



UNIVERSIDAD DE CHILE

# Minería de Datos

Welcome to the Machine Learning class

---

Valentin Barriere

Universidad de Chile – DCC

CC5205, Otoño 2024

# Introducción al aprendizaje profundo

# Perceptrón

---

# Outline : Perceptrón

## Perceptrón

Perceptrón multicapa

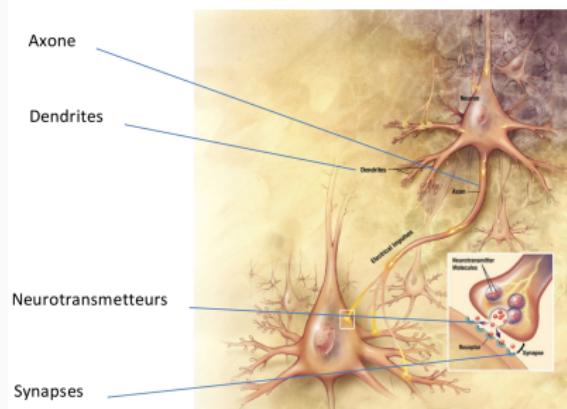
Recuerdos y Entrenamiento

Representaciones

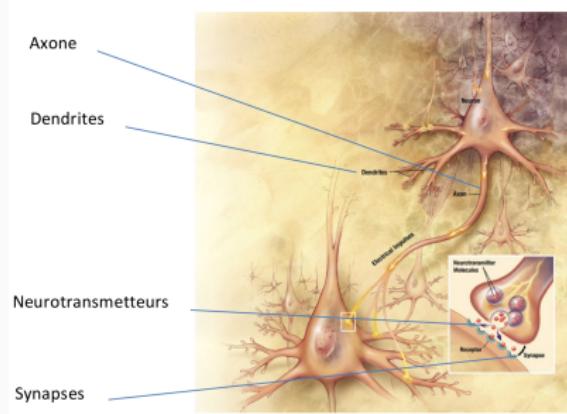
Frameworks

# Primero, ¿por qué este nombre?

Una neurona recibe varias informaciones (neurotransmisores) a nivel de sus **dendritas**. Estos neurotransmisores son liberados por las **sinapsis**. Cuando la cantidad de información supera un cierto **umbral**, la neurona se "activa", enviando una corriente eléctrica a través de su axón, lo que le permite **emitir a su vez** neurotransmisores a través de sus sinapsis.



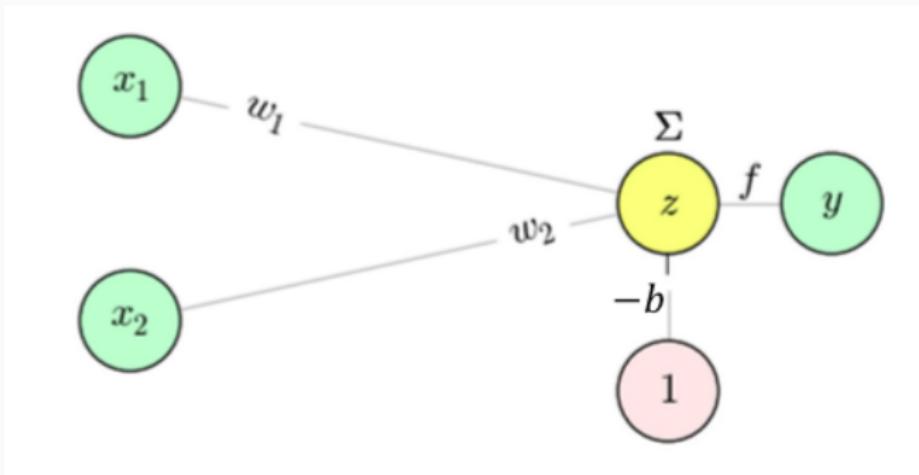
# Primero, ¿por qué este nombre?



## Para resumir:

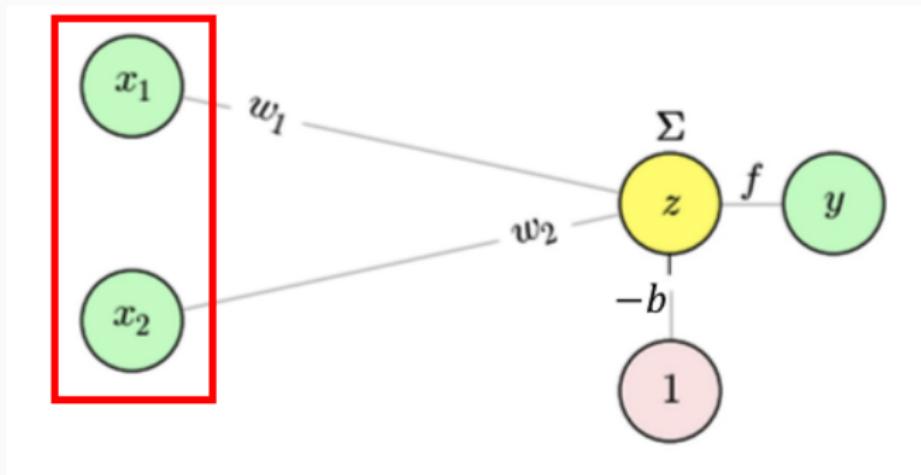
- Las salidas de una neurona son las entradas de otra.
- Una neurona emite cuando recibe una cantidad de información que supera un umbral.
- La cantidad de información emitida a la siguiente neurona es gestionada por las sinapsis, que se activan a partir de un umbral.

