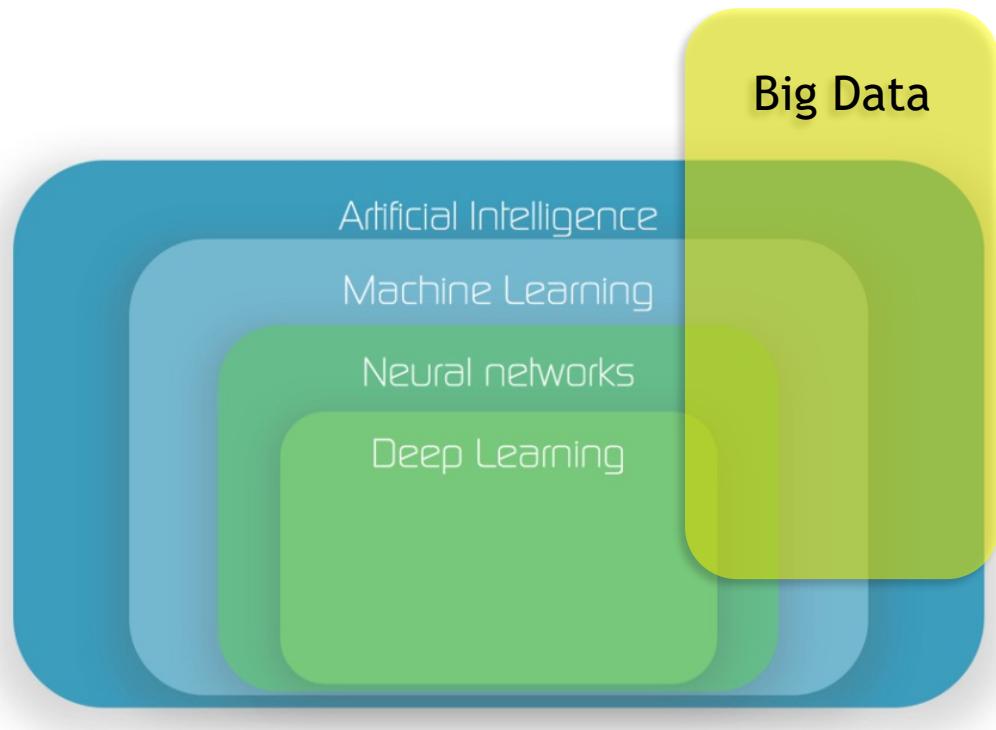


Redes Neuronales

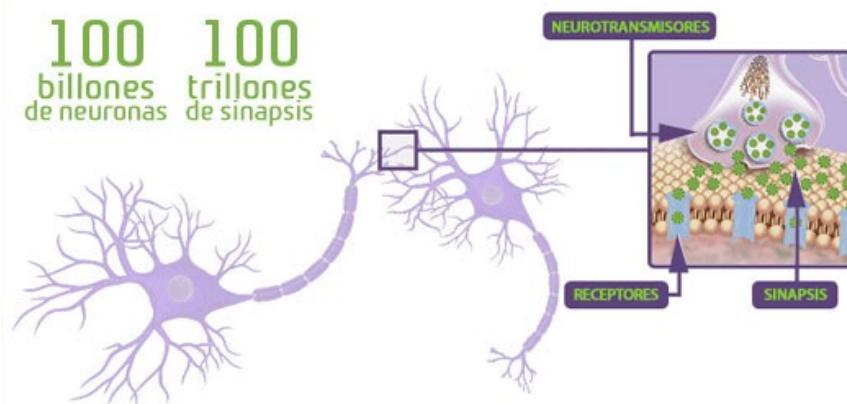
Conceptos básicos

- **Big data:** almacenamiento y procesado de grandes volúmenes de datos
- **Inteligencia artificial:** Resolver problemas mediante máquinas
- **Machine Learning:** Conjunto de algoritmos capaces de identificar y aprender patrones en datos



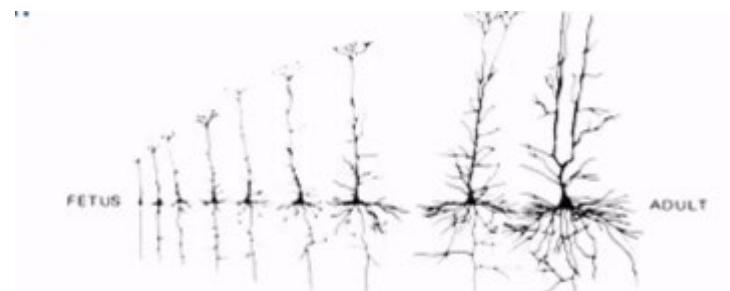
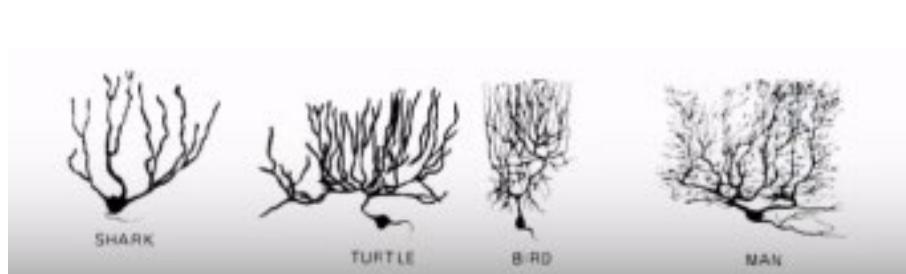
Redes Neuronales

- Algoritmos que simulan la manera en la que los organismos inteligentes procesan la información, a través de **neuronas**
- Una neurona recibe muchas entradas y genera una única salida, que también constituirá una de las muchas entradas de otras neuronas
- La comunicación entre neuronas se realiza a través de una conexión llamada **sinapsis**, donde se encuentra la memoria



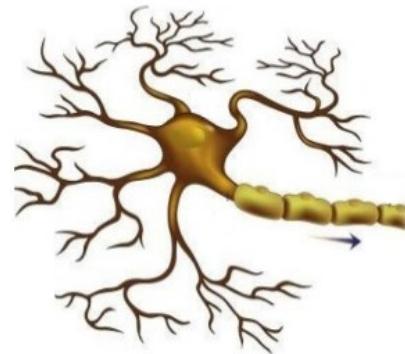
Redes Neuronales

- Entrenar la red neuronal significa alterar sus sinapsis, de forma que aprenda lo que queremos enseñarle
- Cuando vivimos alguna experiencia, se almacena en nuestro cerebro creando nuevas conexiones neuronales, deshaciendo otras y alterando la ponderación de cada una de esas conexiones
- Permiten realizar regresiones y clasificaciones

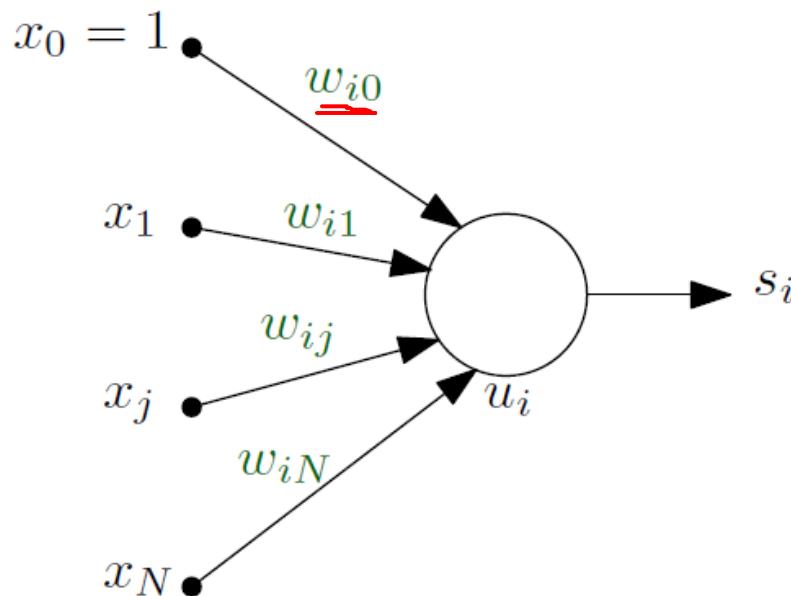


Redes Neuronales

- Modelo de neurona:



Neurona i



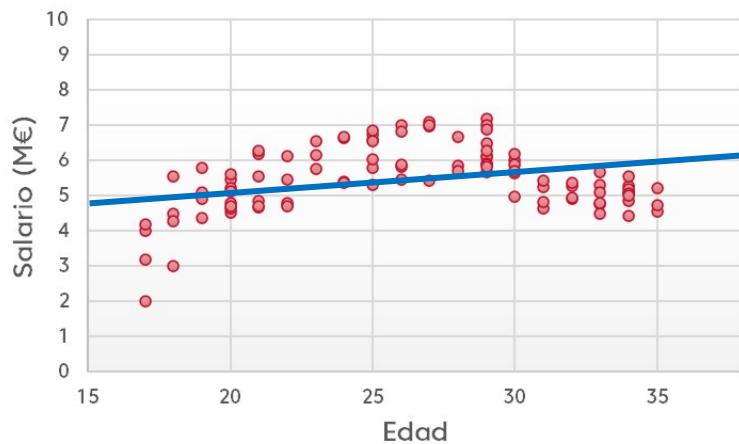
$$u_i = \sum_{j=1}^N w_{ij}x_j + w_{i0} = \mathbf{w}_i^T \mathbf{x} + w_{i0}$$

$$s_i = s(u_i)$$



Redes Neuronales

- Ejemplo: Queremos predecir el salario de los jugadores en función de su edad, utilizando datos históricos



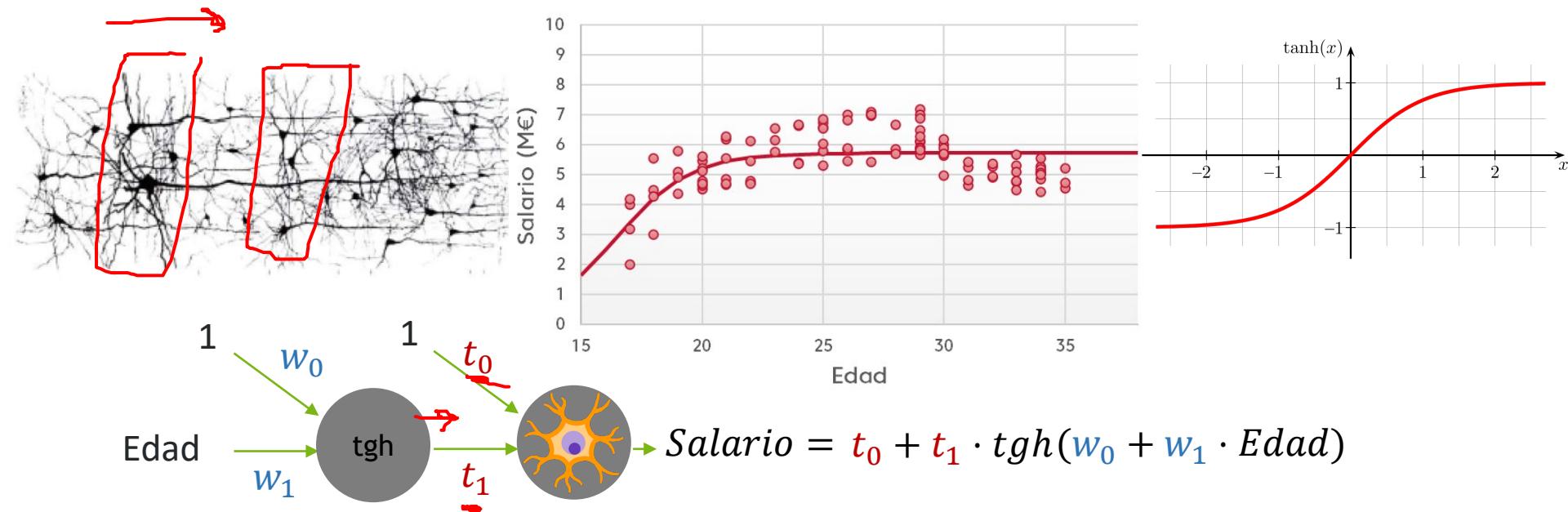
- Se buscan los valores de las sinapsis w_0 y w_1 que minimizan el error

$$Salario = 4.51 + 0.036 \cdot Edad$$



Redes Neuronales

- Podemos mejorar el desempeño incluyendo más capas de neuronas y funciones de activación no lineales (por ejemplo, tangentes hiperbólicas)

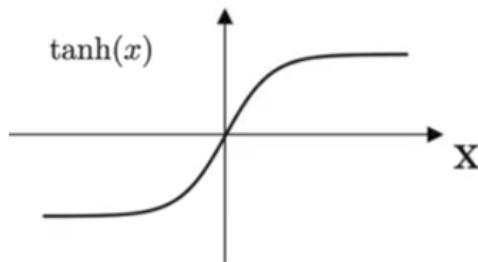


- Se buscan los valores de las sinapsis w_0 w_1 t_0 y t_1 que minimizan el error

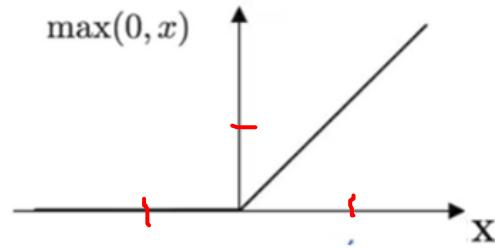
Redes Neuronales

- Función de activación

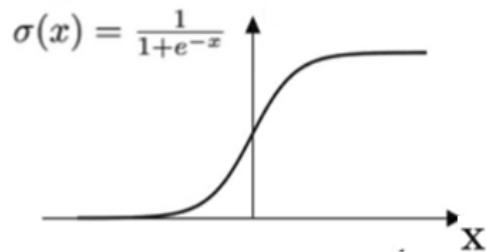
Hyper Tangent Function



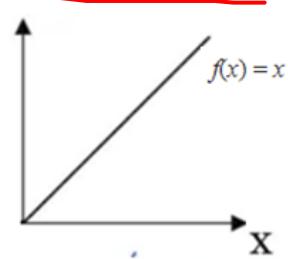
ReLU Function



Sigmoid Function

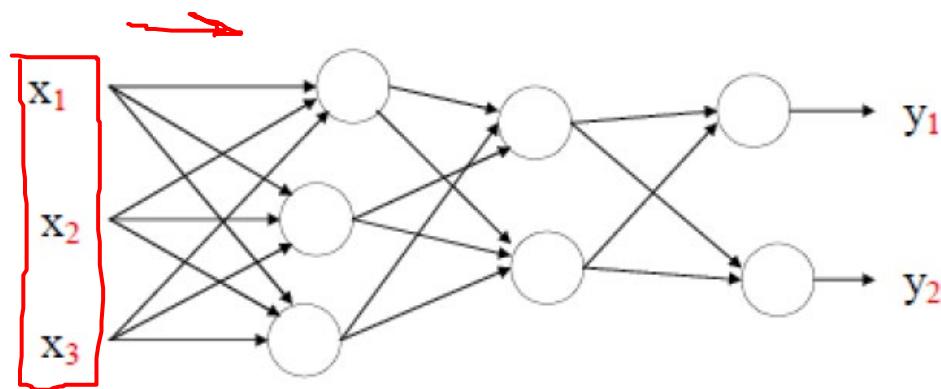
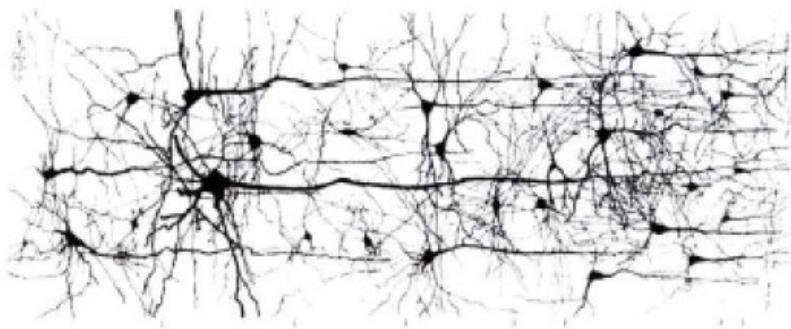


Identity Function



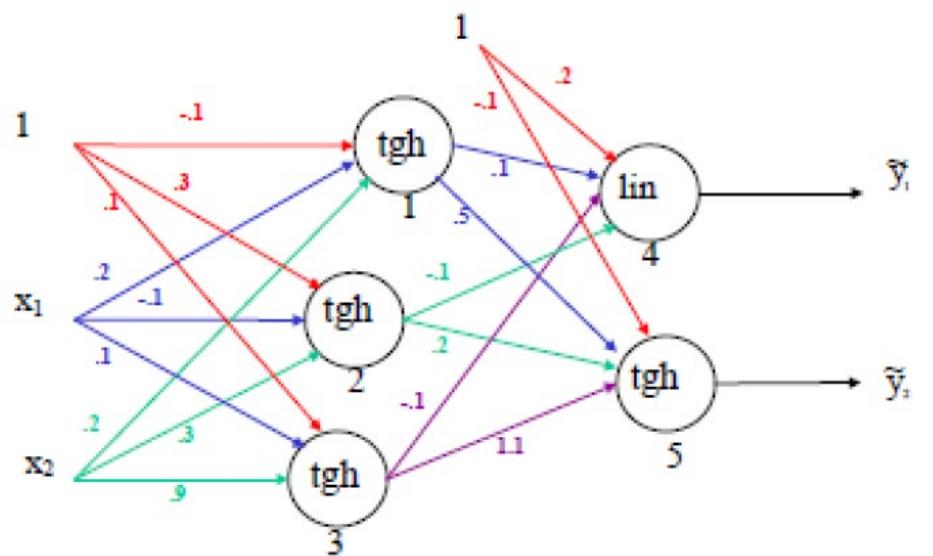
Redes Neuronales

- Arquitectura de la red (*feedforward*)



Redes Neuronales

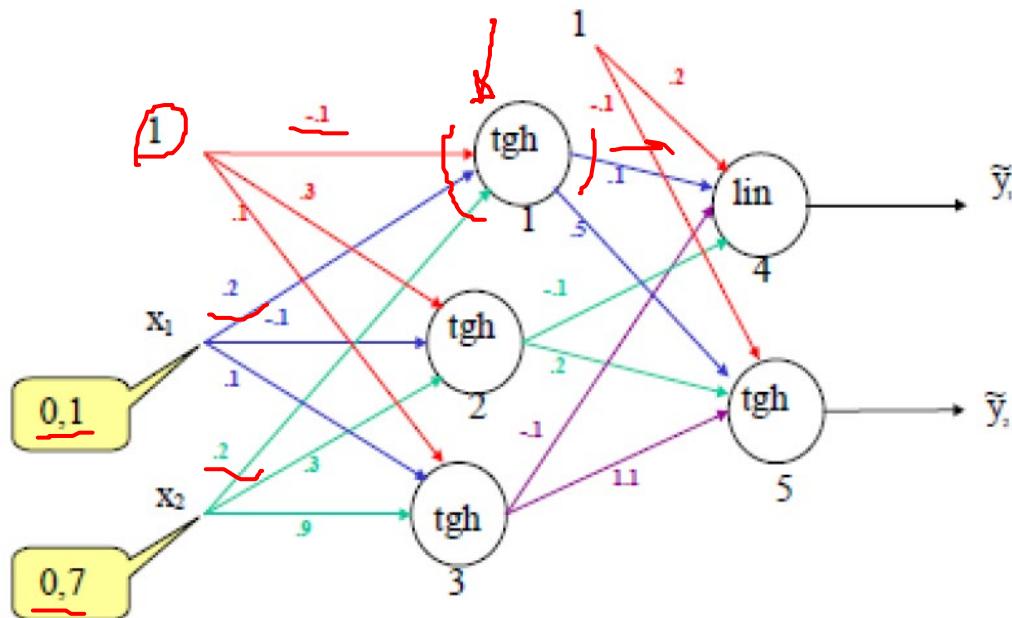
- Ejemplo:



$$\underline{x} = \begin{bmatrix} 0, 1 \\ 0, 7 \end{bmatrix} \quad \tilde{\mathbf{y}} = ?$$

Redes Neuronales

- Ejemplo:

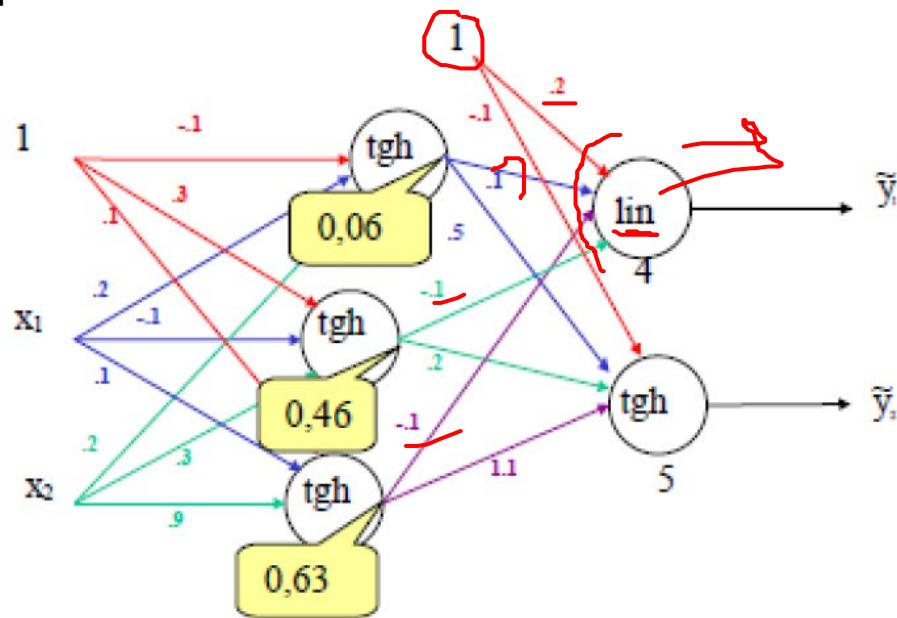


$$u_1 = \underline{-0,1} + (\underline{0,2})(\underline{0,1}) + (\underline{0,2})(\underline{0,7}) = \underline{0,06}$$

$$v_1 = \text{tgh}(0,06) = 0,06$$

Redes Neuronales

- Ejemplo:

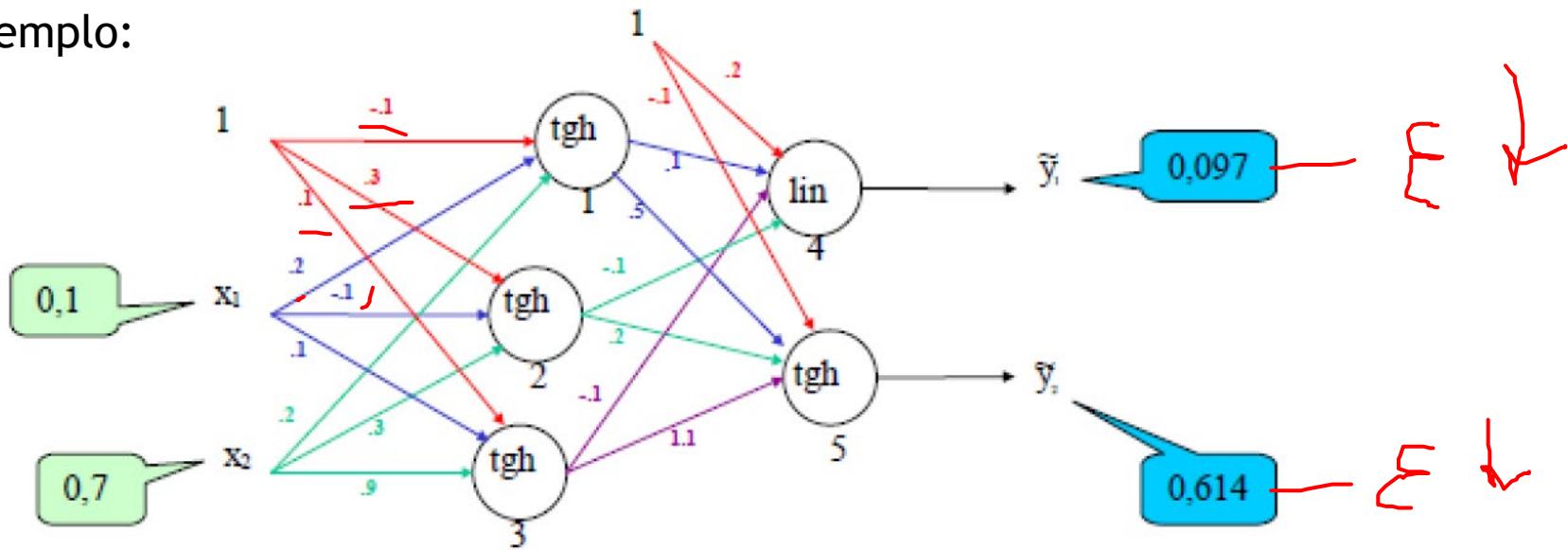


$$u_4 = \underline{0,2} + (0,1)(\underline{0,06}) + (-0,1)(0,46) + (-0,1)(0,63) = \underline{0,097}$$

$$v_4 = 0,097 \quad (\text{linear!})$$

Redes Neuronales

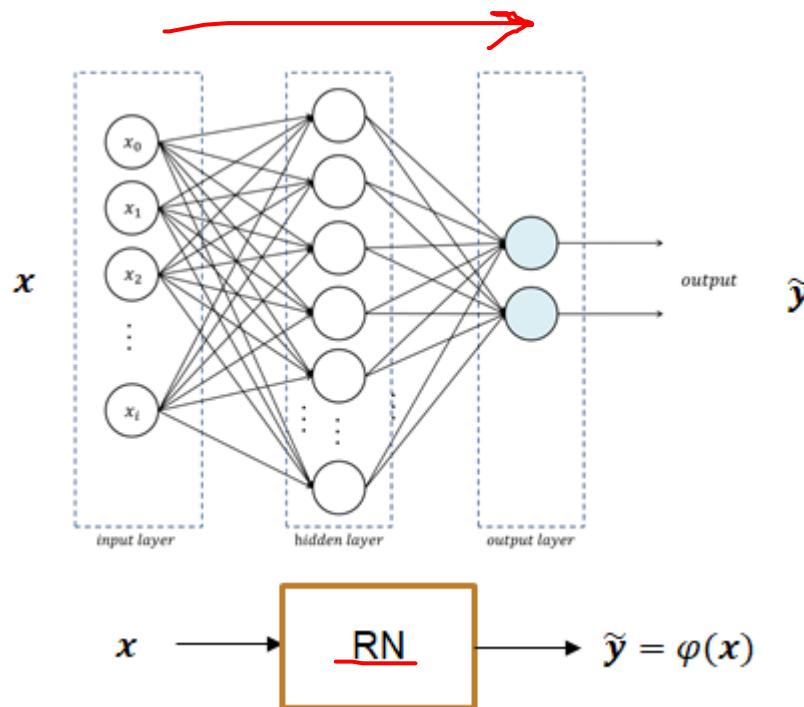
- Ejemplo:



$$\underline{x} = \begin{bmatrix} 0,1 \\ 0,7 \end{bmatrix} \quad \tilde{\underline{y}} = \begin{bmatrix} 0,097 \\ 0,614 \end{bmatrix}$$

Redes Neuronales

- Las redes neuronales *feedforward* permiten aproximar relaciones no lineales entre los datos de entrada y salida



