

# Unidad IV - Redes Neuronales

Germán Braun

Facultado de Informática - Universidad Nacional del Comahue

*german.braun@fi.uncoma.edu.ar*

3 de octubre de 2025

# Agenda

- 1 Aprendizaje Profundo
- 2 Redes Neuronales
- 3 Arquitectura
- 4 Backpropagation  
Paul Werbos (1974)
- 5 Capacidad de Generalización
- 6 Variantes de las Redes Neuronales
- 7 Conclusiones

## machine learning

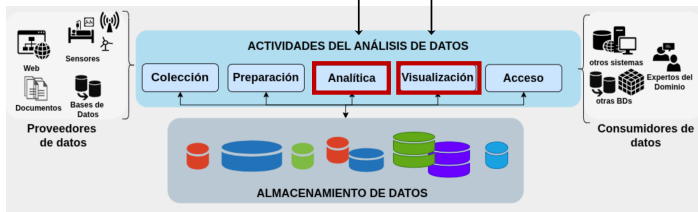
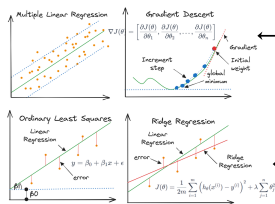
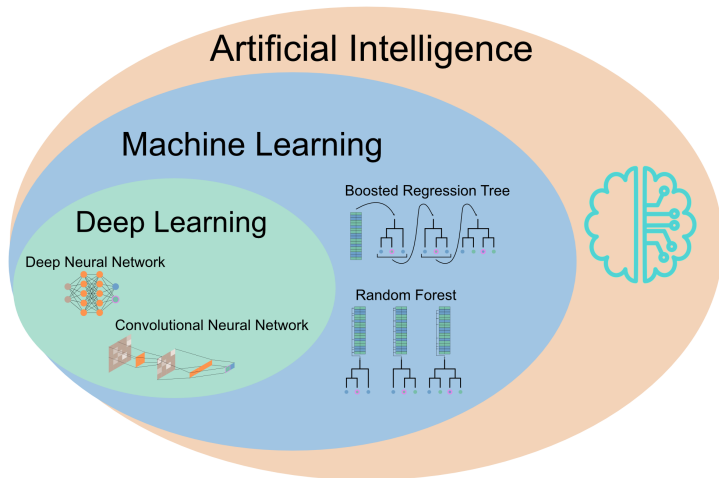


Figura 0.1: Proceso de Análisis de Datos

# Aprendizaje Profundo



(\*) image de [Machine Learning and Deep Learning with R](#)

# Deep Learning (cont'd)

- El **aprendizaje profundo** es un subcampo específico del aprendizaje automático

# Deep Learning (cont'd)

- El **aprendizaje profundo** es un subcampo específico del aprendizaje automático
- Hace énfasis en el aprendizaje en sucesivas **capas** de representación (de allí su nombre “profundo”)

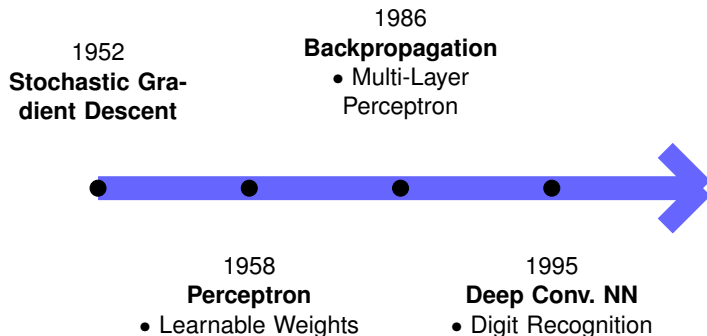
# Deep Learning (cont'd)

- El **aprendizaje profundo** es un subcampo específico del aprendizaje automático
- Hace énfasis en el aprendizaje en sucesivas **capas** de representación (de allí su nombre “profundo”)
- El modelo de aprendizaje es la **red neuronal**, la cual se estructura siguiendo esta idea de capas



# Deep Learning (cont'd)

- El **aprendizaje profundo** es un subcampo específico del aprendizaje automático
- Hace énfasis en el aprendizaje en sucesivas **capas** de representación (de allí su nombre “profundo”)
- El modelo de aprendizaje es la **red neuronal**, la cual se estructura siguiendo esta idea de capas
- El término red neuronal hace referencia a cómo los humanos pensamos



# Redes Neuronales

# Intuición (biológica)

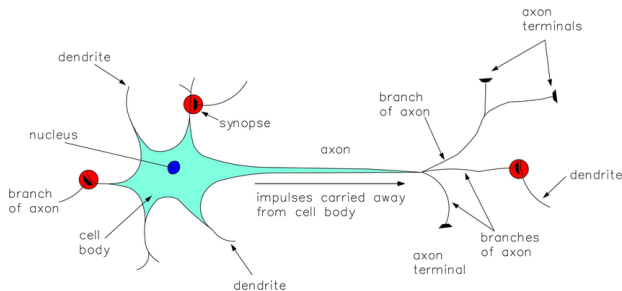


Figura 2.1: image de [paper](#)

- Una neurona consiste de un **cuerpo**, **dendritas** (dendrites) y una **axon**

# Intuición (biológica)

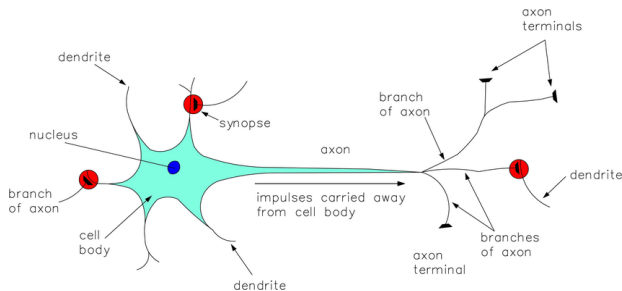


Figura 2.1: image de [paper](#)

- Una neurona consiste de un **cuerpo**, **dendritas** (dendrites) y una **axon**
- Recibe señales a través de las dendritas (sensores) y las envían por el **axon**

# Intuición (biológica)

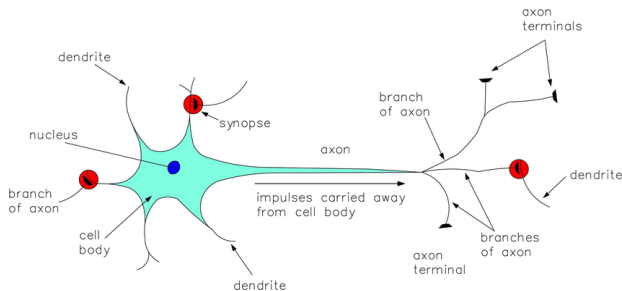
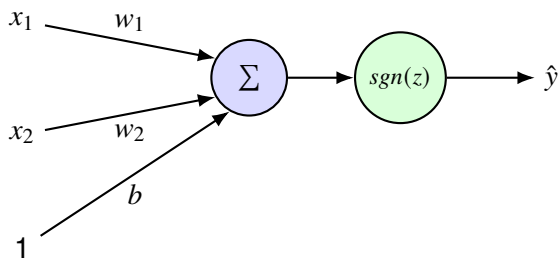


Figura 2.1: image de [paper](#)

- Una neurona consiste de un **cuerpo**, **dendritas** (dendrites) y una **axon**
- Recibe señales a través de las dendritas (sensores) y las envían por el **axon**
- Son *electricamente excitables*: si su polaridad cambia, la neurona genera un pulso electro-químico por el **axon** (se genera una salida)

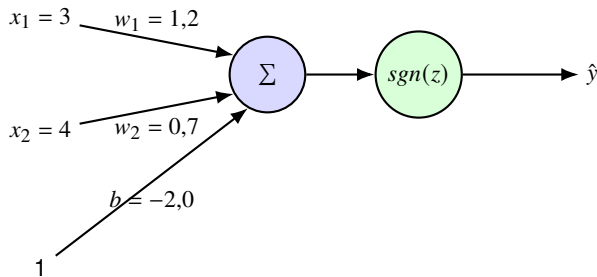
# Arquitectura

# Perceptrón Simple



$$sgn(z) = \begin{cases} 1 & \text{si } \Sigma > 0 \\ -1 & \text{si otherwise} \end{cases}$$

# Perceptrón: Clasificación de Email (Spam)



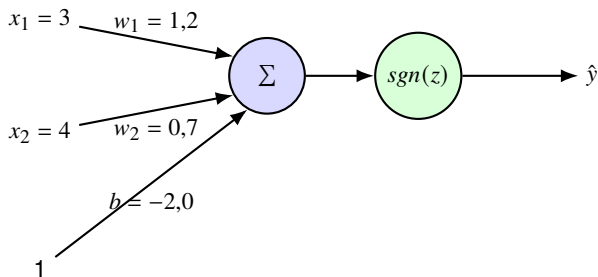
- $x_1$  = veces que aparece la palabra “gratis”
- $x_2$  = # de links
- 

$$sgn(z) = \begin{cases} 1 & \text{si } \Sigma > 0 \\ -1 & \text{si otherwise} \end{cases}$$



# Perceptrón: Clasificación de Email (Spam)

$$1) z = w_1x_1 + w_2x_2 + b$$

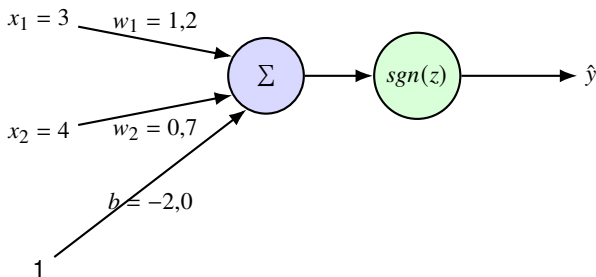


- $x_1$  = veces que aparece la palabra “gratis”
- $x_2$  = # de links
- 

$$sgn(z) = \begin{cases} 1 & \text{si } \Sigma > 0 \\ -1 & \text{si } otherwise \end{cases}$$