# Perceptrón – Presentación



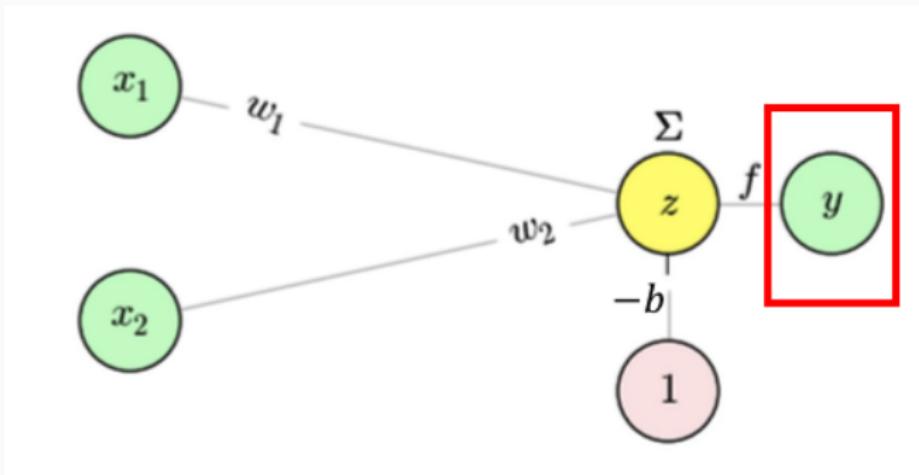
$$y = f(x_1 w_1 + x_2 w_2 - b)$$

# Perceptrón – Presentación



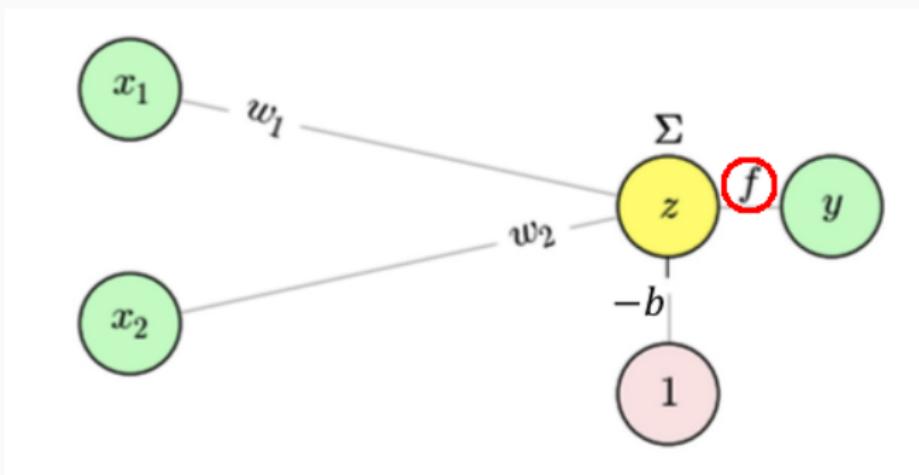
$$y = f(x_1 w_1 + x_2 w_2 - b)$$

## Perceptrón – Presentación



$$y = f(x_1 w_1 + x_2 w_2 - b)$$

## Perceptrón – Presentación



$$y = f(x_1 w_1 + x_2 w_2 - b)$$

En teoría  $f$  es la función de Heaviside:  $f(z) = \mathbb{1}_{\mathbb{R}_+} = \begin{cases} 1 & \text{si } z \geq 0 \\ 0 & \text{si } z < 0 \end{cases}$