Entrenamiento *backpropagation*

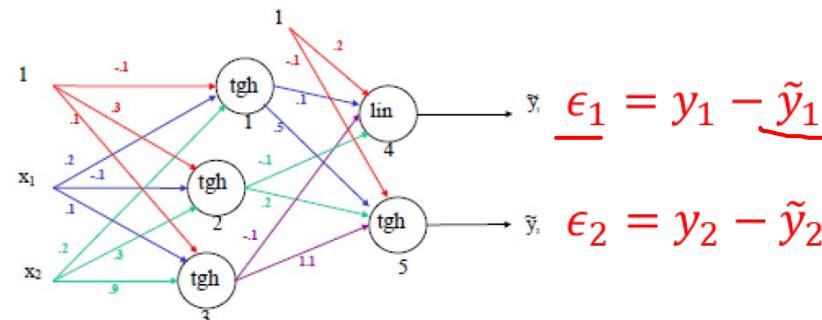
- La red se entrena a partir de ejemplos

Pares entrada - salida (filas)

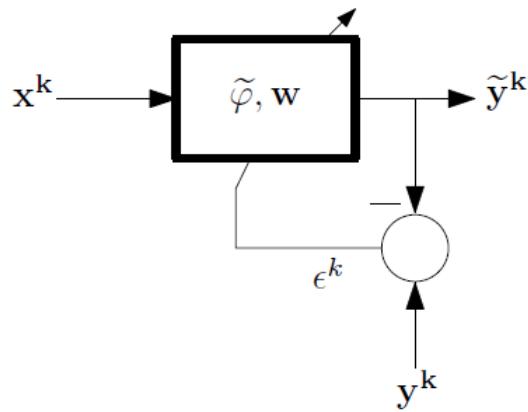
x^k			y^k		
x_1	x_2	\dots	y_1	y_2	\dots
2	45	\dots	3	6	\dots
\dots	\dots	\dots	\dots	\dots	\dots

$$\mathbf{x}^1 = \begin{bmatrix} x_1 \\ x_2 \\ \dots \end{bmatrix} = \begin{bmatrix} 2 \\ 45 \\ \dots \end{bmatrix}$$

$$\mathbf{y}^1 = \begin{bmatrix} y_1 \\ y_2 \\ \dots \end{bmatrix} = \begin{bmatrix} 3 \\ 6 \\ \dots \end{bmatrix}$$



Aprendizaje: minimización del error en la salida



Error a minimizar:

$$\varepsilon^{2^k} = \|\mathbf{y}^k - \tilde{\mathbf{y}}^k\|^2 = \sum_{l=1}^m (y_l^k - \tilde{y}_l^k)^2 \text{ Para todas las neuronas } \epsilon^{2^k} = \epsilon_1^2 + \epsilon_2^2$$

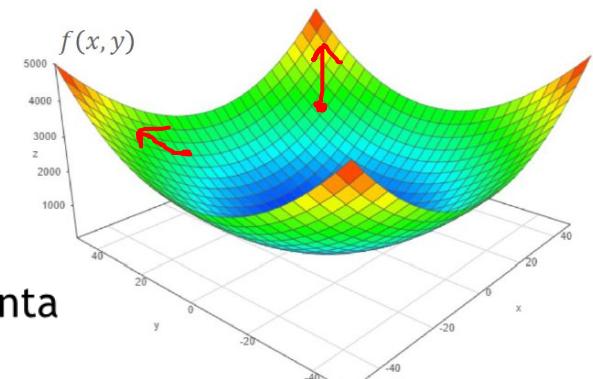
$$F_o = E[\varepsilon^{2^k}] = \left[\frac{1}{P} \sum_{k=1}^P \varepsilon^{2^k} \right] = F_o(\mathbf{w}) \geq 0 \text{ Para todos los pares}$$

Entrenamiento *backpropagation*

- Objetivo: Minimizar $F_o(\mathbf{w}) = E[\epsilon^{2^k}]$
- Se utiliza el algoritmo del **gradiente descendente**
 - El gradiente es la generalización de derivada para funciones de más de una variable
 - Ejemplo: Sea $f(x, y)$ una función de dos variables. Su gradiente es un vector compuesto por las derivadas parciales:

$$f(x, y) = \underline{x^2} + \underline{y^2}$$
$$\frac{\partial f}{\partial x} = \underline{2x} \quad \Rightarrow \nabla f(x, y) = \begin{bmatrix} 2x \\ 2y \end{bmatrix}$$
$$\frac{\partial f}{\partial y} = \underline{2y}$$

- En cada punto del plano XY, el vector gradiente apunta en la dirección de máximo crecimiento de $f(x, y)$



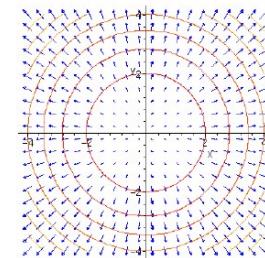
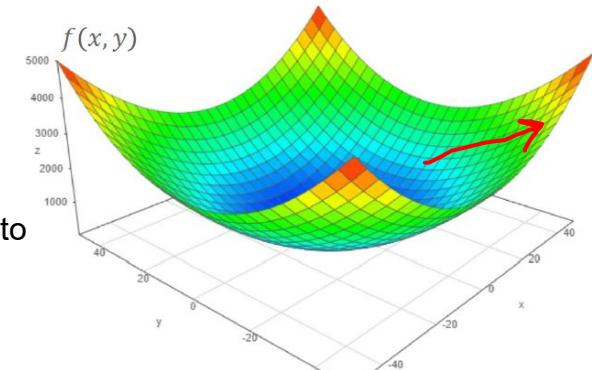
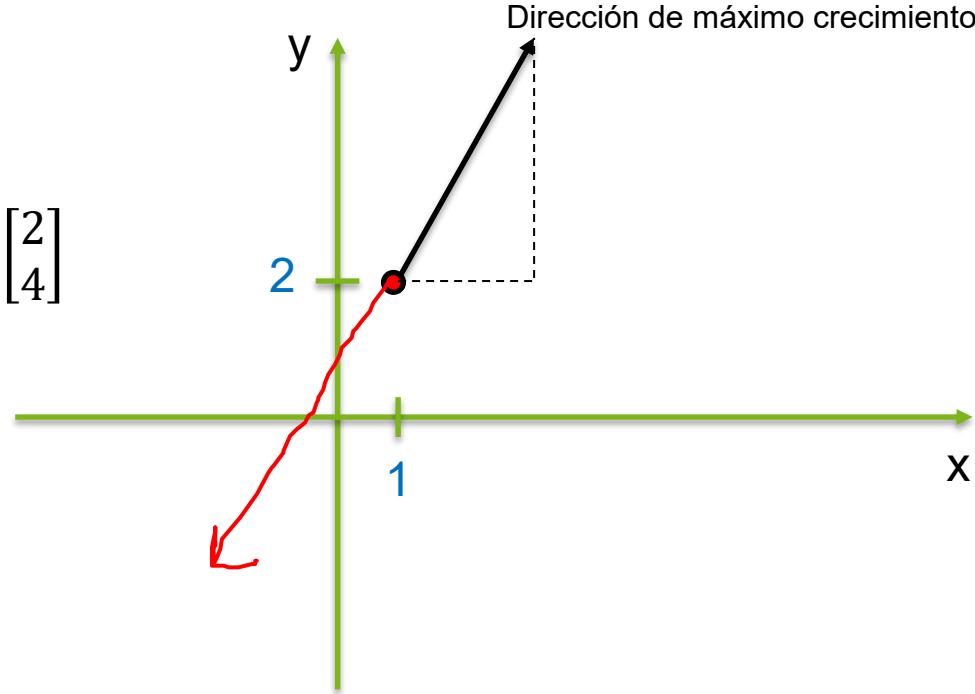
Entrenamiento *backpropagation*

$$f(x, y) = x^2 + y^2$$

$$\begin{aligned}\frac{\partial f}{\partial x} &= 2x \\ \frac{\partial f}{\partial y} &= 2y\end{aligned}$$

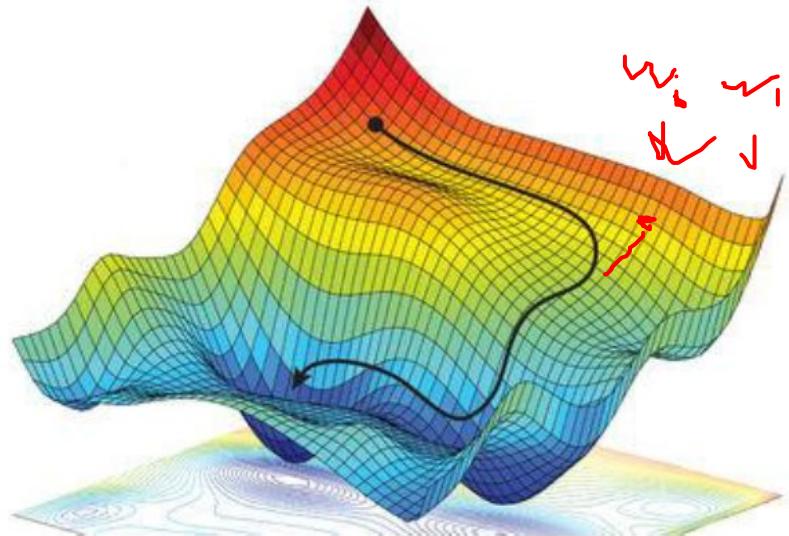
$$\Rightarrow \nabla f(x, y) = \begin{bmatrix} 2x \\ 2y \end{bmatrix}$$

$$\nabla f(1, 2) = \begin{bmatrix} 2 \\ 4 \end{bmatrix}$$



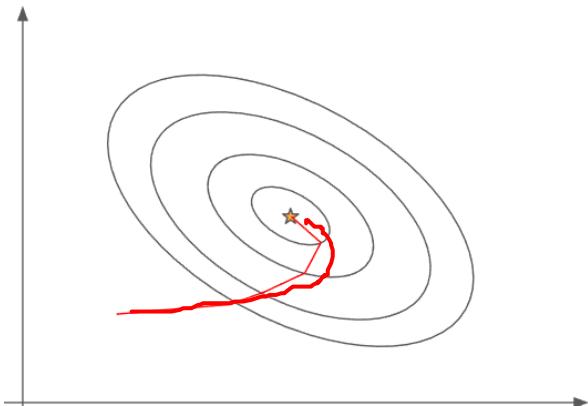
Entrenamiento *backpropagation*

- Objetivo: Minimizar $F_o(\mathbf{w}) = E[\epsilon^{2k}]$
- Se utiliza el algoritmo del **gradiente descendente**

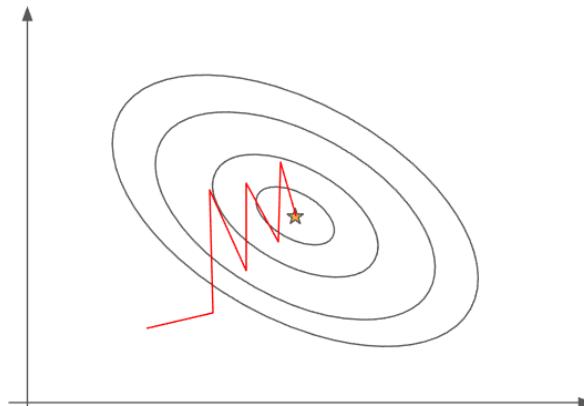


Entrenamiento batch

- Para acelerar el entrenamiento, suelen utilizarse subconjuntos de datos (*batches*) durante el proceso del gradiente descendente.
- Existen dos variaciones del algoritmo del gradiente descendente:
 - **Stochastic Gradient Descent.** Batch Size = 1
 - **Mini-Batch Gradient Descent.** $1 < \text{Batch Size} < \text{Size of Training Set}$. El tamaño de los batches suele ser de 32, 64 o 128 muestras



Gradient Descent



Stochastic Gradient Descent

Entrenamiento batch

Existen múltiples algoritmos de optimización, cada una con diferentes maneras de llegar al punto mínimo.

Gradient Descent

Stochastic Gradient Descent

Mini-Batch Gradient Descent

Momentum

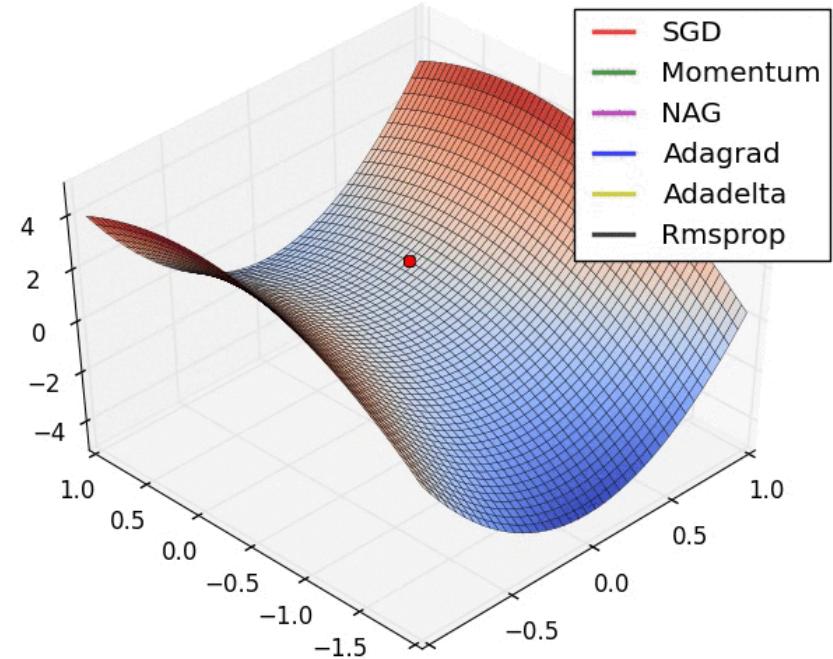
Nesterov Accelerated Gradient

Adagrad

AdaDelta

Adam

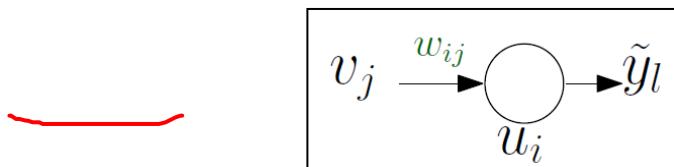
(...)



Entrenamiento *backpropagation*

- Objetivo: Minimizar $F_o(\mathbf{w}) = E[\epsilon^{2^k}]$
- Se utiliza el algoritmo del **gradiente descendente**: $\underline{\mathbf{w}} \rightarrow \underline{\mathbf{w}} + \Delta \mathbf{w}$; $\Delta w_{ij} = -\alpha \frac{\partial F_o}{\partial w_{ij}}$

α : learning rate



Entrenamiento backpropagation

- En resumen:

$$\underline{v}_j \xrightarrow{w_{ij}} \textcircled{u}_i \rightarrow \tilde{y}_l = s(u) = s(v \cdot w)$$

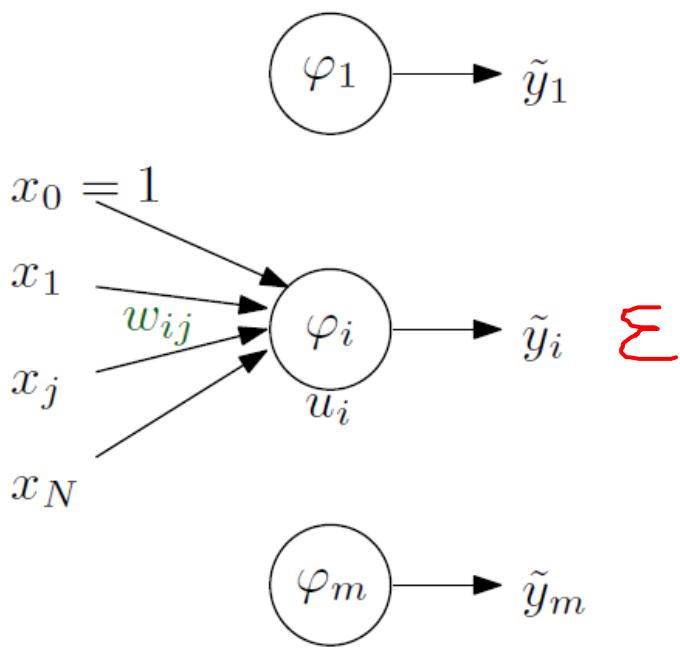
$$\tilde{y} = \tgh(u) \Rightarrow \frac{\partial \tilde{y}_l}{\partial u_i} = 1 - \tilde{y}^2$$

$$\Delta w_{ij}^p = \underline{2\alpha v_j \delta_i}$$

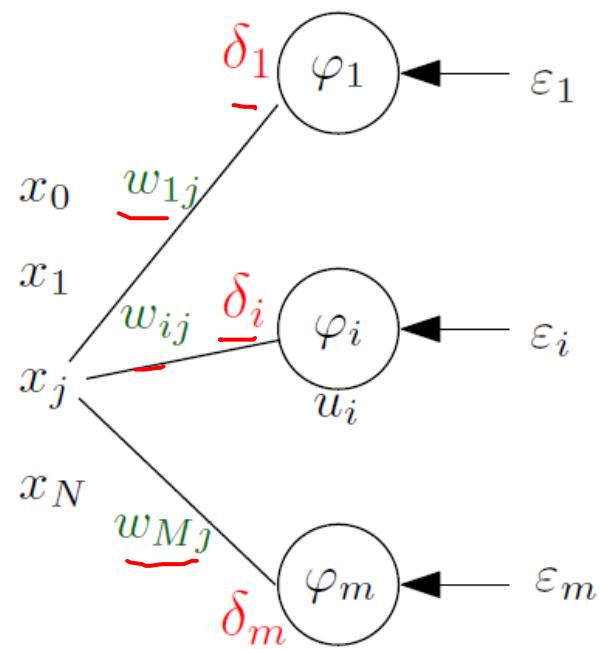
$$\underline{\delta}_i = \underline{\varepsilon}_i \frac{\partial \tilde{y}_l}{\partial u_i}$$

$$v_j = \frac{\partial u_i}{\partial w_{ij}}$$

Entrenamiento *backpropagation*



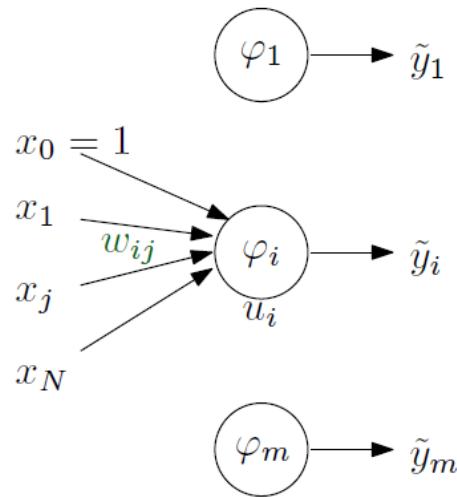
Red original



Red asociada y retropropagación del error

Redes de una capa

- Para cada par (x, y)



$$u_i = \sum_{j=0}^N w_{ij} x_j$$
$$\tilde{y}_i = \varphi_i(u_i) = \begin{cases} u_i & \text{(neurona lineal)} \\ \tgh(u_i) & \text{(neurona tgh)} \end{cases}$$

- Error retropropagado:

$$\delta_i = \begin{cases} \underline{\varepsilon_i} \cdot 1 & \text{(neurona lineal)} \\ \underline{\varepsilon_i} \cdot (1 - \tilde{y}_i^2) & \text{(neurona tgh)} \end{cases}$$

- Incremento en cada sinapsis debido al par p : $\Delta w_{ij}^p = 2\alpha x_j \delta_i$

Redes de una capa (ejemplo numérico)

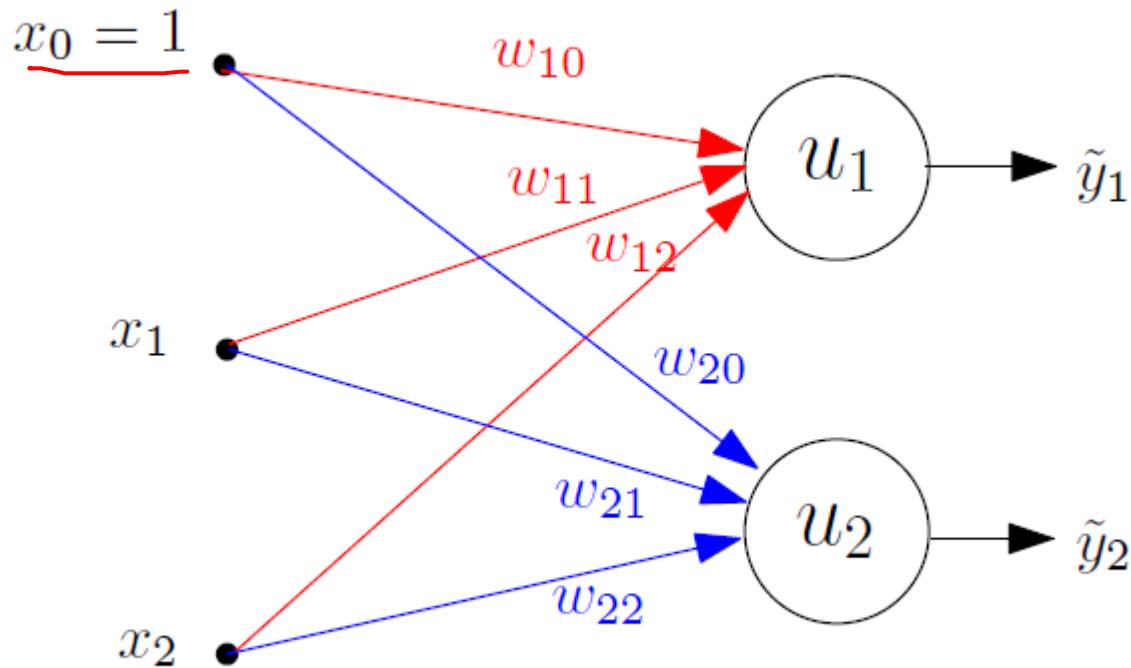
- Vamos a implementar una red neuronal con una capa y dos neuronas, la primera de tipo lineal y la segunda de tipo tangente hiperbólica
- Se presenta el par de entrada-salida (x, y) , donde $x = [0.1 \ 0.7]^T$ e $y = [0.2 \ 1]^T$
- $\alpha = 0.1$ y El valor inicial de las sinapsis es:

$$\left\{ \begin{array}{l} w_{10} = 0.1 \\ w_{11} = -0.2 \\ w_{12} = 0.3 \\ w_{20} = 0.5 \\ w_{21} = 0.4 \\ w_{22} = -0.6 \end{array} \right.$$

- Vamos a calcular los nuevos valores de las sinapsis tras el primer paso del entrenamiento

Redes de una capa (ejemplo numérico)

1) Dibuja la red neuronal



Redes de una capa (ejemplo numérico)

2) Calcula los valores u_1 y u_2

$$u_1 = w_{10} \cdot 1 + w_{11} \cdot x_1 + w_{12} \cdot x_2 = 0.1 \cdot 1 + (-0.2) \cdot 0.1 + 0.3 \cdot 0.7 = 0.29$$

$$u_2 = w_{20} \cdot 1 + w_{21} \cdot x_1 + w_{22} \cdot x_2 = 0.5 \cdot 1 + 0.4 \cdot 0.1 + (-0.6) \cdot 0.7 = 0.12$$

3) Calcula los valores \hat{y}_1 e \hat{y}_2

$$\tilde{y}_1 = u_1 = \underline{\underline{0.29}}$$

$$\tilde{y}_2 = tgh(u_2) = \underline{\underline{tgh(0.12)}} \approx \underline{\underline{0.12}}$$

4) Calcula los errores ε_1 y ε_2

$$\varepsilon_1 = y_1 - \tilde{y}_1 = 0.2 - 0.29 = -0.09 \quad -$$

$$\varepsilon_2 = \underline{\underline{y_2}} - \tilde{y}_2 = 1 - 0.12 = 0.88 \quad -$$

Redes de una capa (ejemplo numérico)

5) Calcula los errores retropropagados δ_1 y δ_2

$$\delta_1 = \boxed{\varepsilon_1} = -0.09$$

$$\delta_2 = \boxed{\varepsilon_2(1 - \tilde{y}_2^2)} = 0.88 \cdot (1 - 0.12^2) = 0.87$$

6) Calcula el nuevo valor de las sinapsis $w_{ij} \rightarrow w_{ij} + \Delta w_{ij}$

$$w_{10} \rightarrow \underline{w_{10}} + 2\alpha x_0 \delta_1 = 0.1 + 2 \cdot 0.1 \cdot 1 \cdot (-0.09) = 0.1 - 0.018 = 0.082$$

$$\underline{w_{11}} \rightarrow \underline{w_{11}} + \underline{2\alpha x_1} \underline{\delta_1} = -0.2 + 2 \cdot 0.1 \cdot 0.1 \cdot (-0.09) = -0.2 - 0.0018 = -0.2018$$

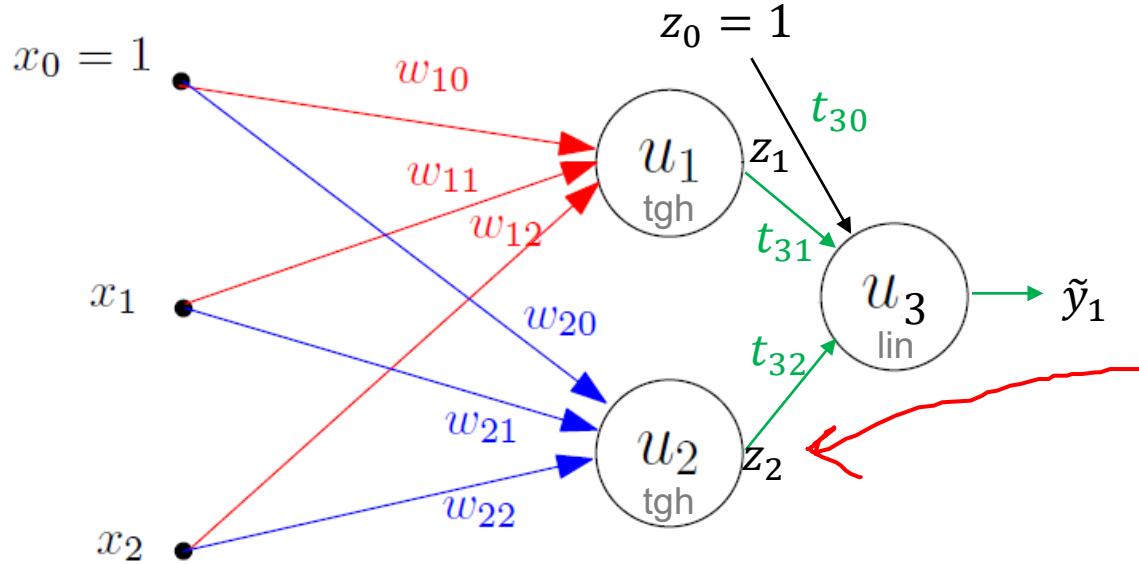
$$w_{12} \rightarrow w_{12} + 2\alpha x_2 \delta_1 = 0.3 + 2 \cdot 0.1 \cdot 0.7 \cdot (-0.09) = 0.3 - 0.0126 = 0.2874$$

$$w_{20} \rightarrow w_{20} + 2\alpha x_0 \delta_2 = 0.5 + 2 \cdot 0.1 \cdot 1 \cdot (0.87) = 0.5 + 0.174 = 0.674$$

$$w_{21} \rightarrow w_{21} + 2\alpha x_1 \delta_2 = 0.4 + 2 \cdot 0.1 \cdot 0.1 \cdot (0.87) = 0.4 + 0.0174 = 0.4174$$

$$w_{22} \rightarrow w_{22} + 2\alpha x_2 \delta_2 = -0.6 + 2 \cdot 0.1 \cdot 0.7 \cdot (0.87) = -0.6 + 0.1218 = -0.4784$$

Redes de dos capas (ejemplo)



$$\delta_3 = \epsilon_1$$

$$t_{31} = t_{31} + 2\alpha z_1 \delta_3$$

$$t_{32} = t_{32} + 2\alpha z_2 \delta_3$$

$$t_{30} = t_{30} + 2\alpha z_0 \delta_3$$



$$\gamma_1 = (1 - z_1^2) \delta_3 t_{31}$$

$$\gamma_2 = (1 - z_2^2) \delta_3 t_{32}$$



$$w_{10} = w_{10} + 2\alpha x_0 \gamma_1$$

$$w_{11} = w_{11} + 2\alpha x_1 \gamma_1$$

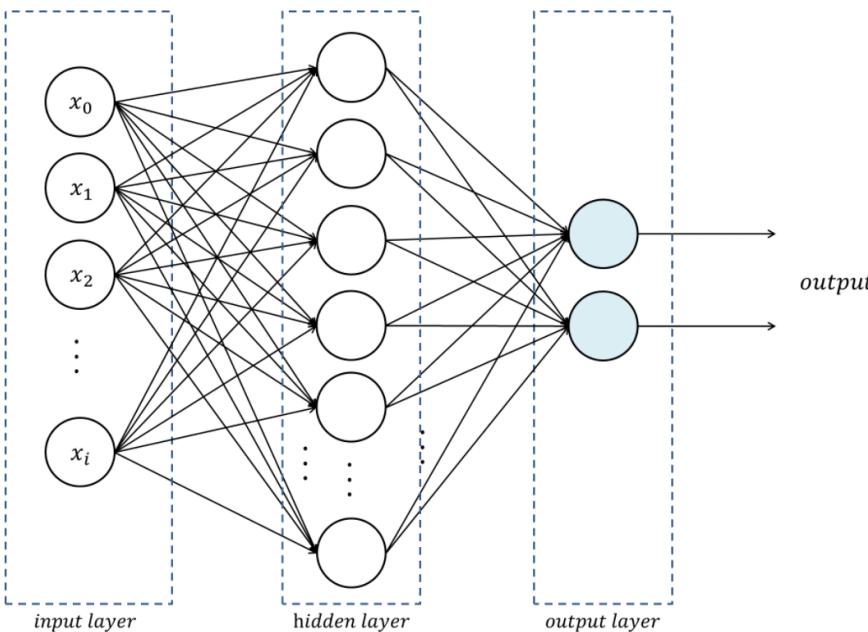
$$w_{12} = w_{12} + 2\alpha x_2 \gamma_1$$

$$w_{20} = w_{20} + 2\alpha x_0 \gamma_2$$

$$w_{21} = w_{21} + 2\alpha x_1 \gamma_2$$

$$w_{22} = w_{22} + 2\alpha x_2 \gamma_2$$

Dimensionando las Redes Neuronales

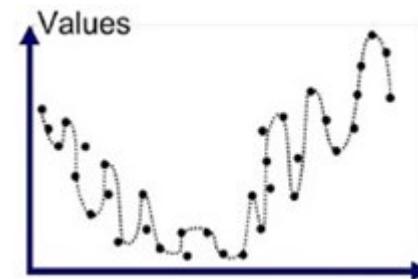


- Red sub-dimensionada
 - No tiene capacidad de representar el mapeo $x \rightarrow y$
 - No aprende, el error se mantiene elevado
- Red sobredimensionada
 - Lenta
 - Tendencia al overfitting

¿Cuántas capas de neuronas?

