



**Rodolfo Ferro**

ferro@cimat.mx

<https://rodolfoferro.xyz>

Clubes de Ciencia  
México

*"Satélites y Neuronas: Explorando lo Invisible con AI"*

Julio, 2025

# Introducción a DL

*Día 3*

### Rodolfo Ferro (ferro@cimat.mx)

- › Sr. SWE (Data Engineer) @ Bionic México
- › Tutor de Ciencia de Datos en Código Facilito y en el Diplomado en Ciencia de Datos de la ENES UNAM León
- › Profesor de AI & Coordinador del TomorrowLab @ EdgeHub School of Innovation (Ags.)
- › **Formación:** BMATH @ UG, CSysEng @ UVEG, StatMethods Specialist @ CIMAT
- › **Experiencia:** ML Engineer @ Vindoo.ai (España), Sherpa Digital en IA @ Microsoft México, AI Research Assistant @ CIMAT & AI Research Intern @ Harvard.



ferro@cimat.mx



# Tabla de contenidos

**1** Repaso

**2** Introducción al aprendizaje profundo

## 1 Repaso

- Definición de dataset
- Consideraciones al crear un dataset
- ¿Cómo se divide un dataset?
- ¿Qué es la IA?

## 2 Introducción al aprendizaje profundo

- Introducción
- Contexto histórico
- Perceptrón
- Perceptrón multicapa
- Aprendizaje
- Regularización



# Section 1: **Repasso**

Un dataset  $D$  es un conjunto de observaciones estructuradas que se utilizan para entrenar, validar o evaluar modelos de aprendizaje automático.  
Matemáticamente, lo podemos representar como:

$$D = \{(x_i, y_i)\}_{i=1}^n$$

donde:

- ›  $x_i \in \mathbb{R}^d$  es el vector de características (o entrada) del ejemplo  $i$ -ésimo
- ›  $y_i$  es la etiqueta o salida correspondiente (puede ser una clase, un valor continuo, una máscara, etc.)
- ›  $n$  es el número total de ejemplos/muestras en el dataset

### Tipos de datasets:

- › **Datos etiquetados:**  $(x_i, y_i)$  – aprendizaje supervisado
- › **Datos no etiquetados:** sólo  $x_i$  (sin  $y_i$ ) – aprendizaje no supervisado
- › **Datos parcialmente etiquetados:** aprendizaje semi-supervisado.

# Consideraciones al crear un dataset

Creación de datasets para ML

## 1 Relevancia del problema:

- » ¿Qué queremos que el modelo aprenda?
- » Definir claramente la variable objetivo  $y$  y las variables predictoras  $x$

## 2 Calidad de los datos:

- » Datos limpios y representativos
- » Etiquetas en formato correcto
- » Evitar sesgos y duplicados

## 3 Tamaño del dataset:

- » Suficientes para capturar la variabilidad del problema
- » Pero no excesivo si se pueden extraer patrones antes (data augmentation o técnicas de balanceo)

## 4 Balance de clases (para clasificación):

- » Evitar clases desbalanceadas

# ¿Cómo se divide un dataset?

Creación de datasets para ML

$$D = D_{train} \cup D_{val} \cup D_{test}$$

- › **Entrenamiento (train):** Para ajustar los parámetros del modelo.
- › **Validación (validation):** Para ajustar hiperparámetros y prevenir sobreajuste.
- › **Prueba (test):** Para evaluar el rendimiento final del modelo.

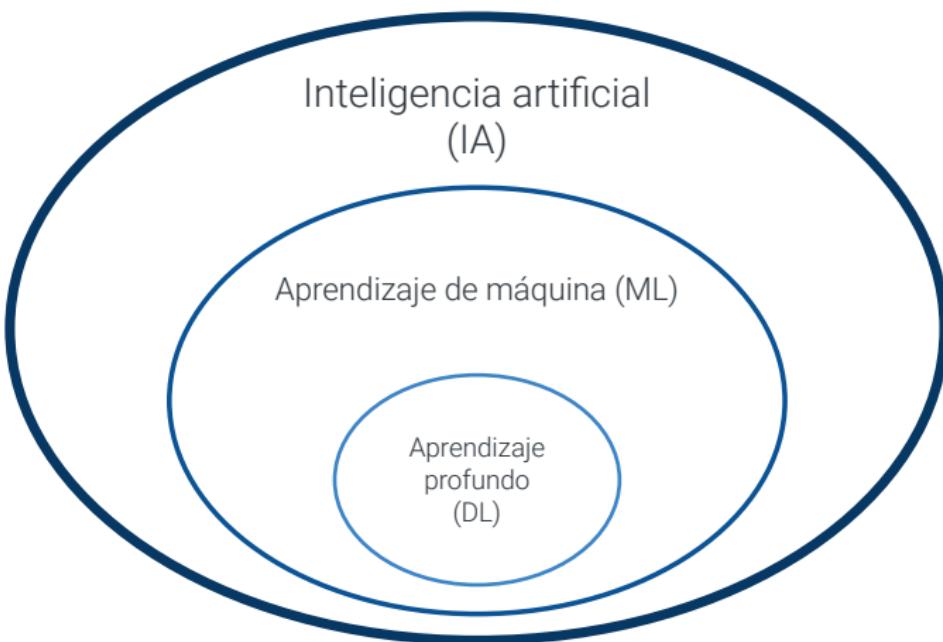
Ejemplo típico de partición:

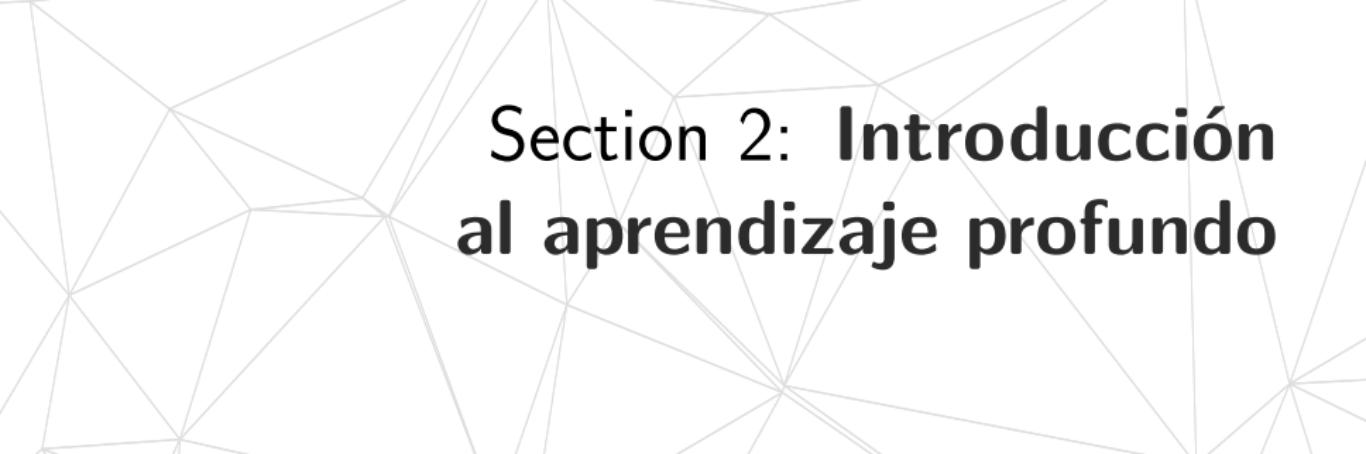
- › 70% **entrenamiento**
- › 15% **validación**
- › 15% **prueba**

- › **Inteligencia artificial:** Cualquier técnica que permita a las computadoras emular o imitar el comportamiento humano.
- › **Aprendizaje de máquina:** Capacidad de aprender sin ser programado explícitamente, enfoque en los algoritmos y la matemática.
- › **Aprendizaje profundo:** Extrae patrones de datos utilizando redes neuronales.