## Perceptrón multicapa

---

# Outline : Perceptrón multicapa

Perceptrón

**Perceptrón multicapa**

Recuerdos y Entrenamiento

Representaciones

Frameworks

# MLP – Presentación

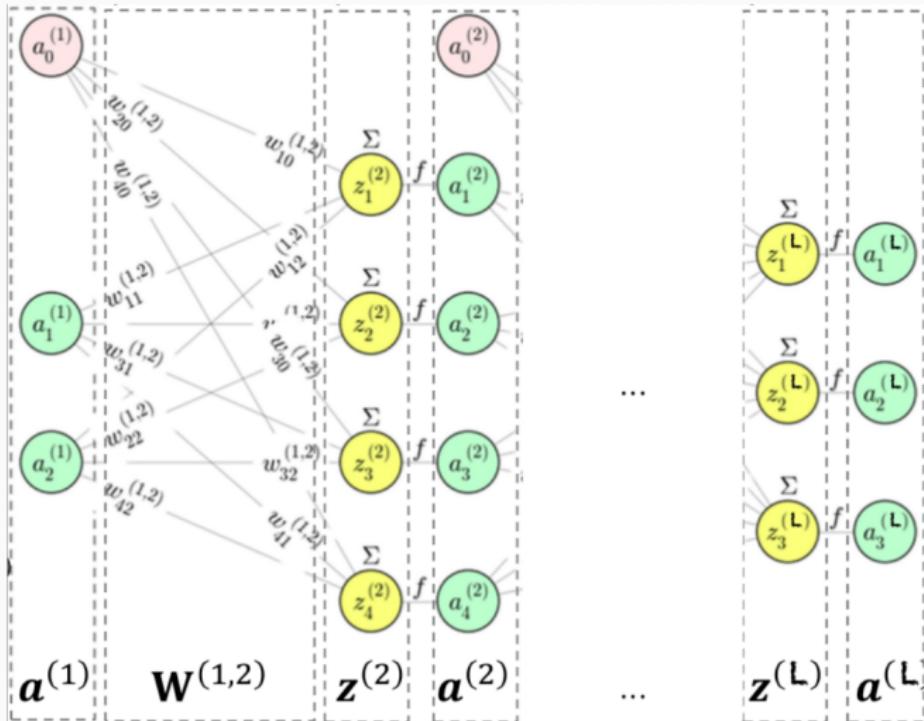


Figure 1: Propagación en un MLP

# MLP – Presentación

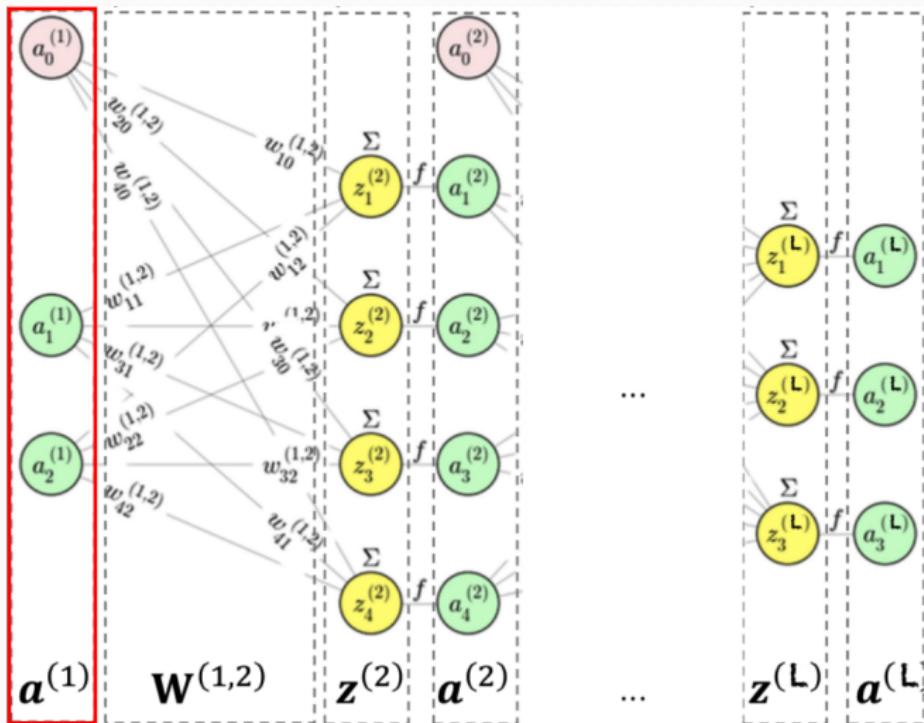


Figure 1: Propagación en un MLP: 1ra capa

# MLP – Presentación

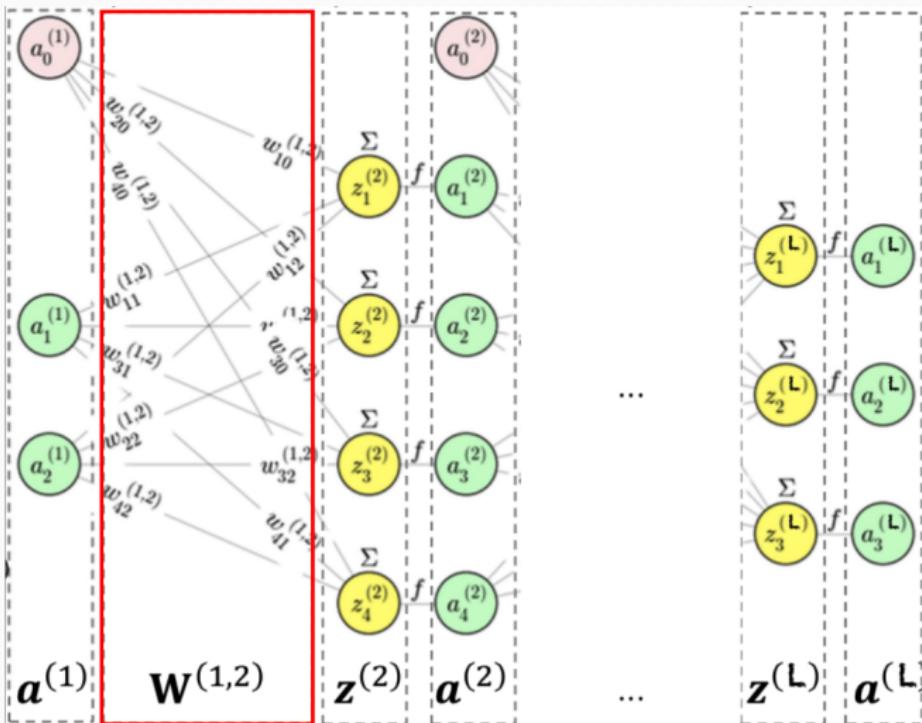


Figure 1: Propagación en un MLP: Matriz de paso

# MLP – Presentación

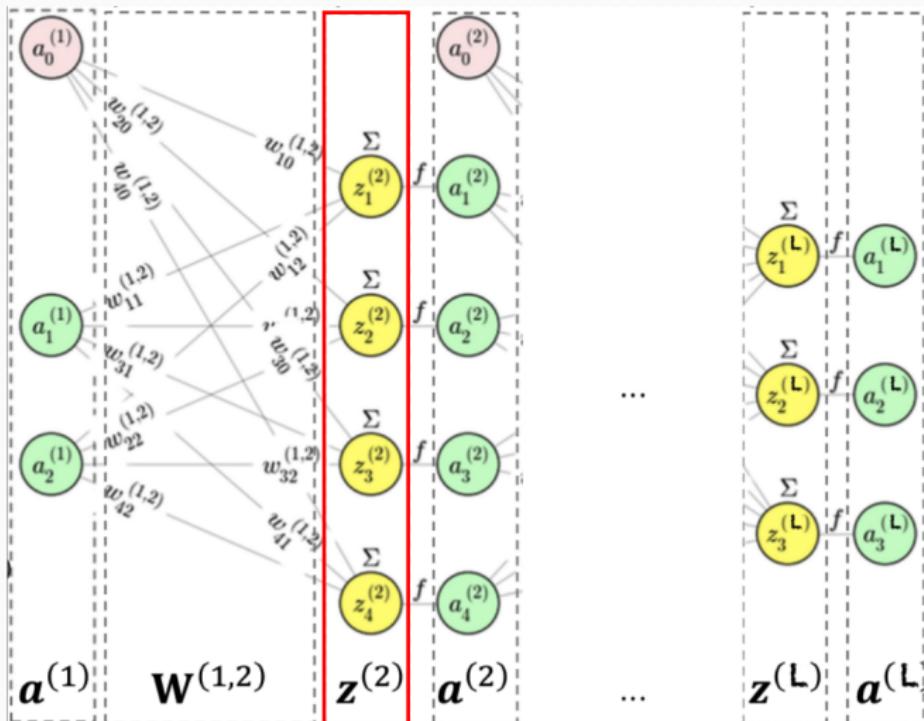
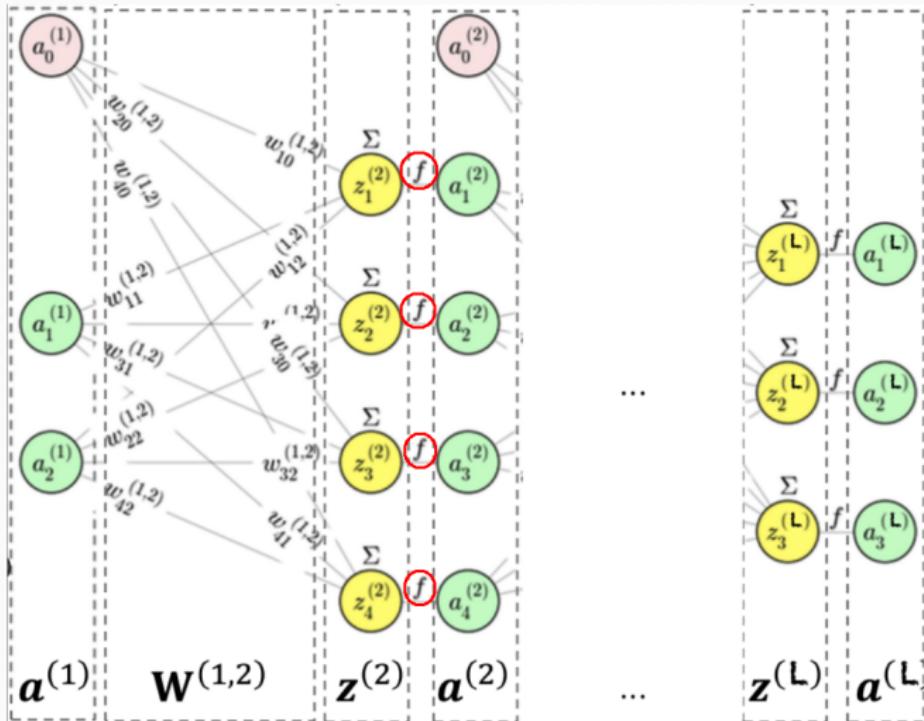