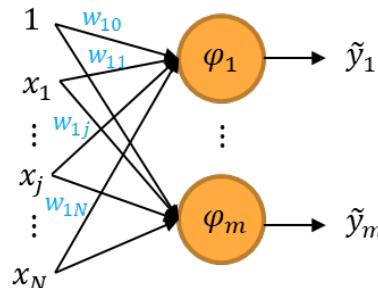
¿Cuántas neuronas por capa?

¿Qué tipo de neurona?



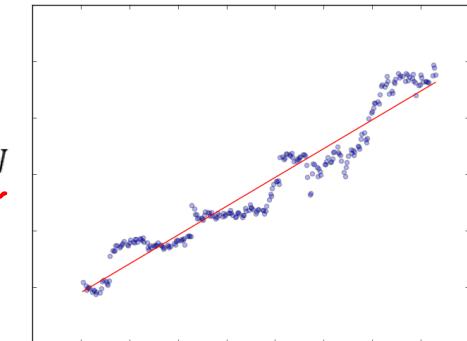
Dimensionando las Redes Neuronales

Redes de una capa



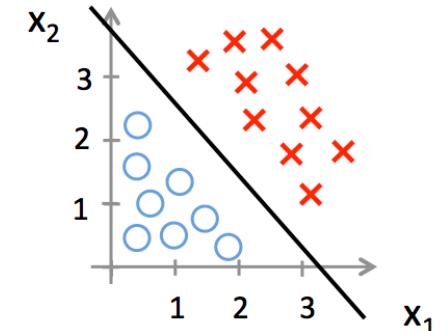
- Si las neuronas son lineales, la red es un regresor lineal

$$\tilde{y}_1 = \sum x_j w_{1j} + w_{1o} = w_{10} + w_{11}x_1 + \dots + w_{1j}x_j + \dots + w_{1N}x_N$$



- Si las neuronas son de tipo $\text{tgh}()$, la salida es un clasificador lineal

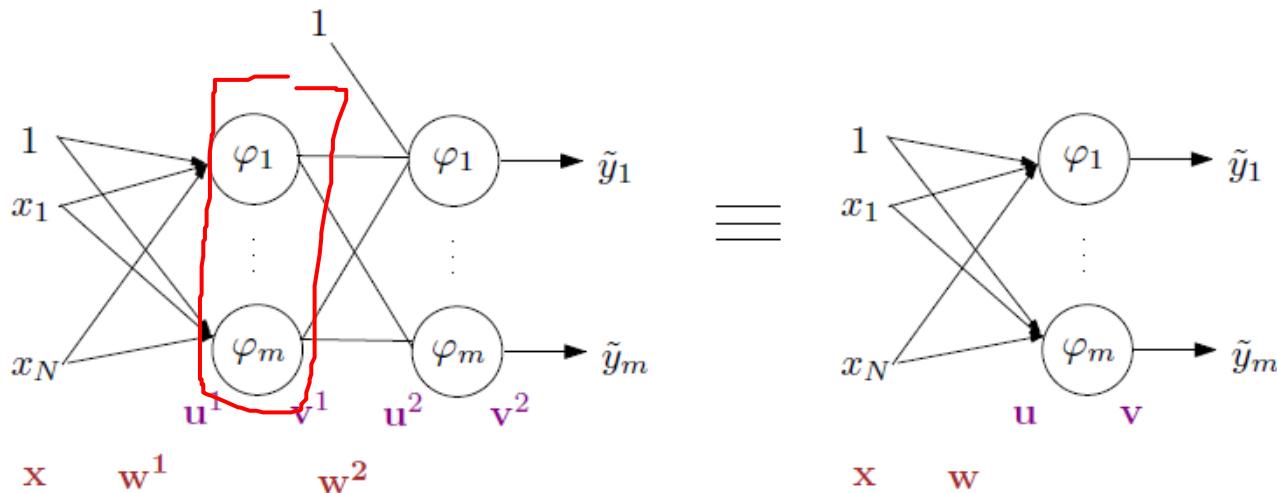
$$\tilde{y}_1 = \text{tgh}(\sum x_j w_{1j} + w_{1o})$$



Dimensionando las Redes Neuronales

Redes de dos capas: Multilayer perceptron (MLP)

- Si todas las neuronas son lineales, la red es idéntica a una red con una sola capa



$$u^2 = w^2v^1 = \underline{w^2} \underline{w^1} x$$

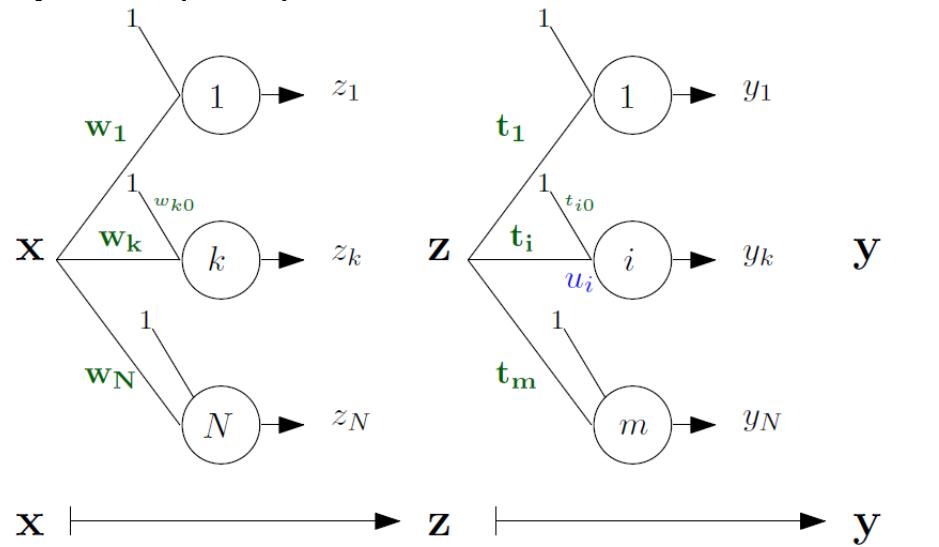
↳ ↳
↙ ↘

$$u = wx = w^2w^1x$$

Dimensionando las Redes Neuronales

Redes de dos capas: Multilayer perceptron (MLP)

- La capa intermedia debe tener neuronas de tipo $\tanh()$
- Teorema de aproximación universal: Si una función es L^2 ($\int |f|^2 < \infty$) en un dominio, entonces una serie de tangentes hiperbólicas es un aproximador universal para la función en el dominio.
- Una red neuronal con dos capas, la primera con neuronas de tipo $\tanh()$ y la segunda con neuronas lineales es un aproximador universal para todas las funciones de interés práctico



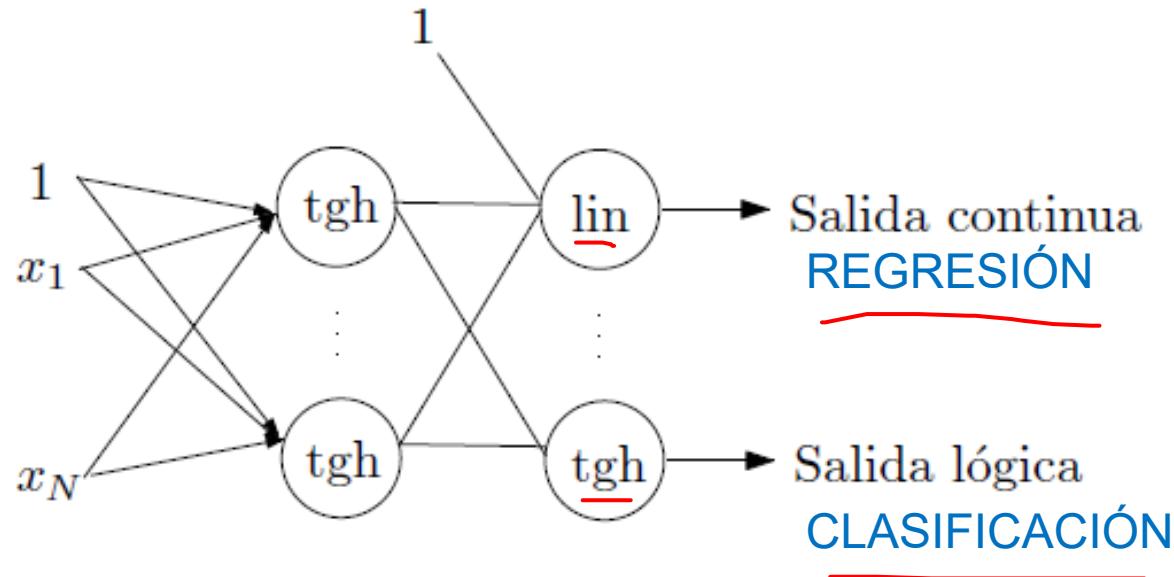
$$z_k = \tanh(x^T w_k + w_{k0})$$

$$u_i = z^T t_i + t_{i0} = t_{i0} + \sum_k t_{ik} \tanh(x^T w_k + w_{k0})$$

$$\tilde{y}_i = \begin{cases} u_i & \text{Neurona de salida lineal} \\ \tanh(u_i) & \text{Neurona de salida } \tanh \text{ (usada en clasificadores)} \end{cases}$$

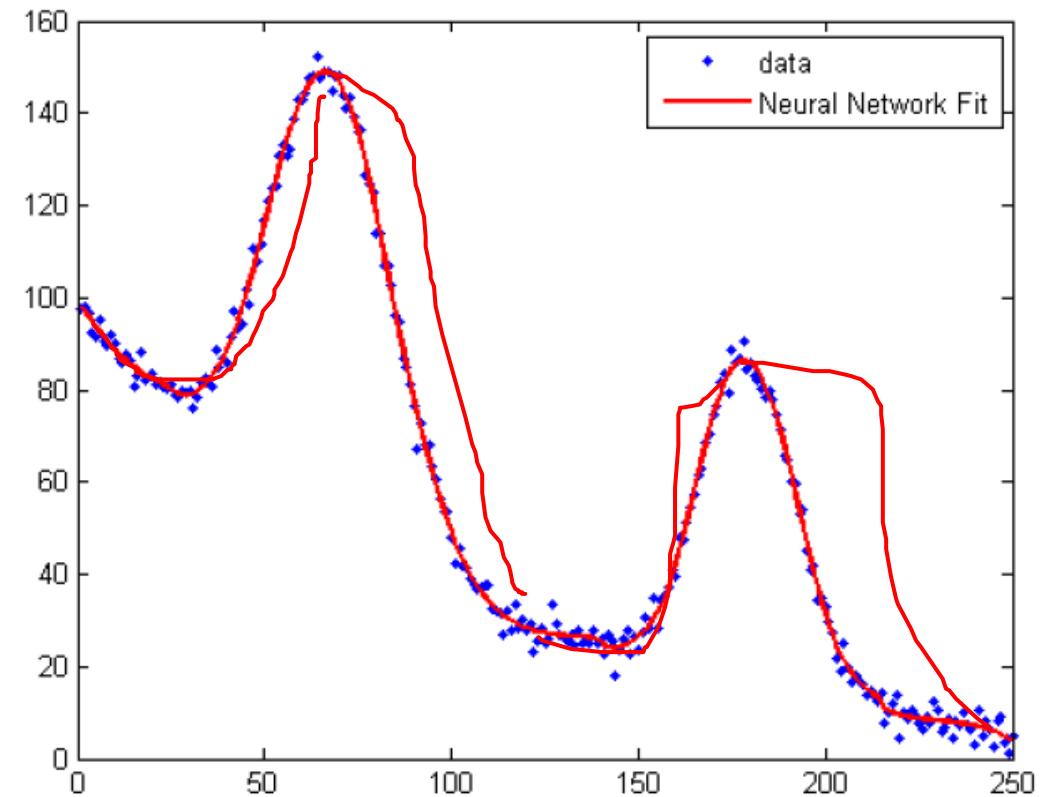
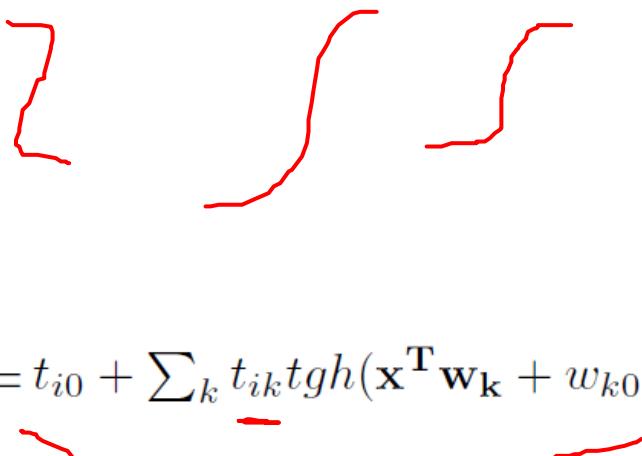
Dimensionando las Redes Neuronales

Redes de dos capas: Multilayer perceptron (MLP)



Dimensionando las Redes Neuronales

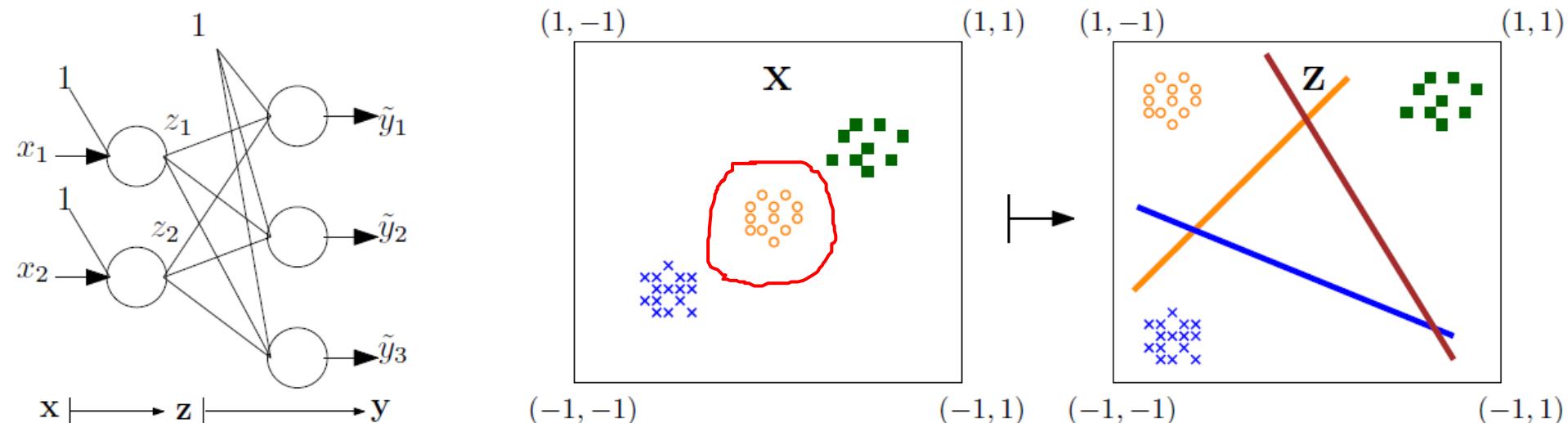
Redes de dos capas (regresión)



Dimensionando las Redes Neuronales

Redes de dos capas (clasificación)

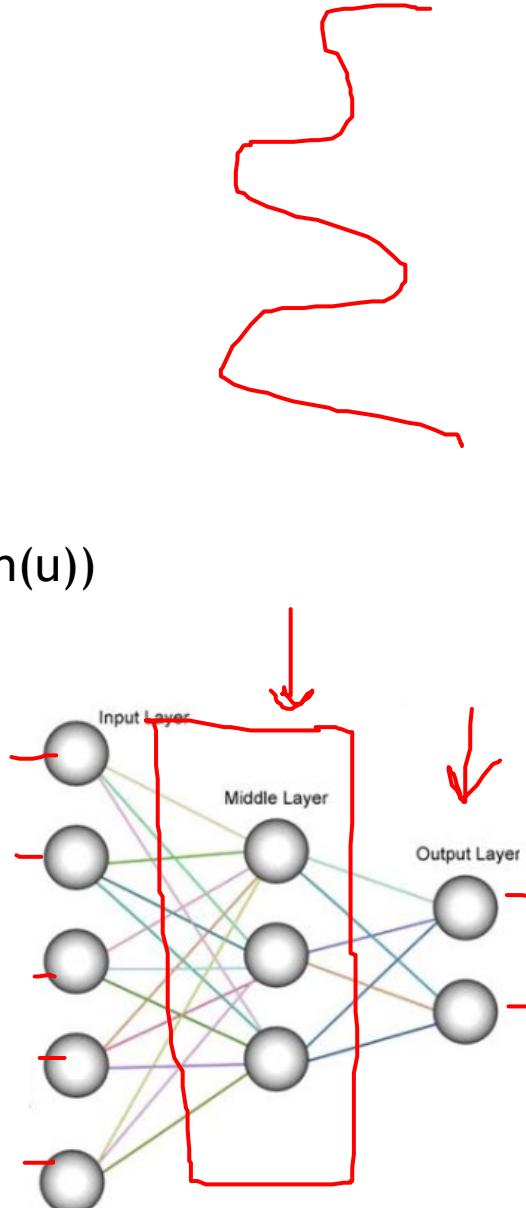
$$\tilde{y}_i = \text{tgh}(\tilde{t}_{i0} + \sum_k t_{ik} \text{tgh}(\mathbf{x}^T \mathbf{w}_k + w_{k0}))$$



La capa de salida separa clases que sí son linealmente separables en el dominio \mathbf{z}
La capa intermedia mapea clases no linealmente separables de \mathbf{x} en clases
linealmente separables en \mathbf{z}

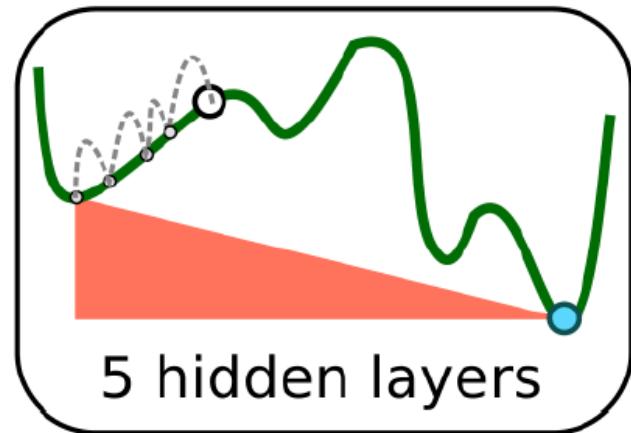
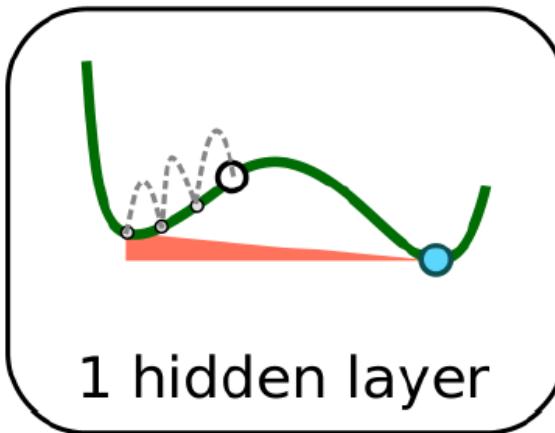
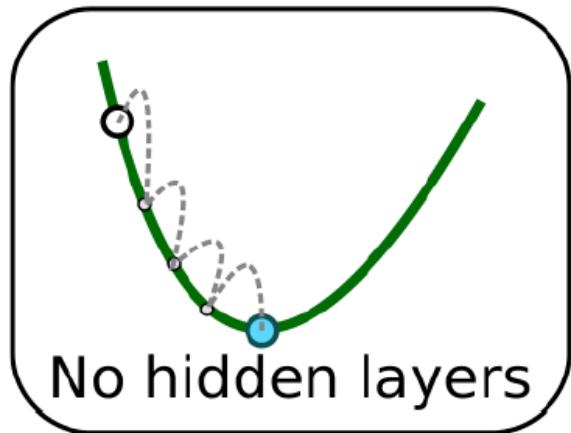
Dimensionando las Redes Neuronales

- ~~Usar una o dos capas~~
- En la capa de salida:
 - Regresión (salida continua): neurona lineal ($y = u$)
 - Clasificación (salida categórica): neurona tgh ($y = tgh(u)$)
- En la capa intermedia usar siempre neuronas de tipo tgh
- Redes de una capa: si el error es alto, utilizar dos capas
- Redes de dos capas:
 - Número de neuronas en la capa intermedia: m_1
 - Regla general: $n > m_1 > m$
 - Si el error es grande, aumentar m_1
- Redes de tres o más capas no son necesarias



Dimensionando las Redes Neuronales

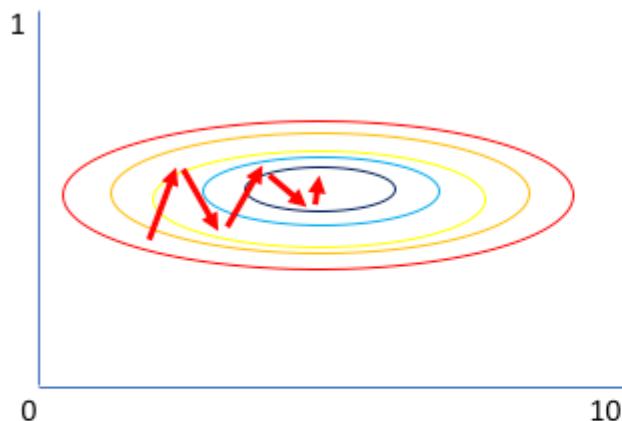
- El algoritmo del gradiente descendente puede finalizar en un mínimo local de la función de error
- Cuando introducimos más no-linealidad en la red (más capas), se multiplican los mínimos locales



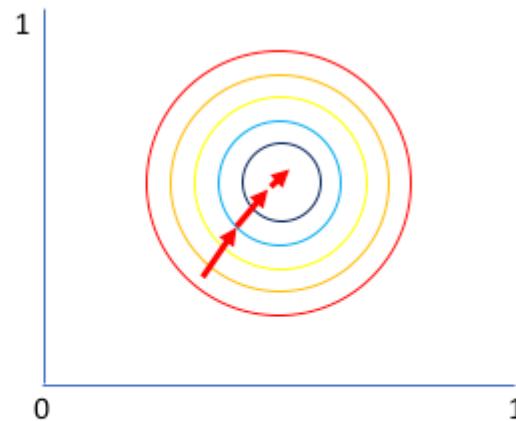
Normalización de variables

- Fundamental para el buen condicionamiento del proceso numérico de optimización
- Queremos valores en el intervalo (-1,+1)

$$\begin{aligned} w &= 0.3 \\ x_1 &= 1.2 \\ x_2 &= 3500.0 \end{aligned}$$



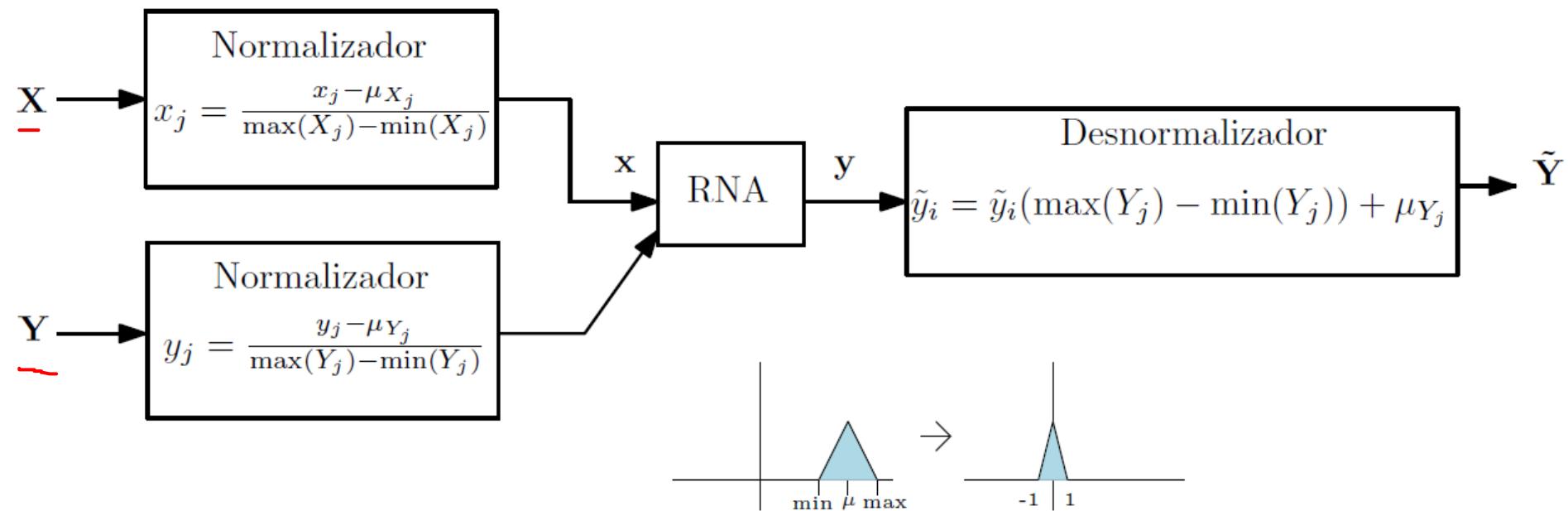
Gradient of larger parameter
dominates the update



Both parameters can be
updated in equal proportions

Normalización de variables

- Fundamental para el buen condicionamiento del proceso numérico de optimización
- Queremos valores en el intervalo (-1,+1)



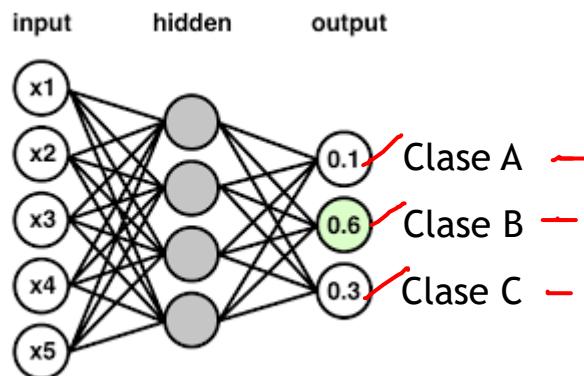
Normalización de variables

- Para variables categóricas, usar el método one-hot encoding

The diagram illustrates the one-hot encoding process. On the left, a table titled "Color" lists five entries: Red, Red, Yellow, Green, and Yellow. A red curly brace groups the first two "Red" entries. A large yellow arrow points from this group to a second table on the right. The second table has columns labeled "Red", "Yellow", and "Green". The rows correspond to the entries in the first table. The "Red" column contains values 1, 1, 0, 0, and 0 respectively. The "Yellow" column contains 0, 0, 1, 0, and 1 respectively. The "Green" column contains 0, 0, 0, 1, and 0 respectively.

