

2022-2023

Aprendizaje Automático

5. Redes Neuronales Multicapa



Francisco Casacuberta Nolla
(fcn@dsic.upv.es)

Enrique Vidal Ruiz
(evidal@dsic.upv.es)

José Ramón Prieto
(joprfon@prhlt.upv.es)

Departament de Sistemes Informàtics i Computació (DSIC)

Universitat Politècnica de València (UPV)

Index

- 1 Redes neuronales multicapa ▷ 2
- 2 Algoritmo de retropropagación del error (BackProp) ▷ 19
- 3 Aspectos de uso y propiedades del BackProp ▷ 34
- 4 Introducción a las redes profundas ▷ 49
- 5 PyTorch para redes neuronales ▷ 60
- 6 Perceptrón con varias capas en PyTorch ▷ 63
- 7 Redes convolucionales en PyTorch ▷ 65
- 8 Visión por computador ▷ 71
- 9 Modelos secuenciales ▷ 93
- 10 ¿Que sigue? ▷ 110
- 11 Notación ▷ 112

Index

- 1 *Redes neuronales multicapa* ▷ 2
 - 2 Algoritmo de retropropagación del error (BackProp) ▷ 19
 - 3 Aspectos de uso y propiedades del BackProp ▷ 34
 - 4 Introducción a las redes profundas ▷ 49
 - 5 PyTorch para redes neuronales ▷ 60
 - 6 Perceptrón con varias capas en PyTorch ▷ 63
 - 7 Redes convolucionales en PyTorch ▷ 65
 - 8 Visión por computador ▷ 71
 - 9 Modelos secuenciales ▷ 93
 - 10 ¿Que sigue? ▷ 110
 - 11 Notación ▷ 112

Modelo conexionista

- Un conjunto de procesadores elementales densamente interconectados.
- Nombres alternativos:
 - Modelo conexionista.
 - Red neuronal artificial.
 - Procesado distribuido y paralelo.
- Perceptrón multicapa:
 - Modelo conexionista simple.
 - Fronteras de decisión complejas.
 - Optimización no convexa.
 - Entrenamiento de los pesos mediante descenso por gradiente: algoritmo de retropropagación del error.

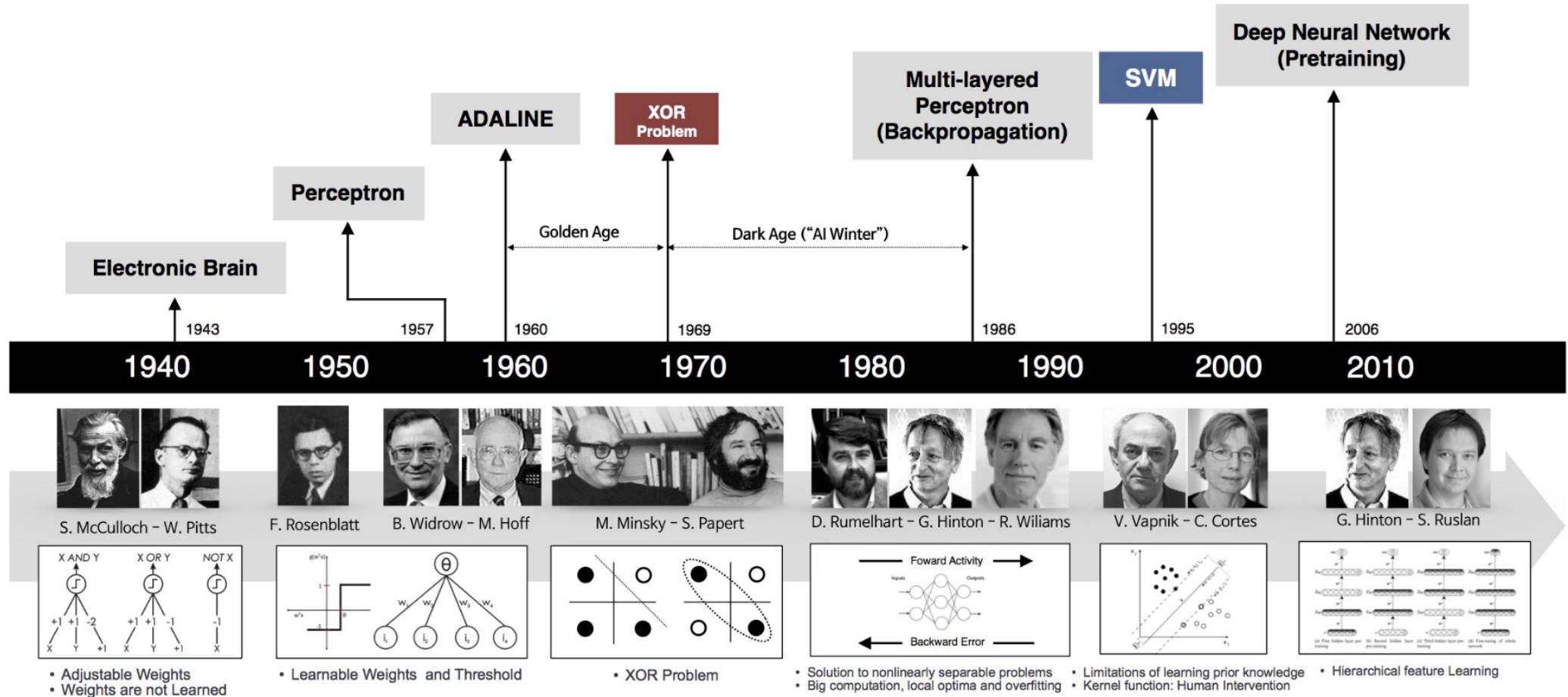
Introducción: historia (I)