# ¿Qué es la IA?

## Intro al aprendizaje de máquina





## **Section 2: Introducción al aprendizaje profundo**

# ¿Qué es el *Deep Learning*?

Intro al aprendizaje profundo

*El aprendizaje profundo (Deep Learning) comprende algoritmos de Machine Learning que (particularmente) utilizan múltiples capas apiladas de unidades de procesamiento para aprender representaciones en un alto nivel sobre datos no estructurados.*

David Foster (Generative Deep Learning)



# ¿Por qué el *Deep Learning*?

Intro al aprendizaje profundo

- La ingeniería de características requiere mucho tiempo, es susceptible a errores y no es escalable consistentemente con datos complejos. Mejor busquemos aprender las características subyacentes directamente de los datos.
- Las redes neuronales artificiales existen desde hace décadas, pero su predominio actual reside principalmente en los siguientes tres aspectos:
  - 1 Hardware (GPUs, etc. + Paralelización)
  - 2 Software (Frameworks para trabajar con NNs)
  - 3 Grandes cantidades de datos
- El aprendizaje profundo está revolucionando muchos campos.

## Intro al aprendizaje profundo

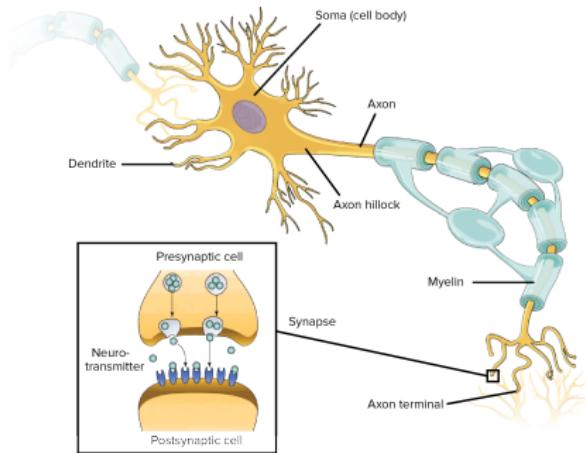
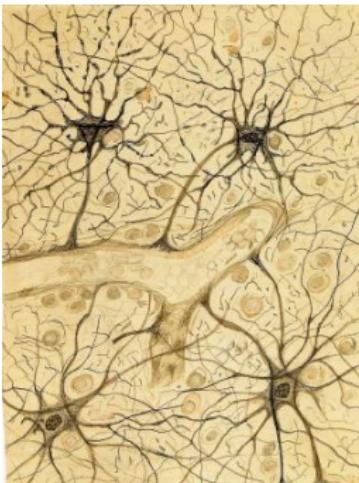


ferrero@cimat.mx

Figure: Santiago Ramón y Cajal

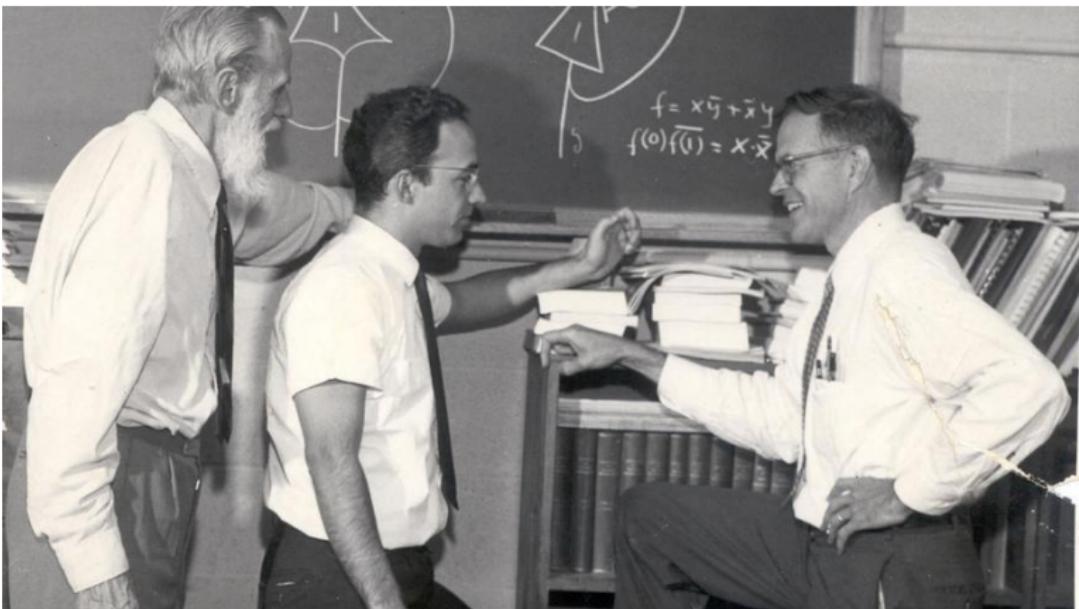
# Contexto histórico

## Intro al aprendizaje profundo



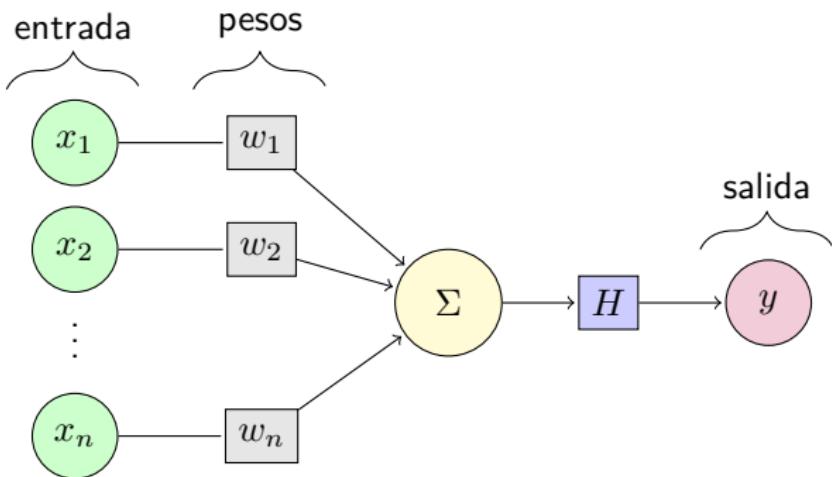
"Santiago Ramón y Cajal Drawings." Janelia Research Campus. Accessed July 5, 2024. <https://www.janelia.org>

"Overview of Neuron Structure and Function (Article)." Khan Academy. Accessed July 5, 2024. <https://www.khanacademy.org>



ferro@cimat.mx

Figure: Warren McCulloch & Walter Pitts



$$\Sigma \rightarrow \sum_{i=1}^n w_i x_i$$

# Bias y función de activación

Intro al aprendizaje profundo

La operación matemática que realiza la neurona para la decisión de umbralización se puede escribir como:

$$f(\mathbf{x}) = \begin{cases} 0 & \text{si } \sum_i w_i x_i < \text{umbral o threshold} \\ 1 & \text{si } \sum_i w_i x_i \geq \text{umbral o threshold} \end{cases}$$

donde  $i \in \{1, 2, \dots, n\}$ , y así,  $\mathbf{x} = (x_1, x_2, \dots, x_n)$ .