Figure 1: Propagación en un MLP: Entrada 2da capa

# MLP – Presentación



**Figure 1:** Propagación en un MLP: Activación

# MLP – Presentación

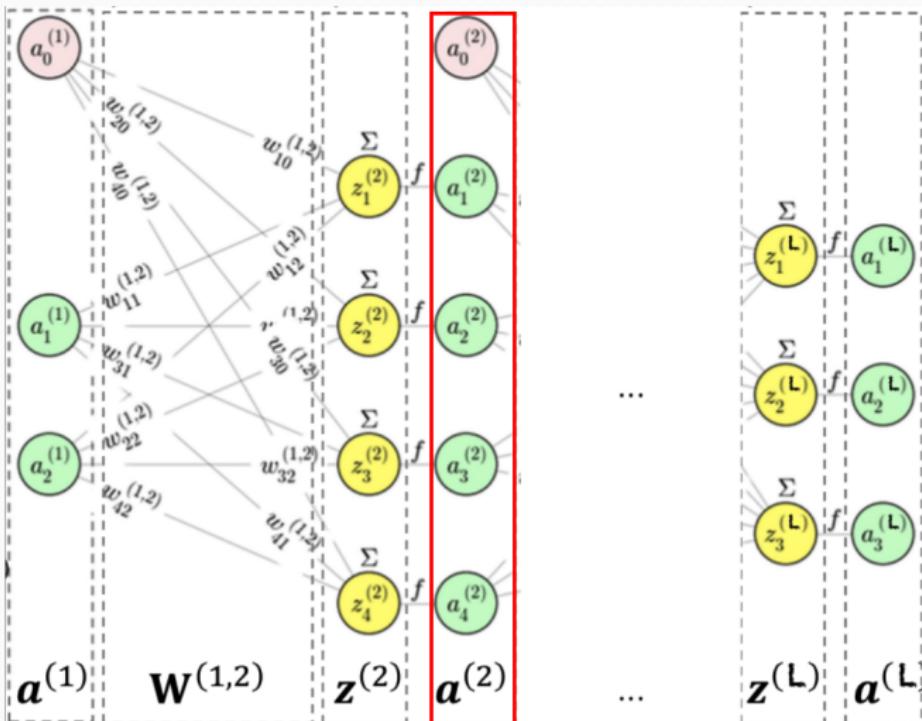


Figure 1: Propagación en un MLP: Salida 2da capa

# MLP – Presentación

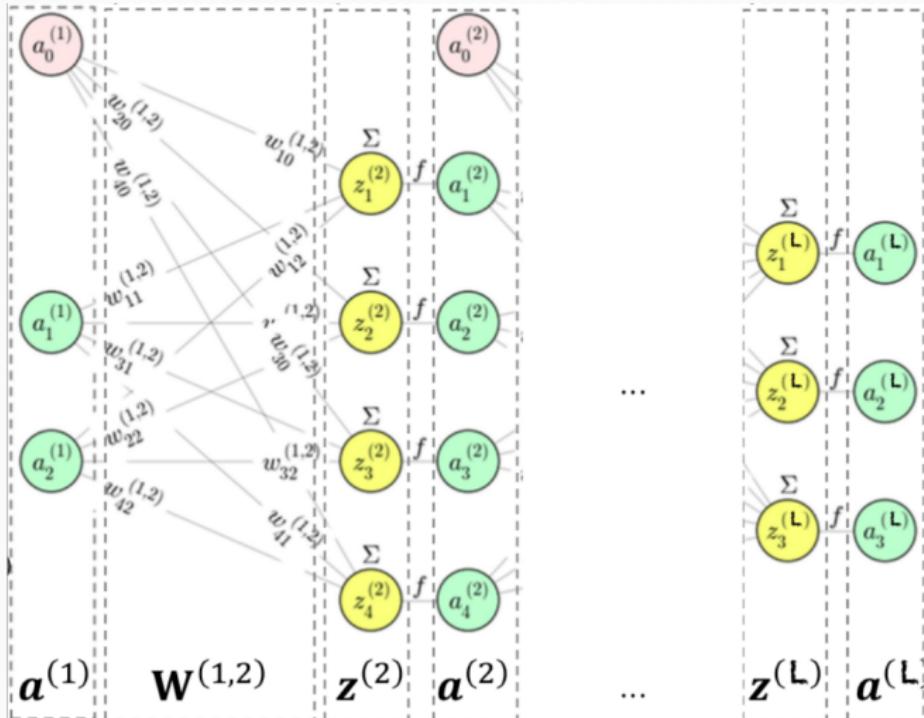
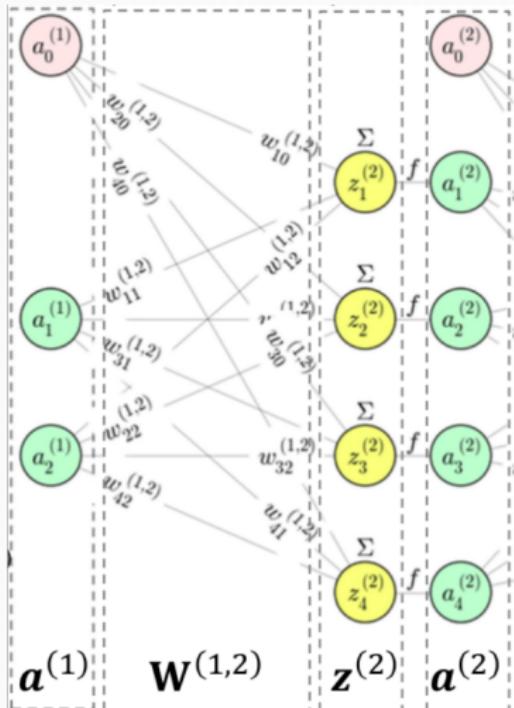


Figure 1: Propagación en un MLP:  $\mathbf{a}^{(\ell+1)} = f(\mathbf{W}_{(\ell,\ell+1)} \mathbf{a}^{(\ell)})$

# MLP: Principio de una capa



$$a_1^{(2)} = f(\sum_j w_{1j}^{(1,2)} a_j^{(1)}) = f(\langle \mathbf{W}_1^{(1,2)}, \mathbf{a}^{(1)} \rangle)$$

$$a_2^{(2)} = f(\sum_j w_{2j}^{(1,2)} a_j^{(1)}) = f(\langle \mathbf{W}_2^{(1,2)}, \mathbf{a}^{(1)} \rangle)$$

$$a_3^{(2)} = f(\sum_j w_{3j}^{(1,2)} a_j^{(1)}) = f(\langle \mathbf{W}_3^{(1,2)}, \mathbf{a}^{(1)} \rangle)$$

$$a_4^{(2)} = f(\sum_j w_{4j}^{(1,2)} a_j^{(1)}) = f(\langle \mathbf{W}_4^{(1,2)}, \mathbf{a}^{(1)} \rangle)$$

Por la capa  $\ell$ , con  $n_\ell$  neuronas:

$$\mathbf{a}^{(\ell)} = (a_0^{(\ell)} \dots a_{n_\ell}^{(\ell)})$$

$$\mathbf{W}^{(\ell, \ell+1)} = (\mathbf{W}_1^{(\ell, \ell+1)} \dots \mathbf{W}_{n_k}^{(\ell, \ell+1)})$$

**Calculo matricial de una capa, similar a LogReg!**

Tenemos  $\mathbf{a}^{(\ell+1)} = f(\mathbf{z}^{(\ell+1)}) = f(\mathbf{W}^{(\ell, \ell+1)} \mathbf{a}^{(\ell)})$

# MLP: Principio

Apilamos capas una sobre otra: la salida de la  $\ell^{\text{ésima}}$  capa es la entrada de la  $\ell + 1^{\text{ésima}}$  capa. Si consideramos la función  $h^{(1,2)}$  que representa el paso de la capa 1 a la capa 2.