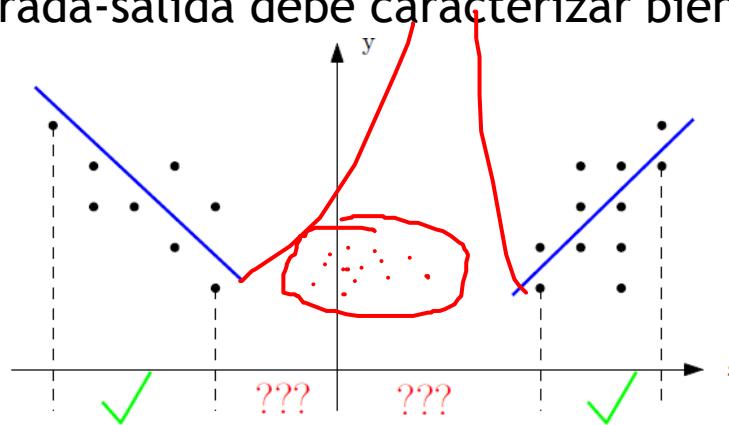
Color	Red	Yellow	Green
Red	1	0	0
Red	1	0	0
Yellow	0	1	0
Green	0	0	1
Yellow	0	1	0

- En clasificadores, las neuronas de salida representan cada una de las categorías

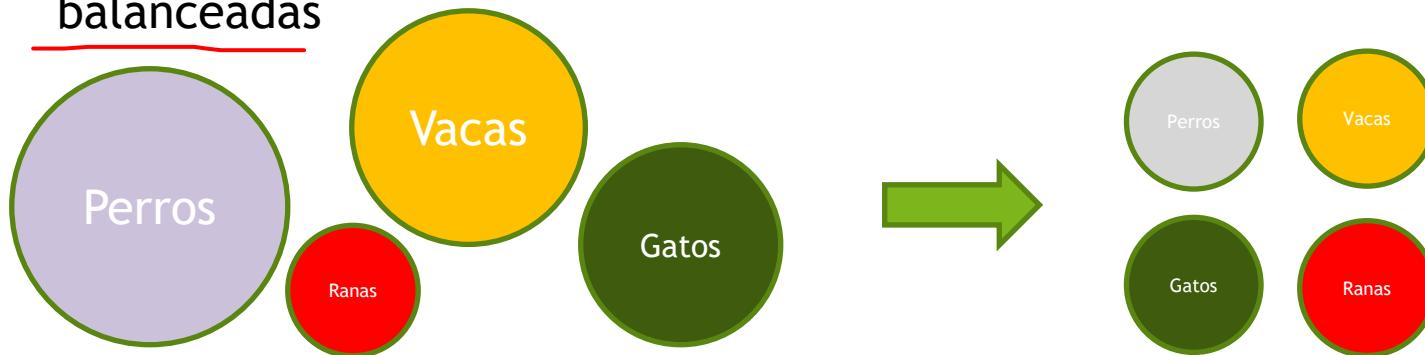


Pares entrada-salida

- El número de pares entrada-salida debe caracterizar bien el dominio de operación



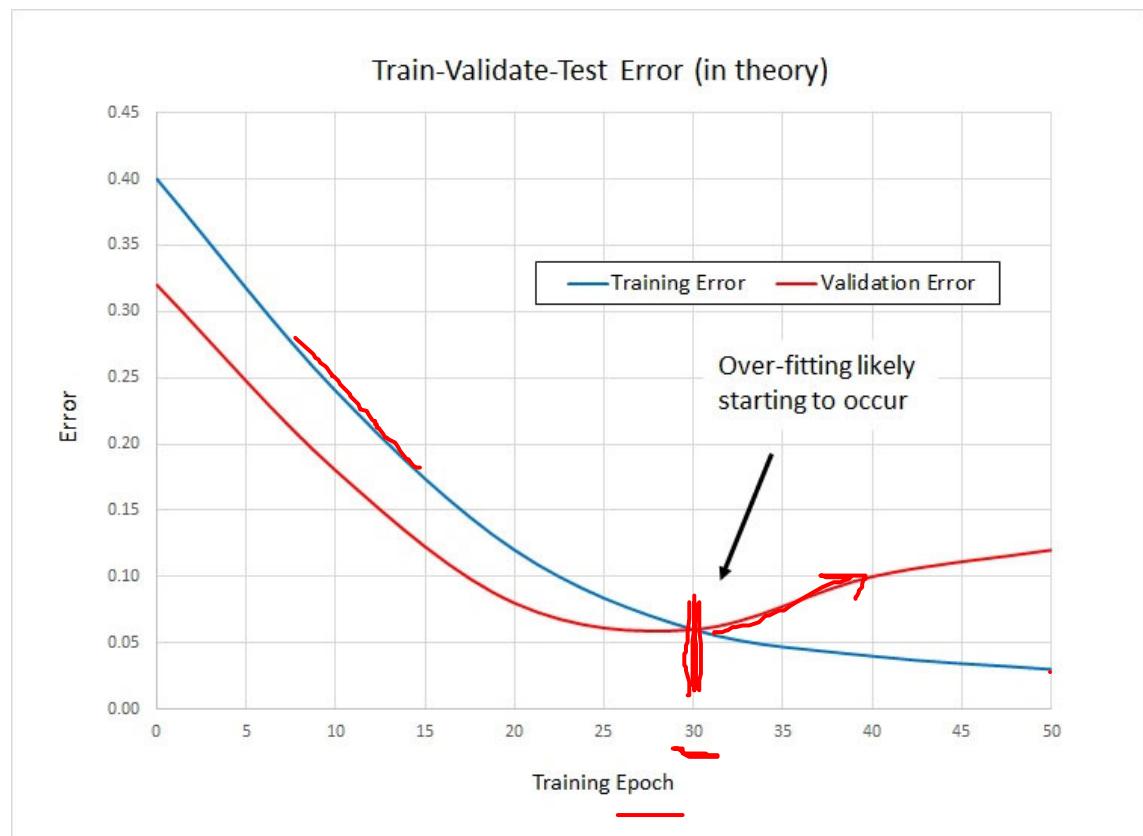
- En problemas de clasificación, es recomendable que las clases estén balanceadas



Entrenamiento en Redes Neuronales

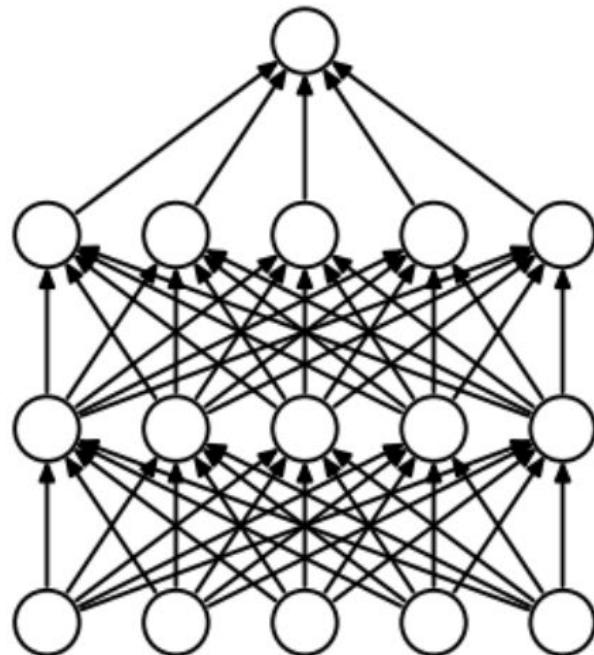
- El conjunto de pares entrada-salida se divide en:

- Entrenamiento (60%)
- Validación (20%)
- Test (20%)

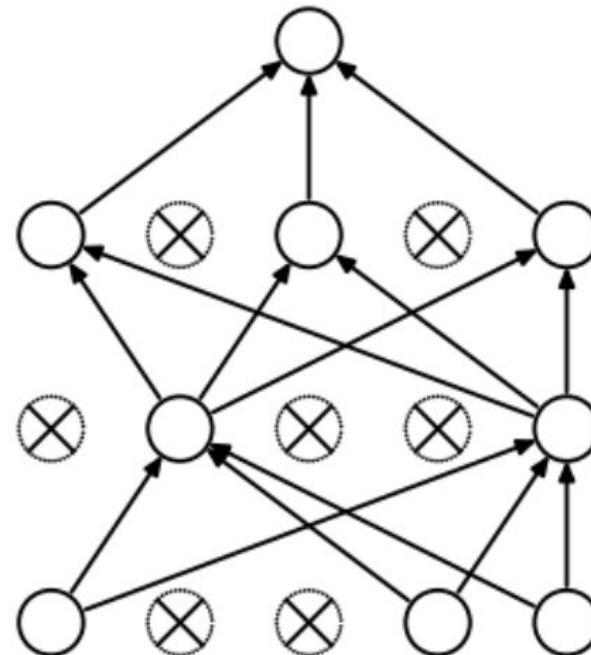


Regularización

- Para evitar el overfitting, a las redes neuronales se les puede aplicar el método *dropout*, que consiste en desactivar neuronas aleatoriamente basándose en una probabilidad



(a) Standard Neural Net



(b) After applying dropout.

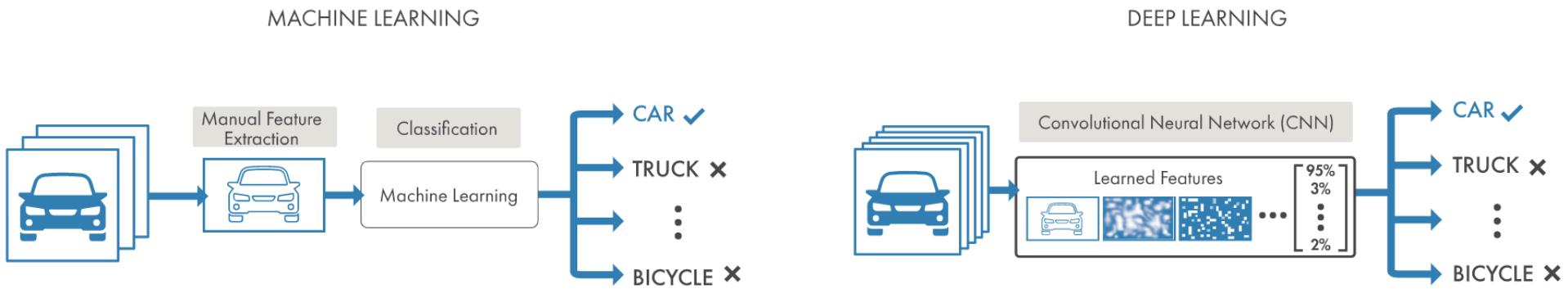
Demo

<https://playground.tensorflow.org/#activation=tanh&batchSize=10&dataset=circle&egDataset=reg-plane&learningRate=0.03®ularizationRate=0&noise=0&networkShape=&seed=0.14894&showTestData=false&discretize=false&percTrainData=50&x=true&y=false&xTimesY=false&xSquared=false&ySquared=false&cosX=false&sinX=false&cosY=false&sinY=false&collectStats=false&problem=classification&initZero=false&hideText=false>

Deep Learning

Deep Learning

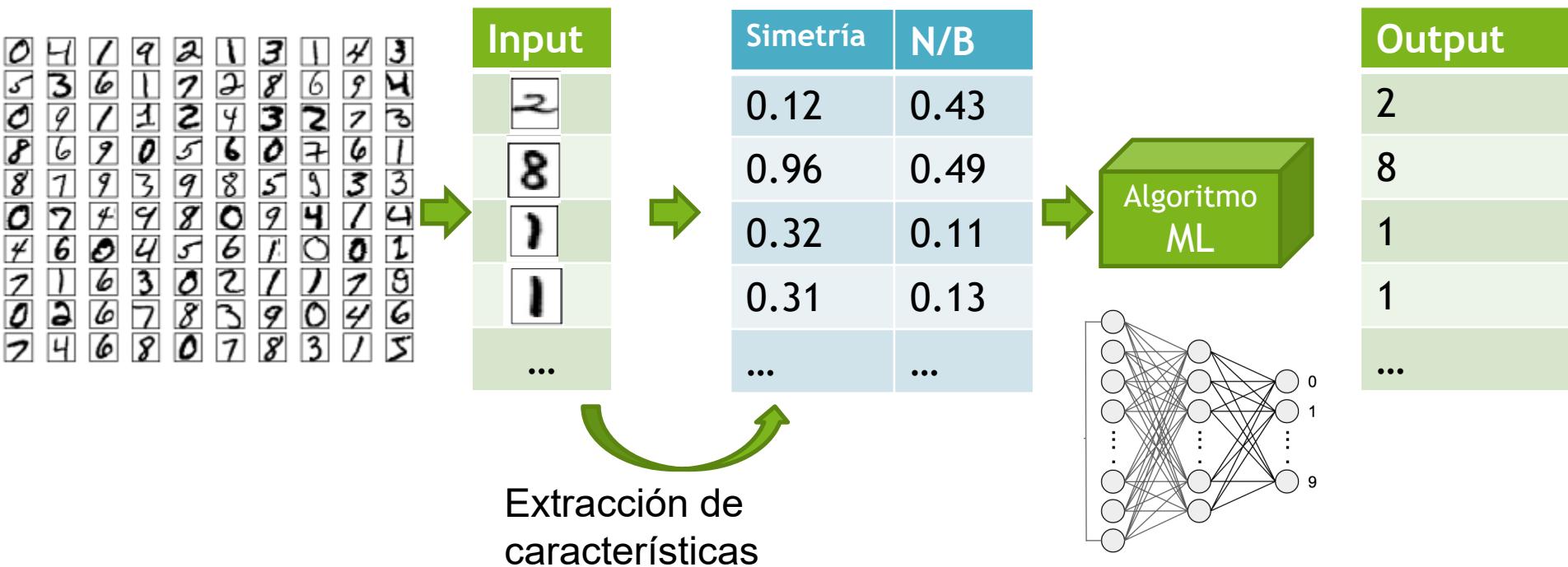
- Machine Learning vs. Deep Learning



- En Machine Learning, se seleccionan manualmente las características
- Con el aprendizaje profundo, los pasos de extracción de características y modelización son automáticos (redes neuronales profundas)
- Para utilizar deep learning, es necesario disponer de GPUs y una gran cantidad de datos etiquetados

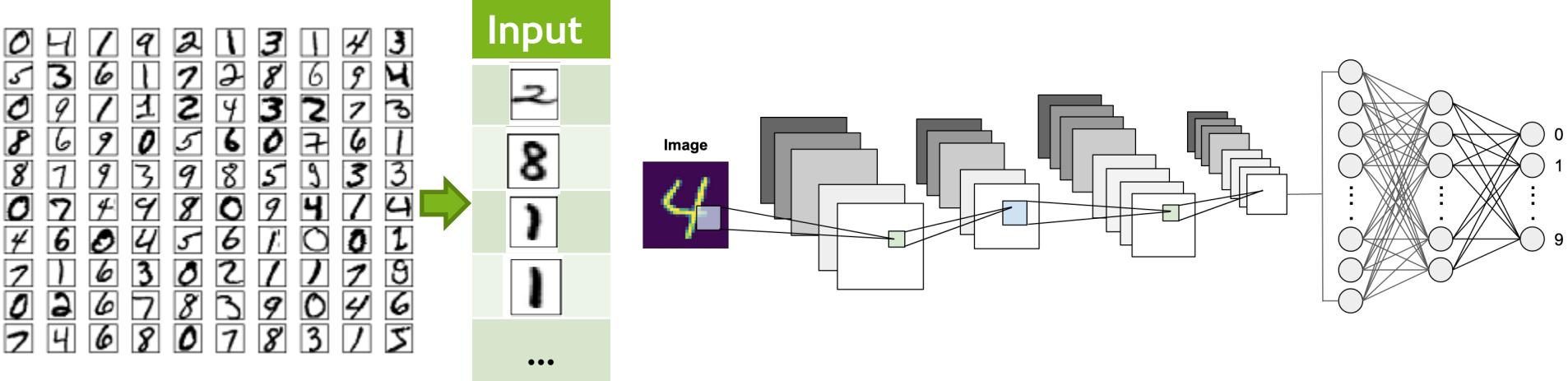
Deep Learning

- Machine Learning vs. Deep Learning



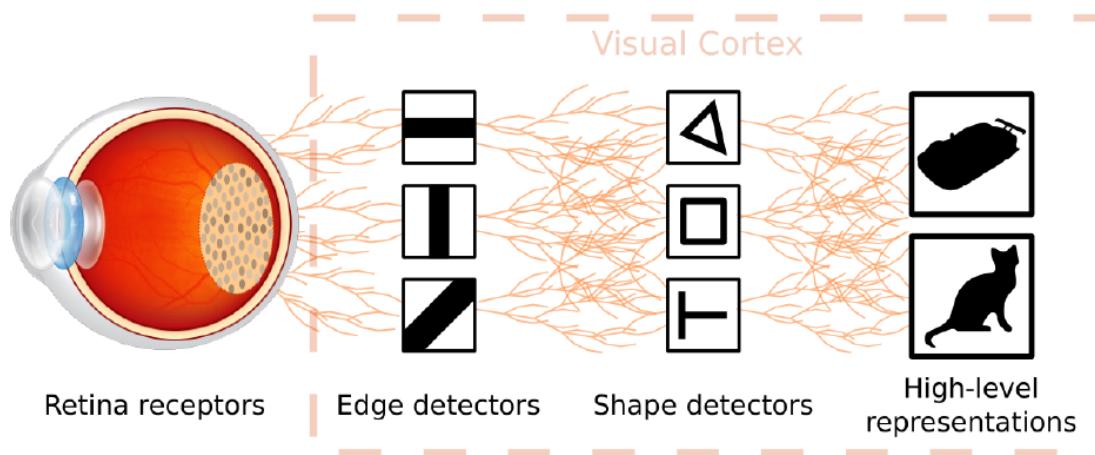
Deep Learning

- Machine Learning vs. Deep Learning



Redes Neuronales Convolucionales (CNN)

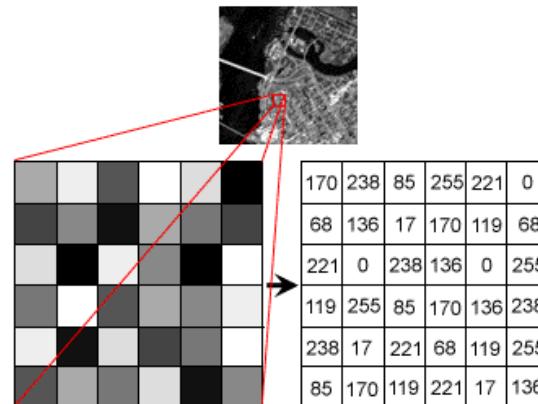
- Uno de los algoritmos más populares de redes neuronales profundas
- Son especialmente eficaces en la **clasificación de imágenes**, aunque también pueden utilizarse con series temporales o señales de audio y vídeo
- Imitan el funcionamiento del sistema visual humano



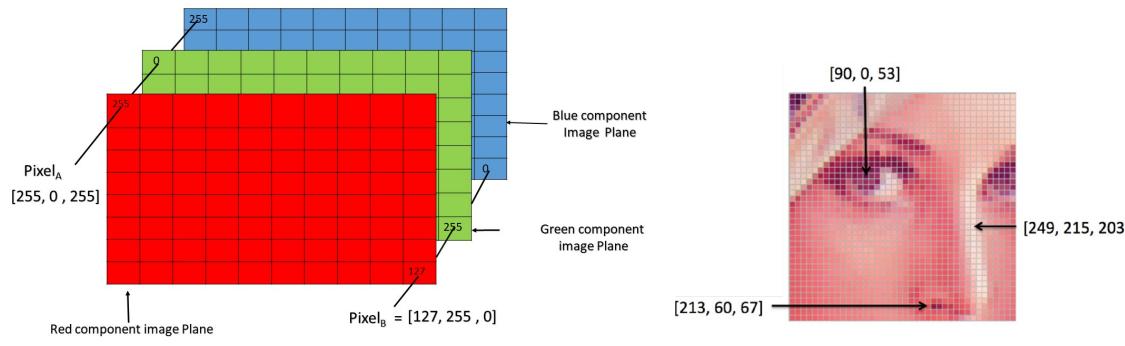
- Son robustas ante cambios de tamaño, contraste, rotación u orientación

Redes Neuronales Convolucionales (CNN)

- Representación digital de imágenes: matriz de píxeles con valores de intensidad entre 0 y 255



- Las imágenes en color se representan con tres capas superpuestas (RGB)

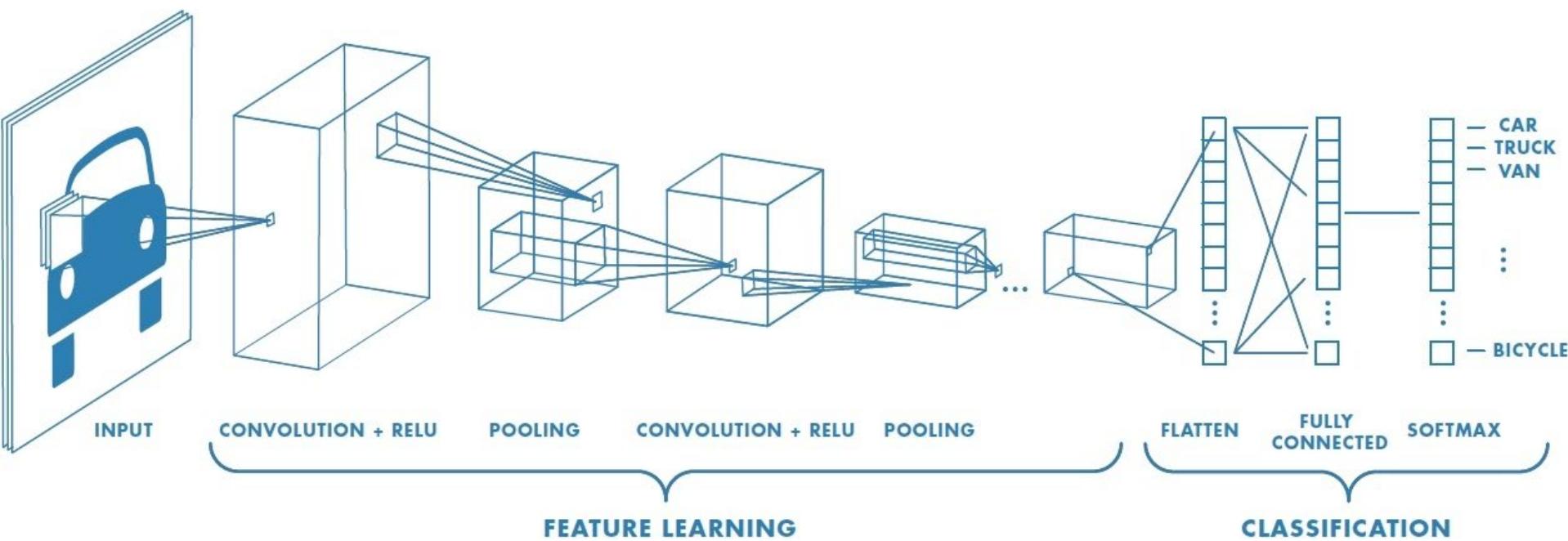


Redes Neuronales Convolucionales (CNN)

Etapas en CNN:

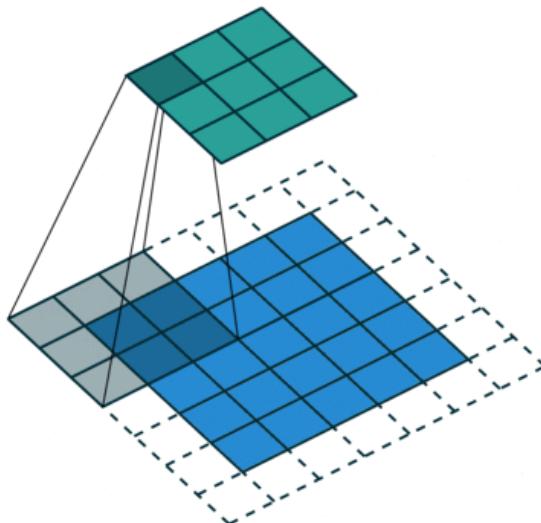
- Convolución
- Normalización (ReLU)
- Pooling
- Regularización

Pensemos en
capas, no en
neuronas



Redes Neuronales Convolucionales (CNN)