# Bias y función de activación

Intro al aprendizaje profundo

De lo anterior, podemos despejar el umbral y escribirlo como  $b$ , obteniendo:

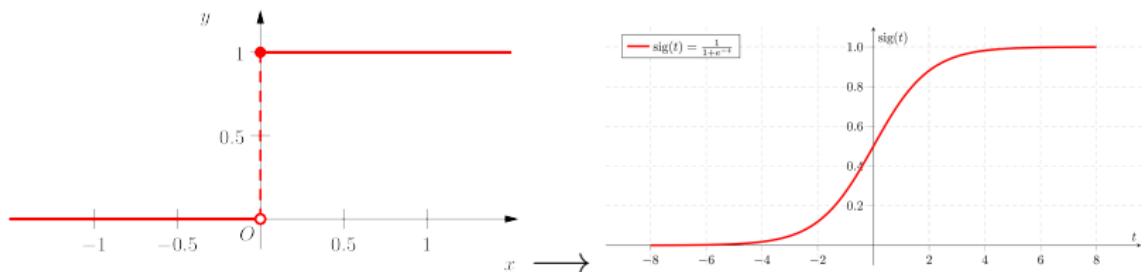
$$f(\mathbf{x}) = \begin{cases} 0 & \text{si } \sum_i w_i x_i + b < 0 \\ 1 & \text{si } \sum_i w_i x_i + b > 0 \end{cases}$$

donde  $\mathbf{x} = (x_1, x_2, \dots, x_n)$  y  $i \in \{1, 2, \dots, n\}$ .

A esto que escribimos como  $b$ , también se le conoce como **bias**, y una interpretación es describir *qué tan susceptible es la neurona a dispararse* (como se exploró en el ejemplo práctico de la identificación de la actividad de Julieta).

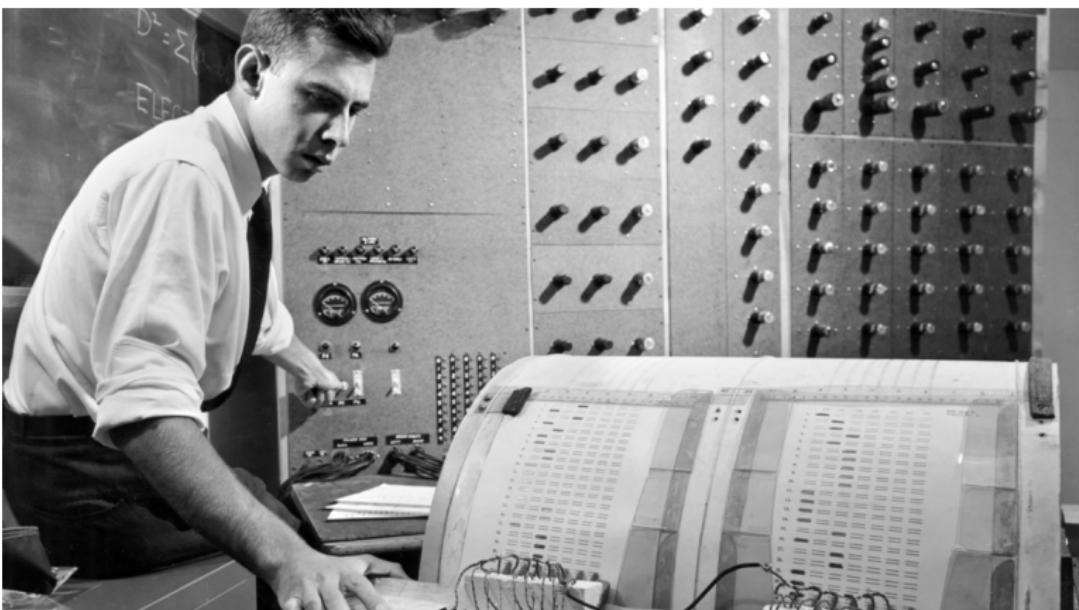
# Bias y función de activación

## Intro al aprendizaje profundo



# El Perceptrón

## Intro al aprendizaje profundo

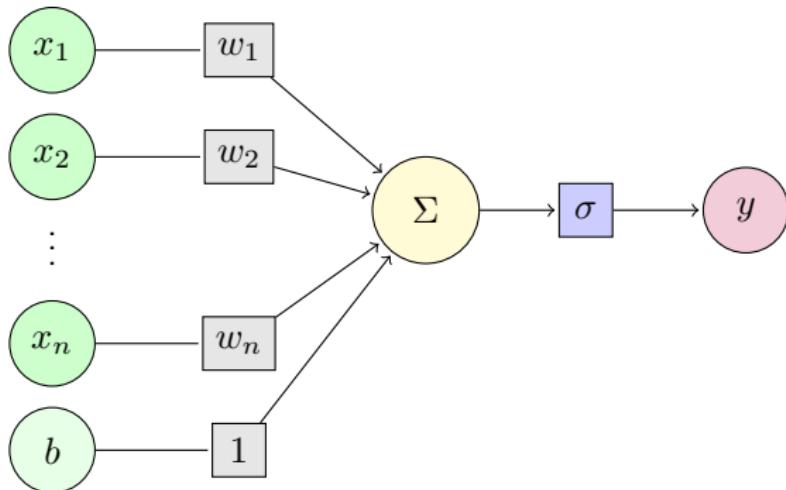


ferro@cimat.mx

Figure: Frank Rosenblatt



Clubes de  
Ciencia  
Mexico



¡Y se agrega un algoritmo formal de entrenamiento!  
(*Backpropagation*)