# Perceptrón: Clasificación de Email (Spam)

$$\begin{aligned} 1) \ z &= w_1x_1 + w_2x_2 + b \\ &= (1,2)(3) + (0,7)(4) - 2,0 \end{aligned}$$

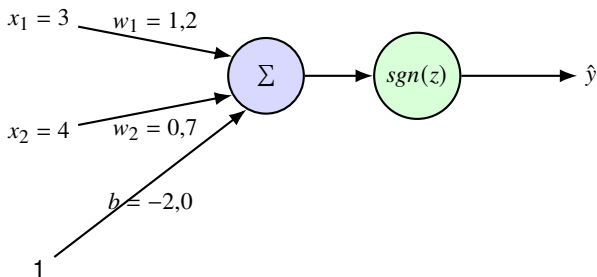


- $x_1$  = veces que aparece la palabra “gratis”
- $x_2$  = # de links
- 

$$sgn(z) = \begin{cases} 1 & \text{si } \Sigma > 0 \\ -1 & \text{si } otherwise \end{cases}$$

# Perceptrón: Clasificación de Email (Spam)

$$\begin{aligned} 1) \ z &= w_1x_1 + w_2x_2 + b \\ &= (1,2)(3) + (0,7)(4) - 2,0 \\ &= 3,6 + 2,8 - 2,0 = 4,4 \end{aligned}$$

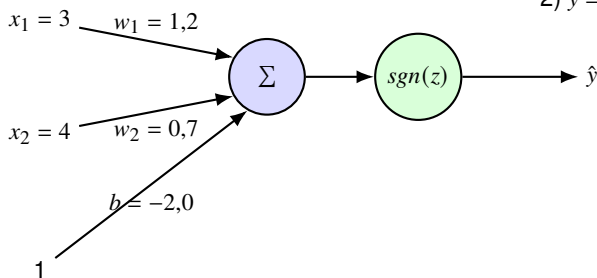


- $x_1$  = veces que aparece la palabra “gratis”
- $x_2$  = # de links
- 

$$sgn(z) = \begin{cases} 1 & \text{si } \Sigma > 0 \\ -1 & \text{si } otherwise \end{cases}$$

# Perceptrón: Clasificación de Email (Spam)

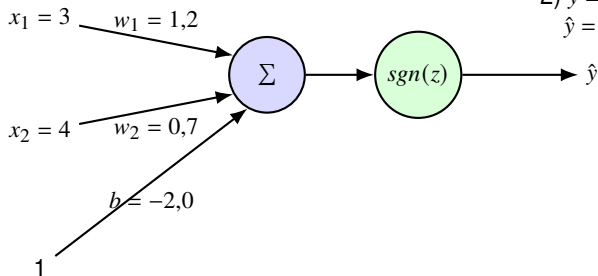
$$\begin{aligned} 1) z &= w_1x_1 + w_2x_2 + b \\ &= (1,2)(3) + (0,7)(4) - 2,0 \\ &= 3,6 + 2,8 - 2,0 = 4,4 \\ 2) \hat{y} &= \text{sgn}(z) \end{aligned}$$



- $x_1$  = veces que aparece la palabra “gratis”
- $x_2$  = # de links
- 

$$\text{sgn}(z) = \begin{cases} 1 & \text{si } \Sigma > 0 \\ -1 & \text{si } \textit{otherwise} \end{cases}$$

# Perceptrón: Clasificación de Email (Spam)



$$\begin{aligned} 1) \quad z &= w_1 x_1 + w_2 x_2 + b \\ &= (1,2)(3) + (0,7)(4) - 2,0 \\ &= 3,6 + 2,8 - 2,0 = 4,4 \\ 2) \quad \hat{y} &= \text{sgn}(z) \\ \hat{y} &= 1 \quad (\text{ya que } z \geq 0) \end{aligned}$$

- $x_1$  = veces que aparece la palabra “gratis”
- $x_2$  = # de links
- 

$$\text{sgn}(z) = \begin{cases} 1 & \text{si } \Sigma > 0 \\ -1 & \text{si } \textit{otherwise} \end{cases}$$

Entrenamiento por *corrección de error*

# Aprendizaje y poder expresivo del perceptrón

## Entrenamiento por *corrección de error*

- 1 Iniciación al azar

## Entrenamiento por *corrección de error*

- 1 Iniciación al azar
- 2 Se muestran muchos ejemplos con las salidas esperadas y en cada caso:



## Entrenamiento por *corrección de error*

- 1 Iniciación al azar
- 2 Se muestran muchos ejemplos con las salidas esperadas y en cada caso:
  - Se obtiene la salida

## Entrenamiento por *corrección de error*

- 1 Iniciación al azar
- 2 Se muestran muchos ejemplos con las salidas esperadas y en cada caso:
  - Se obtiene la salida
  - Si la salida es **correcta** → no se hacen cambios

## Entrenamiento por *corrección de error*

- 1 Iniciación al azar
- 2 Se muestran muchos ejemplos con las salidas esperadas y en cada caso:
  - Se obtiene la salida
  - Si la salida es **correcta** → no se hacen cambios
  - Si la salida es **incorrecta** → *penalización*: se actualizan los pesos en el sentido opuesto al que contribuyó a la salida incorrecta:

## Entrenamiento por *corrección de error*

- 1 Iniciación al azar
- 2 Se muestran muchos ejemplos con las salidas esperadas y en cada caso:
  - Se obtiene la salida
  - Si la salida es **correcta**  $\rightarrow$  no se hacen cambios
  - Si la salida es **incorrecta**  $\rightarrow$  *penalización*: se actualizan los pesos en el sentido opuesto al que contribuyó a la salida incorrecta:
    - $y^{(i)} = -1$  y  $\hat{y}^{(i)} = 1 \rightarrow w = w - \eta x^{(i)}$

## Entrenamiento por *corrección de error*

- 1 Iniciación al azar
- 2 Se muestran muchos ejemplos con las salidas esperadas y en cada caso:
  - Se obtiene la salida
  - Si la salida es **correcta** → no se hacen cambios
  - Si la salida es **incorrecta** → *penalización*: se actualizan los pesos en el sentido opuesto al que contribuyó a la salida incorrecta:
    - $y^{(i)} = -1$  y  $\hat{y}^{(i)} = 1 \rightarrow w = w - \eta x^{(i)}$
    - $y^{(i)} = 1$  y  $\hat{y}^{(i)} = -1 \rightarrow w = w + \eta x^{(i)}$

## Entrenamiento por *corrección de error*

- 1 Iniciación al azar
- 2 Se muestran muchos ejemplos con las salidas esperadas y en cada caso:
  - Se obtiene la salida
  - Si la salida es **correcta** → no se hacen cambios
  - Si la salida es **incorrecta** → *penalización*: se actualizan los pesos en el sentido opuesto al que contribuyó a la salida incorrecta:
    - $y^{(i)} = -1$  y  $\hat{y}^{(i)} = 1 \rightarrow w = w - \eta x^{(i)}$
    - $y^{(i)} = 1$  y  $\hat{y}^{(i)} = -1 \rightarrow w = w + \eta x^{(i)}$
    - En concreto:

$$\epsilon^{(i)} = \frac{y^{(i)} - \hat{y}^{(i)}}{2}$$

## Entrenamiento por *corrección de error*

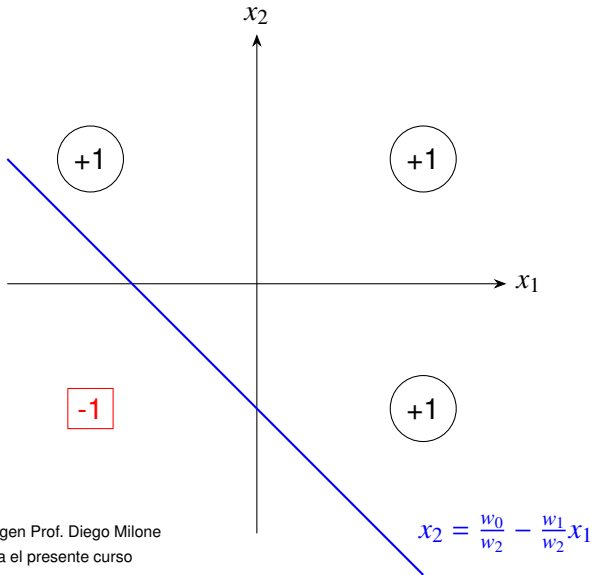
- 1 Iniciación al azar
- 2 Se muestran muchos ejemplos con las salidas esperadas y en cada caso:
  - Se obtiene la salida
  - Si la salida es **correcta** → no se hacen cambios
  - Si la salida es **incorrecta** → *penalización*: se actualizan los pesos en el sentido opuesto al que contribuyó a la salida incorrecta:
    - $y^{(i)} = -1$  y  $\hat{y}^{(i)} = 1 \rightarrow w = w - \eta x^{(i)}$
    - $y^{(i)} = 1$  y  $\hat{y}^{(i)} = -1 \rightarrow w = w + \eta x^{(i)}$
    - En concreto:

$$\epsilon^{(i)} = \frac{y^{(i)} - \hat{y}^{(i)}}{2}$$

- Poder expresivo **limitado a funciones booleanas**
- Puede expresar **solo decisiones lineales**

