZE = 64$

$EPOCHS = 512$

$LR = 0.001$

¿Función de perdida?

Clasificación o Regresión

$LOSS = 'sparse_categorical_crossentropy'$

Datos de entrenamiento

- Imágenes aleatorias de tamaño 256x256
- Texto aleatorio en formato ASCII
[30, 250, 240, 100, 30, 40, 60]

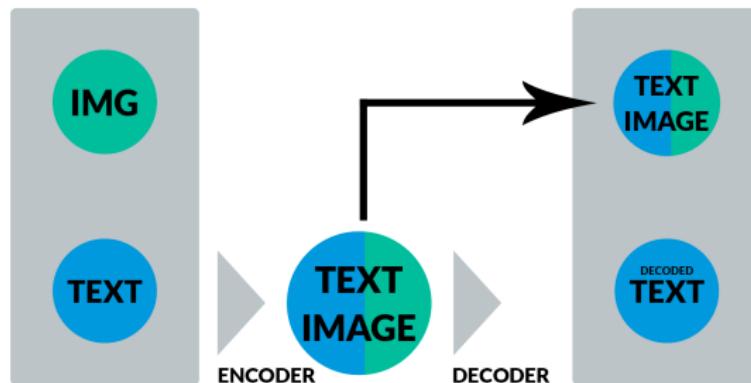


Arquitectura

AutoEncoder

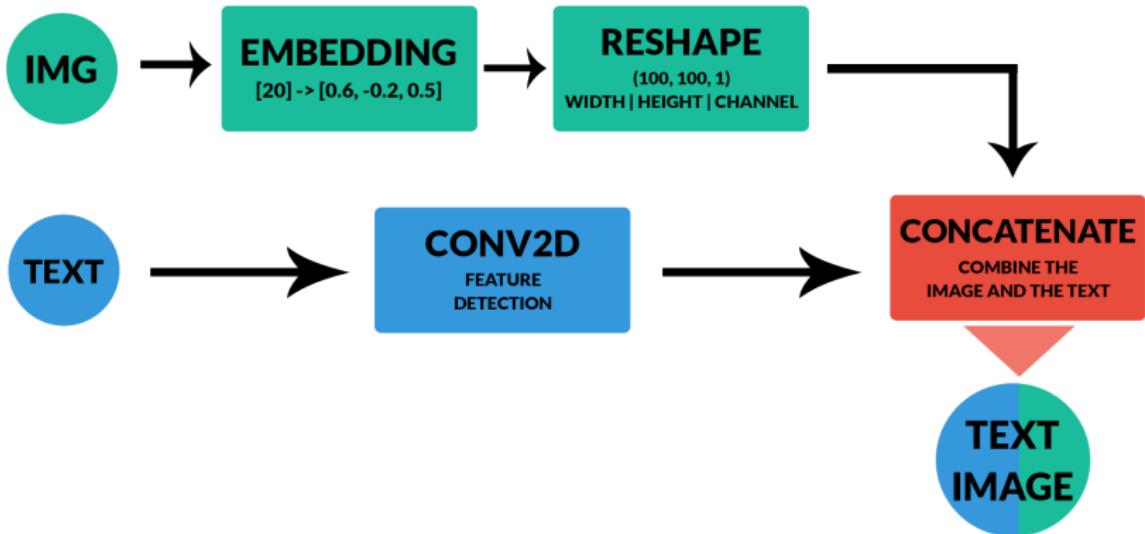
- **Encoder:** Incrusta el texto en la imagen devolviendo una imagen modificada.
- **Decoder:** Devuelve el texto incrustado en la imagen.