- Las CNN realizan operaciones de convolución, una técnica ampliamente utilizada en procesamiento de señales e imágenes
- La convolución permite preservar la relación entre diferentes partes de una imagen
- Para realizar las convoluciones, se utilizan pequeñas matrices denominadas kernels, que recorren la imagen original, aplican productos escalares y producen una nueva imagen con características diferentes



$$\begin{array}{|c|c|c|c|c|} \hline 7 & 2 & 3 & 3 & 8 \\ \hline 4 & 5 & 3 & 8 & 4 \\ \hline 3 & 3 & 2 & 8 & 4 \\ \hline 2 & 8 & 7 & 2 & 7 \\ \hline 5 & 4 & 4 & 5 & 4 \\ \hline \end{array} * \begin{array}{|c|c|c|} \hline 1 & 0 & -1 \\ \hline 1 & 0 & -1 \\ \hline 1 & 0 & -1 \\ \hline \end{array} = \begin{array}{|c|c|c|} \hline 6 & & \\ \hline & & \\ \hline & & \\ \hline \end{array}$$

$$7 \times 1 + 4 \times 1 + 3 \times 1 + 2 \times 0 + 5 \times 0 + 3 \times 0 + 3 \times 1 + 3 \times 1 + 2 \times 1 = 6$$

Redes Neuronales Convolucionales (CNN)

- Dependiendo del kernel, se resaltarán diferentes aspectos de una imagen



$$\begin{matrix} * & \begin{array}{|c|c|c|} \hline -1 & -2 & -1 \\ \hline 0 & 0 & 0 \\ \hline 1 & 2 & 1 \\ \hline \end{array} & = & \begin{array}{|c|c|c|} \hline \text{(bordes)} \\ \hline \end{array} \end{matrix}$$



$$\begin{matrix} * & \begin{array}{|c|c|c|} \hline -2 & -1 & 0 \\ \hline -1 & 1 & 1 \\ \hline 0 & 1 & 2 \\ \hline \end{array} & = & \begin{array}{|c|c|c|} \hline \text{(relieve)} \\ \hline \end{array} \end{matrix}$$



Redes Neuronales Convolucionales (CNN)

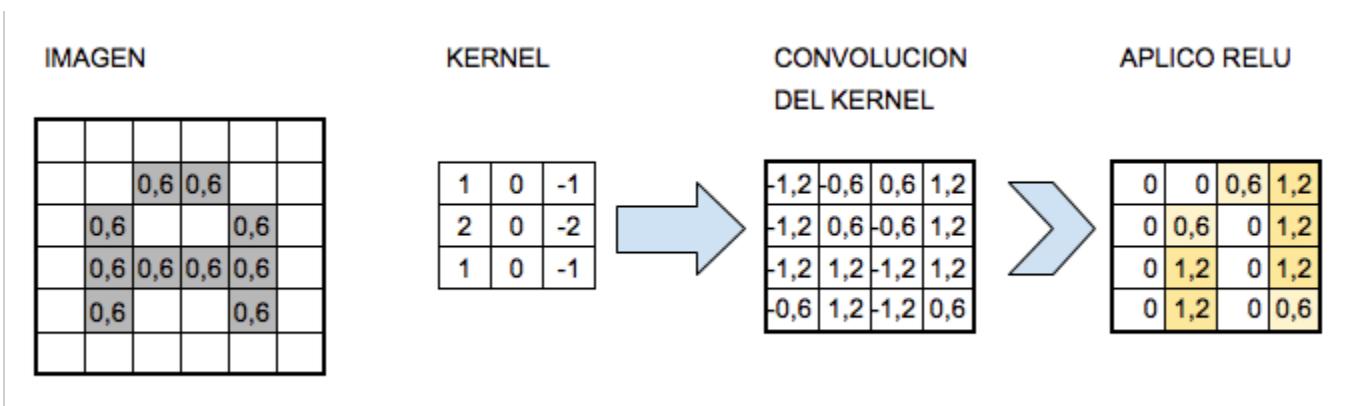
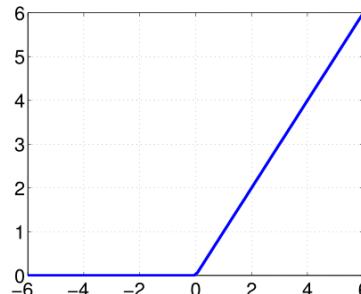
- En una capa convolucional, suelen utilizarse varios kernels
- Los pesos del kernel se aprenden durante el entrenamiento (backpropagation)

	Operation	Filter	Convolved Image
	Identity	$\begin{bmatrix} 0 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 0 \end{bmatrix}$	
Edge detection	Sharpen	$\begin{bmatrix} 0 & -1 & 0 \\ -1 & 5 & -1 \\ 0 & -1 & 0 \end{bmatrix}$	
	Box blur (normalized)	$\frac{1}{9} \begin{bmatrix} 1 & 1 & 1 \\ 1 & 1 & 1 \\ 1 & 1 & 1 \end{bmatrix}$	
	Gaussian blur (approximation)	$\frac{1}{16} \begin{bmatrix} 1 & 2 & 1 \\ 2 & 4 & 2 \\ 1 & 2 & 1 \end{bmatrix}$	

Redes Neuronales Convolucionales (CNN)

- La función de activación más utilizada tras aplicar la convolución se denomina Rectifier Linear Unit (ReLU), que facilita el entrenamiento

$$f(x) = \max \{0, x\}$$

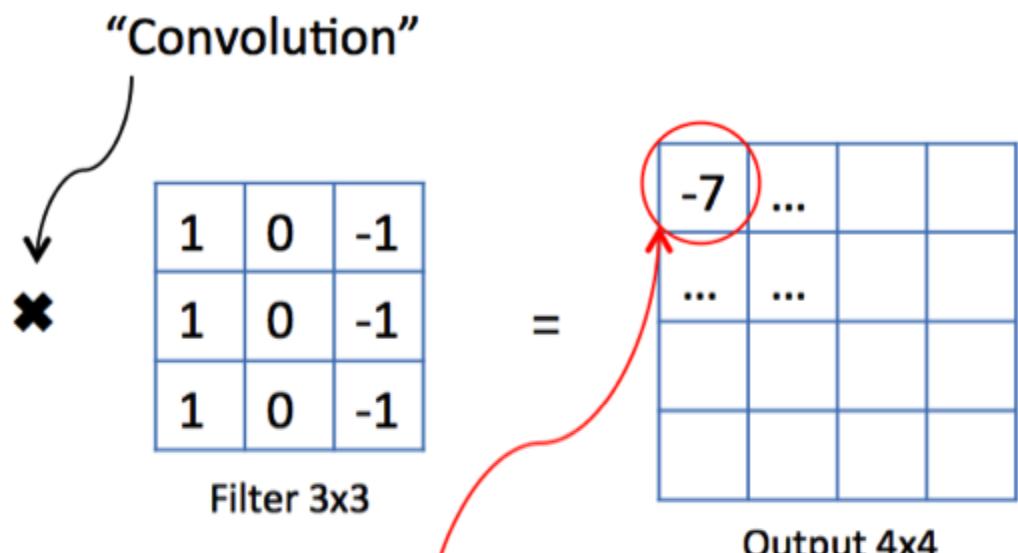


Redes Neuronales Convolucionales (CNN)

- ¿Cuál es el resultado de la siguiente convolución?

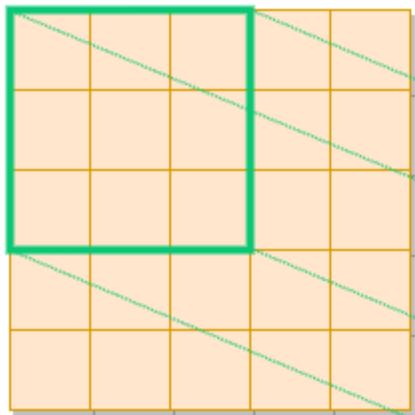
3	1	1	2	8	4
1	0	7	3	2	6
2	3	5	1	1	3
1	4	1	2	6	5
3	2	1	3	7	2
9	2	6	2	5	1

Original image 6x6



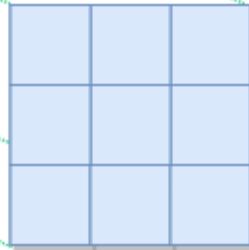
Result of the element-wise product and sum of the filter matrix and the orginal image

Input D x D: 5 x 5

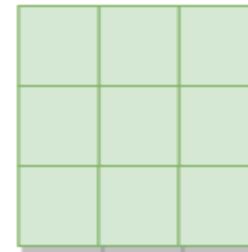


Padding VALID
Output dimension = $D - N + 1$
 $5 - 3 + 1 = 3$

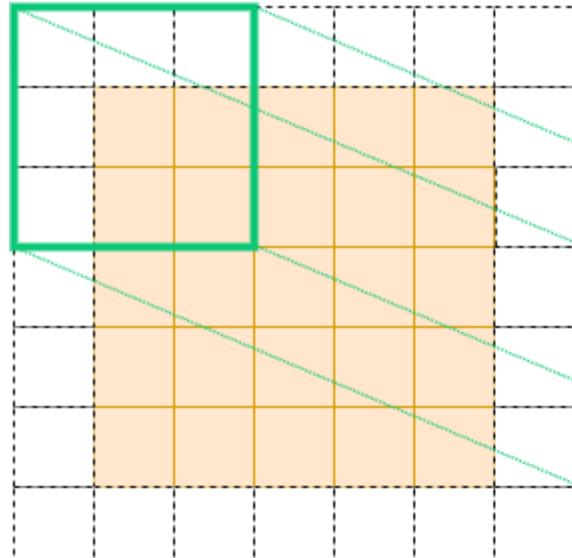
Filter N x N: 3 x 3



Output: 3 x 3



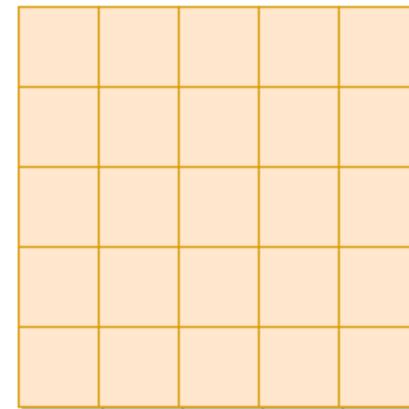
Input D x D: 5 x 5
Plus added padding of size 1



Padding SAME
Output dimension = Input dimension

Filter N x N: 3 x 3

Output: 5 x 5

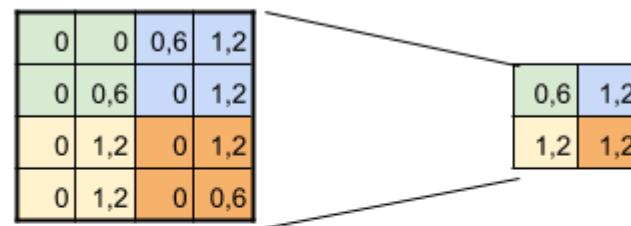


Podemos elegir diferentes estrategias de convolución

Redes Neuronales Convolucionales (CNN)

Pooling

- Calcula estadísticas por grupos de píxeles
- Reduce complejidad computacional y evita el overfitting
- Es invariante al escalado y pequeñas traslaciones, manteniendo las características más importantes que detectó cada convolución
- Suele utilizarse el max-pooling, que mantiene las características más destacadas



SUBSAMPLING:
Aplico Max-Pooling de 2x2
y reduzco mi salida a la mitad

Redes Neuronales Convolucionales (CNN)

Ejemplo

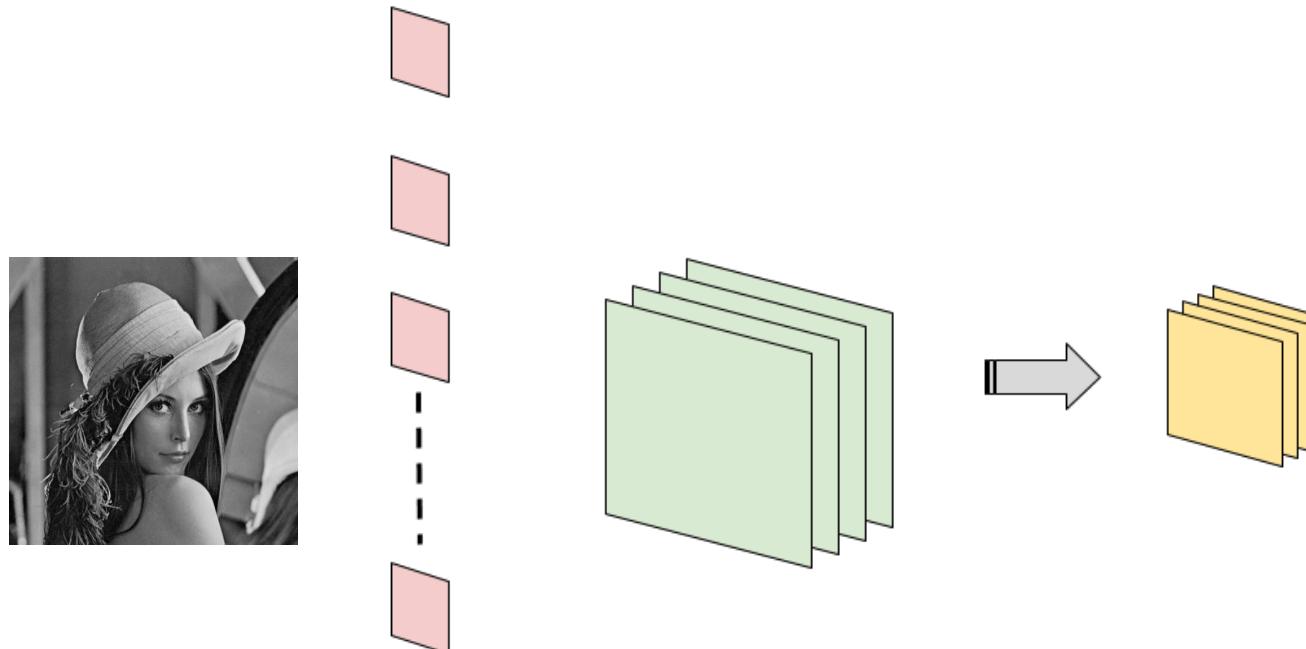


IMAGEN DE
ENTRADA
 $28 \times 28 \times 1$

784 px

Aplico 32 kernels
de 3x3
y Función de
Activación ReLu

Obtengo 32
Feature
Mapping de
 $28 \times 28 \times 1$

25.088 px

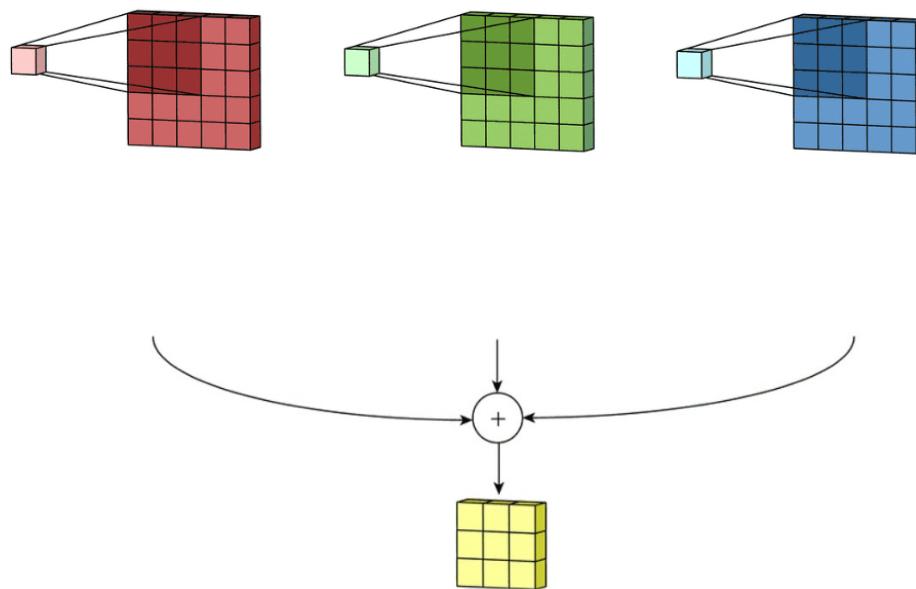
Aplico
Max-Pooling
 2×2

Obtengo 32
Salidas
 $14 \times 14 \times 1$

6.272 px

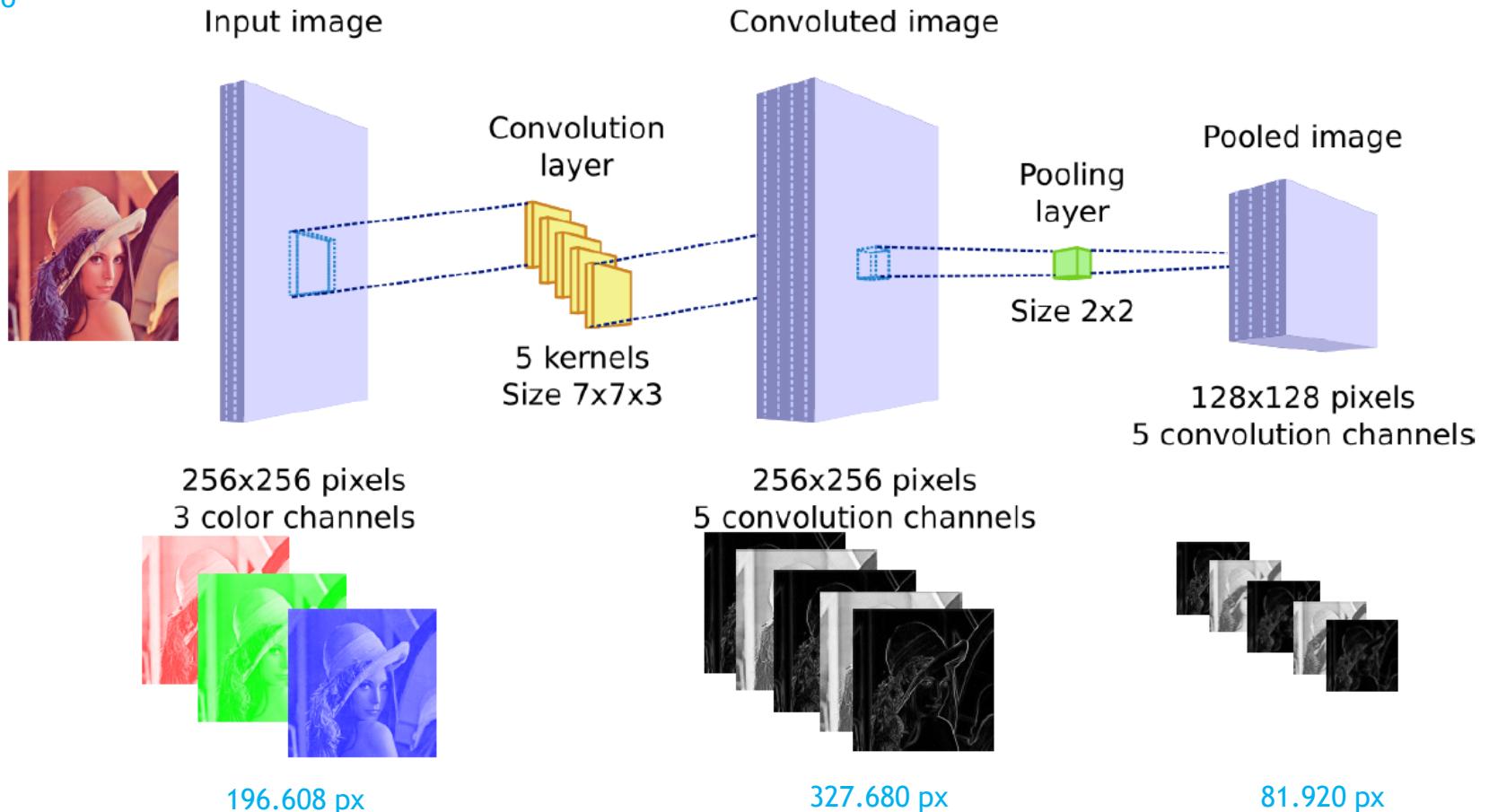
Redes Neuronales Convolucionales (CNN)

- Con imágenes RGB, cada kernel se aplica a cada canal separadamente, y posteriormente se suman los resultados



Redes Neuronales Convolucionales (CNN)

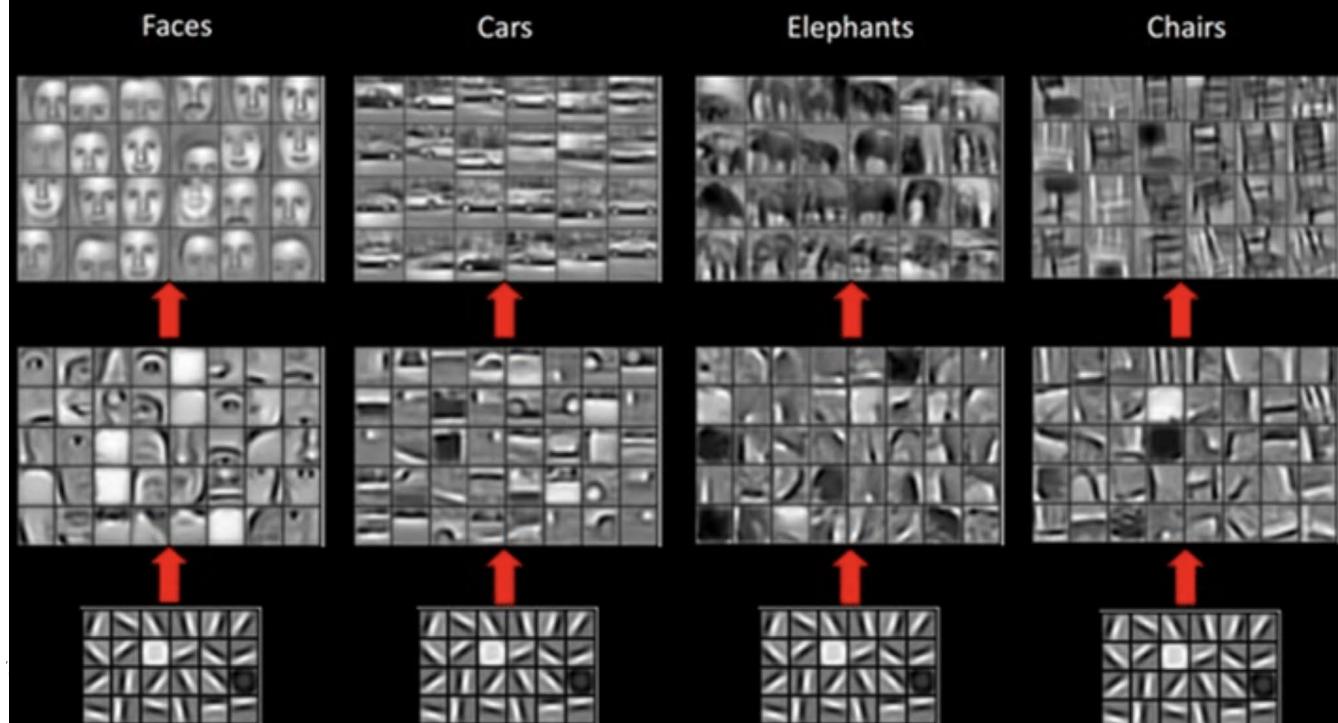
Ejemplo



Redes Neuronales Convolucionales (CNN)

- La primera convolución es capaz de detectar características primitivas como líneas o curvas
- A medida que se realicen más convoluciones, la red reconocerá formas más complejas

Tercera capa

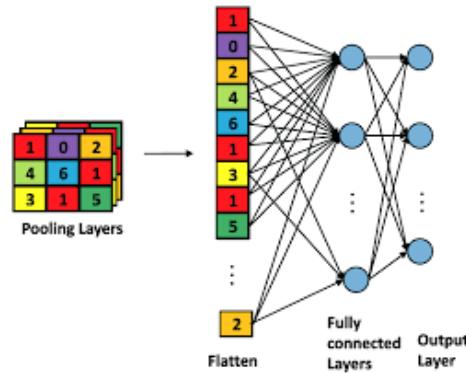


Segunda capa

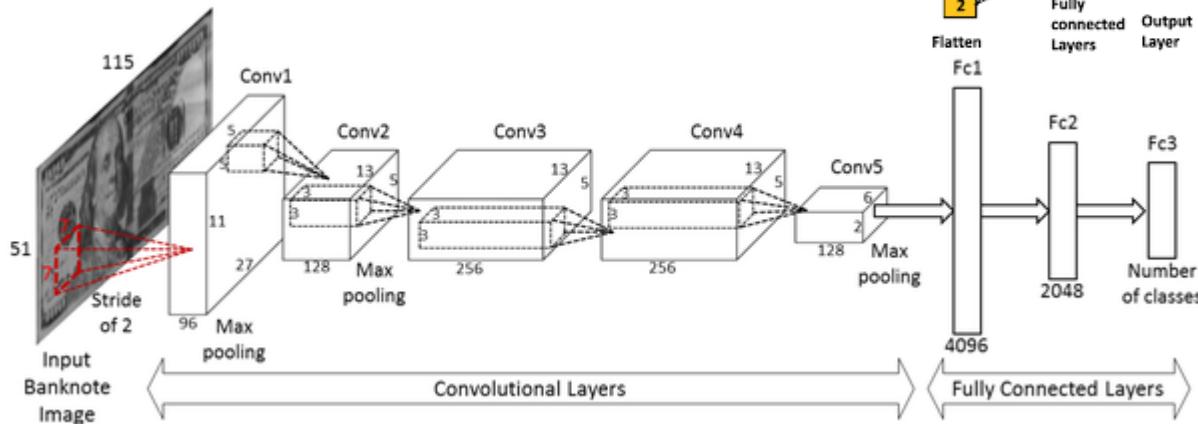
Primera capa

Redes Neuronales Convolucionales (CNN)

- Tras las capas de convolución+pooling, se aplana los datos y se utiliza una red neuronal “tradicional”, teniendo una neurona de salida por cada categoría
- Al final de la red, se aplica una función denominada *softmax*, encargada de transformar los valores de entrada a probabilidades
- La clase con mayor probabilidad será el resultado de la predicción

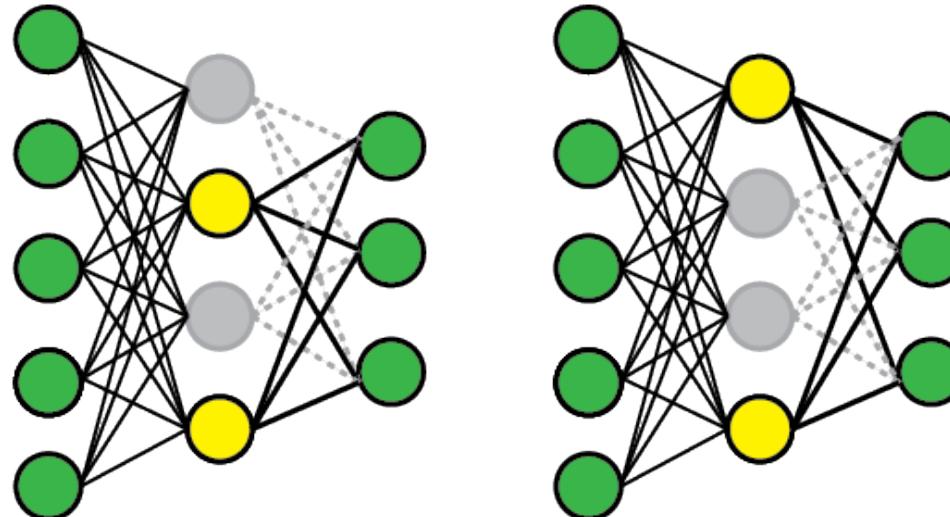


LOGITS SCORES	SOFTMAX	PROBABILITIES
2.0 →	$S(y_i) = \frac{e^{y_i}}{\sum_j e^{y_j}}$	→ $p = 0.7$
1.0 →		→ $p = 0.2$
0.1 →		→ $p = 0.1$



Redes Neuronales Convolucionales (CNN)