# Poder expresivo del perceptrón (OR)

$$w_1x_1 + w_2x_2 - w_0 = 0 \rightarrow (w^T x - b = 0)$$

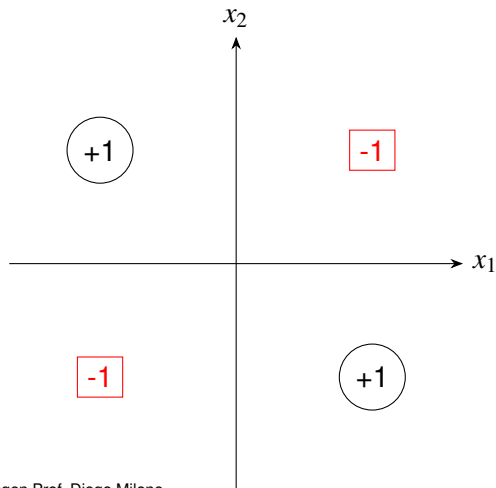


(\*) créditos de la imagen Prof. Diego Milone

(\*) adaptada para el presente curso



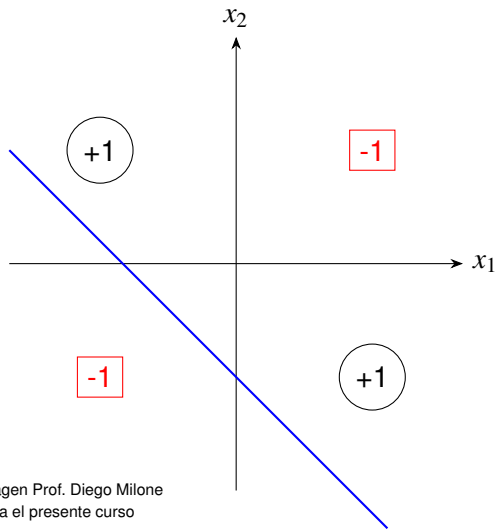
# Poder expresivo del perceptrón (XOR)



(\*) créditos de la imagen Prof. Diego Milone

(\*) adaptada para el presente curso

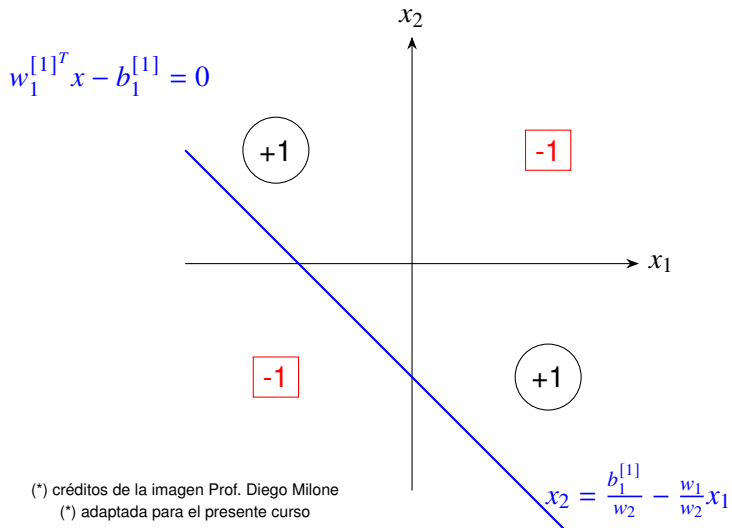
# Poder expresivo del perceptrón (XOR)



(\*) créditos de la imagen Prof. Diego Milone

(\*) adaptada para el presente curso

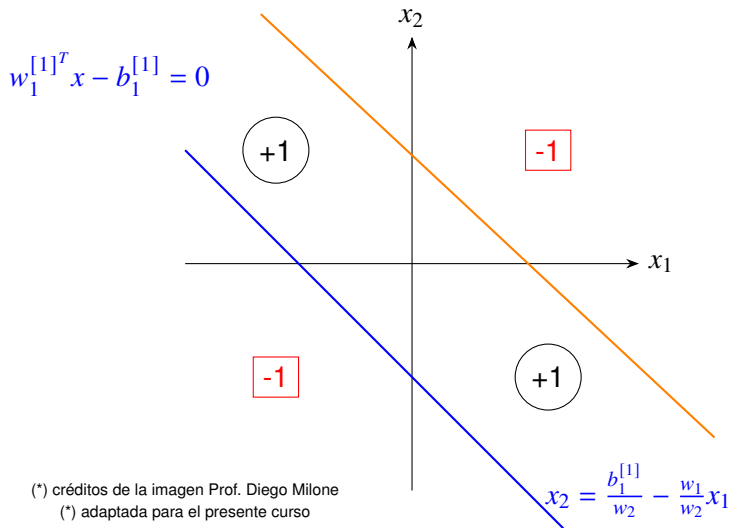
# Poder expresivo del perceptrón (XOR)



(\*) créditos de la imagen Prof. Diego Milone

(\*) adaptada para el presente curso

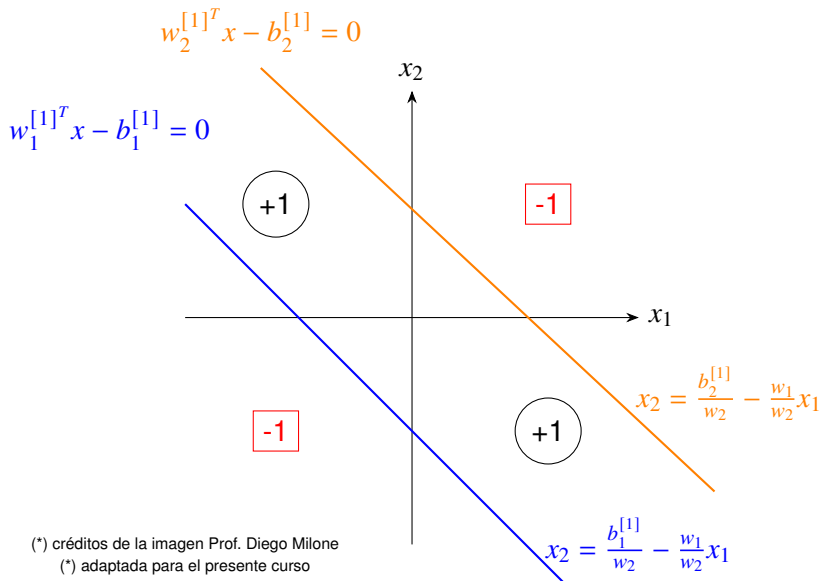
# Poder expresivo del perceptrón (XOR)



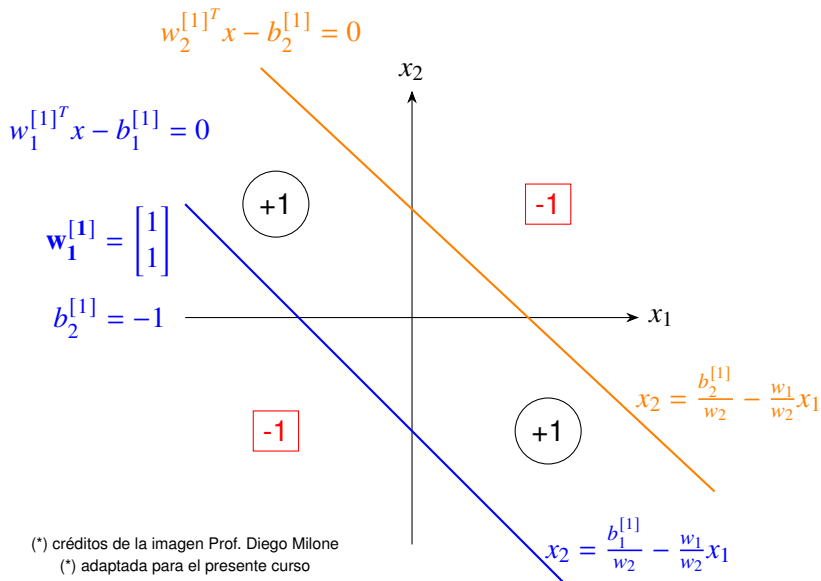
(\*) créditos de la imagen Prof. Diego Milone

(\*) adaptada para el presente curso

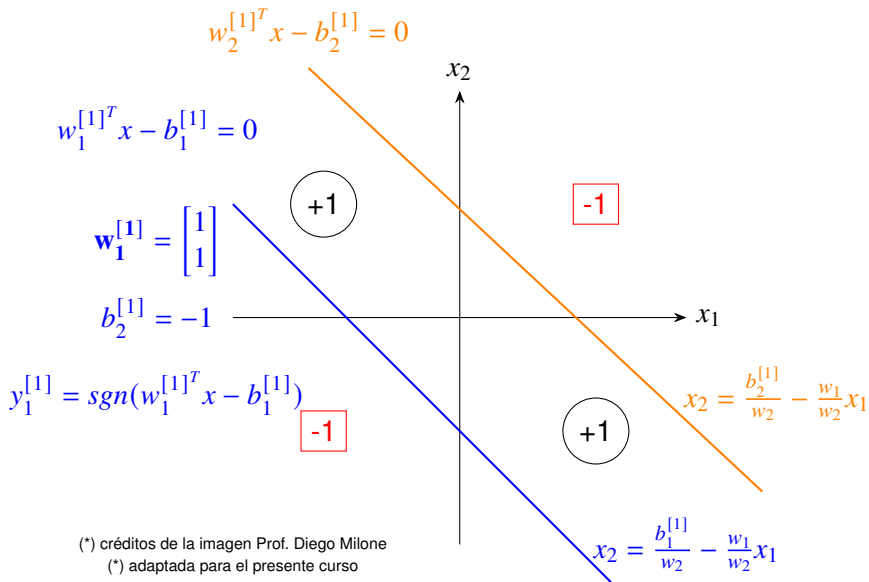
# Poder expresivo del perceptrón (XOR)



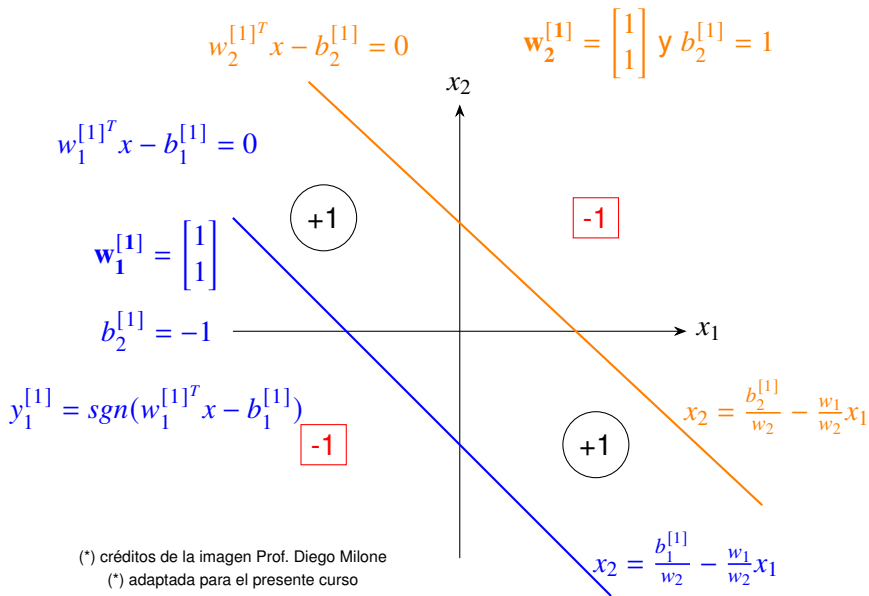
# Poder expresivo del perceptrón (XOR)