- $\mathbf{a}^{(2)} = h^{(1,2)}(\mathbf{a}^{(1)})$
- $\mathbf{a}^{(3)} = h^{(2,3)}(\mathbf{a}^{(2)})$
- ...
- $\mathbf{a}^{(L)} = h^{(L-1,L)}(\mathbf{a}^{(L-1)})$
- $\mathbf{a}^{(L)} = h^{(L-1,L)} \circ h^{(L-2,L-1)} \circ \dots \circ h^{(1,2)}(\mathbf{a}^{(1)})$
- $MLP = h^{(L-1,L)} \circ h^{(L-2,L-1)} \circ \dots \circ h^{(1,2)}$  tal que  $MLP(\mathbf{a}^{(1)}) = \mathbf{a}^{(L)}$

## Una función final como una composición de funciones

El MLP puede verse como una composición de funciones no lineales.

Esto permite obtener una función que pasa de los datos de entrada a la salida deseada, que es lo más compleja posible.

# Funciones de activación

## Función de Heaviside

- En teoría  $f$  es la función de Heaviside:  $f(z) = \mathbb{1}_{\mathbb{R}_+} = \begin{cases} 0 & \text{si } z < 0 \\ 1 & \text{si } z \geq 0 \end{cases}$
- Problema:** esta función no es derivable.

En la práctica se utilizan funciones derivables como:

Nombre	Gráfico	Ecuación	Valores en $\pm\infty$	Valores derivada en $\pm\infty$
Sigmoide		$f(z) = \frac{1}{1+e^{-z}}$	0; 1	0; 0
Tangente hiperbólica		$f(z) = \frac{e^z - e^{-z}}{e^z + e^{-z}}$	-1; 1	0; 0
ReLU <sup>1</sup>		$f(z) = \begin{cases} 0 & \text{si } z < 0 \\ z & \text{si } z \geq 0 \end{cases}$	0; $z$	0; 1
ELU <sup>2</sup>		$f(z) = \begin{cases} \alpha(e^z - 1) & \text{si } z < 0 \\ z & \text{si } z \geq 0 \end{cases}$	$-\alpha; z$	0; 1

<sup>1</sup>Unidad de Rectificación Lineal

<sup>2</sup>Unidad Exponencial Lineal

## **Recuerdos y Entrenamiento**

---

# Outline : Recuerdos y Entrenamiento

Perceptrón

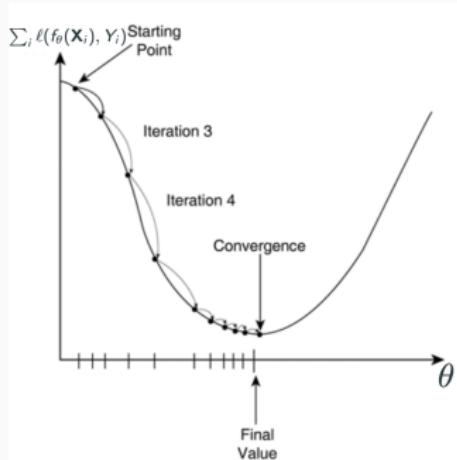
Perceptrón multicapa

**Recuerdos y Entrenamiento**

Representaciones

Frameworks

# Recuerdos: Optimización



## Optimización de la función de costo

- Sirve para converger al valor mínimo de la función de costo en el conjunto de datos de entrenamiento
- Mejor caso: rápido y preciso
- A menudo se hacen aproximaciones para ser más rápidos

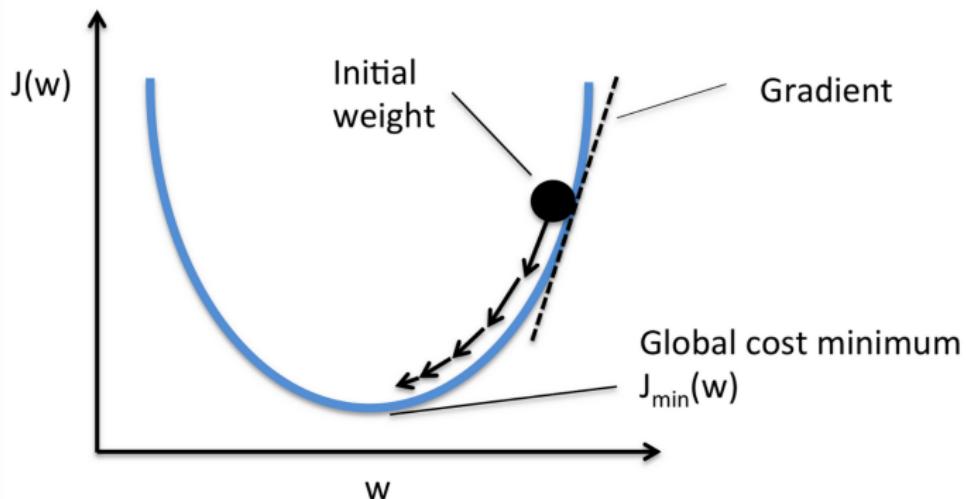
# Gradiente

Una función multivariante  $f(x)$  puede escribirse como una serie de Taylor:

$$f(x + \delta_x) = f(x) + \nabla_x f(x)\delta_x + \mathcal{O}(\|\delta_x^2\|)$$

## Definición

El gradiente de una función  $\nabla_x f(x) = (\frac{\partial f}{\partial_i})_{i=1..n}$  es su derivativa según cada dimensión. Es una **aproximación lineal de la función al nivel local**.