# Idea intuitiva de entrenamiento

Intro al aprendizaje profundo

Arroja una  
predicción

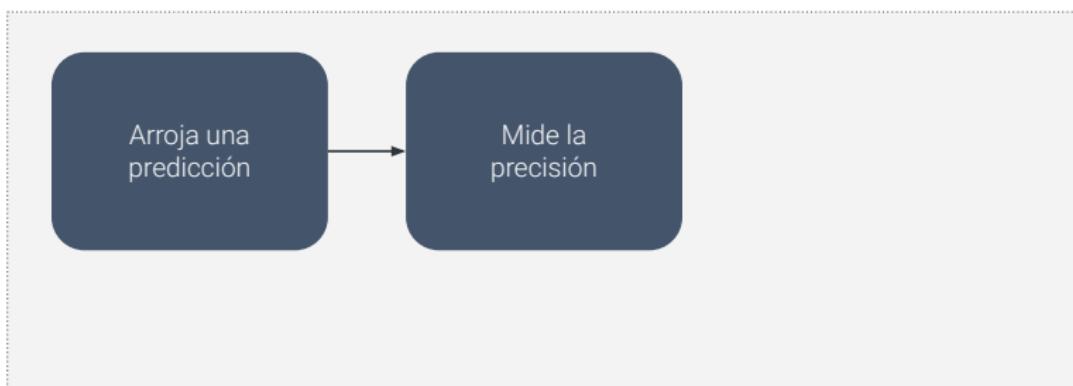
ferro@cimat.mx



Clubes de Ciencia  
Mexico

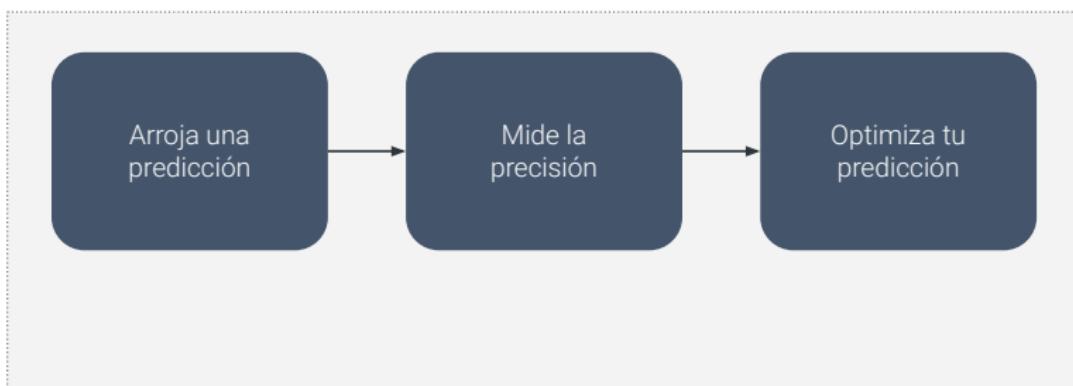
# Idea intuitiva de entrenamiento

Intro al aprendizaje profundo



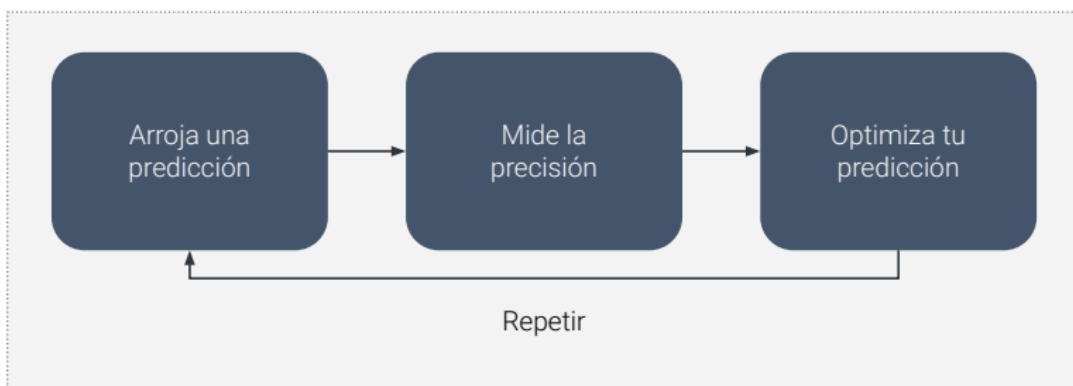
# Idea intuitiva de entrenamiento

## Intro al aprendizaje profundo



# Idea intuitiva de entrenamiento

## Intro al aprendizaje profundo



Dado el vector  $X$ , **¿qué vector  $(A, B, C)$  se le parece más?**

$$X = \begin{bmatrix} 0.5 \\ 0.3 \\ 0.7 \end{bmatrix}$$

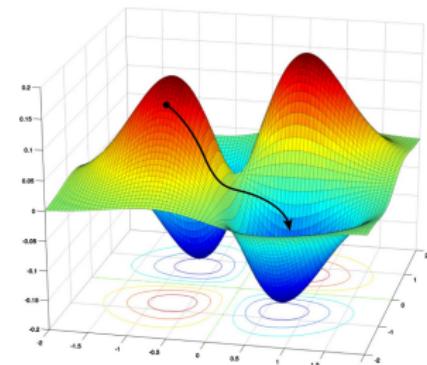
$$A = \begin{bmatrix} 0.3 \\ 0.3 \\ 0.3 \end{bmatrix}, B = \begin{bmatrix} 0.6 \\ 0.2 \\ 0.6 \end{bmatrix}, C = \begin{bmatrix} -0.5 \\ -0.3 \\ -0.7 \end{bmatrix}$$

Dado el vector  $X$ , **¿qué vector  $(A, B, C)$  se le parece más?**

$$X = \begin{bmatrix} 0.5 \\ 0.3 \\ 0.7 \end{bmatrix}$$

$$A = \begin{bmatrix} 0.3 \\ 0.3 \\ 0.3 \end{bmatrix}, B = \begin{bmatrix} 0.6 \\ 0.2 \\ 0.6 \end{bmatrix}, C = \begin{bmatrix} -0.5 \\ -0.3 \\ -0.7 \end{bmatrix}$$

- › **Error:** Es una función.
- › **Optimizar:** Maximizar o minimizar.
- › **Gradiente:** Derivada de una función vectorial, proporciona información sobre máximos o mínimos.
- › **Descenso de gradiente:** Algoritmo para, iterativamente, buscar optimizar una función.
- › **Limitantes:**
  - » Max's/min's locales.
  - » Tamaño de salto en gradiente



ferro@cmat.mx

Hasta este punto, debemos notar que hay algunas observaciones importantes:

- › **TLUs:**
  - » No existe un algoritmo de aprendizaje formal → Búsqueda de pesos.
  - » Se limita a propagación hacia adelante (*forward pass/forward propagation*)
- › **Perceptrón:** Puede utilizar retropropagación, introducido en 1958.
- › **Retropropagación:** Algoritmo para realizar ajustes en los valores de los pesos.
- › **Limitantes:** Separabilidad lineal.
- › **¿Alguna otra observación?**



### Ejercicio: Índice de aprobación de estudiantes



- › Breve historia sobre el perceptrón
- › Post sobre el perceptrón de Rosenblatt
- › Post sobre la función de activación
- › Selección de threshold para clasificadores binarios
- › Post sobre redes neuronales por IBM

Recordemos cómo opera el producto matricial:

$$\begin{bmatrix} 2 & 5 & 2 \\ 1 & 0 & -2 \\ 3 & 1 & 1 \end{bmatrix} \begin{bmatrix} -2 & 1 & 0 \\ -2 & 2 & 1 \\ 0 & 0 & 3 \end{bmatrix} = \begin{bmatrix} \quad & \quad & \quad \\ \quad & \quad & \quad \\ \quad & \quad & \quad \end{bmatrix}$$

Recordemos cómo opera el producto matricial:

$$\begin{bmatrix} 2 & 5 & 2 \\ 1 & 0 & -2 \\ 3 & 1 & 1 \end{bmatrix} \begin{bmatrix} -2 & 1 & 0 \\ -2 & 2 & 1 \\ 0 & 0 & 3 \end{bmatrix} = \begin{bmatrix} 14 & & \\ & & \end{bmatrix}$$

$$(2 \cdot -2) + (5 \cdot -2) + (2 \cdot 0) = -14$$

Recordemos cómo opera el producto matricial:

$$\begin{bmatrix} 2 & 5 & 2 \\ 1 & 0 & -2 \\ 3 & 1 & 1 \end{bmatrix} \begin{bmatrix} -2 & 1 & 0 \\ -2 & 2 & 1 \\ 0 & 0 & 3 \end{bmatrix} = \begin{bmatrix} 14 & 12 & \end{bmatrix}$$

$$(2 \cdot 1) + (5 \cdot 2) + (2 \cdot 0) = 12$$

Recordemos cómo opera el producto matricial:

$$\begin{bmatrix} 2 & 5 & 2 \\ 1 & 0 & -2 \\ 3 & 1 & 1 \end{bmatrix} \begin{bmatrix} -2 & 1 & 0 \\ -2 & 2 & 1 \\ 0 & 0 & 3 \end{bmatrix} = \begin{bmatrix} 14 & 12 & 11 \end{bmatrix}$$

$$(2 \cdot 0) + (5 \cdot 1) + (2 \cdot 3) = 11$$

Recordemos cómo opera el producto matricial:

$$\begin{bmatrix} 2 & 5 & 2 \\ 1 & 0 & -2 \\ 3 & 1 & 1 \end{bmatrix} \begin{bmatrix} -2 & 1 & 0 \\ -2 & 2 & 1 \\ 0 & 0 & 3 \end{bmatrix} = \begin{bmatrix} 14 & 12 & 11 \\ -2 & & \end{bmatrix}$$

$$(1 \cdot -2) + (0 \cdot -2) + (-2 \cdot 0) = -2$$

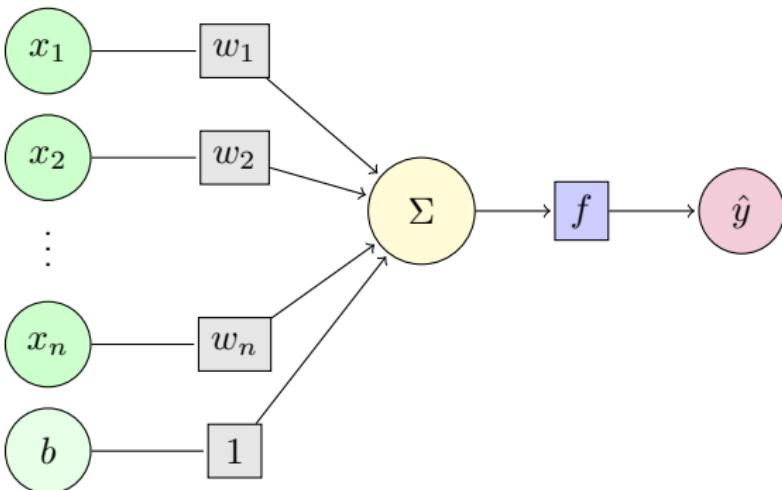
Recordemos cómo opera el producto matricial:

$$\begin{bmatrix} 2 & 5 & 2 \\ 1 & 0 & -2 \\ 3 & 1 & 1 \end{bmatrix} \begin{bmatrix} -2 & 1 & 0 \\ -2 & 2 & 1 \\ 0 & 0 & 3 \end{bmatrix} = \begin{bmatrix} 14 & 12 & 11 \\ -2 & & \\ & & \end{bmatrix}$$