# Poder expresivo del perceptrón (XOR)

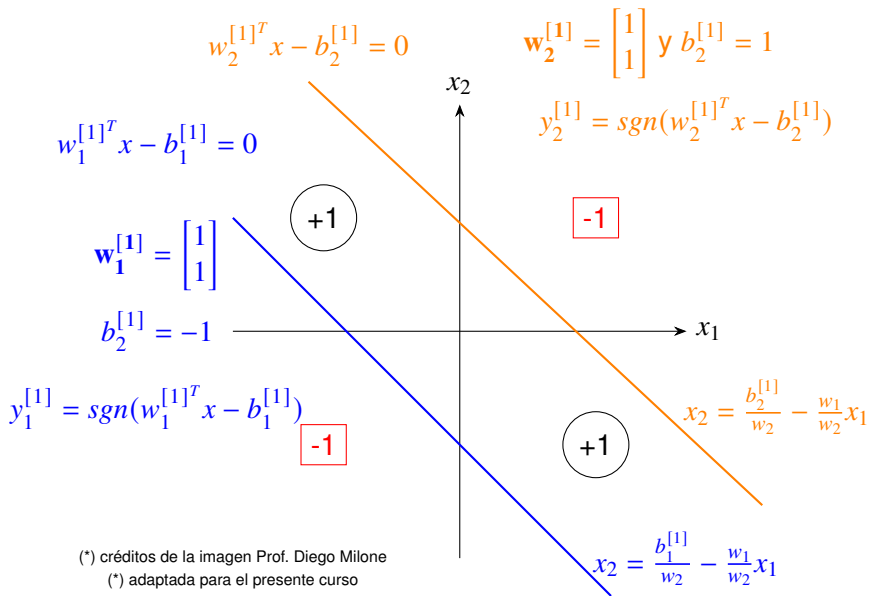


# Poder expresivo del perceptrón (XOR)

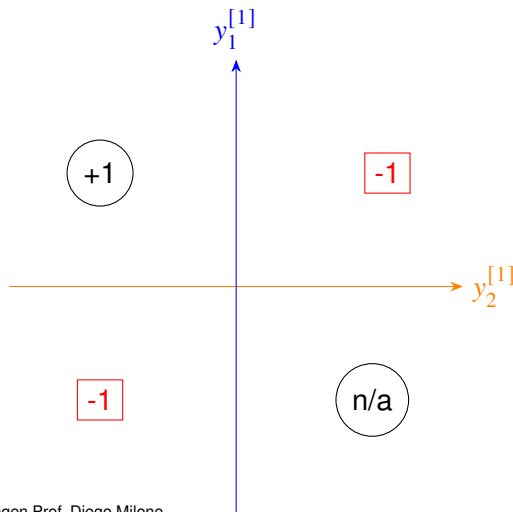




# Poder expresivo del perceptrón (XOR)



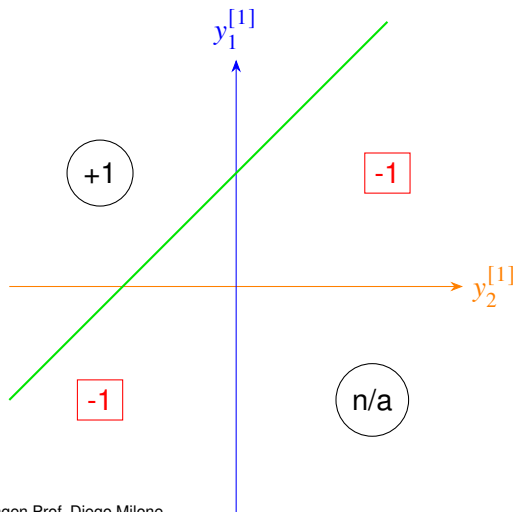
# Poder expresivo del perceptrón (XOR)



(\*) créditos de la imagen Prof. Diego Milone

(\*) adaptada para el presente curso

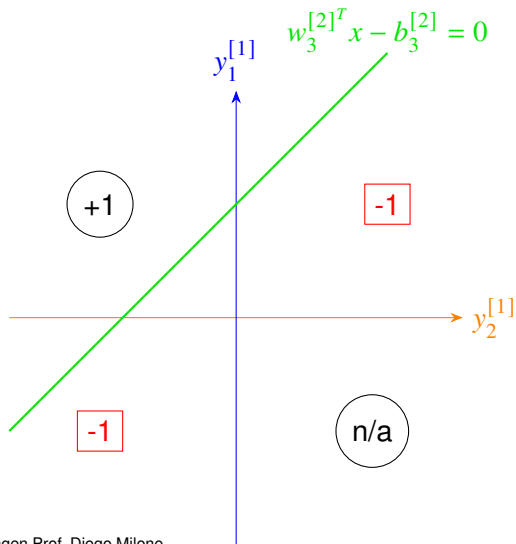
# Poder expresivo del perceptrón (XOR)



(\*) créditos de la imagen Prof. Diego Milone

(\*) adaptada para el presente curso

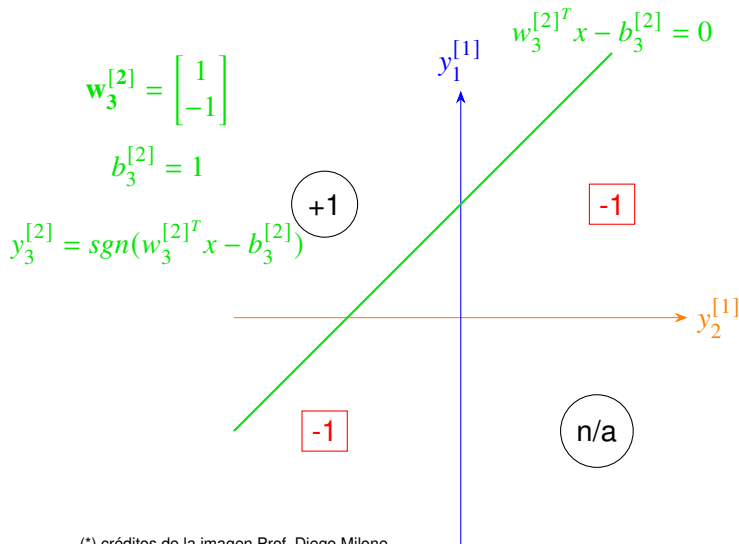
# Poder expresivo del perceptrón (XOR)



(\*) créditos de la imagen Prof. Diego Milone

(\*) adaptada para el presente curso

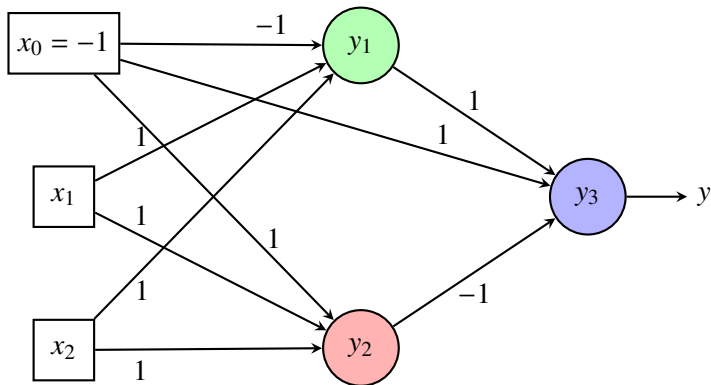
# Poder expresivo del perceptrón (XOR)



(\*) créditos de la imagen Prof. Diego Milone

(\*) adaptada para el presente curso

# Red neuronal XOR



- $y_1$  aprende OR
- $y_2$  aprende AND
- $y_3$  aprende OR - AND

# Tabla de verdad XOR con neuronas OR y AND

$x_1$	$x_2$	$y_1$ (OR)	$y_2$ (AND)	$y_3$ (XOR)
-1	-1	-1	-1	-1
-1	1	1	-1	1
1	-1	1	-1	1
1	1	1	1	-1

■  $y_1 = \text{sgn}(x_1 + x_2 + 1)$

■  $y_2 = \text{sgn}(x_1 + x_2 - 1)$

■  $y_3 = \text{sgn}(y_1 - y_2 - 1)$

# Poder expresivo


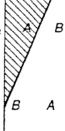
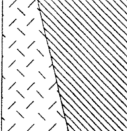
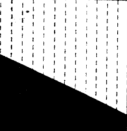


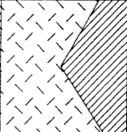

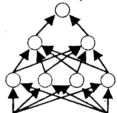

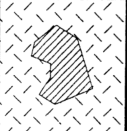

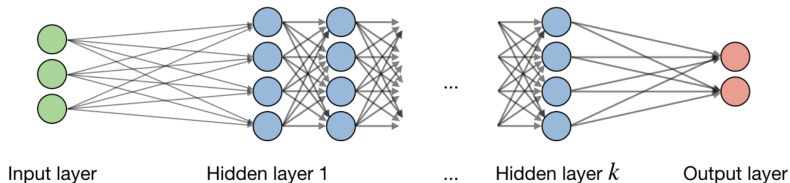
Structure	Types of decision regions	Exclusive OR problem	Classes with meshed regions	Most general region shapes
<p>Single-layer</p> 	Half-plane bounded by a hyper-plane			
<p>Two-layer</p> 	Convex, open, or closed regions			
<p>Three-layer</p> 	Arbitrary regions whose complexity is determined by number of nodes			