- 1943: McCulloch y Pitt introducen un modelo matemático simple de “neurona”.
- 1949: Hebb propone una regla que modela el aprendizaje en las neuronas. Rochester realiza una simulación en un computador IBM en 1950.
- 1957: *Rosenblatt* introduce *el Perceptrón* como un dispositivo hardware con capacidad de autoaprendizaje y Widrow y Hoff proponen el *Adaline* para la cancelación de ecos en redes telefónicas.
- 1969: *Minsky y Papert* demuestran que un perceptrón solo puede implementar funciones discriminantes lineales y que esta limitación no se puede superar mediante multiples perceptrones organizados en cascada: Para ello sería necesario introducir funciones no-lineales.
- 1970-1975: Diversos autores tratan de desarrollar algoritmos de descenso por gradiente adecuados para multiples perceptrones en cascada con funciones no-lineales. El cálculo de derivadas parciales se muestra esquivo.

Introducción: historia (II)

- 1986: *Rumelhart, Hinton y Williams* popularizan la técnica de *retropropagación del error*. Se basa en el uso de cierto tipo de funciones no lineales, llamadas “funciones de activación”, con las que se simplifica el cálculo de derivadas parciales necesarias para descenso por gradiente. Al parecer, técnicas similares habían sido ya propuestas por *Linnainmaa* en 1970 y *Werbos* en 1974.
- 1996: *Bishop, Ripley, Ney*, entre otros, dan una interpretación probabilística a las redes neuronales y al algoritmo de retropropagación del error.
- 2000: Limitaciones en el uso del algoritmo de retropropagación del error en redes de muchas capas.
- 2006: *Hinton* publica en *Science* un artículo que inaugura una nueva tendencia denominada “redes profundas”. Posteriormente, en 2015 *LeCun, Bengio y Hinton* publican un artículo sobre estas técnicas en *Nature*. Se desarrollan diversas técnicas que mejoran el uso del algoritmo de retropropagación en redes profundas y en redes recurrentes.
- 2015-: Aplicación con gran éxito en múltiples problemas. Uso de grandes redes pre-entrenadas.