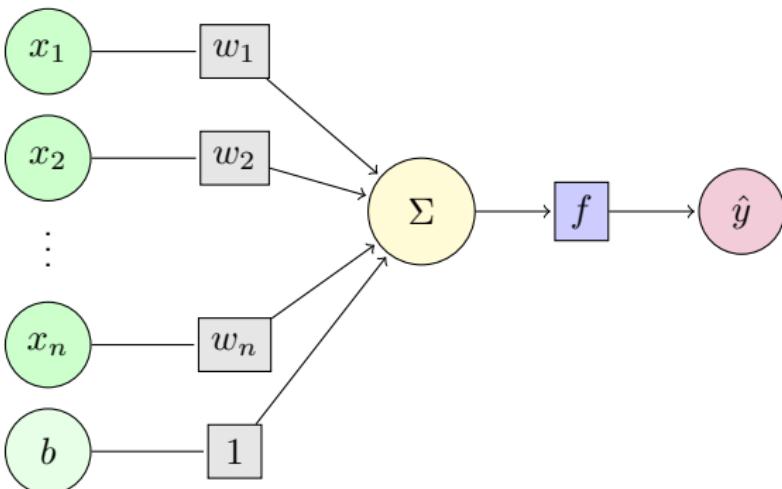
...

Recordemos cómo opera el producto matricial:

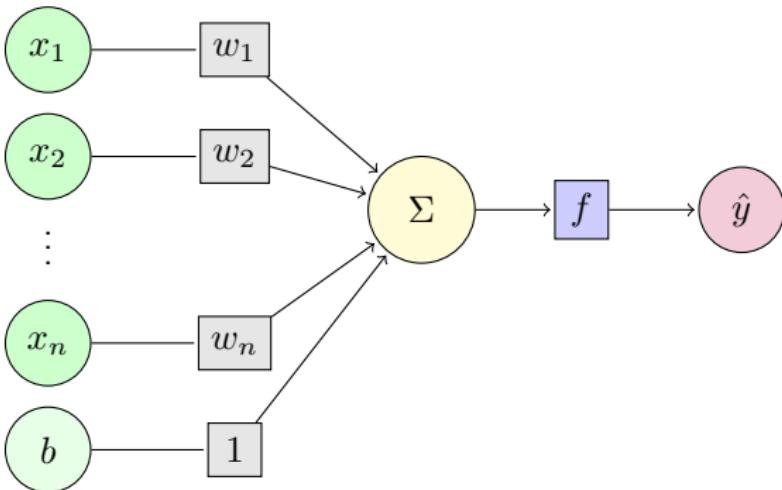
$$\begin{bmatrix} 2 & 5 & 2 \\ 1 & 0 & -2 \\ 3 & 1 & 1 \end{bmatrix} \begin{bmatrix} -2 & 1 & 0 \\ -2 & 2 & 1 \\ 0 & 0 & 3 \end{bmatrix} = \begin{bmatrix} 14 & 12 & 11 \\ -2 & 1 & -6 \\ -8 & 5 & 4 \end{bmatrix}$$



$$\hat{y} = f \left( \sum_i^n w_i x_i + b \right)$$



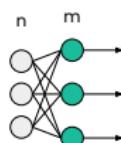
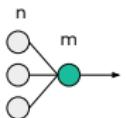
$$\sum_i^n w_i x_i = w_1 x_1 + w_2 x_2 + \cdots + w_n x_n$$



$$\mathbf{w}^T \mathbf{x} = [w_1 w_2 \cdots w_n] \begin{bmatrix} x_1 \\ x_2 \\ \vdots \\ x_n \end{bmatrix} = w_1 x_1 + w_2 x_2 + \cdots + w_n x_n$$

# Producto matricial

## Intro al aprendizaje profundo

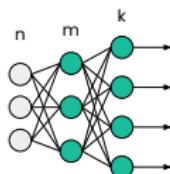


$$\mathbf{W} \cdot \mathbf{X} + \mathbf{b} = [w_1 \ w_2 \ \dots \ w_n] \cdot \begin{bmatrix} x_1 \\ x_2 \\ \vdots \\ x_n \end{bmatrix} + b \longrightarrow \mathbf{W}_{Layer} \cdot \mathbf{X} + \mathbf{b} = \begin{bmatrix} w_{1,1} & w_{1,2} & \dots & w_{1,n} \\ w_{2,1} & w_{2,2} & \dots & w_{2,n} \\ \vdots & \vdots & \ddots & \vdots \\ w_{m,1} & w_{m,2} & \dots & w_{m,n} \end{bmatrix} \cdot \begin{bmatrix} x_1 \\ x_2 \\ \vdots \\ x_n \end{bmatrix} + \begin{bmatrix} b_1 \\ b_2 \\ \vdots \\ b_m \end{bmatrix}$$

An arrow points from the term  $\mathbf{W} \cdot \mathbf{X}$  in the first equation to the term  $\mathbf{W}_{Layer} \cdot \mathbf{X}$  in the second equation. A dashed box contains the function  $f(\cdot)$ , with an arrow pointing from it to the addition operation between the two equations.

# Producto matricial

## Intro al aprendizaje profundo



$$\mathbf{W}_{Layer_k} \cdot (\mathbf{W}_{Layer_n} \cdot \mathbf{X} + \mathbf{b}_m) + \mathbf{b}_k = \begin{bmatrix} w_{1,1} & w_{1,2} & \dots & w_{1,m} \\ w_{2,1} & w_{2,2} & \dots & w_{2,m} \\ \vdots & \vdots & \ddots & \vdots \\ w_{k,1} & w_{k,2} & \dots & w_{k,m} \end{bmatrix} \left( \begin{bmatrix} w_{1,1} & w_{1,2} & \dots & w_{1,n} \\ w_{2,1} & w_{2,2} & \dots & w_{2,n} \\ \vdots & \vdots & \ddots & \vdots \\ w_{m,1} & w_{m,2} & \dots & w_{m,n} \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \\ \vdots \\ x_n \end{bmatrix} + \begin{bmatrix} b_1 \\ b_2 \\ \vdots \\ b_m \end{bmatrix} \right) + \begin{bmatrix} b_1 \\ b_2 \\ \vdots \\ b_k \end{bmatrix}$$

$f(\cdot)$

# Composición de funciones

Intro al aprendizaje profundo

$$\mathbf{W}_{Layer_k} \cdot (\mathbf{W}_{Layer_m} \cdot \mathbf{X} + \mathbf{b}_m) + \mathbf{b}_k = \begin{bmatrix} w_{1,1} & w_{1,2} & \dots & w_{1,m} \\ w_{2,1} & w_{2,2} & \dots & w_{2,m} \\ \vdots & \vdots & \ddots & \vdots \\ w_{k,1} & w_{k,2} & \dots & w_{k,m} \end{bmatrix} \left( \begin{bmatrix} w_{1,1} & w_{1,2} & \dots & w_{1,n} \\ w_{2,1} & w_{2,2} & \dots & w_{2,n} \\ \vdots & \vdots & \ddots & \vdots \\ w_{m,1} & w_{m,2} & \dots & w_{m,n} \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \\ \vdots \\ x_n \end{bmatrix} + \begin{bmatrix} b_1 \\ b_2 \\ \vdots \\ b_m \end{bmatrix} \right) + \begin{bmatrix} b_1 \\ b_2 \\ \vdots \\ b_k \end{bmatrix}$$