Figura 3.1: Capacidad de clasificación según dimension de la red





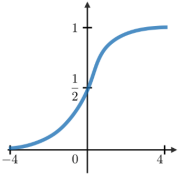
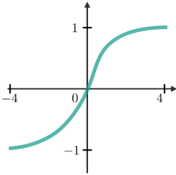
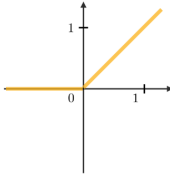
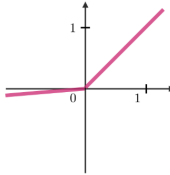
$$z_j^{[i]} = w_j^{[i]T} x - b_j^{[i]}$$

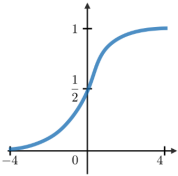
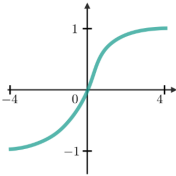
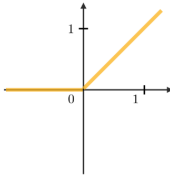
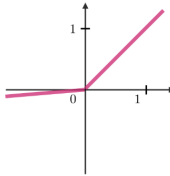
dónde

- $i$  es la  $i^{th}$  capa de la red and  $j$  la  $j^{th}$  neurona oculta de la capa
- $w$ ,  $b$ ,  $z$  los pesos, bias y la salida de la red, respectivamente

(\*) image de [cheatsheet-deep-learning](https://cheatsheet-deep-learning.github.io/)

# Activación / Pérdida

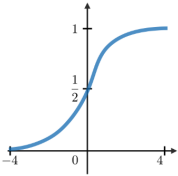
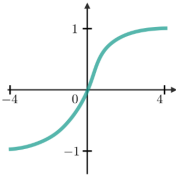
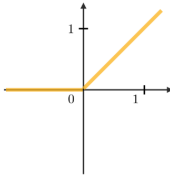
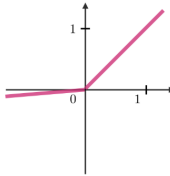
Sigmoid	Tanh	ReLU	Leaky ReLU
$g(z) = \frac{1}{1 + e^{-z}}$	$g(z) = \frac{e^z - e^{-z}}{e^z + e^{-z}}$	$g(z) = \max(0, z)$	$g(z) = \max(\epsilon z, z)$ with $\epsilon \ll 1$
			

Sigmoid	Tanh	ReLU	Leaky ReLU
$g(z) = \frac{1}{1 + e^{-z}}$	$g(z) = \frac{e^z - e^{-z}}{e^z + e^{-z}}$	$g(z) = \max(0, z)$	$g(z) = \max(\epsilon z, z)$ with $\epsilon \ll 1$
			

## Pérdida por entropía cruzada (**clasificación**)

$$L(z, y) = -[y \log(z) + (1 - y) \log(1 - z)]$$

# Activación / Pérdida

Sigmoid	Tanh	ReLU	Leaky ReLU
$g(z) = \frac{1}{1 + e^{-z}}$	$g(z) = \frac{e^z - e^{-z}}{e^z + e^{-z}}$	$g(z) = \max(0, z)$	$g(z) = \max(\epsilon z, z)$ with $\epsilon \ll 1$
			

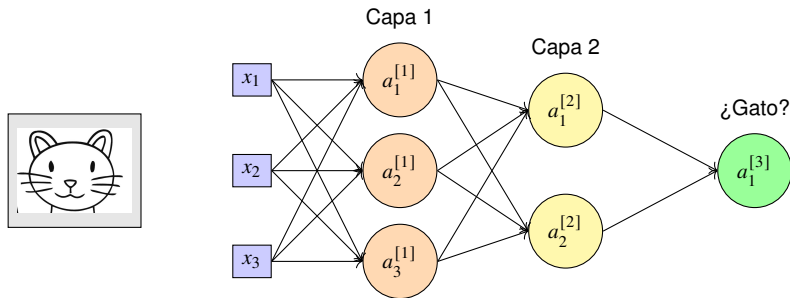
## Pérdida por entropía cruzada (**clasificación**)

$$L(z, y) = -[y \log(z) + (1 - y) \log(1 - z)]$$

## Error cuadrático (**regresión**)

$$J(\theta) = \frac{1}{2m} \sum_{i=1}^m (y^{(i)} - \hat{y}^{(i)})^2$$

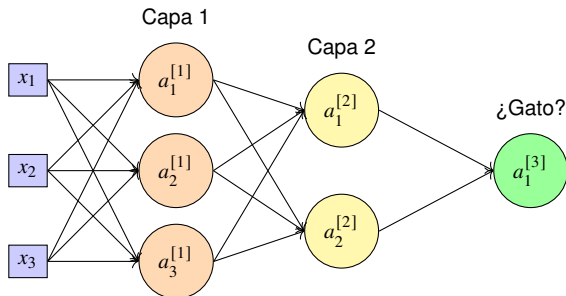
# Generalización del perceptrón



$$\blacksquare a^{[1]} = \sigma(z^{[1]})$$

$$z^{[1]} = w^{[1]}x - b^{[1]}$$

# Generalización del perceptrón



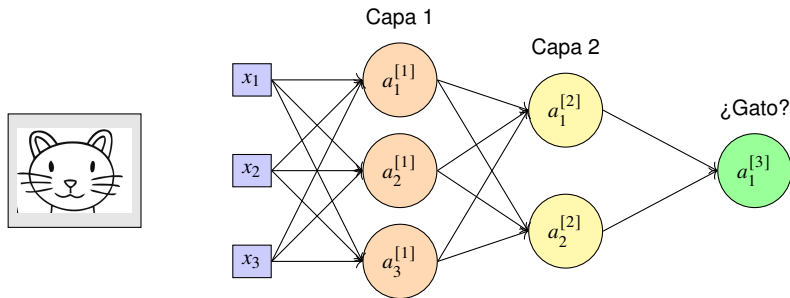
- $a^{[1]} = \sigma(z^{[1]})$

$$z^{[1]} = w^{[1]}x - b^{[1]}$$

- $a^{[2]} = \sigma(z^{[2]})$

$$z^{[2]} = w^{[2]}a^{[1]} - b^{[2]}$$

# Generalización del perceptrón



- $a^{[1]} = \sigma(z^{[1]})$

$$z^{[1]} = w^{[1]}x - b^{[1]}$$

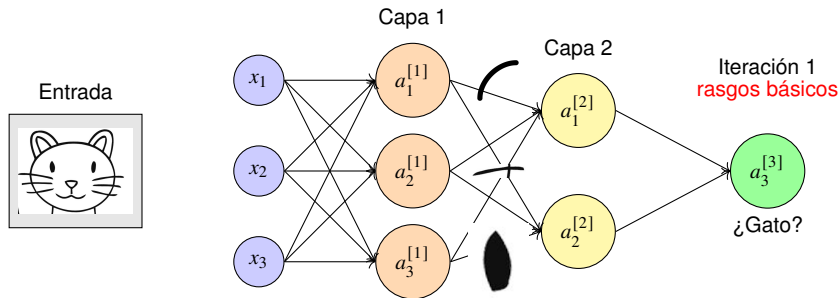
- $a^{[2]} = \sigma(z^{[2]})$

$$z^{[2]} = w^{[2]}a^{[1]} - b^{[2]}$$

- $a^{[3]} = \sigma(z^{[3]})$

$$z^{[3]} = w^{[3]}a^{[2]} - b^{[3]}$$

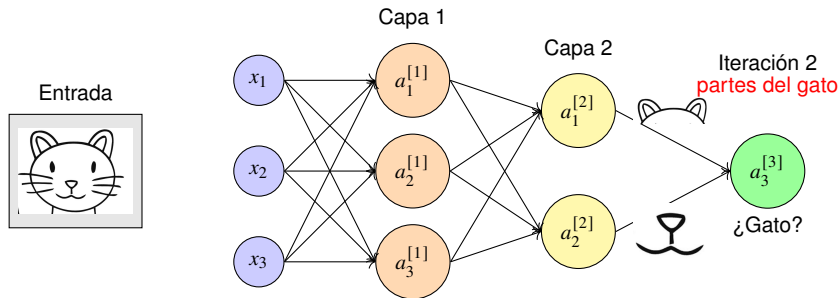
# ¿Cómo aprende una red?



- Ejemplo: para una imagen de 64x64 pixels



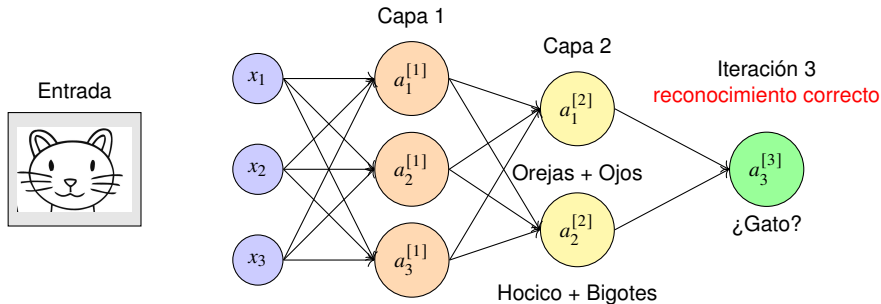
# ¿Cómo aprende una red?