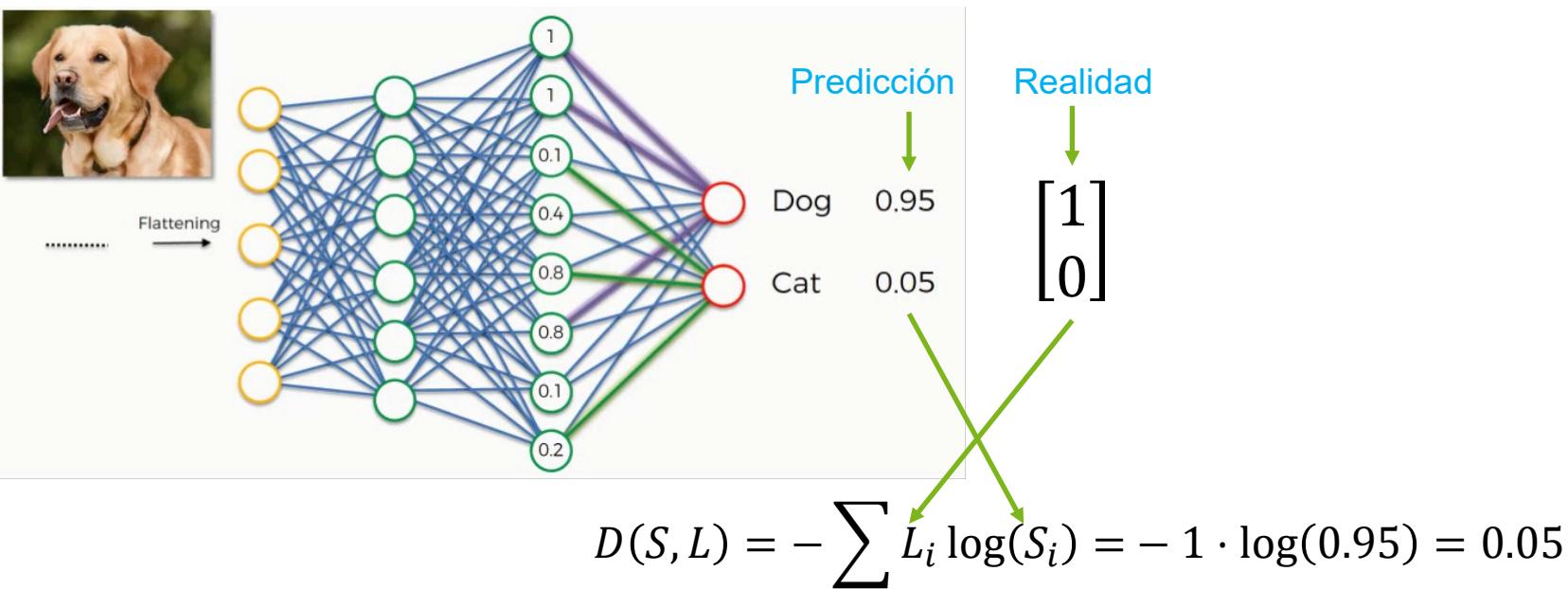
- En la red neuronal final, se suele utilizar un proceso de regularización conocido como **dropout**, con el objetivo de evitar overfitting
- Esta técnica deshabilita aleatoriamente neuronas durante el entrenamiento, forzando a la red a aprender múltiples representaciones independientes del mismo dato



Redes Neuronales Convolucionales (CNN)

Función de error: Cross Entropy

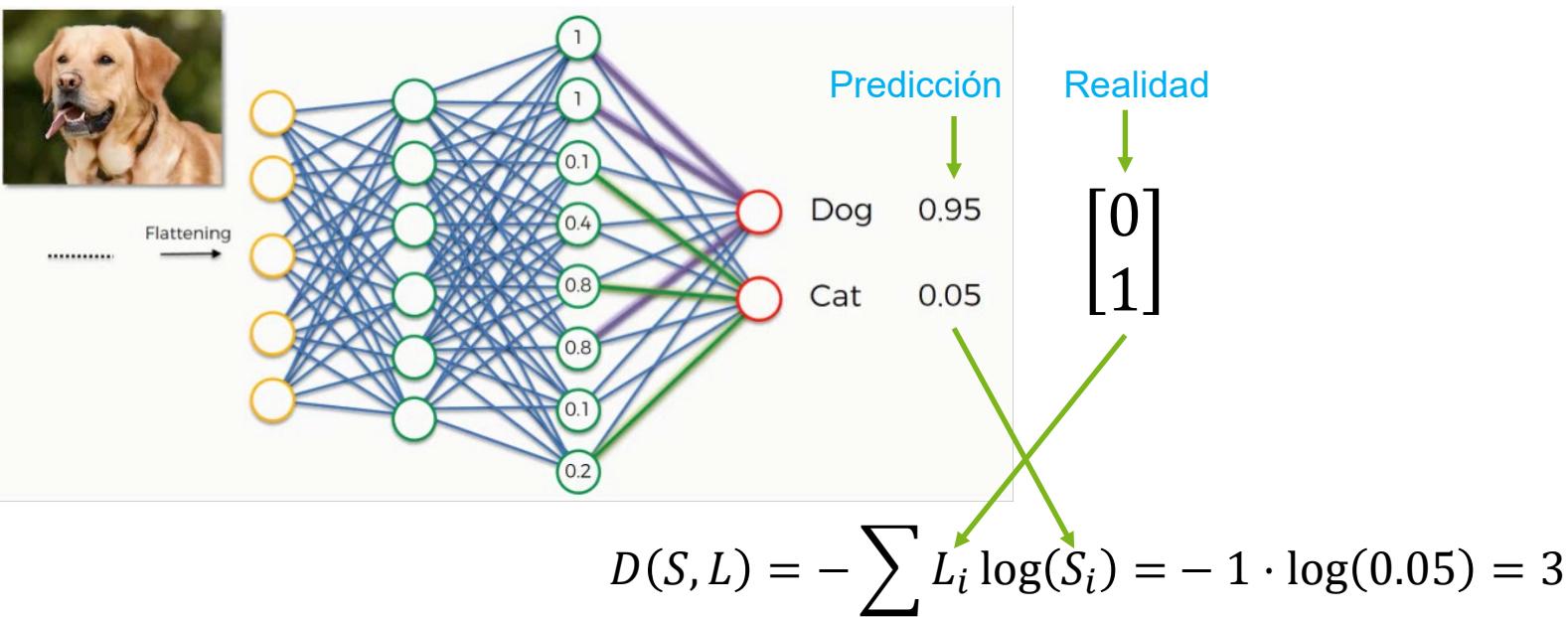
- Suele emplearse esta función de error para entrenar la red neuronal



Redes Neuronales Convolucionales (CNN)

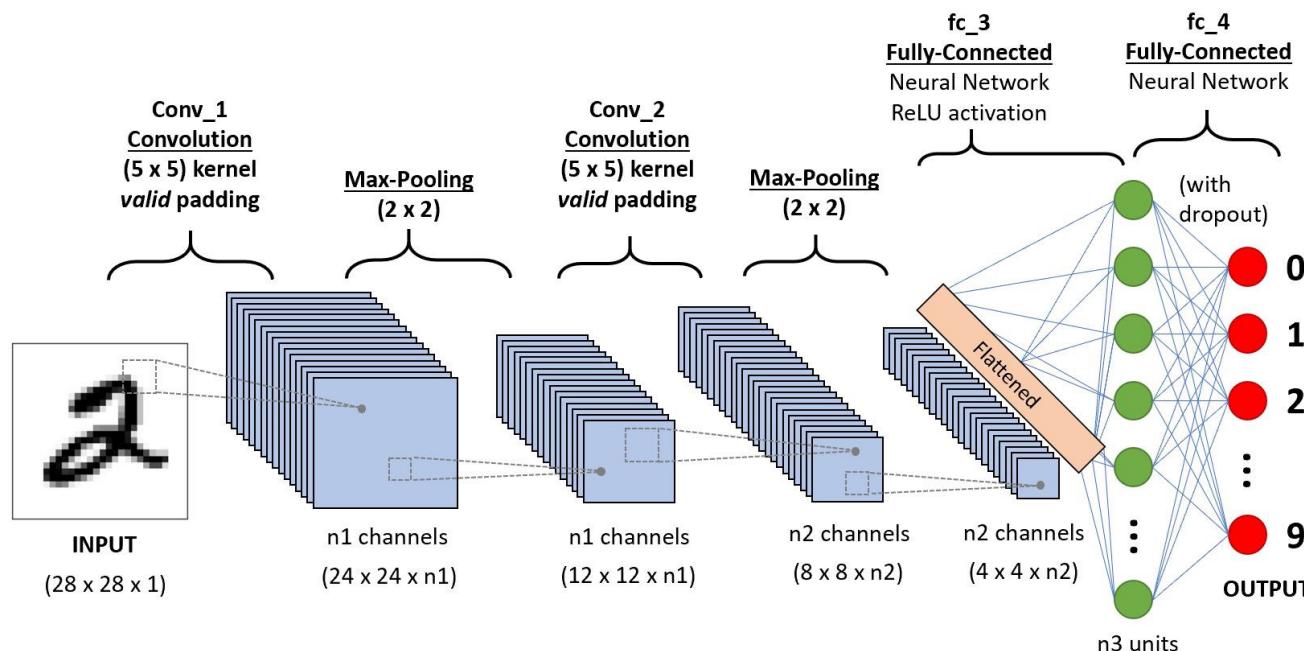
Función de error: Cross Entropy

- Suele emplearse esta función de error para entrenar la red neuronal



Redes Neuronales Convolucionales (CNN)

- Por tanto, la arquitectura consiste en capas anidadas de convoluciones+pooling, y finalmente capas interconectadas (“fully connected”)
- El tipo de arquitectura variará según el tipo de problema



Redes Neuronales Convolucionales (CNN)

- Ejemplo: ImageNet (concurso de reconocimiento de imágenes)
- Arquitectura ganadora (2012):

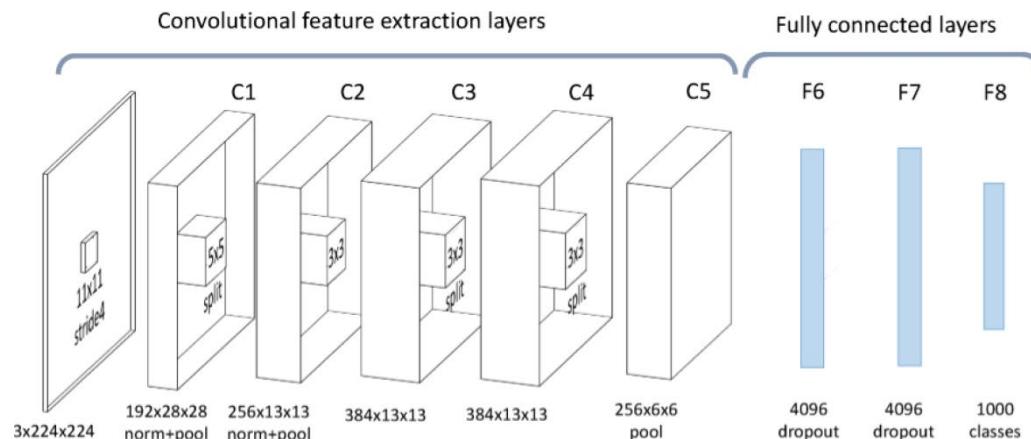
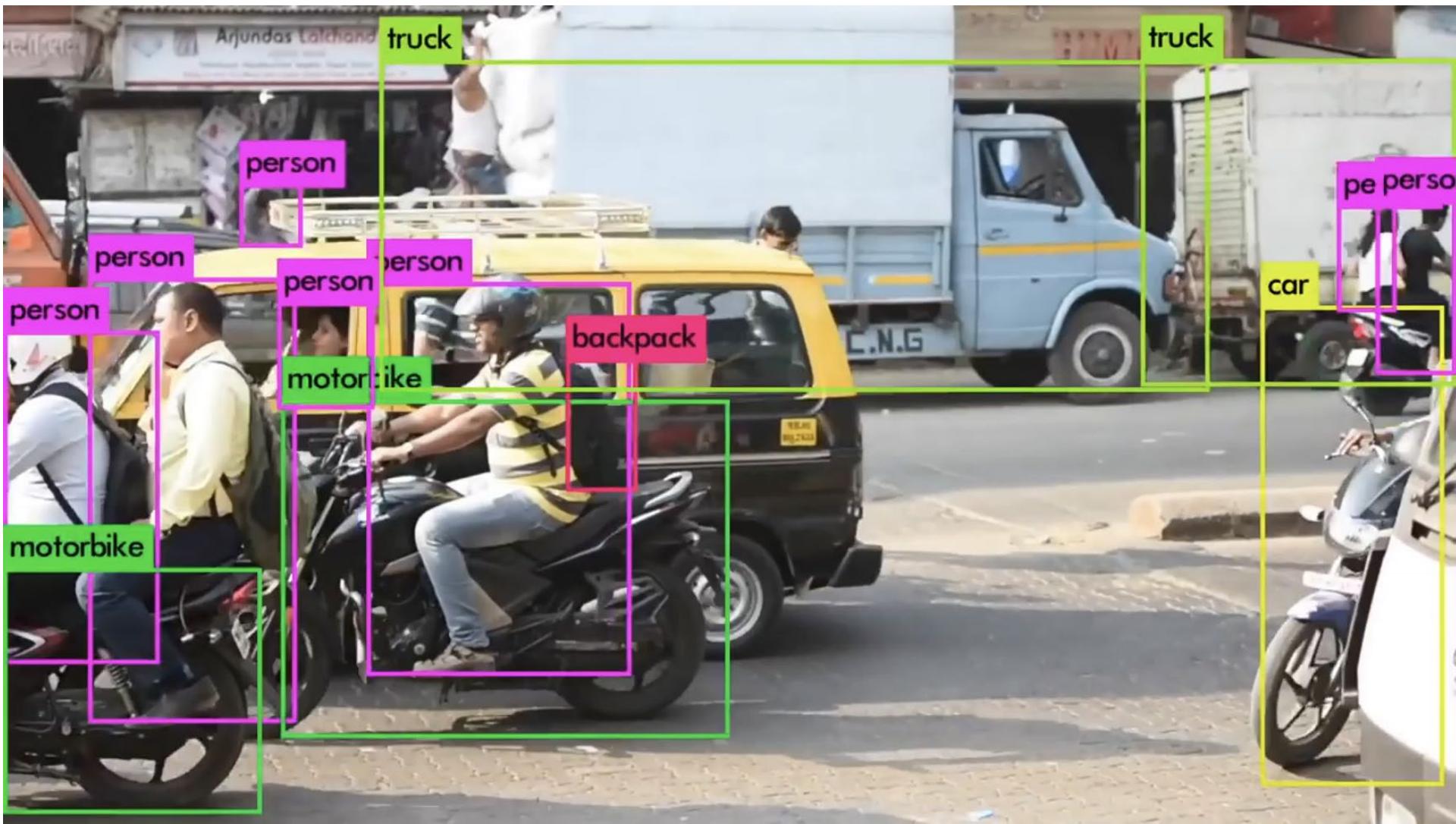


Fig. 2.3: Architecture for image recognition. The 2012 ILSVRC winner consists of eight layers [82]. Each layer performs a linear transformation (specifically, convolutions in layers C1–C5 and matrix multiplication in layers F6–F8) followed by nonlinear transformations (rectification in all layers, contrast normalization in C1–C2, and pooling in C1–C2 and C5). Regularization with dropout noise is used in layers F6–F7.

Redes Neuronales Convolucionales (CNN)



Redes Neuronales Convolucionales (CNN)

Algoritmo You Only Look Once (YOLO)

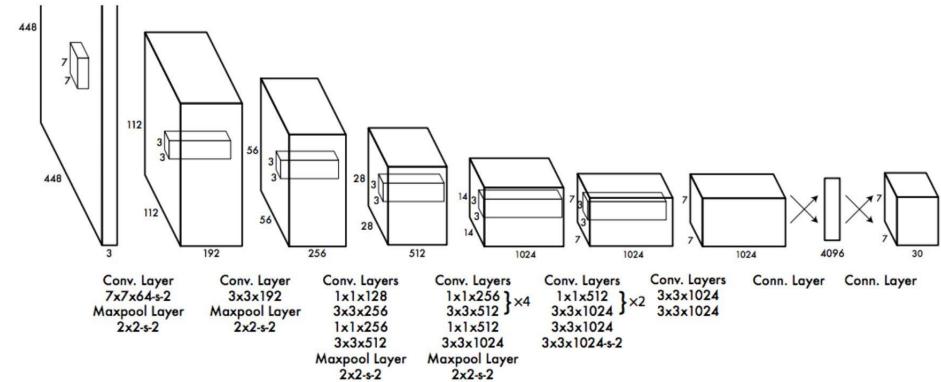
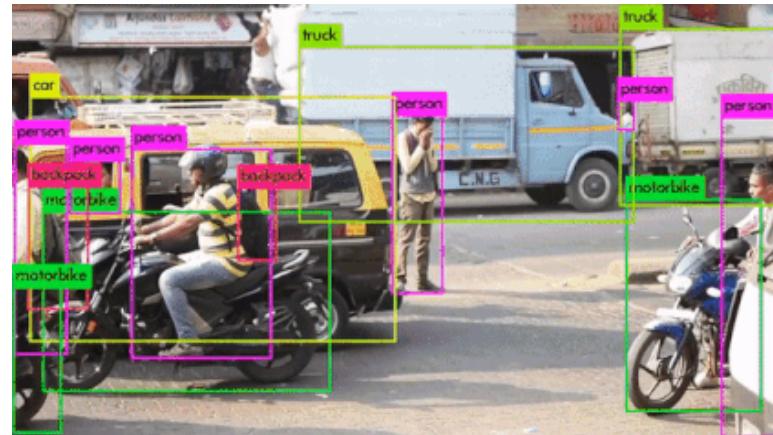


Figure 3: The Architecture. Our detection network has 24 convolutional layers followed by 2 fully connected layers. Alternating 1×1 convolutional layers reduce the features space from preceding layers. We pretrain the convolutional layers on the ImageNet classification task at half the resolution (224×224 input image) and then double the resolution for detection.

Redes Neuronales Convolucionales (CNN)



"man in black shirt is playing guitar."



"construction worker in orange safety vest is working on road."



"two young girls are playing with lego toy."



"girl in pink dress is jumping in air."

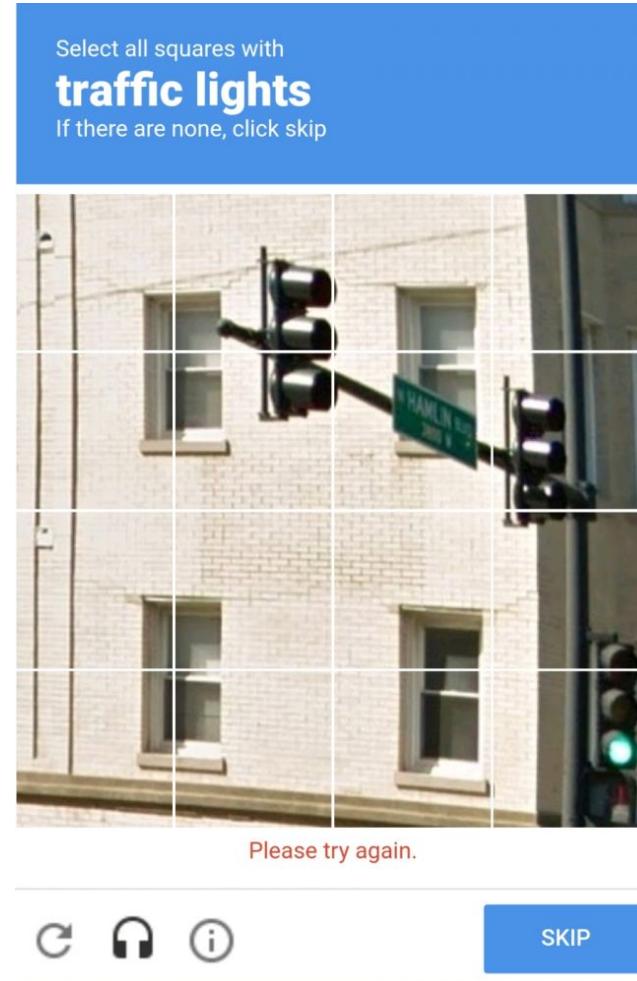
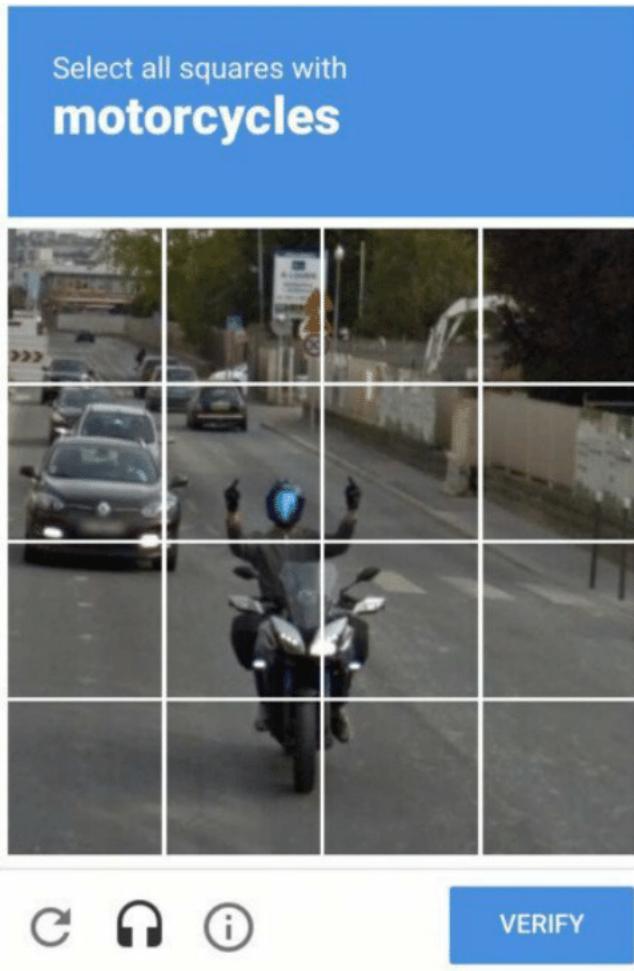


"black and white dog jumps over bar."

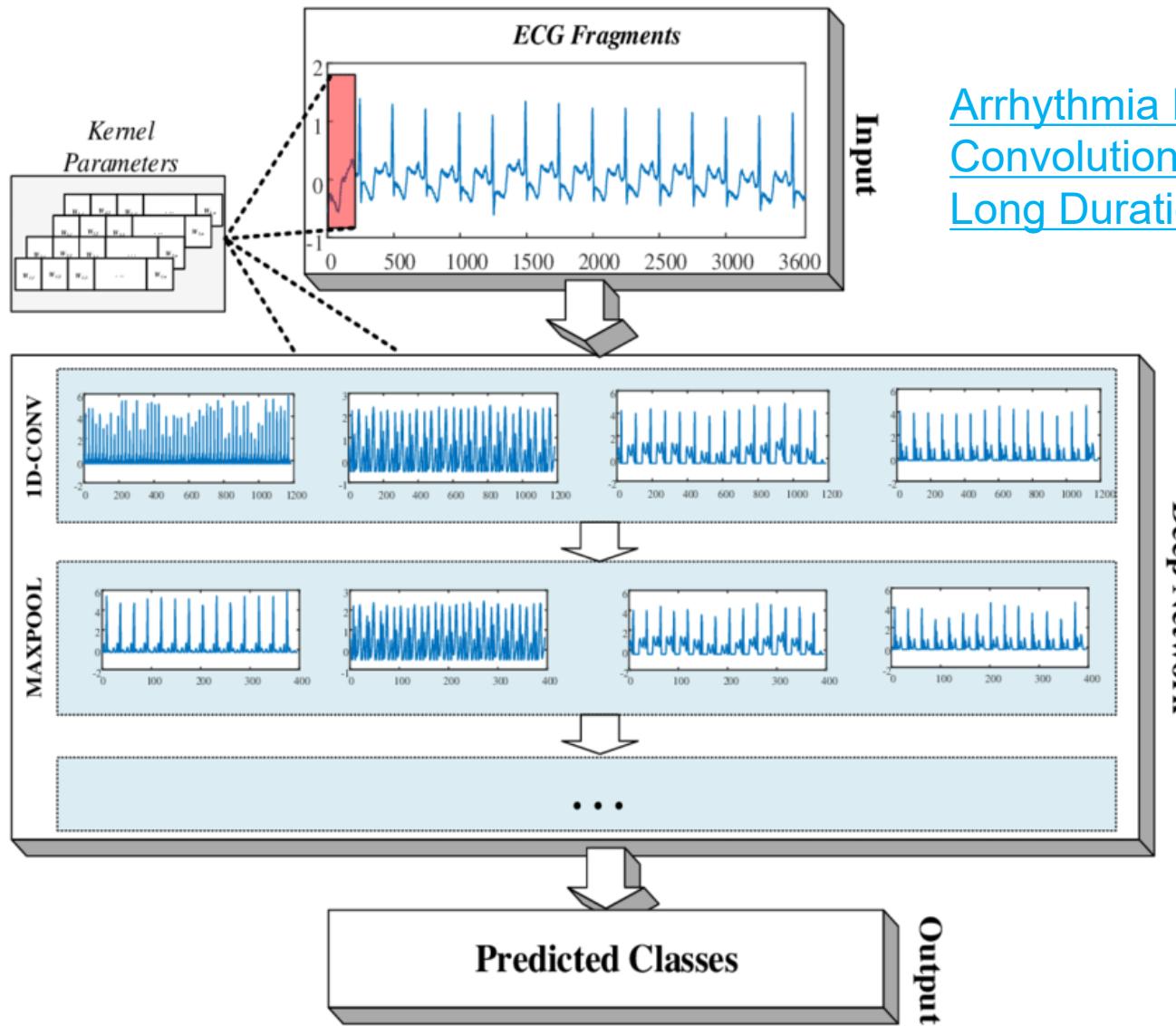


"young girl in pink shirt is swinging on swing."

Redes Neuronales Convolucionales (CNN)



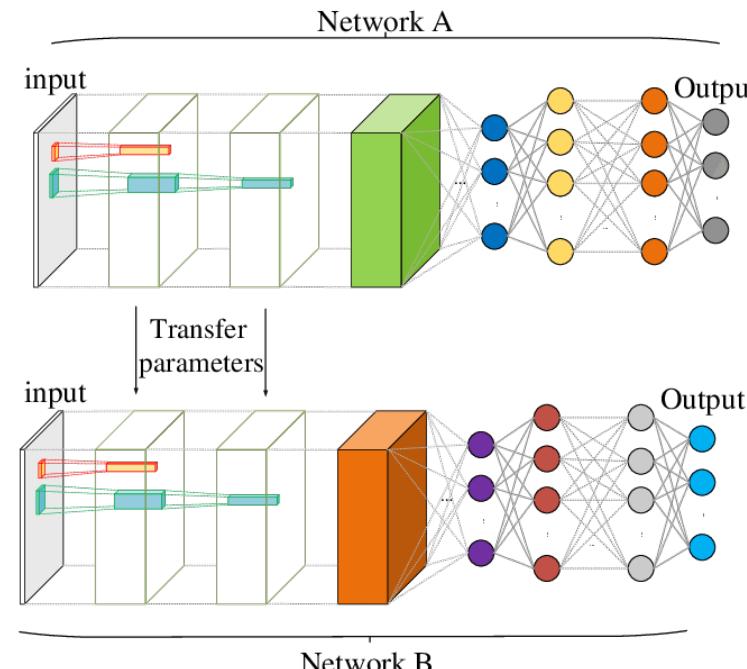
No solo en imágenes...