↓

$$f_k(\mathbf{W}_{Layer_k} \cdot f_m(\mathbf{W}_{Layer_m} \cdot \mathbf{X} + \mathbf{b}_m) + \mathbf{b}_k)$$

↓

$$f_n(\dots (f_2(\mathbf{W}_{Layer_k} \cdot f_1(\mathbf{W}_{Layer_1} \cdot \mathbf{X} + \mathbf{b}_1) + \mathbf{b}_2)) + \mathbf{b}_n)$$

# Composición de funciones

Intro al aprendizaje profundo

$$f_n(\dots(f_2(\mathbf{W}_{Layer_k} \cdot f_1(\mathbf{W}_{Layer_1} \cdot \mathbf{X} + \mathbf{b}_1) + \mathbf{b}_2)) + \mathbf{b}_n)$$



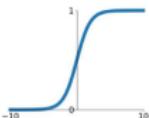
$$F'(x) = f'(g(x)) \cdot g'(x)$$

# Funciones de activación

Intro al aprendizaje profundo

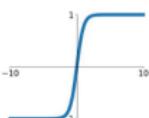
## Sigmoid

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



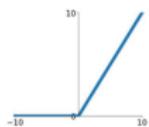
## tanh

$$\tanh(x)$$



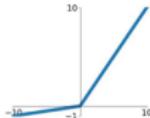
## ReLU

$$\max(0, x)$$



## Leaky ReLU

$$\max(0.1x, x)$$

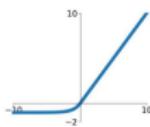


## Maxout

$$\max(w_1^T x + b_1, w_2^T x + b_2)$$

## ELU

$$\begin{cases} x & x \geq 0 \\ \alpha(e^x - 1) & x < 0 \end{cases}$$



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

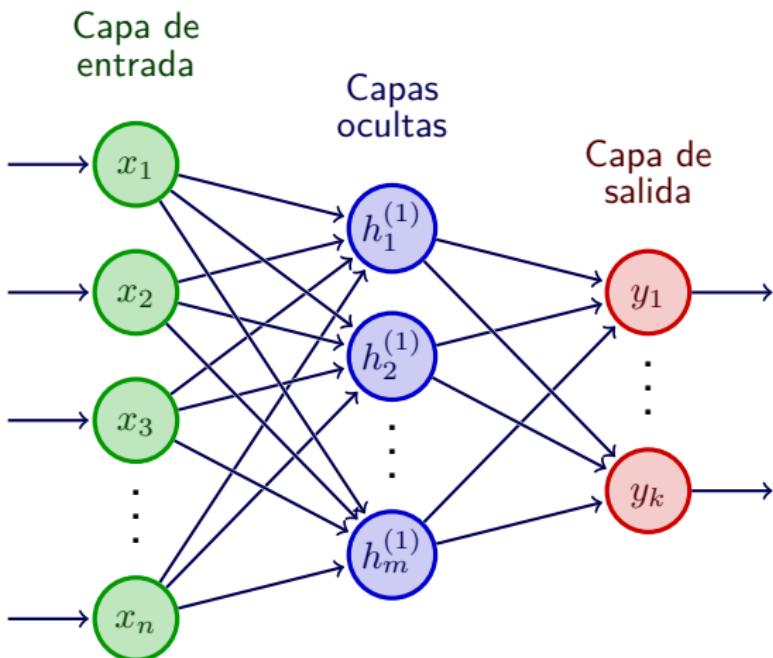
$$f(x) = \tanh(x) = \frac{e^x - e^{-x}}{e^x + e^{-x}} \Rightarrow f'(x) = 1 - f(x)^2$$



Clubes de  
Ciencia  
Mexico

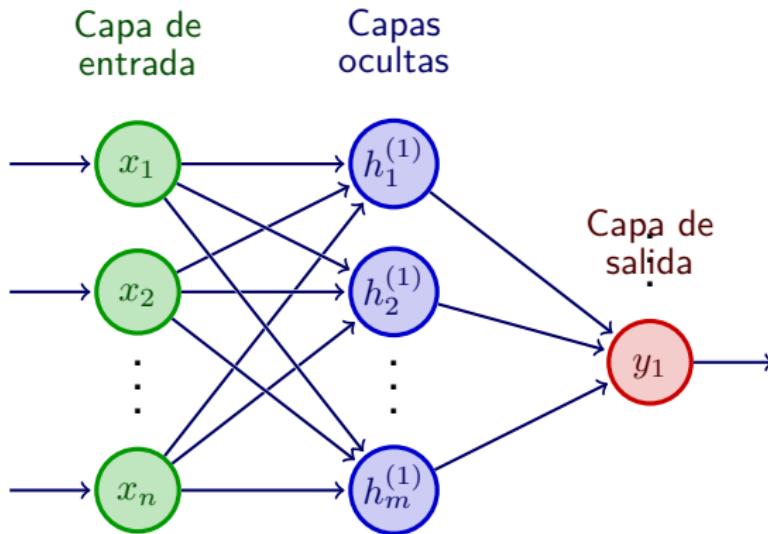
# El perceptrón multicapa

Intro al aprendizaje profundo



# El perceptrón multicapa

Intro al aprendizaje profundo



Para  $\mathbf{x}^{(i)} = (x_1^{(i)}, x_2^{(i)}, \dots, x_n^{(i)}), y^{(i)}$ , tendríamos que la salida es  
 $\hat{y}_1^{(i)} = f(x^{(i)})$ .

- › La **función de pérdida (loss)** de nuestra red neuronal *mide el costo* asociado a predicciones incorrectas.
- › Si observaciones (de entrada y salida)  $(x^{(i)}, y^{(i)})$  y consideramos a la salida como función de  $x^{(i)}$  y  $\mathbf{W}$ , entonces las salidas son  $\hat{y} = f(x^{(i)}; \mathbf{W})$  y la función de pérdida puede escribirse como:

$$\mathcal{L}(f(x^{(i)}; \mathbf{W}), y^{(i)})$$

Es decir, una función que mida la salida *real* con la *predicción*.  
Todo esto para una observación  $i$ .

- » Para todas las observaciones:

$$\mathbf{J}(\mathbf{W}) = \frac{1}{n} \sum_{i=1}^n \mathcal{L}(f(x^{(i)}; \mathbf{W}), y^{(i)})$$

A esta función se le conoce como función de costo o función objetivo (lo que queremos minimizar).