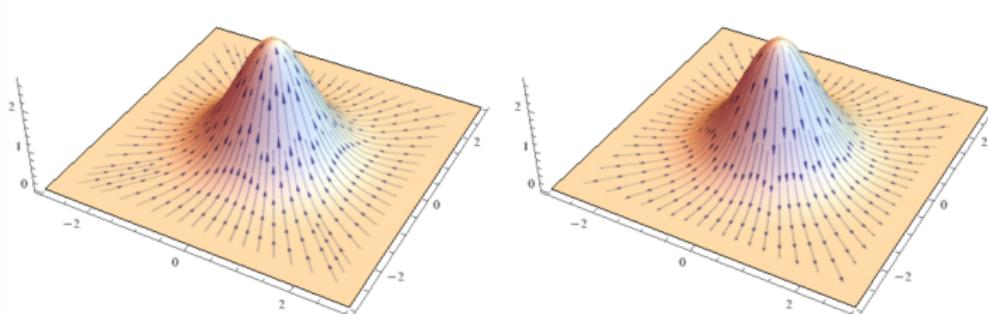
# Gradiente

Una función multivariante  $f(x)$  puede escribirse como una serie de Taylor:

$$f(x + \delta_x) = f(x) + \nabla_x f(x) \delta_x + \mathcal{O}(\|\delta_x^2\|)$$

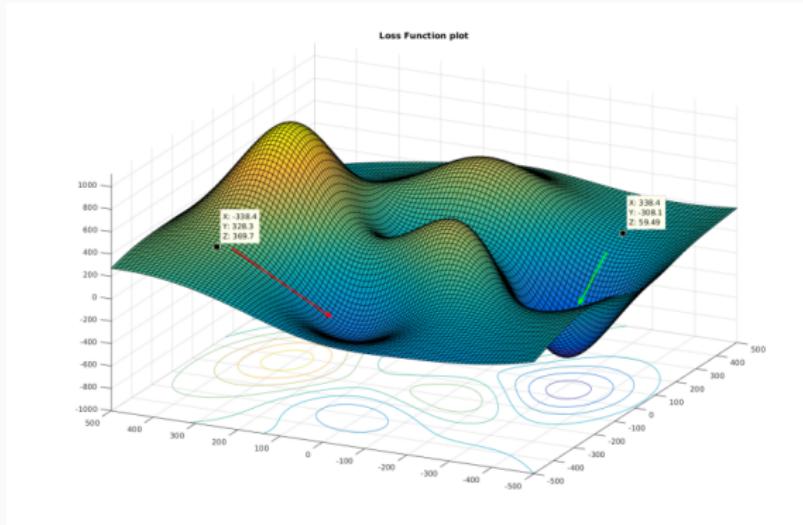
## Definición

El gradiente de una función  $\nabla_x f(x) = (\frac{\partial f}{\partial_i})_{i=1..n}$  es su derivativa según cada dimensión. Es una **aproximación lineal de la función al nivel local**.



Mas detalles [aca.](#)

# Visualización de la función de costo



Se puede visualizar el valor de la función de costo como una superficie:

- Los valores de los parámetros  $\theta$  varían en el plano, y el valor de la función  $\ell(\mathcal{D}_n; \theta)$  varía en altura.
- La convergencia se produce cuando se tienen parámetros que están en un hueco de esta superficie (mínimo local o global, dependiendo del modelo)

# Optimización : visualización

# Optimización : Descenso del Gradiente Estocástico

## Descenso del gradiente

Después de cada cálculo de la función de costo  $\ell(Y_i, f_\theta(\mathbf{X}_i); \theta)$ , se calcula el gradiente de esta función para actualizar los parámetros  $\theta$ :

$$\theta \leftarrow \theta - \alpha * \nabla_{\theta} \ell(Y_i, f_\theta(\mathbf{X}_i); \theta)$$

## Ejemplo

- Sean  $f_\theta(\mathbf{X}) = \theta^T \mathbf{X} = \sum_k^d \theta_k X^{(k)}$  y  $\ell(Y, f_\theta(\mathbf{X})) = \frac{1}{2}(Y - f_\theta(\mathbf{X}))^2$

$$\nabla_{\theta} \ell(Y_i, f_\theta(\mathbf{X}_i); \theta) = \begin{pmatrix} \frac{\partial}{\partial \theta_1} \\ \vdots \\ \frac{\partial}{\partial \theta_d} \end{pmatrix} \cdot \ell(Y_i, f_\theta(\mathbf{X}_i); \theta) = \begin{pmatrix} X_i^{(1)}(Y - f_\theta(\mathbf{X}_i)) \\ \vdots \\ X_i^{(d)}(Y - f_\theta(\mathbf{X}_i)) \end{pmatrix}$$

# Optimización : Descenso del Gradiente Estocástico

## Descenso del gradiente

Después de cada cálculo de la función de costo  $\ell(Y_i, f_\theta(\mathbf{X}_i); \theta)$ , se calcula el gradiente de esta función para actualizar los parámetros  $\theta$ :

$$\theta \leftarrow \theta - \alpha * \nabla_{\theta} \ell(Y_i, f_\theta(\mathbf{X}_i); \theta)$$

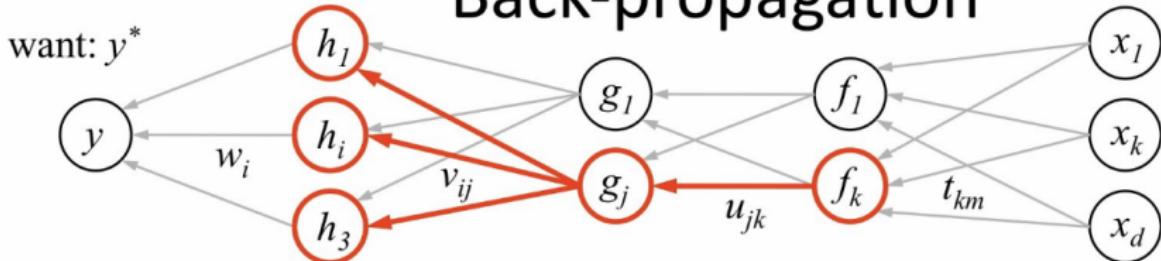
## Ejemplo

- Sean  $f_\theta(\mathbf{X}) = \theta^T \mathbf{X} = \sum_k^d \theta_k X^{(k)}$  y  $\ell(Y, f_\theta(\mathbf{X})) = \frac{1}{2}(Y - f_\theta(\mathbf{X}))^2$

$$\begin{pmatrix} \theta_1 \\ \vdots \\ \theta_d \end{pmatrix} \leftarrow \begin{pmatrix} \theta_1 \\ \vdots \\ \theta_d \end{pmatrix} - \alpha * \begin{pmatrix} X_i^{(1)}(Y - f_\theta(\mathbf{X}_i)) \\ \vdots \\ X_i^{(d)}(Y - f_\theta(\mathbf{X}_i)) \end{pmatrix}$$

# Backpropagation: Príncipe

## Back-propagation



1. receive new observation  $x = [x_1 \dots x_d]$  and target  $y^*$
2. **feed forward:** for each unit  $g_j$  in each layer  $1 \dots L$   
compute  $g_j$  based on units  $f_k$  from previous layer:  $g_j = \sigma\left(u_{j0} + \sum_k u_{jk} f_k\right)$
3. get prediction  $y$  and error ( $y - y^*$ )
4. **back-propagate error:** for each unit  $g_j$  in each layer  $L \dots 1$