Transfer Learning



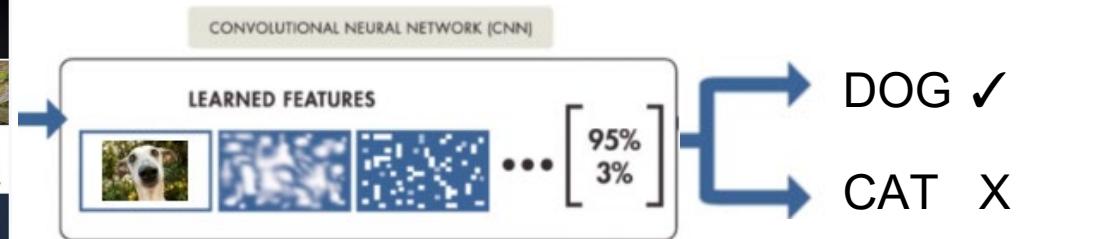
- Si no tenemos suficientes ejemplos de entrenamiento, podemos utilizar una red pre-entrenada y cambiarle las etiquetas salida.
- Las características de bajo nivel extraídas por la primera red, pueden servir para la nueva tarea



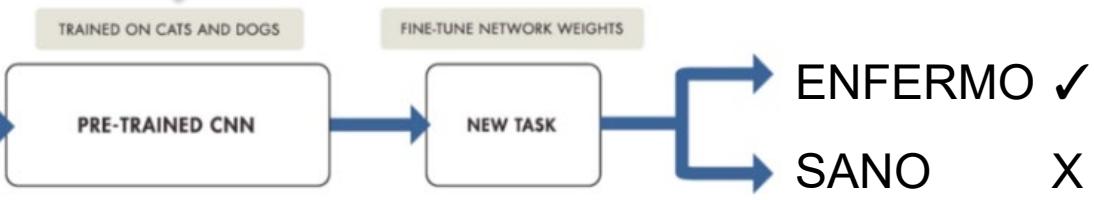
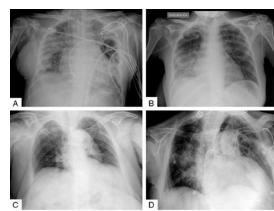


Transfer Learning

1 MILLÓN DE IMÁGENES

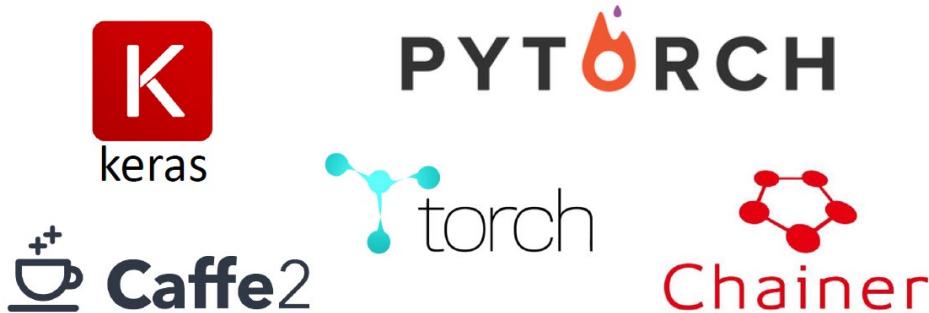


100 IMÁGENES



Deep Learning frameworks

- Alto nivel



- Bajo nivel



Lectura recomendada

- **How to Grid Search Hyperparameters for Deep Learning Models in Python With Keras**

<https://machinelearningmastery.com/grid-search-hyperparameters-deep-learning-models-python-keras/>

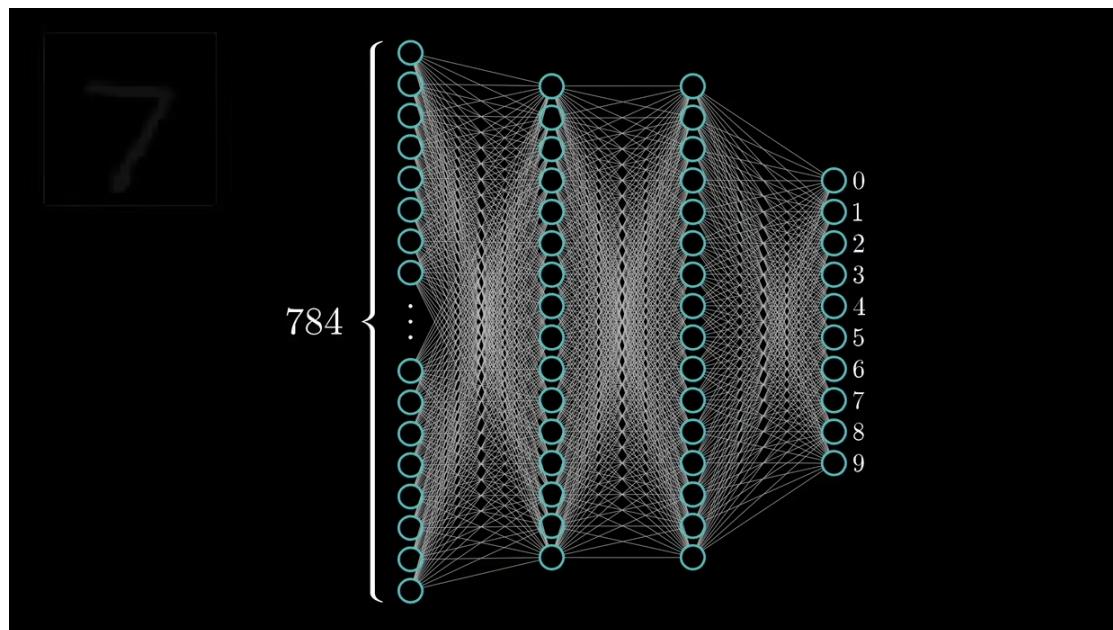


Práctica

Reconocimiento de dígitos manuscritos

- El dataset [MNIST](#) contiene 70.000 imágenes de tamaño 28x28 píxeles con dígitos del 0 al 9 escritos a mano
- Utilizaremos la librería [keras](#) de Python para crear clasificadores basados en redes neuronales
- [Enlace al notebook de Python](#)

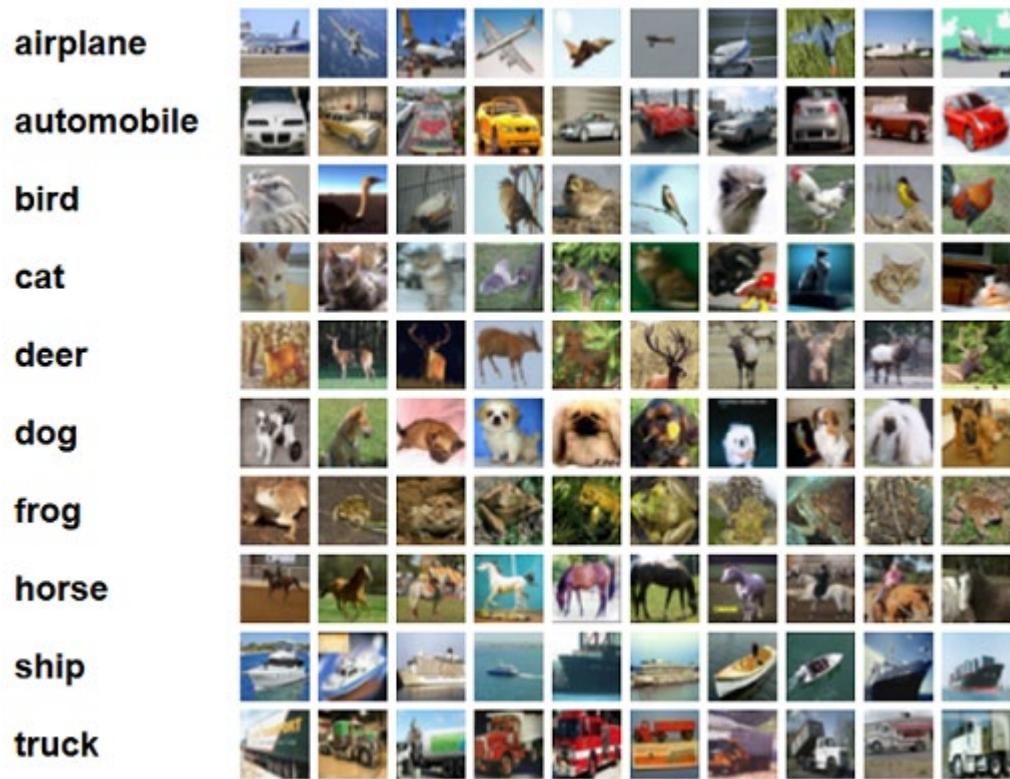
0 0 0 0 0 0 0 0 0 0 0 0 0 0
1 1 1 1 1 1 1 1 1 1 1 1 1 1
2 2 2 2 2 2 2 2 2 2 2 2 2 2
3 3 3 3 3 3 3 3 3 3 3 3 3 3
4 4 4 4 4 4 4 4 4 4 4 4 4 4
5 5 5 5 5 5 5 5 5 5 5 5 5 5
6 6 6 6 6 6 6 6 6 6 6 6 6 6
7 7 7 7 7 7 7 7 7 7 7 7 7 7
8 8 8 8 8 8 8 8 8 8 8 8 8 8
9 9 9 9 9 9 9 9 9 9 9 9 9 9



Do it yourself

Reconocimiento de imágenes

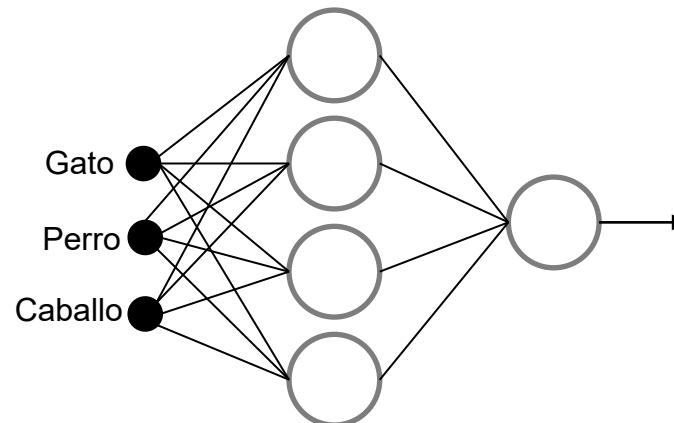
- El conjunto de datos CIFAR-10 consta de 60000 imágenes en color de 32x32 con 10 clases, y 6000 imágenes por clase. Hay 50000 imágenes de entrenamiento y 10000 imágenes de prueba.
- [Enlace al notebook de Python](#)



Embeddings

- ¿Cómo codificarías esta variable de entrada a una red neuronal?

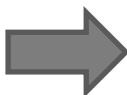
Variable 1	Gato	Perro	Caballo
Gato	1	0	0
Perro	0	1	0
Gato	1	0	0
Caballo	0	0	1



Embeddings

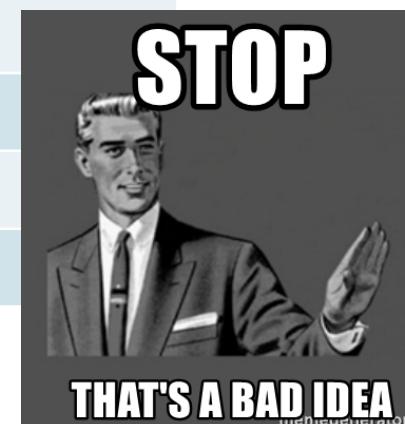
- ¿Y esta?

	Variable1
1	Abeja
2	Alce
3	Ardilla
4	Águila
5	Avispa
6	Avestruz
	...
949	Zarigüeya
950	Zorro



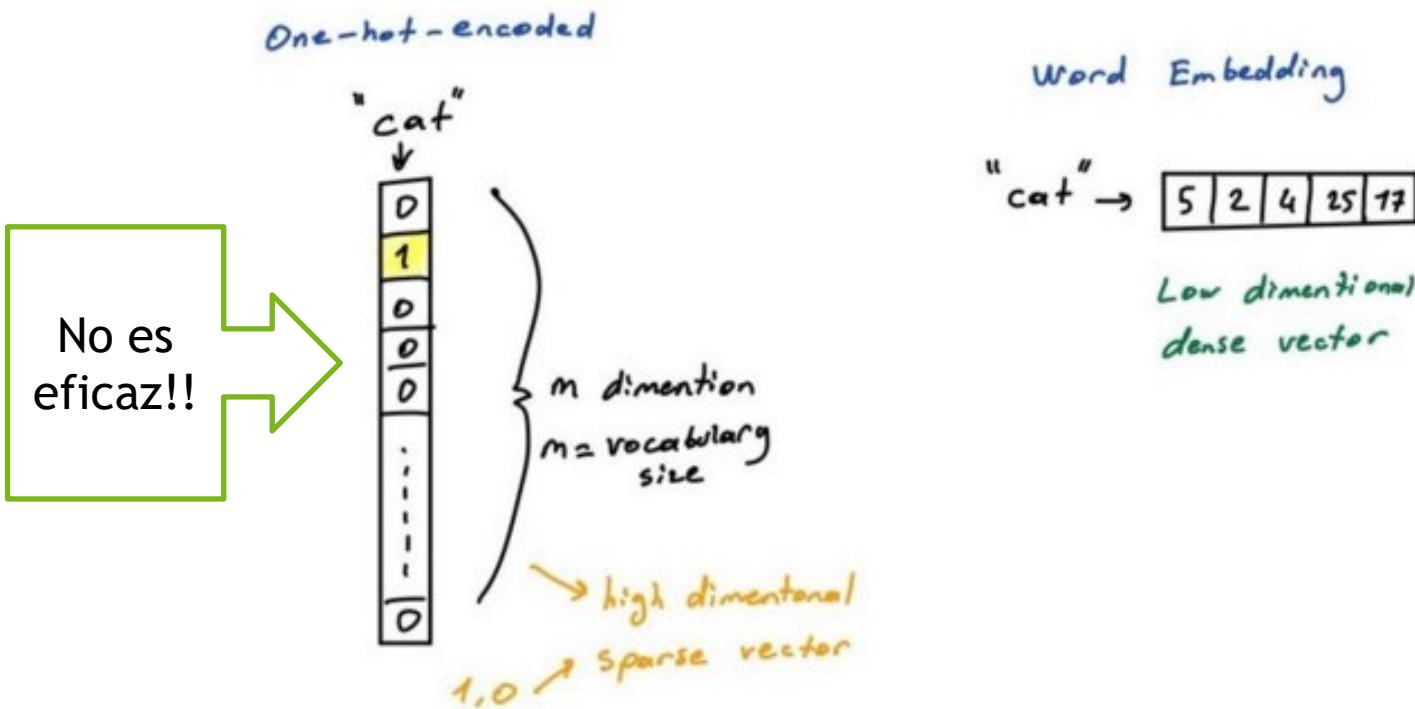
	Abeja	Alce	Ardilla	...
1	1	0	0	0
2	0	1	0	0
3	0	0	1	0
4	0	0	0	0
5	0	0	0	0
6	0	0	0	0

949	0	0	0	0
950	0	0	0	0

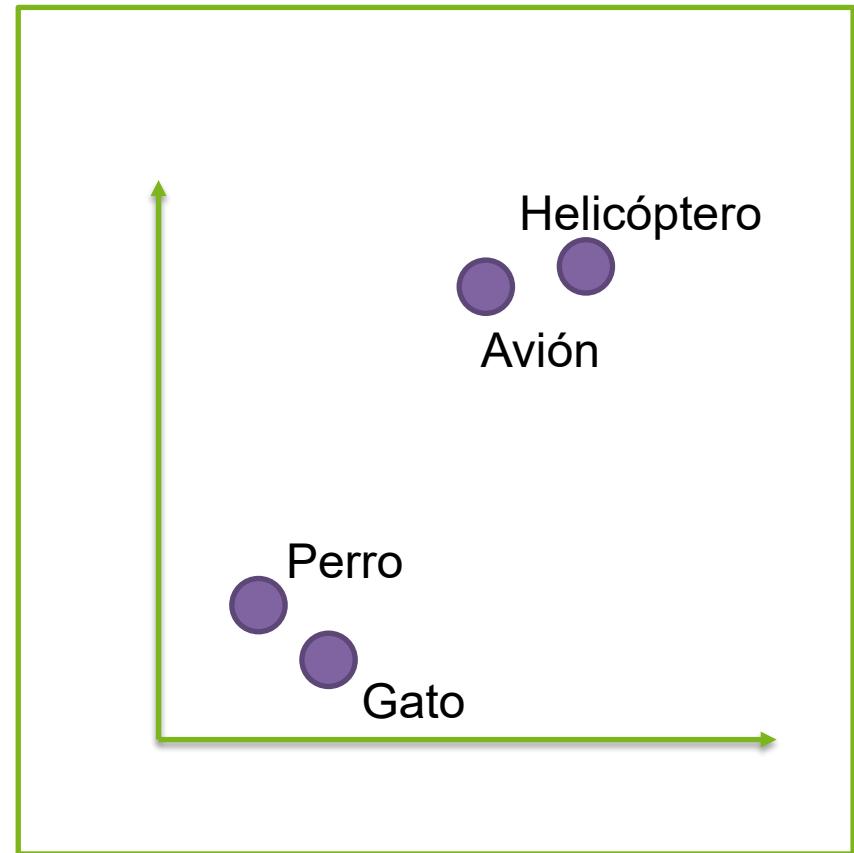
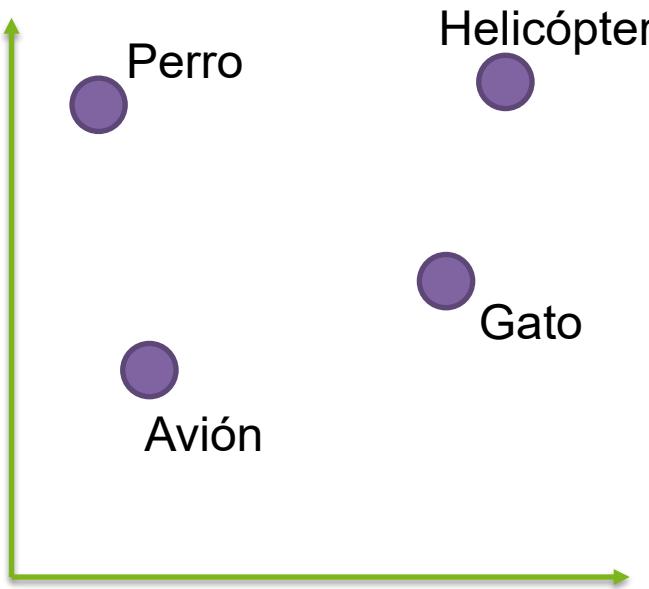


Embeddings

- La técnica de embedding consiste en proyectar una entrada en un espacio diferente
- En redes neuronales, los embeddings son representaciones de variables discretas en un espacio de dimensión baja



Embeddings

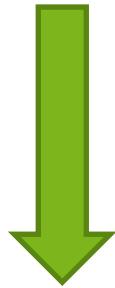


¿Qué representación vectorial es mejor?

Embeddings

TOKENIZAR

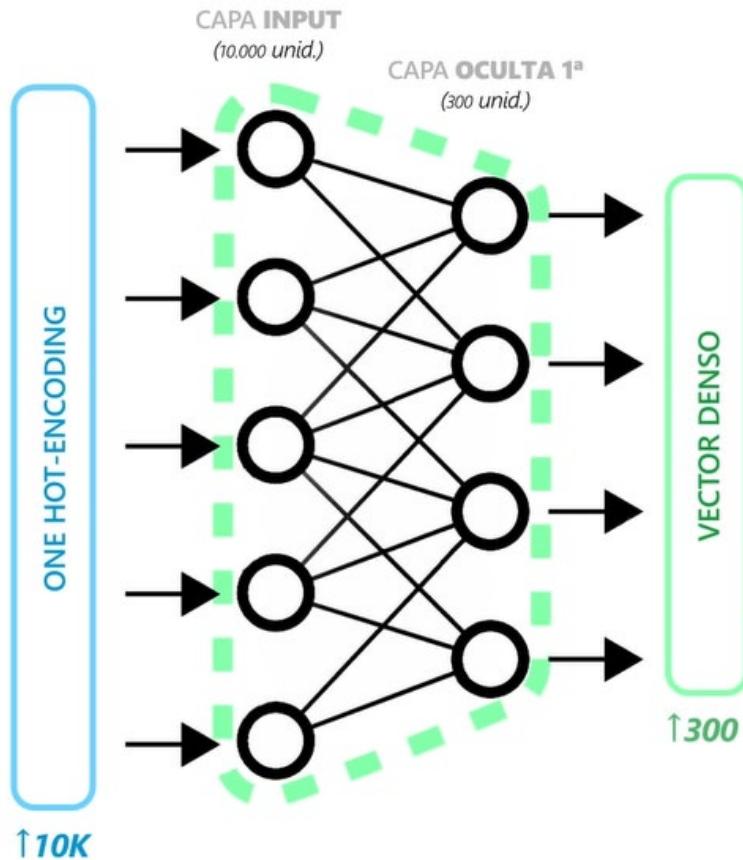
```
sentences = [  
    'Today is a sunny day',  
    'Today is a rainy day',  
    'Is it sunny today?'  
]  
  
{'today': 1, 'is': 2, 'a': 3, 'sunny': 4, 'day': 5, 'rainy': 6, 'it': 7}
```



```
[ [1, 2, 3, 4, 5], [1, 2, 3, 6, 5], [2, 7, 4, 1] ]
```

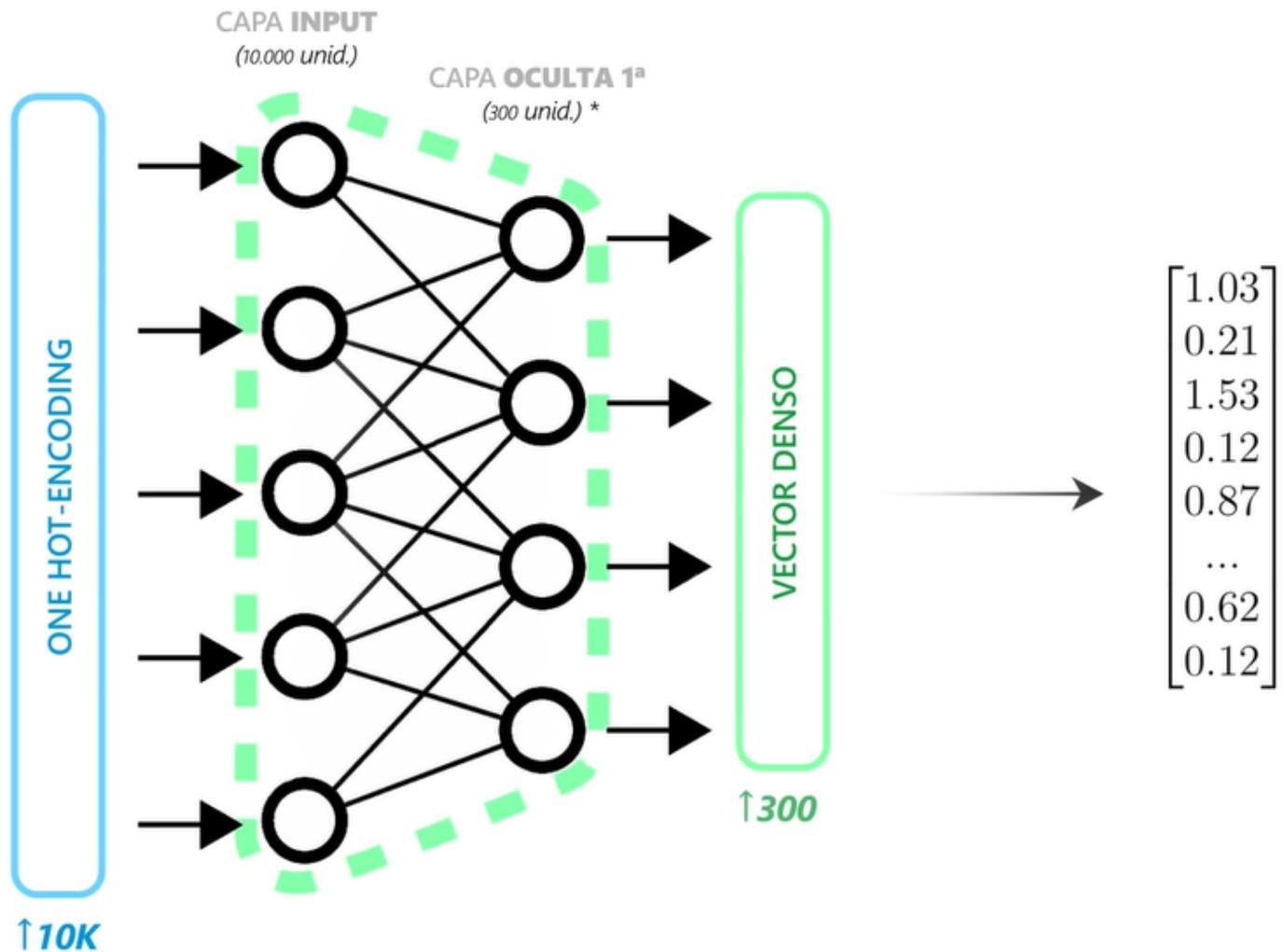
Embeddings

- Imaginemos que tenemos un vocabulario con 10.000 palabras



- Al pasar las palabras por una red neuronal, podemos añadir capas con menos neuronas para comprimir la información
- Tenemos que entrenar la red, dependiendo del tipo de tarea que queramos resolver (análisis de sentimiento, reviews positivas/negativas, etc.)