- › **Binary Cross Entropy Loss:** Se puede utilizar con modelos que devuelven como salida una probabilidad entre 0 y 1.

$$\mathbf{J}(\mathbf{W}) = \frac{1}{n} \sum_{i=1}^n y^{(i)} \log(f(x^{(i)}; \mathbf{W})) + (1 - y^{(i)}) \log(1 - f(x^{(i)}; \mathbf{W}))$$

- › **Mean Squared Error (MSE) Loss:** Se puede utilizar con modelos de regresión que generan números reales continuos.

$$\mathbf{J}(\mathbf{W}) = \frac{1}{n} \sum_{i=1}^n \left( y^{(i)} - f(x^{(i)}; \mathbf{W}) \right)^2$$

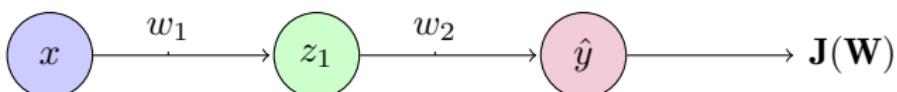
- › Queremos encontrar los pesos ideales de la red neuronal, los cuales minimizan  $\mathbf{J}(\mathbf{W})$ , es decir:

$$\mathbf{W}^* = \operatorname{argmin}_{\mathbf{W}} \frac{1}{n} \sum_{i=1}^n \mathcal{L}(f(x^{(i)}; \mathbf{W}), y^{(i)})$$

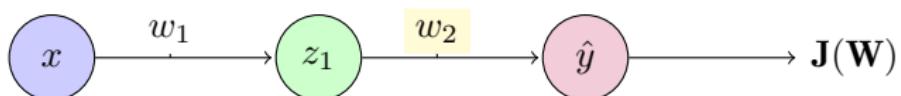
$$\mathbf{W}^* = \operatorname{argmin}_{\mathbf{W}} \mathbf{J}(\mathbf{W})$$

### Algoritmo: Descenso de gradiente

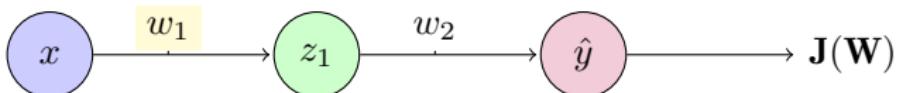
- 1 Inicializar los pesos aleatoriamente  $\sim \mathcal{N}(0, \sigma^2)$
- 2 Repetir hasta converger:
- 3 Calcular el gradiente  $\frac{\partial \mathbf{J}(\mathbf{W})}{\partial \mathbf{W}}$
- 4 Actualizar los pesos  $\mathbf{W} \leftarrow \mathbf{W} - \eta \frac{\partial \mathbf{J}(\mathbf{W})}{\partial \mathbf{W}}$
- 5 Devolver pesos *óptimos*



¿Cómo se calculan los gradientes?  
Con la regla de la cadena.



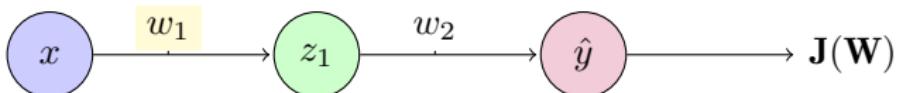
$$\frac{\partial \mathbf{J}(\mathbf{W})}{\partial w_2} = \frac{\partial \mathbf{J}(\mathbf{W})}{\partial \hat{y}} \cdot \frac{\partial \hat{y}}{\partial w_2}$$



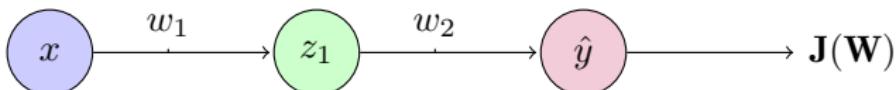
$$\frac{\partial \mathbf{J}(\mathbf{W})}{\partial w_1} = \frac{\partial \mathbf{J}(\mathbf{W})}{\partial \hat{y}} \cdot \frac{\partial \hat{y}}{\partial w_1}$$

# Retropropagación

## Intro al aprendizaje profundo



$$\frac{\partial \mathbf{J}(\mathbf{W})}{\partial w_1} = \frac{\partial \mathbf{J}(\mathbf{W})}{\partial \hat{y}} \cdot \frac{\partial \hat{y}}{\partial z_1} \cdot \frac{\partial z_1}{\partial w_1}$$



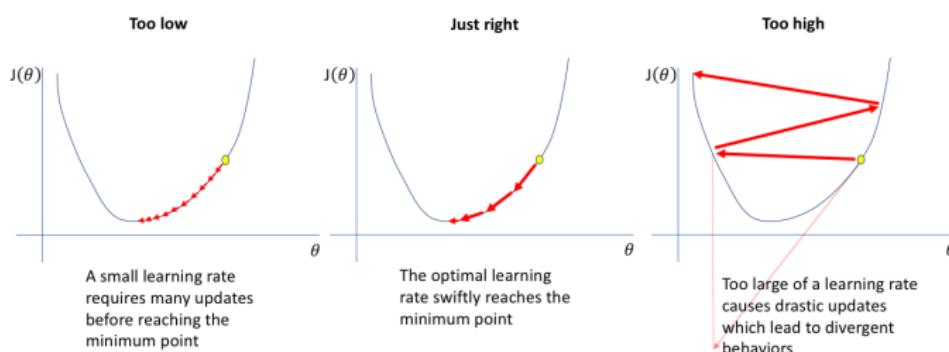
¿Cómo se calculan los gradientes?

Con la regla de la cadena.

Esto se repite para **cada peso** en la red neuronal, usando los gradientes de las capas posteriores.

La actualización de pesos está dada por:

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



## Algoritmo: Descenso de gradiente

- 1 Inicializar los pesos aleatoriamente  $\sim \mathcal{N}(0, \sigma^2)$
  - 2 Repetir hasta converger:
  - 3 Calcular el gradiente  $\frac{\partial \mathbf{J}(\mathbf{W})}{\partial \mathbf{W}}^*$
  - 4 Actualizar los pesos  $\mathbf{W} \leftarrow \mathbf{W} - \eta \frac{\partial \mathbf{J}(\mathbf{W})}{\partial \mathbf{W}}$
  - 5 Devolver pesos *óptimos*
- › \*Esto es muy pesado de calcular (computacionalmente).



# Descenso de gradiente estocástico

Intro al aprendizaje profundo

## Algoritmo: Descenso de gradiente estocástico

- 1 Inicializar los pesos aleatoriamente  $\sim \mathcal{N}(0, \sigma^2)$
- 2 Repetir hasta converger:
- 3 Seleccionar observación  $i$
- 4 Calcular el gradiente  $\frac{\partial \mathbf{J}_i(\mathbf{W})}{\partial \mathbf{W}}^*$
- 5 Actualizar los pesos  $\mathbf{W} \leftarrow \mathbf{W} - \eta \frac{\partial \mathbf{J}(\mathbf{W})}{\partial \mathbf{W}}$
- 6 Devolver pesos *óptimos*

➤ \*Esto es muy sencillo de calcular (computacionalmente), pero es estocástico.



Clubes de Ciencia  
Mexico

## Algoritmo: Descenso de gradiente estocástico - *Mini batches*

1 Inicializar los pesos aleatoriamente  $\sim \mathcal{N}(0, \sigma^2)$

2 Repetir hasta converger:

3 Seleccionar un batch  $B$  de observaciones

4 Calcular el gradiente 
$$\frac{\partial \mathbf{J}(\mathbf{W})}{\partial \mathbf{W}} = \frac{1}{B} \sum_{k=1}^B \frac{\partial \mathbf{J}_k(\mathbf{W})}{\partial \mathbf{W}}^*$$

5 Actualizar los pesos  $\mathbf{W} \leftarrow \mathbf{W} - \eta \frac{\partial \mathbf{J}(\mathbf{W})}{\partial \mathbf{W}}$

6 Devolver pesos óptimos

› \*Esto es rápido de calcular (computacionalmente), y da una mejor estimación del gradiente.



Clubes de  
Ciencia  
Mexico

- › Los **frameworks** para aprendizaje profundo (como TensorFlow, PyTorch, etc.) ya hacen la diferenciación y optimización por nosotros, es decir, ya calculan el gradiente y actualizan los pesos.
- › Nosotros exploraremos el uso de **TensorFlow** a través de su API de alto nivel, **Keras**, para las redes neuronales que estaremos construyendo.
- › Comenzaremos retomando algunos de los problemas planteados en la sesión anterior.