Introducción: historia (III)



(Serengil, Evolution of Neural Networks 2017)

Funciones discriminantes lineales y función de activación

- FUNCIONES DISCRIMINANTES LINEALES (FDL)

$$\phi : \mathbb{R}^d \rightarrow \mathbb{R} : \quad \phi(\mathbf{x}; \boldsymbol{\theta}) = \boldsymbol{\theta}^t \mathbf{x} = \sum_{i=0}^d \theta_i x_i$$

Notación en coordenadas homogéneas:

- $\mathbf{x} \in \mathbb{R}^{d+1}$, $\mathbf{x} = x_0, x_1, \dots, x_d$, $x_0 \stackrel{\text{def}}{=} 1$
- $\boldsymbol{\theta} \in \mathbb{R}^D$, $\boldsymbol{\theta} = \theta_0, \theta_1, \dots, \theta_d$, $D \stackrel{\text{def}}{=} d + 1$

La componente 0 del *vector de pesos* es el *umbral*, $\theta_0 \in \mathbb{R}$

Funciones discriminantes lineales y función de activación

- FUNCIONES DISCRIMINANTES LINEALES (FDL)

$$\phi : \mathbb{R}^d \rightarrow \mathbb{R} : \quad \phi(\boldsymbol{x}; \boldsymbol{\theta}) = \boldsymbol{\theta}^t \boldsymbol{x} = \sum_{i=0}^d \theta_i x_i$$

Notación en coordenadas homogéneas:

- $\boldsymbol{x} \in \mathbb{R}^{d+1}$, $\boldsymbol{x} = x_0, x_1, \dots, x_d$, $x_0 \stackrel{\text{def}}{=} 1$
- $\boldsymbol{\theta} \in \mathbb{R}^D$, $\boldsymbol{\theta} = \theta_0, \theta_1, \dots, \theta_d$, $D \stackrel{\text{def}}{=} d + 1$

La componente 0 del *vector de pesos* es el *umbral*, $\theta_0 \in \mathbb{R}$

- FUNCIONES DISCRIMINANTES LINEALES CON ACTIVACIÓN (FDLA)

$$g \circ \phi : \mathbb{R}^d \rightarrow \mathbb{R} : \quad g \circ \phi(\boldsymbol{x}; \boldsymbol{\theta}) = g(\boldsymbol{\theta}^t \boldsymbol{x})$$

$g : \mathbb{R} \rightarrow \mathbb{R}$ es una función de activación [†]

[†] también denominada *función logística* y a la FDLA *función discriminante lineal logística*.