(a) compute error on  $g_j$

$$\frac{\partial E}{\partial g_j} = \sum_i \underbrace{\sigma'(h_i)}_{\text{should } g_j \text{ be higher or lower?}} \underbrace{v_{ij}}_{\text{how } h_i \text{ will change as } g_j \text{ changes}} \underbrace{\frac{\partial E}{\partial h_i}}_{\text{was } h_i \text{ too high or too low?}}$$

(b) for each  $u_{jk}$  that affects  $g_j$

(i) compute error on  $u_{jk}$

$$\frac{\partial E}{\partial u_{jk}} = \frac{\partial E}{\partial g_j} \underbrace{\sigma'(g_j)}_{\text{do we want } g_j \text{ to be higher/lower?}} f_k$$

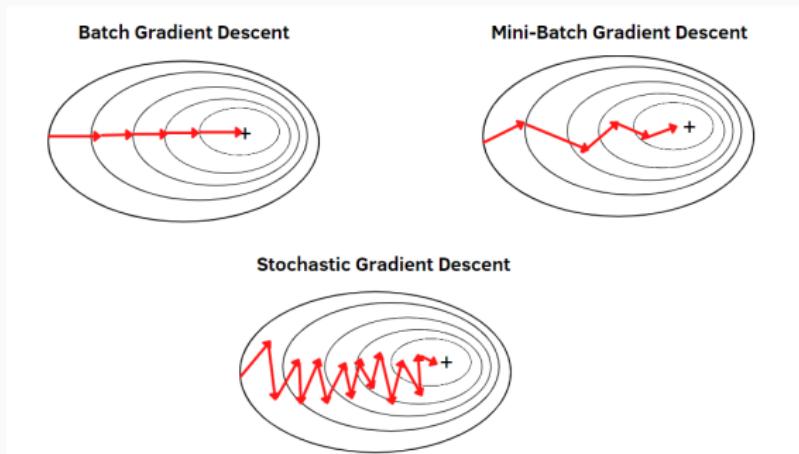
(ii) update the weight

$$u_{jk} \leftarrow u_{jk} - \eta \frac{\partial E}{\partial u_{jk}}$$

Mas detalles sobre backpropagacion en [este video](#), y generalidades [aca](#)

# Stochastic Gradient Descent

- Estas operaciones se pueden hacer en paralelo  $\Rightarrow$  GPU!
- No se puede utilizar todo el dataset de train para updatear los parametros del modelo, se usa la técnica de los **mini-batch**
- Vamos a updatear los pesos del modelo calculando el error sobre **m** ejemplos que vamos a tomar de manera aleatoria en el dataset
- Mas **m** es grande, menos ruido en la convergencia



# Representaciones

---

# Outline : Representaciones

Perceptrón

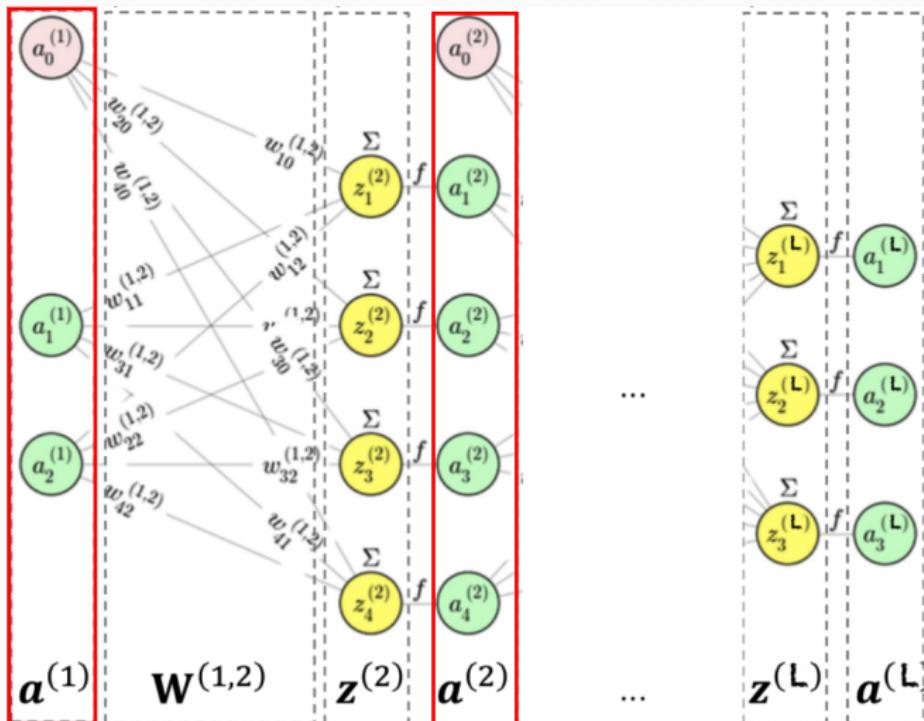
Perceptrón multicapa

Recuerdos y Entrenamiento

**Representaciones**

Frameworks

# MLP – Representaciones



**Figure 2:** Cada capa oculta  $\mathbf{a}^{(\ell)}$  ( $\ell \in \{2..L - 1\}$ ) es una representación vectorial del *feature vector*  $\mathbf{a}^{(1)}$  en entrada del MLP

# MLP – Representaciones

Cada capa oculta  $\mathbf{a}^{(\ell)}$  ( $\ell \in \{2..L - 1\}$ ) es una representación vectorial del *feature vector*  $\mathbf{a}^{(1)}$  en entrada del MLP

## Complejidad de las representaciones

Las representaciones obtenidas  $\mathbf{a}^{(\ell)}$  tienen propiedades diferentes:

- Tenemos en entrada un vector  $\mathbf{a}^{(1)}$  "bruto" y en salida  $\mathbf{a}^{(L)}$  una representación que permite la resolución de una tarea compleja.
- Cuanto más avanzamos en la red neuronal (cuanto mayor es  $\ell$ ), más compleja será la representación:  $\mathbf{a}^{(1)}$  es la representación de nivel más bajo, y  $\mathbf{a}^{(L)}$  es la representación de nivel más alto.

Para imágenes, representan:

I contornos

II nariz, boca, ojos, ...