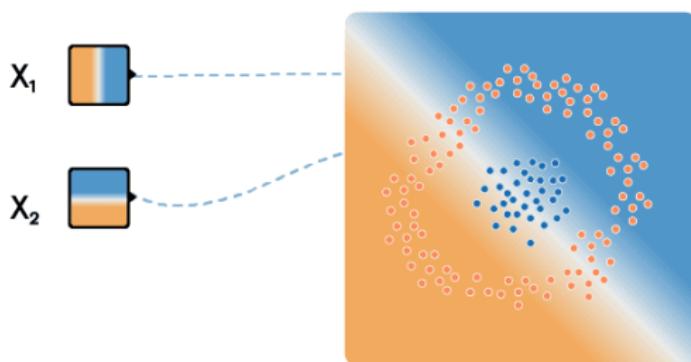
# TensorFlow

TensorFlow es un framework open-source para Machine Learning desarrollado por Google. Utilizado para construir y entrenar redes neuronales artificiales.

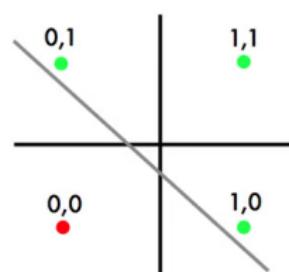


Clubes de Ciencia  
Mexico

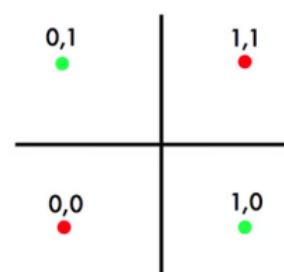
## Ejercicio: Exploración del TensorFlow Playground



## Ejercicio: Problema de separabilidad lineal



OR



XOR

# Ejercicio: Exploración con TensorFlow

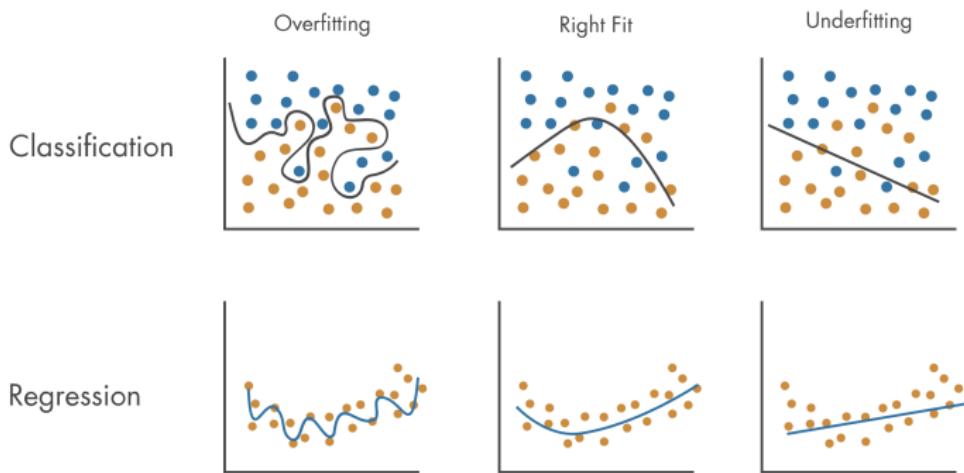


# TensorFlow

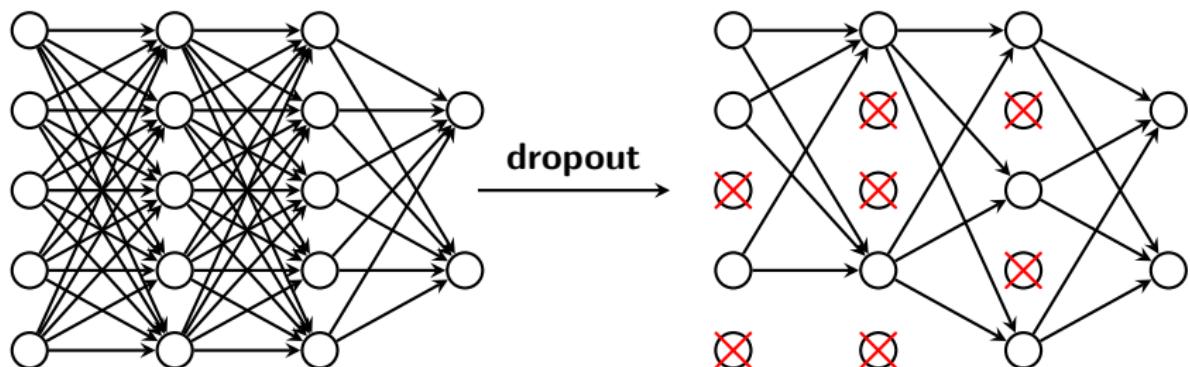
- › Setting the learning rate of your neural network
- › Retropropagación paso a paso
- › TensorFlow Tutorials
- › Libro Neural Networks and Deep Learning

# El problema del overfitting

## Intro al aprendizaje profundo



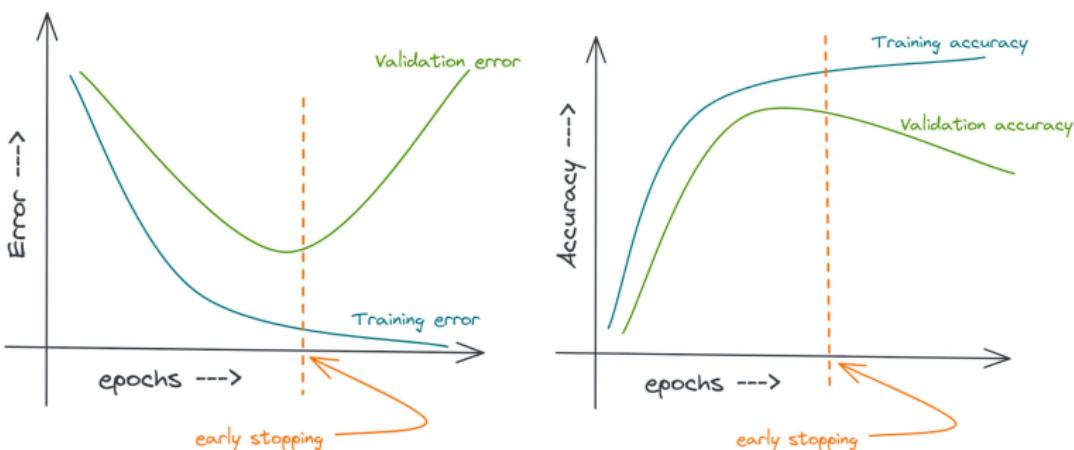
- › La **regularización** consiste en alguna técnica que sirve para evitar que un modelo se sobreajuste.
- › Es necesaria porque ayuda a mejorar la generalización de nuestro modelo con datos no vistos.
- › Los métodos de regularización que exploraremos serán:
  - » *Dropout*
  - » *Early stopping*



- › Durante el entrenamiento, establecemos aleatoriamente algunas activaciones en 0
  - » Típicamente hacemos "drop" del 50% de activaciones en una capa.
  - » Esto forza a la red a no depender de ningún nodo/neurona.
- › Podemos realizar el dropout en TensorFlow utilizando la capa `tf.keras.layers.Dropout(0.5)`, donde el 0.5 puede variar de acuerdo a lo especificado.

# Early stopping

## Intro al aprendizaje profundo



- El *Early Stopping* puede ser realizado en TensorFlow de manera sencilla creando un callback (función que se llama en cada iteración durante el entrenamiento de la red neuronal):

```
model = tf.keras.models.Sequential(...)  
callback = tf.keras.callbacks.EarlyStopping(monitor='loss',  
patience=3)  
history = model.fit(..., callbacks=[callback])
```

- › Artículo "Dropout: A Simple Way to Prevent Neural Networks from Overfitting"
- › Regularización en Redes Neuronales
- › Vanishing and Exploding Gradients in Neural Network Models:  
Debugging, Monitoring, and Fixing