Embeddings



Embeddings

PADDING

```
sentences = [  
    'Today is a sunny day',  
    'Today is a rainy day',  
    'Is it sunny today?',  
    'I really enjoyed walking in the snow today'  
]
```



```
[  
    [2, 3, 4, 5, 6],  
    [2, 3, 4, 7, 6],  
    [3, 8, 5, 2],  
    [9, 10, 11, 12, 13, 14, 15, 2]  
]
```

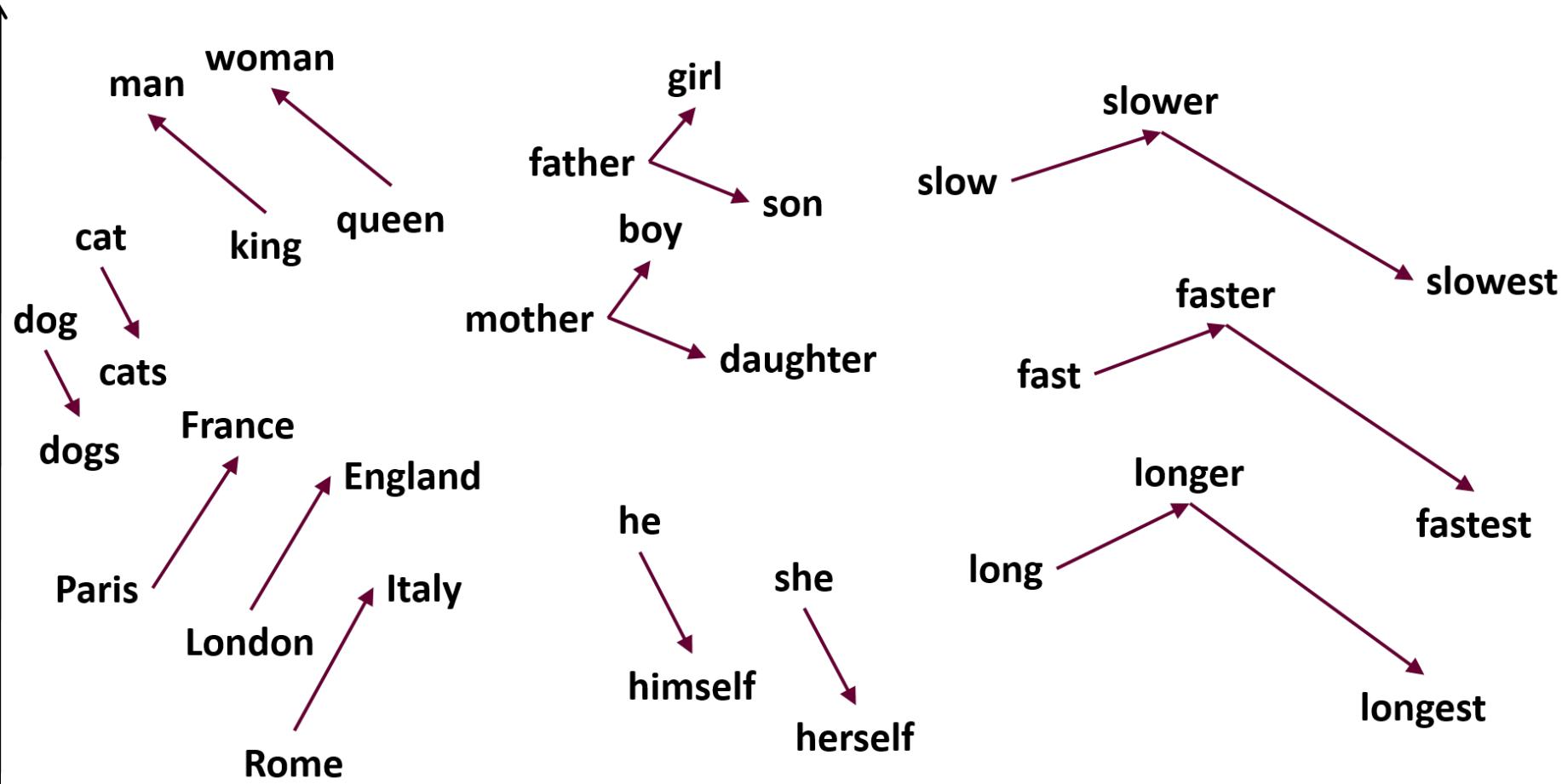
Para entrenar un algoritmo de ML siempre tendremos que establecer unos inputs con el mismo número de dimensiones.

Esto se soluciona rellenando con 0s mediante la técnica “padding”



```
[ [ 0 0 0 2 3 4 5 6 ]  
  [ 0 0 0 2 3 4 7 6 ]  
  [ 0 0 0 0 3 8 5 2 ]  
  [ 9 10 11 12 13 14 15 2 ] ]
```

Embeddings



Embeddings

- Ya existen embeddings pre-entrenados que convierte texto a vectores comprimidos
- El más conocido es Word2vec de Google
- <https://projector.tensorflow.org/>

Embeddings

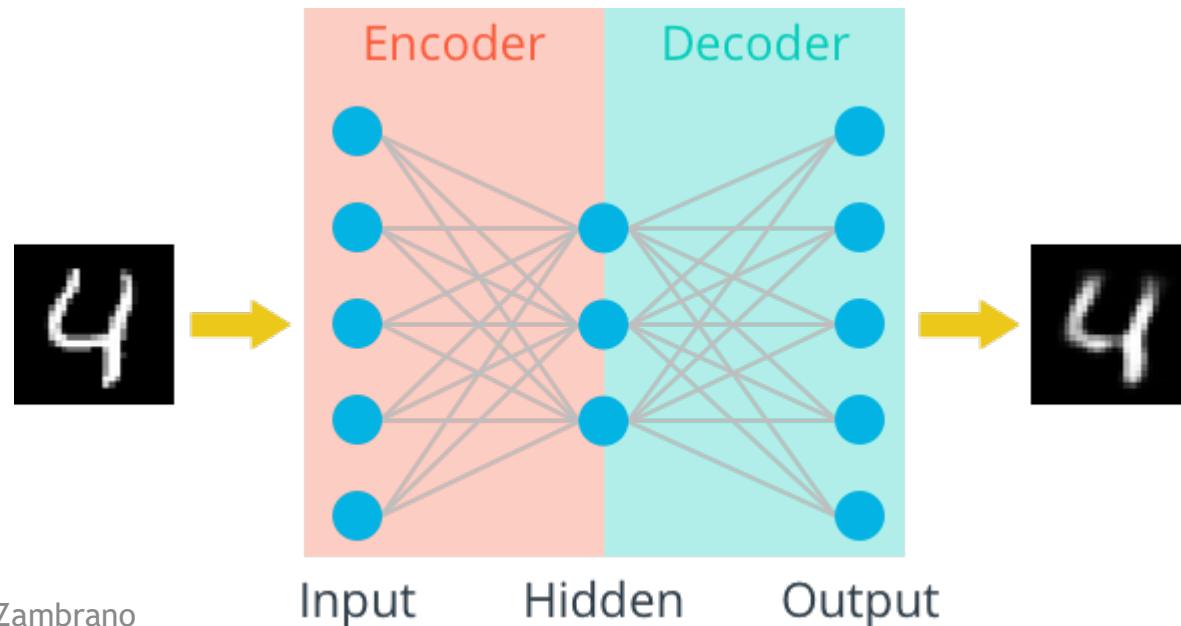
Práctica:

https://www.tensorflow.org/text/guide/word_embeddings

Autoencoders

Tipo de red neuronal que se utiliza para aprender codificaciones eficientes de datos no etiquetados

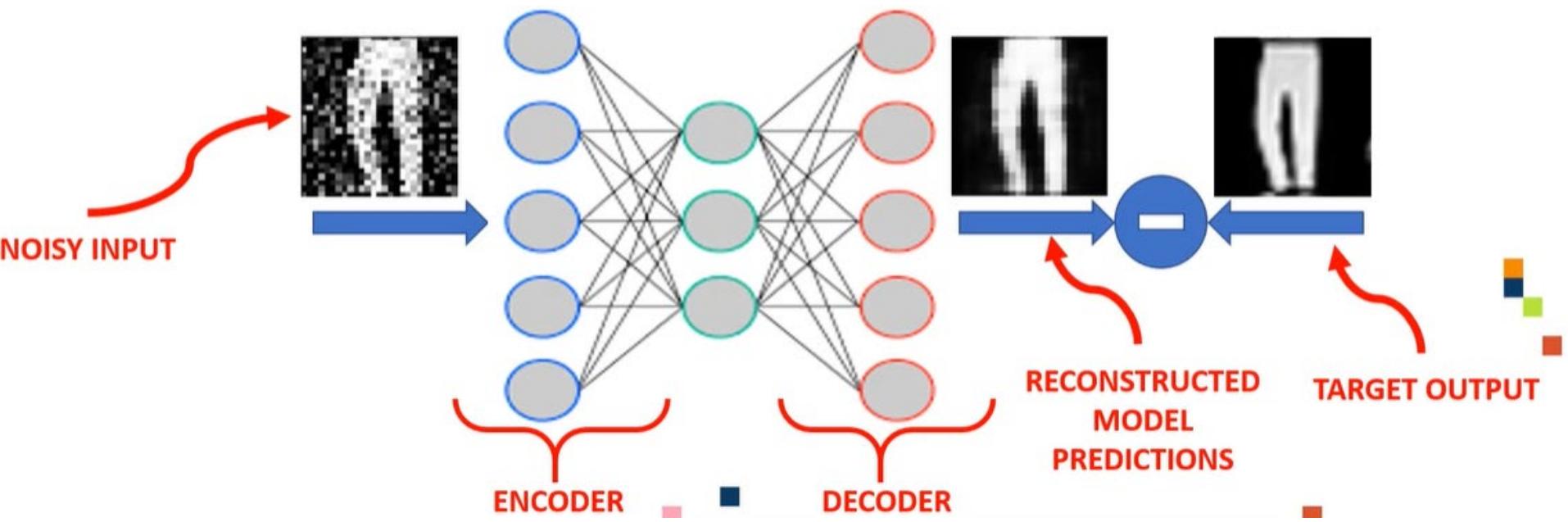
La información se comprime a un espacio de baja dimensión, para posteriormente reconstruirla



Autoencoders

Aplicaciones:

- Extracción de características
- Codificaciones y compresiones
- Detección de anomalías
- Limpieza de imágenes



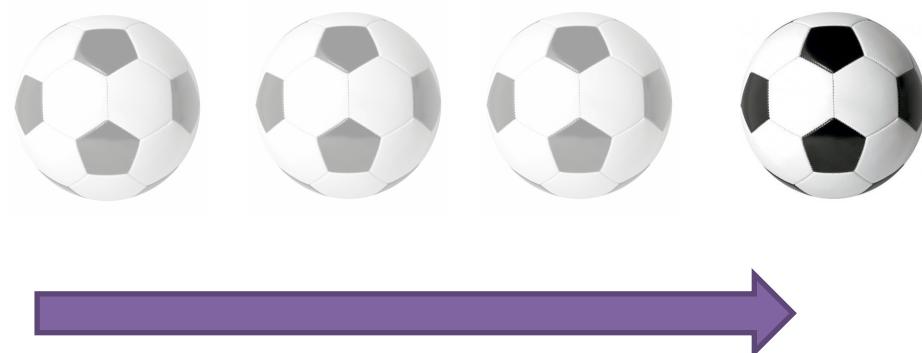
Redes Neuronales Recurrentes

¿Hacia dónde va a moverse el balón?



Redes Neuronales Recurrentes

¿Hacia dónde va a moverse el balón?



En ciertas ocasiones necesitamos conocer **secuencias** para realizar predicciones

Redes Neuronales Recurrentes

Muchos tipos de datos son secuenciales, por ejemplo el audio y el texto

¿Qué letra viene después?

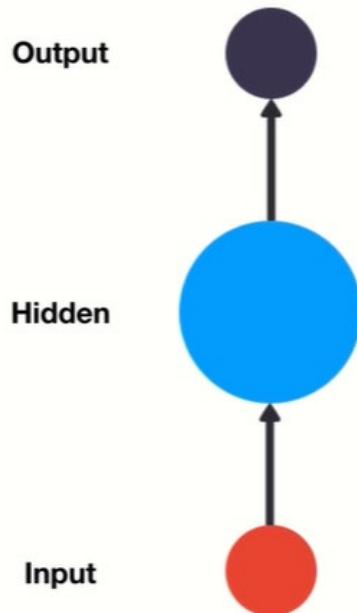
T - E - X - T -

¿Qué palabra viene después?

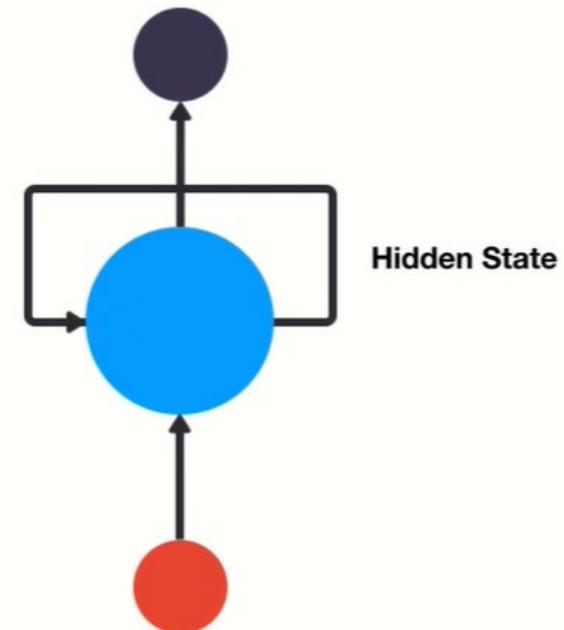
VOY - A - VISITAR -

Redes Neuronales Recurrentes

Las redes neuronales recurrentes son útiles para tratar datos secuenciales



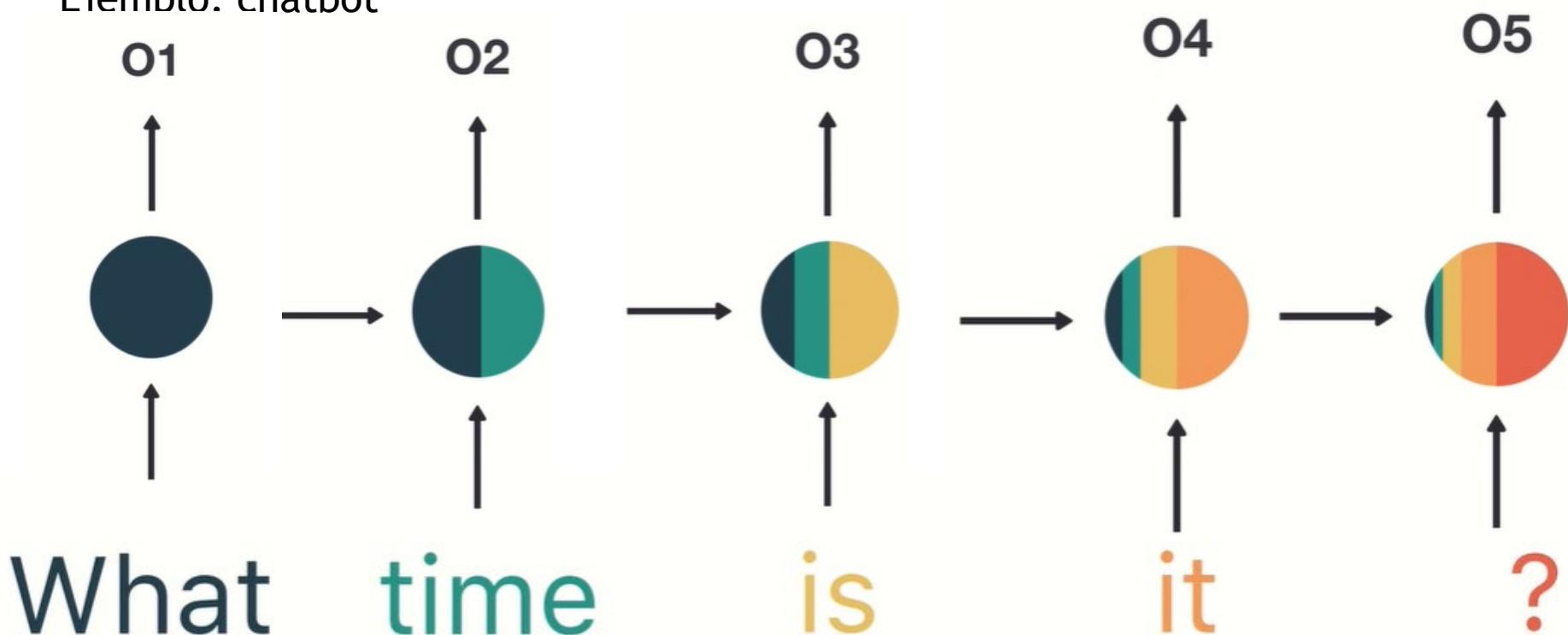
Red neuronal feedforward



Red neuronal recurrente
(almacena representaciones de entradas previas)

Redes Neuronales Recurrentes

Ejemplo: chatbot



Redes Neuronales Recurrentes

Ejemplo: chatbot

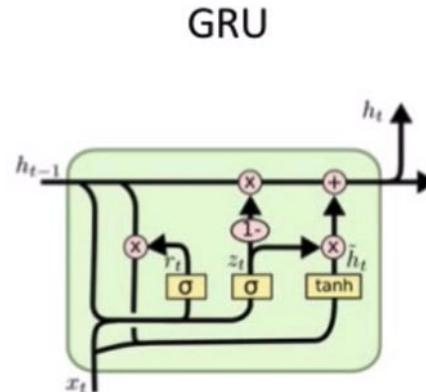
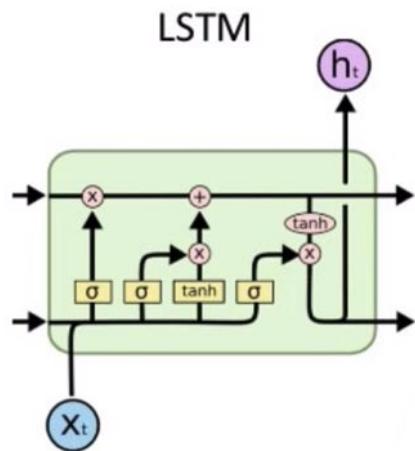
Asking for the time



Redes Neuronales Recurrentes

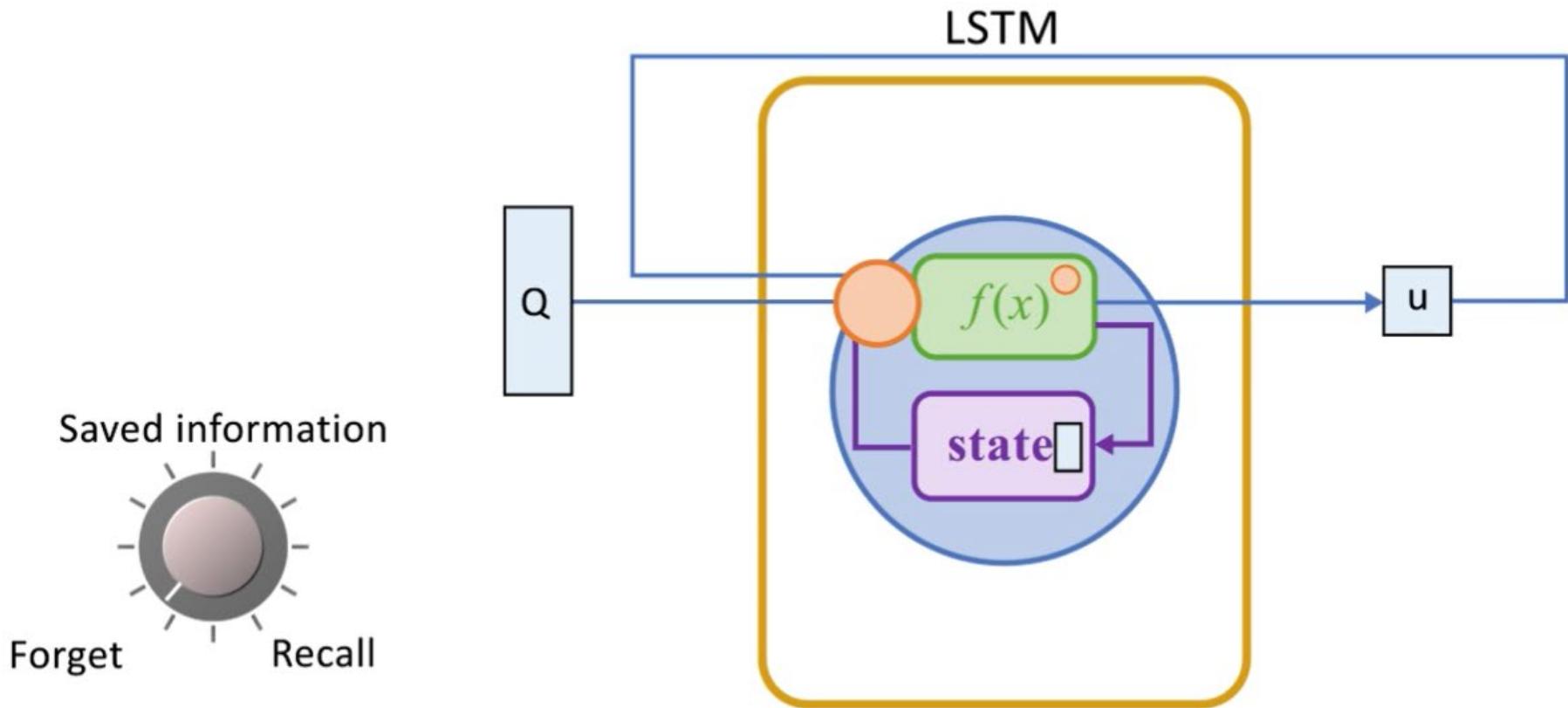
Los tipos de redes neuronales recurrentes más utilizados son:

- LSTM (Long Short Term Memory)
- GRU (Gated Recurrent Unit)

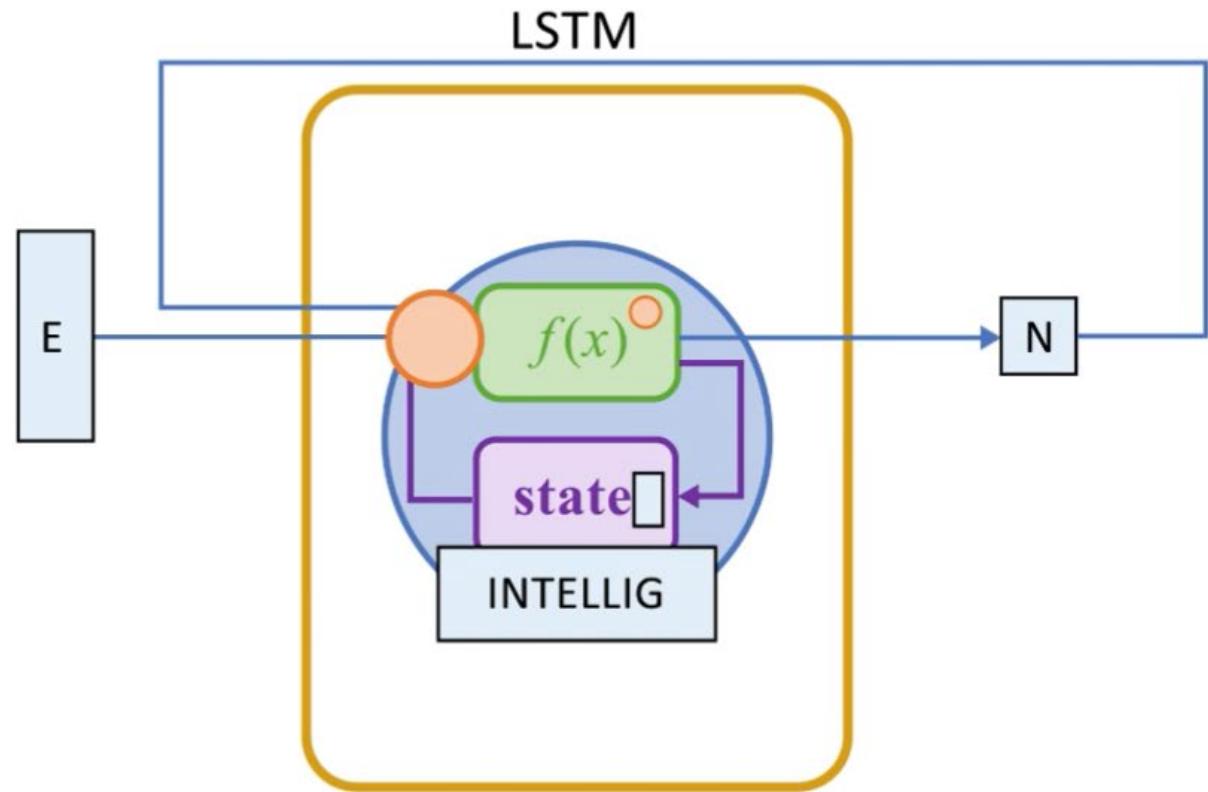
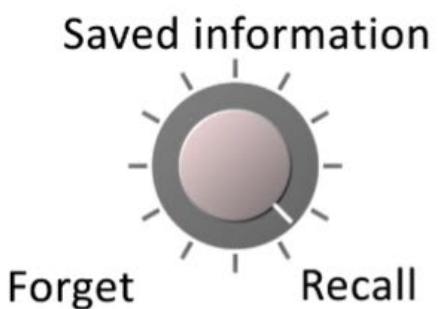


Estas redes son capaces de conservar memoria a largo plazo gracias al uso de puertas, que controlan el flujo de información pasada y actual para mantenerla o eliminarla

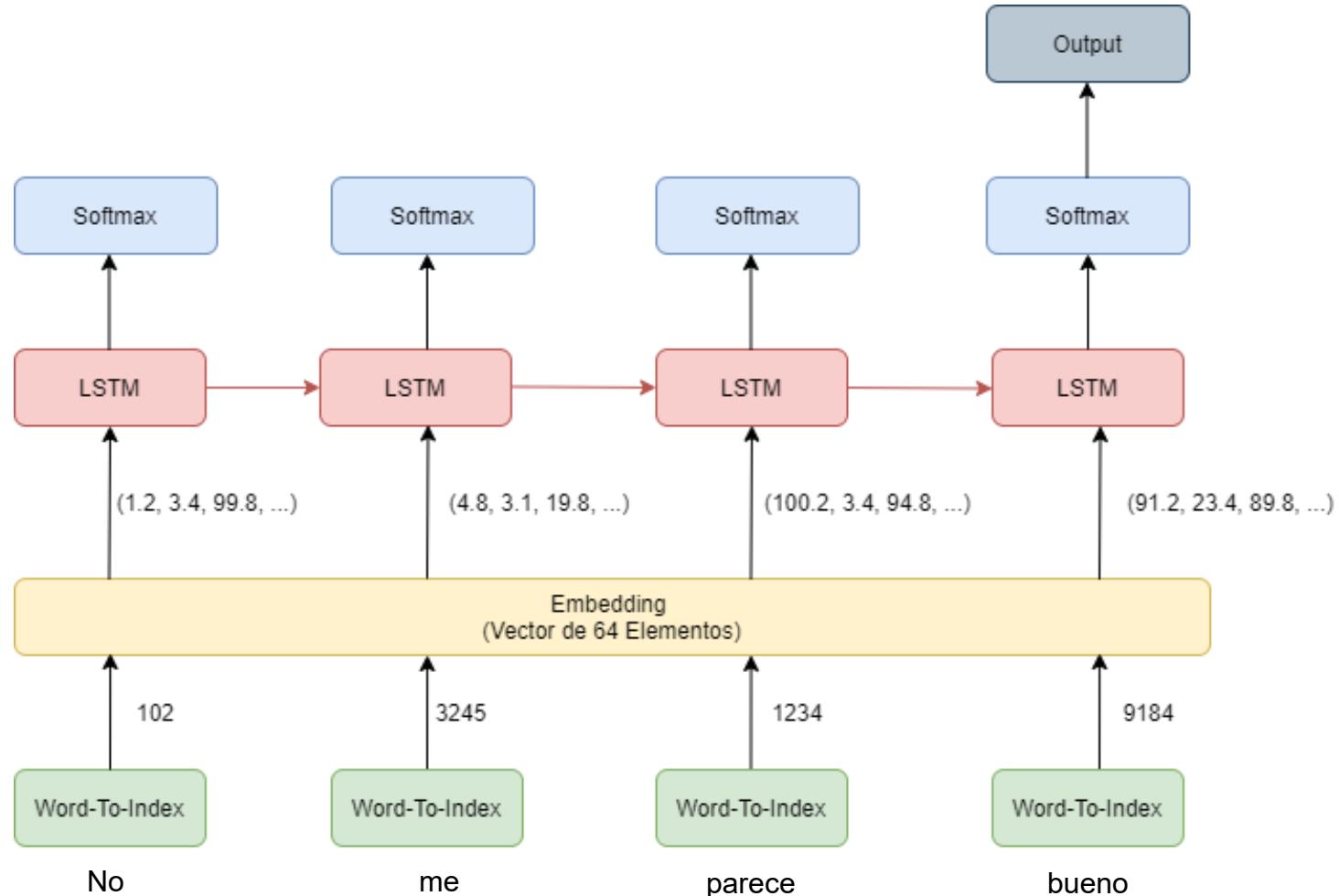
Redes Neuronales Recurrentes



Redes Neuronales Recurrentes



Redes Neuronales Recurrentes



Redes Neuronales Recurrentes (Aplicaciones)