AUTOENCODER ARQUITECTURE



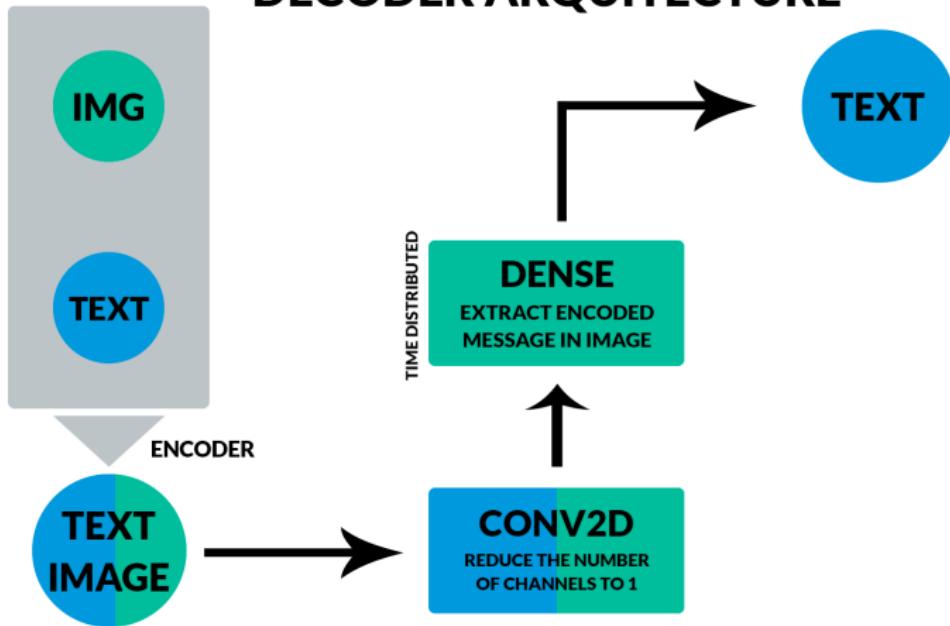
Encoder

ENCODER ARQUITECTURE



Decoder

DECODER ARQUITECTURE



Hiperparámetros

$epochs = 512$

$steps_per_epoch = 512$

$BATCH_SIZE = 64$

$LR = 0.001$

Funciones de perdida

- 1– MSE
- 2– Categorical Cross Entropy

Doodle

Datos de entrenamiento

■ Imagenes (Doodles)⁵



■ Categorias:

[aircraftcarrier, airplane, alarmclock, ambulance, angel...]

⁵<https://quickdraw.withgoogle.com/>

Vamos a resolver la siguiente PDE:

$$\begin{cases} \phi_t + u\phi_x = 0, & (x, t) \in (0, 1) \times (0, 1) \\ \phi(t, 0) = \phi(t, 1), & t \in (0, 1) \\ \phi(0, x) = \sin(2\pi x/L), & x \in (0, 1) \end{cases}$$

Donde $u = 1$ es la velocidad y $L = 1$, son valores escalares constantes.

Objetivo: $f_{NN} \approx \phi$

Datos de Entrenamiento

- $X_0 = [t_0, \{0, 1\}]$ Cantidad: 500

Proviene de: $\phi(t, 0) = \phi(t, 1)$, $t \in (0, 1)$

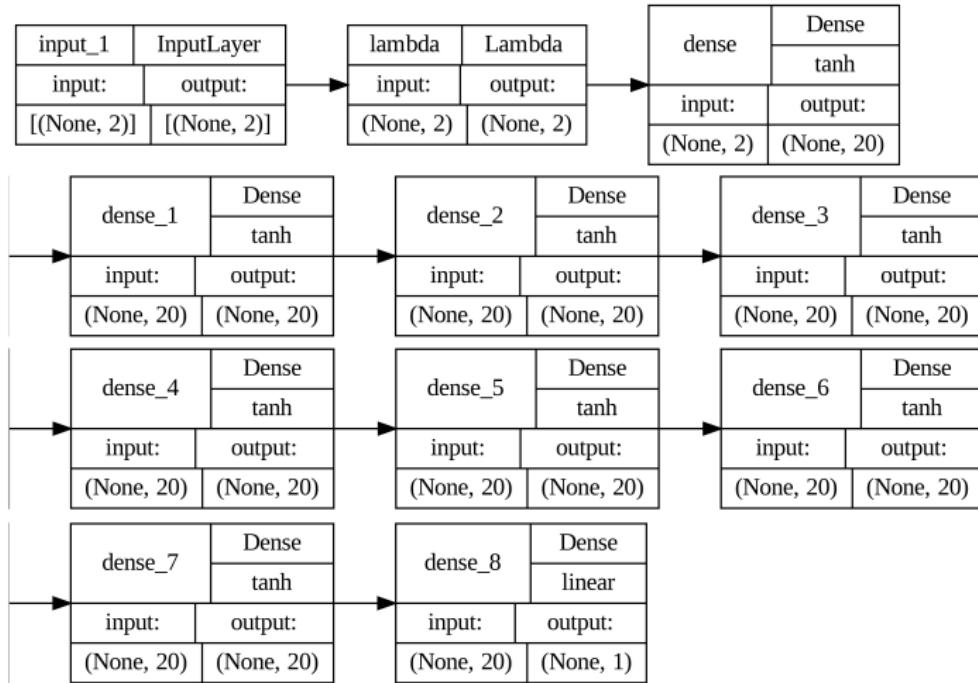
- $X_b = [0, x_b]$ Cantidad: 500

Proviene de: $\phi(0, x) = \sin(2\pi x/L)$, $x \in (0, 1)$

- $X_r = [t_r, x_r]$ Cantidad: 10000

Son puntos aleatorios en $(0, 1) \times (0, 1)$

Arquitectura



¿Cómo calculamos la pérdida?