■ Ejemplo: para una imagen de 64x64 pixels

Si escala de grises →  $\mathbf{x}$  tiene 4096 entradas

# ¿Cómo aprende una red?



■ Ejemplo: para una imagen de 64x64 pixels

SI escala de grises  $\rightarrow$   $x$  tiene 4096 entradas

SI imagen RGB (color)  $\rightarrow$  64 x 64 x 3 (Rojo, Verde, Azul)  $\rightarrow$   $x$  tiene 12288 entradas

# Vectorización de una Red Neuronal

$$\underbrace{\begin{bmatrix} z_1 \\ \vdots \\ z_m \end{bmatrix}}_{\mathbf{z} \in \mathbb{R}^{m \times 1}} = \underbrace{\begin{bmatrix} -\mathbf{w}_1^{[1]\top} & - \\ -\mathbf{w}_2^{[1]\top} & - \\ \vdots & \\ -\mathbf{w}_m^{[1]\top} & - \end{bmatrix}}_{\mathbf{W}^{[1]} \in \mathbb{R}^{m \times d}} \underbrace{\begin{bmatrix} x_1 \\ x_2 \\ \vdots \\ x_d \end{bmatrix}}_{\mathbf{x} \in \mathbb{R}^{d \times 1}} + \underbrace{\begin{bmatrix} b_1^{[1]} \\ b_2^{[1]} \\ \vdots \\ b_m^{[1]} \end{bmatrix}}_{\mathbf{b}^{[1]} \in \mathbb{R}^{m \times 1}}$$

$\mathbf{x} \in \mathbb{R}^{d \times 1}$  ( $d$  features)  
 $\mathbf{W}^{[1]} \in \mathbb{R}^{m \times d}$  ( $m$  neuronas)  
 $\mathbf{b}^{[1]} \in \mathbb{R}^{m \times 1}$  (bias)  
 $\mathbf{z} \in \mathbb{R}^{m \times 1}$  (salida)

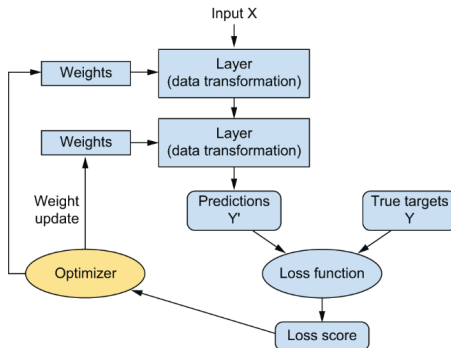
- Imagen:  $28 \times 28 = 784$  píxeles  $\rightarrow d = 784$
- Primera capa oculta con 128 neuronas  $\rightarrow m = 128$

$\mathbf{x} \in \mathbb{R}^{784 \times 1}$   
 $\mathbf{W}^{[1]} \in \mathbb{R}^{128 \times 784}$   
 $\mathbf{b}^{[1]} \in \mathbb{R}^{128 \times 1}$   
 $\mathbf{z} \in \mathbb{R}^{128 \times 1}$

# Backpropagation

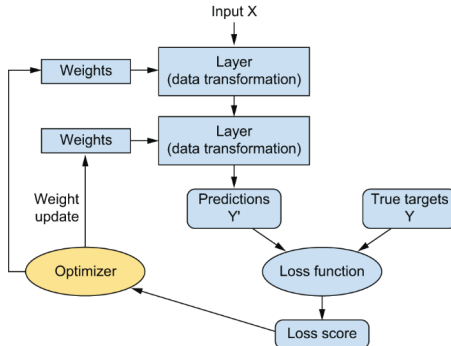
Paul Werbos (1974)

# Motivación



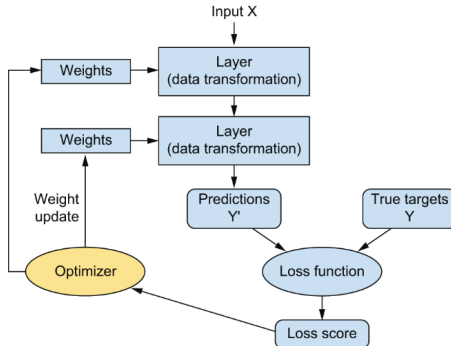
- La **capacidad de clasificación** de las redes neuronales depende de las cantidad de neuronas.

# Motivación



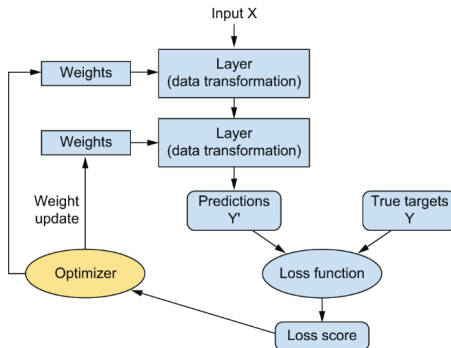
- La **capacidad de clasificación** de las redes neuronales depende de las cantidad de neuronas.
- Es necesario tener un **procedimiento eficiente** para poder entrenar una red

# Motivación



- La **capacidad de clasificación** de las redes neuronales depende de las cantidad de neuronas.
- Es necesario tener un **procedimiento eficiente** para poder entrenar una red

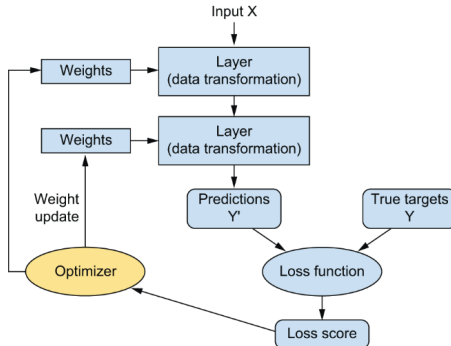
# Motivación



- La **capacidad de clasificación** de las redes neuronales depende de la cantidad de neuronas.
- Es necesario tener un **procedimiento eficiente** para poder entrenar una red
- **idea**: usar **regla de la cadena**, i.e. derivada de funciones compuestas

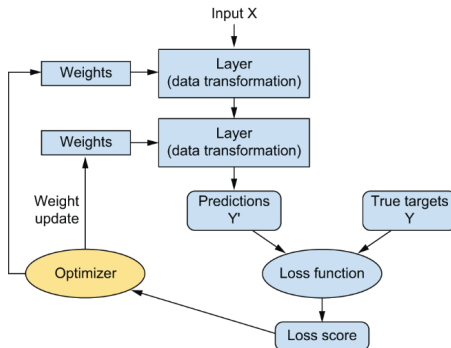


# Motivación



- La **capacidad de clasificación** de las redes neuronales depende de las cantidad de neuronas.
- Es necesario tener un **procedimiento eficiente** para poder entrenar una red
- **idea**: usar **regla de la cadena**, i.e. derivada de funciones compuestas
- Usar *score* como una señal de feedback para ajustar valores de los pesos
- Algoritmo fundamental en aprendizaje profundo

# Motivación



- La **capacidad de clasificación** de las redes neuronales depende de las cantidad de neuronas.
- Es necesario tener un **procedimiento eficiente** para poder entrenar una red
- **idea**: usar **regla de la cadena**, i.e. derivada de funciones compuestas
- Usar *score* como una señal de feedback para ajustar valores de los pesos
- Algoritmo fundamental en aprendizaje profundo
- Escalabilidad y entrenamiento eficiente

- 1 Ejecutar **propagación hacia adelante** (*forward propagation*) para obtener la pérdida correspondiente

- 1 Ejecutar **propagación hacia adelante** (*forward propagation*) para obtener la pérdida correspondiente
- 2 **Back propagar** la pérdida para obtener los gradientes

# Actualización de pesos

- 1 Ejecutar **propagación hacia adelante** (*forward propagation*) para obtener la pérdida correspondiente
- 2 **Back propagar** la pérdida para obtener los gradientes
- 3 Usar los gradientes para actualizar los pesos de la red

# Actualización de pesos

- 1 Ejecutar **propagación hacia adelante** (*forward propagation*) para obtener la pérdida correspondiente
- 2 **Back propagar** la pérdida para obtener los gradientes
- 3 Usar los gradientes para actualizar los pesos de la red

Derivada con respecto a los pesos  $w$ , computado usando la regla de la cadena:

$$\frac{\partial L(z, y)}{\partial w} = \frac{\partial L(z, y)}{\partial a} \times \frac{\partial a}{\partial z} \times \frac{\partial z}{\partial w}$$

# Actualización de pesos

- 1 Ejecutar **propagación hacia adelante** (*forward propagation*) para obtener la pérdida correspondiente
- 2 **Back propagar** la pérdida para obtener los gradientes
- 3 Usar los gradientes para actualizar los pesos de la red

Derivada con respecto a los pesos  $w$ , computado usando la regla de la cadena:

$$\frac{\partial L(z, y)}{\partial w} = \frac{\partial L(z, y)}{\partial a} \times \frac{\partial a}{\partial z} \times \frac{\partial z}{\partial w}$$

Fórmula de actualización de pesos:

$$w \leftarrow w - \alpha \frac{\partial L(z, y)}{\partial w}$$

**Datos iniciales:**

$$\mathbf{x} = \begin{bmatrix} 1 \\ 0 \end{bmatrix}, \quad y = 1, \quad w_1 = 1, \quad w_2 = -1, \quad b = 1$$



**Datos iniciales:**

$$\mathbf{x} = \begin{bmatrix} 1 \\ 0 \end{bmatrix}, \quad y = 1, \quad w_1 = 1, \quad w_2 = -1, \quad b = 1$$

**1. Cálculo hacia adelante**

$$z = w_1 x_1 + w_2 x_2 + b = (1)(1) + (-1)(0) + 1 = 2$$

$$\hat{y} = \sigma(z) = \frac{1}{1 + e^{-z}} = \frac{1}{1 + e^{-2}} \approx 0,8808$$

**Datos iniciales:**

$$\mathbf{x} = \begin{bmatrix} 1 \\ 0 \end{bmatrix}, \quad y = 1, \quad w_1 = 1, \quad w_2 = -1, \quad b = 1$$

**1. Cálculo hacia adelante**

$$z = w_1 x_1 + w_2 x_2 + b = (1)(1) + (-1)(0) + 1 = 2$$

$$\hat{y} = \sigma(z) = \frac{1}{1 + e^{-z}} = \frac{1}{1 + e^{-2}} \approx 0,8808$$

**2. Cálculo de la pérdida**

$$C = \frac{1}{2}(y - \hat{y})^2 = \frac{1}{2}(1 - 0,8808)^2 \approx 0,0072$$

# Actualización de pesos- Error Cuadrático (II)

## 3. Regla de la cadena para actualizar $w_1$

$$\frac{\partial C}{\partial w_1} = \frac{\partial C}{\partial \hat{y}} \cdot \frac{\partial \hat{y}}{\partial z} \cdot \frac{\partial z}{\partial w_1}$$