III rostros

Para texto, representan:

I naturaleza gramatical

II Entidad o no

III Sentimiento, u otro concepto complejo

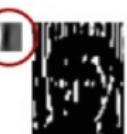
## Complejidad de las representaciones: ejemplo



Layer 3 activation (coefficients)



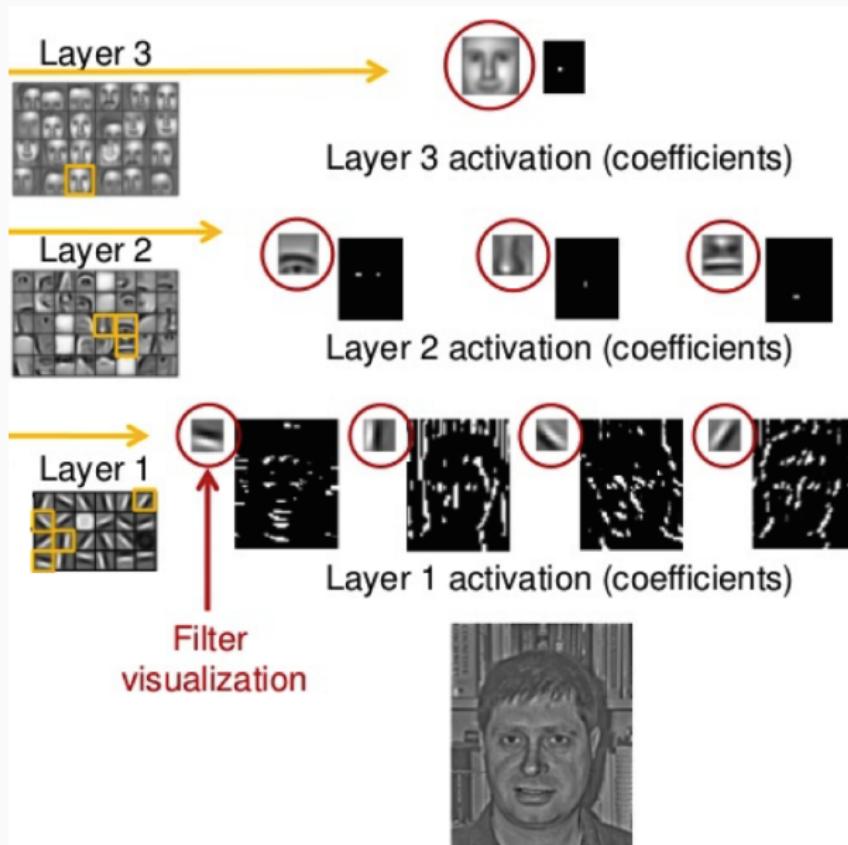
Layer 2 activation (coefficients)



Layer 1 activation (coefficients)



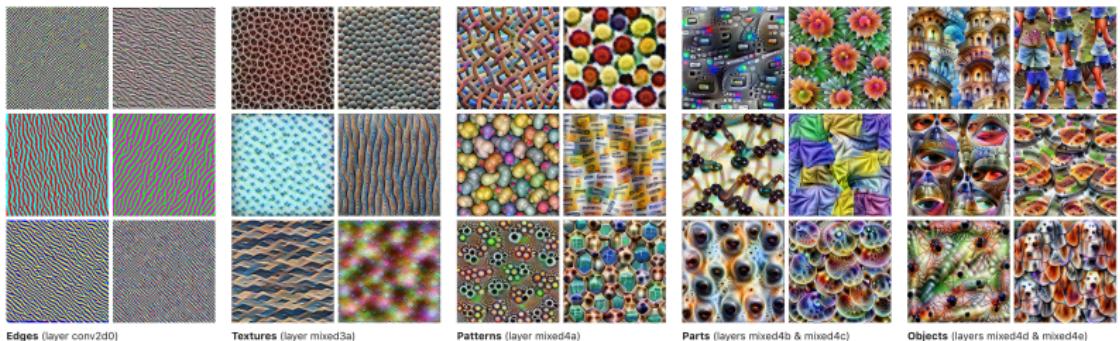
# Complejidad de las representaciones: Composicionalidad



# Visualización de activaciones de un AlexNet

## Feature Visualization

How neural networks build up their understanding of images



Deep Visualization Toolbox

[Distill.pub: Feature Visualization](https://distill.pub/)

# Aprendizaje de representaciones

## Aprendizaje de representaciones

- Cada capa proporciona una representación de los datos de entrada con dimensiones más bajas.
- Esas representaciones provenientes únicamente de los datos brutos a veces superan conjuntos de descriptores clásicos

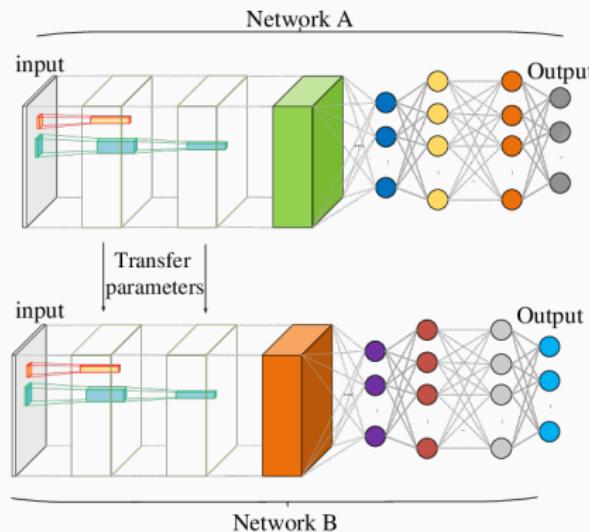


**Figure 3:** Vector de 4096 dimensiones de AlexNet con un reducción en 2D

# Transfer Learning y Fine-tuning

## Fine-tuning

El *fine-tuning* es el proceso de ajuste en el nuevo conjunto de datos  $\mathcal{D}$ . Puede implicar re-entrenar todas las capas, sólo las capas más profundas relacionadas con tareas específicas (congelando las capas iniciales), o re-entrenando las capas profundas y descongelando las capas iniciales de manera gradual.



# Frameworks

---

# Outline : Frameworks

---

Perceptrón

Perceptrón multicapa

Recuerdos y Entrenamiento

Representaciones

**Frameworks**

Biblioteca de Python para Deep Learning (Redes Neuronales Profundas).  
Es de alto nivel que corre sobre Tensorflow y otros backends.



- Visión: [Tutorial de Keras para afinar un VGG16 preentrenado](#), que puede ser utilizado con:  
[Diferentes modelos de CNN preentrenados disponibles en Keras](#)
- Texto y audio:  
[Tutorial RNN-LSTM Seq2seq para traducción automática](#)
- Texto: [Uso de embeddings de palabras preentrenados](#)

Biblioteca de Python para Deep Learning (Redes Neuronales Profundas)  
concurrente de Tensorflow



- Visión: [Tutorial para afinar un ResNet18 preentrenado](#)
- Audio: [Reconocimiento de voz con Wav2Vec2](#)
- Texto: [Tutorial RNN-GRU Seq2seq para traducción automática](#)

# HuggingFace's Transformers

Biblioteca de Python para **Transformers** (Tipo de Modelos de Redes Neuronales Profundas)



## Hugging Face

- Muchos modelos ya pre-entrenados, para imágenes, audio, multimodal, texto, ...
- Modelos clásicos, modelos generativos, embeddings, ... hasta los más grandes y nuevos (tipo LLaMA3-70B)
- Otras librerías: Diffuser, Datasets, Accelerate, PEFT, bitsandbytes, TRL, ...

**Questions?**

## References i