■ Primera Pérdida

Pasamos $X_r = [t_r, x_r]$ por la red neuronal y obtenemos $f_{NN}(t_r, x_r)$

Aplicando diferenciación automática obtenemos $\frac{\partial f_{NN}}{\partial t}$ y $\frac{\partial f_{NN}}{\partial x}$

$$LOSS+ = \left(\frac{\partial f_{NN}}{\partial t} + u \frac{\partial f_{NN}}{\partial x} \right)^2$$

■ Segunda Pérdida

Pasamos $X_b = [t_b, \{0, 1\}]$ por la red neuronal y obtenemos $f_{NN}(t_b, x_b)$

$$LOSS+ = (f_{NN}(t_b, 0) - f_{NN}(t_b, 1))^2$$

■ Tercera Pérdida

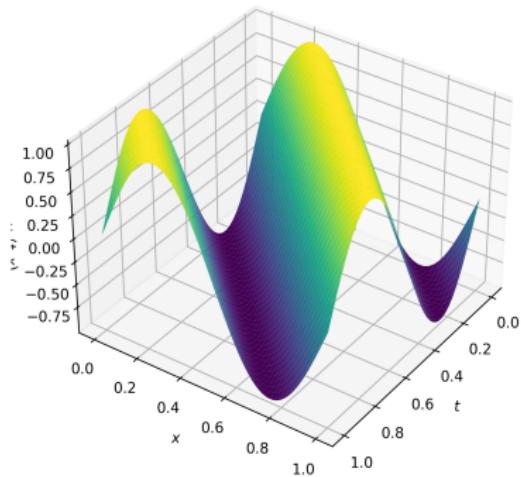
Pasamos $X_0 = [0, x_0]$ por la red neuronal y obtenemos $f_{NN}(t_0, x_0)$

$$LOSS+ = (\sin(2\pi x_0/L) - f_{NN}(t_0, x_0))^2$$

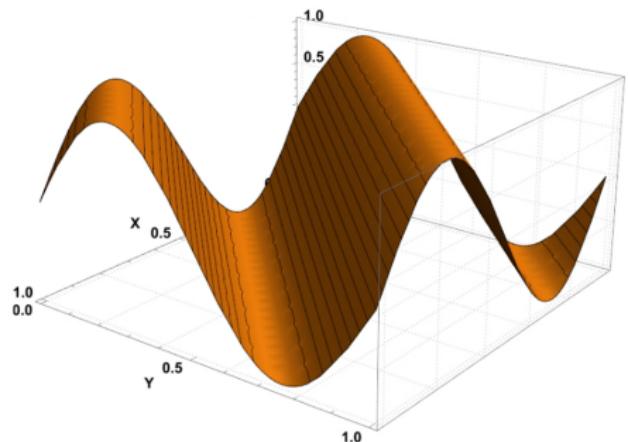
Con las perdidas anteriores lo que estamos imponiendo es cada una de las condiciones de la ecuación diferencial.

$$\begin{cases} \phi_t + u\phi_x = 0, (x, t) \in (0, 1) \times (0, 1) \\ \phi(t, 0) = \phi(t, 1), t \in (0, 1) \\ \phi(0, x) = \sin(2\pi x/L), x \in (0, 1) \end{cases}$$

Resultados



NEURAL NETWORK



MATHEMATICA

Section 4

Ajuste de los parámetros

Aprendizaje supervisado

- En redes supervisadas, se dispone de **datos de entrenamiento**, formados por un conjunto de valores de entrada \hat{x} , junto con los resultados asociados, \hat{y}
- Es usual disponer además de **datos de test**, x_{test} , y_{test}

Función de coste y entrenamiento de la red neuronal

- El proceso de **entrenamiento de la red neuronal** consiste en determinar los parámetros (pesos, $w_{j,k}^{(i)}$ y desplazamientos, $b_j^{(i)}$) que minimizan un funcional, "**función de coste**", sobre los datos de entrenamiento:

$$\Theta^* = \operatorname{argmin}\{J(\Theta; \hat{x}, \hat{y}), \quad \Theta = \left(w_{j,k}^{(i)}, b_j^{(i)}\right)\}$$

- La función de coste varía con cada tipo de red neuronal. Por ejemplo, en problemas de regresión se suelen usar mínimos cuadrados ("MSE: minimum mean square error"):

$$J(\Theta; \hat{x}, \hat{y}) = \frac{1}{N_{data}} \sum_{i=1}^{N_{data}} (\hat{y}_i - f_{NN}(\hat{x}_i))^2$$

Algoritmos de minimización

- Dificultades para la minimización: complejidad del funcional de coste, grandes valores de N_{data}
- Enormes requerimientos de cálculo para el entrenamiento, uso de grandes ordenadores, GPUs
- Se suelen utilizar algoritmos de tipo **descenso de gradiente**⁶

$$\Theta_{k+1} = \Theta_k - \ell_r \nabla_{\Theta} J(\Theta_k; \hat{x}, \hat{y}), \quad \ell_r: \text{"Learning Rate"}$$

- Necesidad de derivar de forma eficiente: **diferenciación automática**⁷
- Algoritmos de **gradiente estocástico**⁸: en cada paso, se calcula el gradiente pero sólo en un subconjunto aleatorio de datos

⁶https://en.wikipedia.org/wiki/Gradient_descent

⁷https://en.wikipedia.org/wiki/Automatic_differentiation

⁸https://en.wikipedia.org/wiki/Stochastic_gradient_descent