# Actualización de pesos- Error Cuadrático (II)

## 3. Regla de la cadena para actualizar $w_1$

$$\frac{\partial C}{\partial w_1} = \frac{\partial C}{\partial \hat{y}} \cdot \frac{\partial \hat{y}}{\partial z} \cdot \frac{\partial z}{\partial w_1}$$

dónde:

$$\frac{\partial C}{\partial \hat{y}} = \hat{y} - y = 0,8808 - 1 = -0,1192$$

# Actualización de pesos- Error Cuadrático (II)

## 3. Regla de la cadena para actualizar $w_1$

$$\frac{\partial C}{\partial w_1} = \frac{\partial C}{\partial \hat{y}} \cdot \frac{\partial \hat{y}}{\partial z} \cdot \frac{\partial z}{\partial w_1}$$

dónde:

$$\frac{\partial C}{\partial \hat{y}} = \hat{y} - y = 0,8808 - 1 = -0,1192$$

$$\frac{\partial \hat{y}}{\partial z} = \hat{y}(1 - \hat{y}) = 0,8808 \times (1 - 0,8808) = 0,1049$$

# Actualización de pesos- Error Cuadrático (II)

## 3. Regla de la cadena para actualizar $w_1$

$$\frac{\partial C}{\partial w_1} = \frac{\partial C}{\partial \hat{y}} \cdot \frac{\partial \hat{y}}{\partial z} \cdot \frac{\partial z}{\partial w_1}$$

dónde:

$$\frac{\partial C}{\partial \hat{y}} = \hat{y} - y = 0,8808 - 1 = -0,1192$$

$$\frac{\partial \hat{y}}{\partial z} = \hat{y}(1 - \hat{y}) = 0,8808 \times (1 - 0,8808) = 0,1049$$

$$\frac{\partial z}{\partial w_1} = x_1 = 1$$

# Actualización de pesos- Error Cuadrático (II)

## 3. Regla de la cadena para actualizar $w_1$

$$\frac{\partial C}{\partial w_1} = \frac{\partial C}{\partial \hat{y}} \cdot \frac{\partial \hat{y}}{\partial z} \cdot \frac{\partial z}{\partial w_1}$$

dónde:

$$\frac{\partial C}{\partial \hat{y}} = \hat{y} - y = 0,8808 - 1 = -0,1192$$

$$\frac{\partial \hat{y}}{\partial z} = \hat{y}(1 - \hat{y}) = 0,8808 \times (1 - 0,8808) = 0,1049$$

$$\frac{\partial z}{\partial w_1} = x_1 = 1$$

$$\Rightarrow \frac{\partial C}{\partial w_1} = (-0,1192)(0,1049)(1) = -0,0125$$

# Actualización de pesos- Error Cuadrático (II)

## 3. Regla de la cadena para actualizar $w_1$

$$\frac{\partial C}{\partial w_1} = \frac{\partial C}{\partial \hat{y}} \cdot \frac{\partial \hat{y}}{\partial z} \cdot \frac{\partial z}{\partial w_1}$$

dónde:

$$\frac{\partial C}{\partial \hat{y}} = \hat{y} - y = 0,8808 - 1 = -0,1192$$

$$\frac{\partial \hat{y}}{\partial z} = \hat{y}(1 - \hat{y}) = 0,8808 \times (1 - 0,8808) = 0,1049$$

$$\frac{\partial z}{\partial w_1} = x_1 = 1$$

$$\Rightarrow \frac{\partial C}{\partial w_1} = (-0,1192)(0,1049)(1) = -0,0125$$

## 4. Actualización del peso ( $\eta$ tasa de aprendizaje)

$$w_1 \leftarrow w_1 - \eta \frac{\partial C}{\partial w_1} = 1 - 0,1 \times (-0,0125) = 1 + 0,00125 = 1,00125$$



**Datos del ejemplo:**

$$\mathbf{x} = \begin{bmatrix} 1 \\ 0 \end{bmatrix}, \quad y = 1, \quad w_1 = 1, \quad w_2 = -1, \quad b = 1$$

**Datos del ejemplo:**

$$\mathbf{x} = \begin{bmatrix} 1 \\ 0 \end{bmatrix}, \quad y = 1, \quad w_1 = 1, \quad w_2 = -1, \quad b = 1$$

1. Cálculo hacia adelante

$$z = w_1 x_1 + w_2 x_2 + b = (1)(1) + (-1)(0) + 1 = 2$$

$$\hat{y} = \sigma(z) = \frac{1}{1 + e^{-2}} \approx 0,8808$$

## Datos del ejemplo:

$$\mathbf{x} = \begin{bmatrix} 1 \\ 0 \end{bmatrix}, \quad y = 1, \quad w_1 = 1, \quad w_2 = -1, \quad b = 1$$

### 1. Cálculo hacia adelante

$$z = w_1 x_1 + w_2 x_2 + b = (1)(1) + (-1)(0) + 1 = 2$$

$$\hat{y} = \sigma(z) = \frac{1}{1 + e^{-2}} \approx 0,8808$$

### 2. Función de pérdida (entropía cruzada)

$$C = -[y \log(\hat{y}) + (1 - y) \log(1 - \hat{y})]$$

$$C = -\log(0,8808) \approx 0,1269$$

## 3. Gradiente con entropía cruzada + sigmoide

$$\frac{\partial C}{\partial w_1} = (\hat{y} - y) \cdot x_1$$

## 3. Gradiente con entropía cruzada + sigmoide

$$\frac{\partial C}{\partial w_1} = (\hat{y} - y) \cdot x_1$$

$$\frac{\partial L}{\partial \hat{y}} = -\frac{y}{\hat{y}} + \frac{1-y}{1-\hat{y}}$$

$$\frac{\partial \hat{y}}{\partial z} = \sigma'(z) = \hat{y}(1 - \hat{y})$$

$$\frac{\partial z}{\partial w} = x \quad (\text{si } w \text{ es } w_1, \text{ entonces } x = x_1; \text{ para } b \text{ es } 1)$$

$$\frac{\partial L}{\partial w} = \left( -\frac{y}{\hat{y}} + \frac{1-y}{1-\hat{y}} \right) \cdot \hat{y}(1 - \hat{y}) \cdot x = (-y(1 - \hat{y}) + (1 - y)\hat{y}) x = (\hat{y} - y)x$$

$$\frac{\partial C}{\partial w_1} = (\hat{y} - y) \cdot x_1$$

Sustituimos valores:

$$\hat{y} - y = 0,8808 - 1 = -0,1192, \quad x_1 = 1$$

$$\Rightarrow \frac{\partial C}{\partial w_1} = -0,1192 \cdot 1 = -0,1192$$

$$\frac{\partial C}{\partial w_1} = (\hat{y} - y) \cdot x_1$$

Sustituimos valores:

$$\hat{y} - y = 0,8808 - 1 = -0,1192, \quad x_1 = 1$$

$$\Rightarrow \frac{\partial C}{\partial w_1} = -0,1192 \cdot 1 = -0,1192$$

## 4. Actualización del peso

$$w_1 \leftarrow w_1 - \eta \cdot \frac{\partial C}{\partial w_1} = 1 - 0,1 \cdot (-0,1192) = 1 + 0,01192 = 1,01192$$

# Capacidad de Generalización



Parece todo lindo pero ...

**Overfitting!**

## Overfitting!

### Solución por defecto

- Aumentar el dataset de entrenamiento

## Overfitting!

### Solución por defecto

- Aumentar el dataset de entrenamiento
- Sin embargo, es costoso y *time-consuming*

- Elección de una apropiada función de pérdida
- Mini-batch
- Otras funciones de activación
- Tasas de aprendizaje adaptativas
- Momentum
- Early Stopping
- Weight Decay
- Dropout

# Regularización y otras yerbas (II)

Elección de una apropiada función de pérdida

Entropía cruzada vs. error cuadrático

# Regularización y otras yerbas (II)

Elección de una apropiada función de pérdida

Entropía cruzada vs. error cuadrático

Mini-batch

- El gradiente *batch* usa todos los ejemplos antes de actualizar los pesos → **muy lento!**
- Actualizar los pesos usando batches de  $m$  ejemplos y entrenar hasta que todos los mini-batches son usados → **mejor performance!**

# Regularización y otras yerbas (II)

## Elección de una apropiada función de pérdida

Entropía cruzada vs. error cuadrático

## Mini-batch

- El gradiente *batch* usa todos los ejemplos antes de actualizar los pesos → **muy lento!**
- Actualizar los pesos usando batches de  $m$  ejemplos y entrenar hasta que todos los mini-batches son usados → **mejor performance!**

## Otras funciones de activación

- El **desvanecimiento del gradiente** (*vanishing gradient problem*) ocurre cuando el gradiente se hace demasiado pequeño al retropropagarse, entonces las capas iniciales no aprenden
- Sucede cuando activación es **sigmoide** o **tanh**
- Otras funciones como **ReLU** permiten atacar este problema

# Regularización y otras yerbas (III)

## Tasas de aprendizaje adaptativas ( $\eta$ )

- Si  $\eta$  muy grande  $\rightarrow$  pérdida total no decrece luego de actualización
- Si  $\eta$  muy chica  $\rightarrow$  el aprendizaje es muy lento
- **alternativa I:** reducir  $\eta$  luego de algunas épocas
- **alternativa II:** Adagrad, RMSprop, Adadelta, Adam, AdaSecant, Nadam



# Regularización y otras yerbas (III)

## Tasas de aprendizaje adaptativas ( $\eta$ )

- Si  $\eta$  muy grande  $\rightarrow$  pérdida total no decrece luego de actualización
- Si  $\eta$  muy chica  $\rightarrow$  el aprendizaje es muy lento
- **alternativa I:** reducir  $\eta$  luego de algunas épocas
- **alternativa II:** Adagrad, RMSprop, Adadelta, Adam, AdaSecant, Nadam

## Momentum

- **NO garantiza mínimos globales**, pero es una optimización para SGD (*Stochastic Gradient Descent*)
- **intuición:** acumular una suma decreciente de los gradientes previos para acelerar el aprendizaje

# Regularización y otras yerbas (III)

## Tasas de aprendizaje adaptativas ( $\eta$ )

- Si  $\eta$  muy grande  $\rightarrow$  pérdida total no decrece luego de actualización
- Si  $\eta$  muy chica  $\rightarrow$  el aprendizaje es muy lento
- **alternativa I:** reducir  $\eta$  luego de algunas épocas
- **alternativa II:** Adagrad, RMSprop, Adadelata, Adam, AdaSecant, Nadam

## Momentum

- **NO garantiza mínimos globales**, pero es una optimización para SGD (*Stochastic Gradient Descent*)
- **intuición:** acumular una suma decreciente de los gradientes previos para acelerar el aprendizaje

## Early Stopping

- **Detener el entrenamiento** cuando la performance mejora con respecto al conjunto de entrenamiento pero no sobre el conjunto de validación  $\rightarrow$  vemos signos de *overfitting*

## Dropout

- **Deactivar**, de manera aleatoria, algunas neuronas durante entrenamiento para forzar a otras a aprender
- El dropout es desactivado durante el testing para usar la red completa

## Dropout

- **Deactivar**, de manera aleatoria, algunas neuronas durante entrenamiento para forzar a otras a aprender
- El dropout es desactivado durante el testing para usar la red completa

## Weight Decay

- También llamada **L2**
- **intuición:** *podar* conexiones entre neuronas
- Penaliza proporcionalmente al cuadrado de la magnitud de los pesos

$$L(w, b) = L_0 + \lambda ||\mathbf{w}||^2$$

# Variantes de las Redes Neuronales

## Redes Neuronales Convolucionales (CNN) [1]

## Redes Neuronales Convolucionales (CNN) [1]

- **aplicación** → Son ampliamente usadas para procesamiento de imágenes

## Redes Neuronales Convolucionales (CNN) [1]

- **aplicación** → Son ampliamente usadas para procesamiento de imágenes
- **intuición I** → cuando se procesa una imagen con una ANN, la primera capa *fully* conectada es muy compleja → conecta cada pixel a cada neurona



## Redes Neuronales Convolucionales (CNN) [1]

- **aplicación** → Son ampliamente usadas para procesamiento de imágenes
- **intuición I** → cuando se procesa una imagen con una ANN, la primera capa *fully* conectada es muy compleja → conecta cada pixel a cada neurona
- **intuición II** → cada neurona oculta será conectada a solo una pequeña región de la imagen

[1] [Introducing Convolutional Networks](#)

## Redes Neuronales Convolucionales (CNN) [1]

- **aplicación** → Son ampliamente usadas para procesamiento de imágenes
- **intuición I** → cuando se procesa una imagen con una ANN, la primera capa *fully* conectada es muy compleja → conecta cada pixel a cada neurona
- **intuición II** → cada neurona oculta será conectada a solo una pequeña región de la imagen

[1] [Introducing Convolutional Networks](#)

## Redes Neuronales Recurrentes (RNN) [2]

## Redes Neuronales Convolucionales (CNN) [1]

- **aplicación** → Son ampliamente usadas para procesamiento de imágenes
- **intuición I** → cuando se procesa una imagen con una ANN, la primera capa *fully* conectada es muy compleja → conecta cada pixel a cada neurona
- **intuición II** → cada neurona oculta será conectada a solo una pequeña región de la imagen

[1] [Introducing Convolutional Networks](#)

## Redes Neuronales Recurrentes (RNN) [2]

- **aplicación** → procesamiento de datos secuenciales (texto, sonido, ...)

## Redes Neuronales Convolucionales (CNN) [1]

- **aplicación** → Son ampliamente usadas para procesamiento de imágenes
- **intuición I** → cuando se procesa una imagen con una ANN, la primera capa *fully* conectada es muy compleja → conecta cada pixel a cada neurona
- **intuición II** → cada neurona oculta será conectada a solo una pequeña región de la imagen

[1] [Introducing Convolutional Networks](#)

## Redes Neuronales Recurrentes (RNN) [2]

- **aplicación** → procesamiento de datos secuenciales (texto, sonido, ...)
- **intuición I** → son redes con **memoria**, es decir que la salida depende el estado actual y del anterior

## Redes Neuronales Convolucionales (CNN) [1]

- **aplicación** → Son ampliamente usadas para procesamiento de imágenes
- **intuición I** → cuando se procesa una imagen con una ANN, la primera capa *fully* conectada es muy compleja → conecta cada pixel a cada neurona
- **intuición II** → cada neurona oculta será conectada a solo una pequeña región de la imagen

[1] [Introducing Convolutional Networks](#)

## Redes Neuronales Recurrentes (RNN) [2]

- **aplicación** → procesamiento de datos secuenciales (texto, sonido, ...)
- **intuición I** → son redes con **memoria**, es decir que la salida depende el estado actual y del anterior
- **intuición II** → modela la probabilidad de que una palabra de entrada pertenezca a un slot. “llegaría a **Neuquén** el **2 de Noviembre**”

[2] [Recurrent Neural Networks](#)

# Conclusiones

- Actualmente, el *deep learning* tiene gran dominancia en el campo de la IA:  
“IA y *deep learning* se consideran equivalentes”

# Consideraciones finales

- Actualmente, el *deep learning* tiene gran dominancia en el campo de la IA:  
*“IA y deep learning se consideran equivalentes”*
- Efectivas para resolver una **gran cantidad de problemas**



# Consideraciones finales

- Actualmente, el *deep learning* tiene gran dominancia en el campo de la IA:  
*“IA y deep learning se consideran equivalentes”*
- Efectivas para resolver una **gran cantidad de problemas**
- **Pero..**

# Consideraciones finales

- Actualmente, el *deep learning* tiene gran dominancia en el campo de la IA:  
*“IA y deep learning se consideran equivalentes”*
- Efectivas para resolver una **gran cantidad de problemas**
- **Pero..**
- Requieren de mucho **computo** paralelo para entrenamiento

# Consideraciones finales

- Actualmente, el *deep learning* tiene gran dominancia en el campo de la IA:  
*“IA y deep learning se consideran equivalentes”*
- Efectivas para resolver una **gran cantidad de problemas**
- **Pero..**
- Requieren de mucho **computo** paralelo para entrenamiento
  - **GPUs se vuelven esenciales**

# Consideraciones finales

- Actualmente, el *deep learning* tiene gran dominancia en el campo de la IA:  
*“IA y deep learning se consideran equivalentes”*
- Efectivas para resolver una **gran cantidad de problemas**
- **Pero..**
- Requieren de mucho **computo** paralelo para entrenamiento
  - **GPUs se vuelven esenciales**
- Requieren **gran cantidad de datos**

# Consideraciones finales

- Actualmente, el *deep learning* tiene gran dominancia en el campo de la IA:  
*“IA y deep learning se consideran equivalentes”*
- Efectivas para resolver una **gran cantidad de problemas**
- **Pero..**
- Requieren de mucho **computo** paralelo para entrenamiento
  - **GPUs se vuelven esenciales**
- Requieren **gran cantidad de datos**
  - **Tenemos la Big Data!**

# Consideraciones finales

- Actualmente, el *deep learning* tiene gran dominancia en el campo de la IA:  
*“IA y deep learning se consideran equivalentes”*
- Efectivas para resolver una **gran cantidad de problemas**
- **Pero..**
- Requieren de mucho **computo** paralelo para entrenamiento
  - **GPUs se vuelven esenciales**
- Requieren **gran cantidad de datos**
  - **Tenemos la Big Data!**
  - **Supervisado require intervención de expertos!**

# Consideraciones finales

- Actualmente, el *deep learning* tiene gran dominancia en el campo de la IA:  
*“IA y deep learning se consideran equivalentes”*
- Efectivas para resolver una **gran cantidad de problemas**
- **Pero..**
- Requieren de mucho **computo** paralelo para entrenamiento
  - **GPUs se vuelven esenciales**
- Requieren **gran cantidad de datos**
  - **Tenemos la Big Data!**
  - **Supervisado require intervención de expertos!**
- Frameworks actuales para construir redes
  - TensorFlow
  - Keras y Deep-dive Keras
  - PyTorch

# Bibliografía y material de referencia



Harrington, Peter. Machine learning in action. *Simon and Schuster*, 2012.



Alpaydin, Ethem. Introduction to machine learning. 3era Edición *MIT Press*, 2020.



Brett Lantz. Machine Learning with R. *Packt Publishing*, 1997.



Tom M. Mitchell. Machine Learning. *WCB McGraw-Hill*, 1997.



Witten I., Frank E., Hall, M., Pal C.. Data Mining: Practical Machine Learning Tools and Techniques. 4th Edition *WMorgan Kaufmann. Elsevier*, 2017.



Michael A. Nielsen. Neural Networks and Deep Learning. 4th Edition *Determination Press*, 2015.

<http://neuralnetworksanddeeplearning.com>



# Bibliografía y material de referencia



Afshine Amidi, Shervine Amidi. CS 229 — Machine Learning.  
<https://stanford.edu/~shervine/teaching/cs-229/>



Andrew Ng. Stanford CS229 - Machine Learning Course.  
<https://www.youtube.com/playlist?list=PLoROMvovdv4rMiGQp3WXShtMGgzqpFVfbU>



Andrew Ng. Deep Learning AI.  
<https://www.deeplearning.ai/resources/>



Kilian Weinberger. Machine Learning for Intelligent Systems. <https://www.cs.cornell.edu/courses/cs4780/2018fa/syllabus/>



Corinna Cortes and Vladimir Vapnik. Support-vector networks. Machine Learning. Springer. 1995



Paul J. Werbos, *Beyond Regression: New Tools for Prediction and Analysis in the Behavioral Sciences*, PhD thesis, Harvard University, 1974.



David E. Rumelhart, Geoffrey E. Hinton, and Ronald J. Williams, *Learning representations by back-propagating errors*, Nature, vol. 323, pp. 533–536, 1986.  
DOI: 10.1038/323533a0