Funciones de activación y sus derivadas

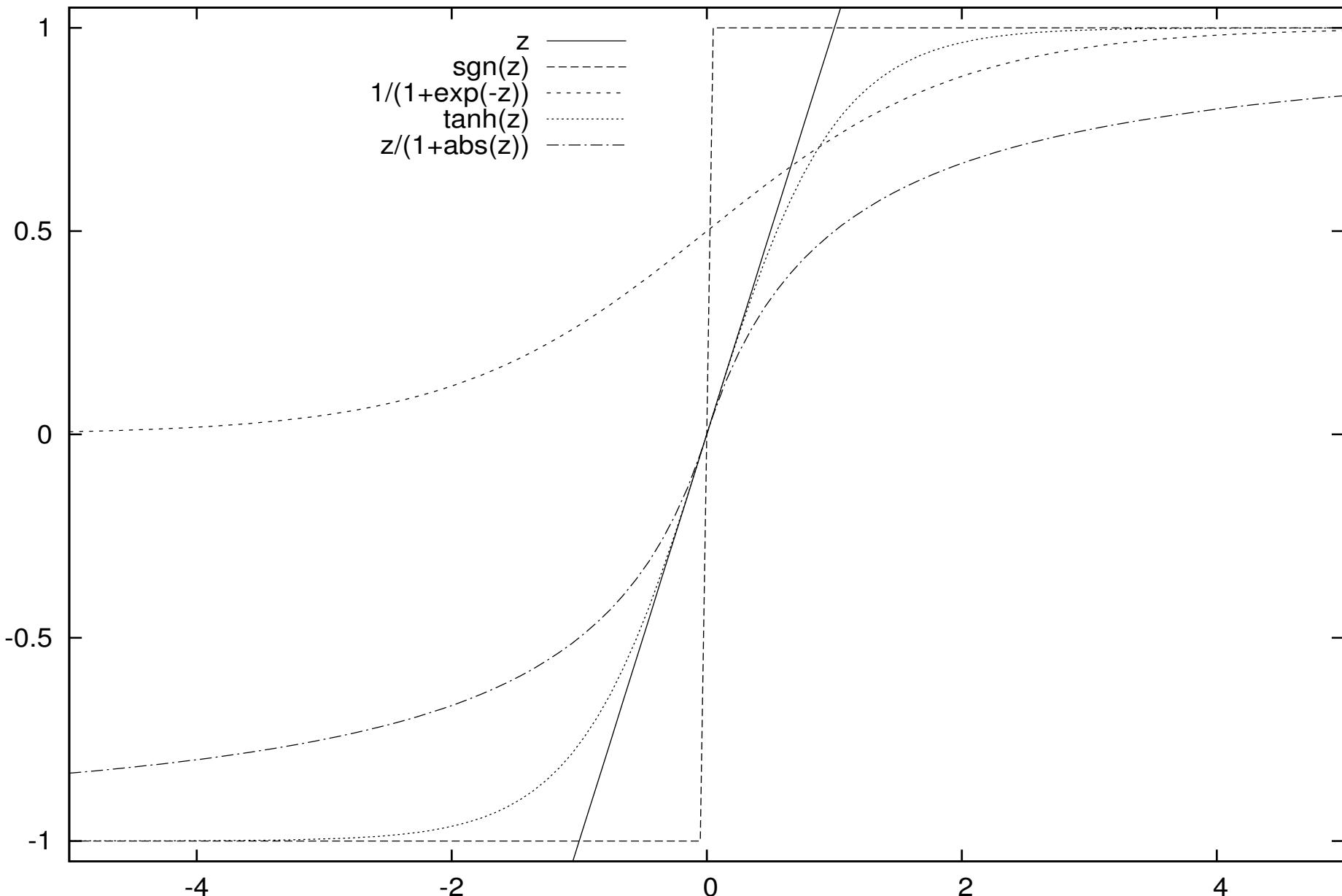
Sea $g : \mathbb{R} \rightarrow \mathbb{R}$ y $z \in \mathbb{R}$:

- **LINEAL**: $g_L(z) = z \Rightarrow g'_L(z) = \frac{d g_L}{d z} = 1$
- **RELU** (rectified linear unit): $g_U(z) = \max(0, z) \Rightarrow g'_U(z) = \begin{cases} 0 & \text{si } z < 0 \\ 1 & \text{si } z > 0 \\ \text{no definida} & \text{si } z = 0 \end{cases}$
- **ESCALÓN[†]**: $g_E(z) = \text{sgn}(z) \stackrel{\text{def}}{=} \begin{cases} +1 & \text{si } z > 0 \\ -1 & \text{si } z < 0 \end{cases} \Rightarrow g'_E(z) = \begin{cases} \text{no definida} & \text{si } z = 0 \\ 0 & \text{si } z \neq 0 \end{cases}$
- **SIGMOID**: $g_S(z) = \frac{1}{1 + \exp(-z)} \Rightarrow g'_S(z) = g_S(z) (1 - g_S(z))$
- **TANGENTE HIPERBÓLICA**: $g_T(z) = \frac{\exp(z) - \exp(-z)}{\exp(z) + \exp(-z)} \Rightarrow g'_T(z) = 1 - (g_T(z))^2$
- **RÁPIDA**: $g_F(z) = \frac{z}{1 + |z|} \Rightarrow g'_F(z) = \frac{1}{(1 + |z|)^2} = \left(\frac{g_F(z)}{z}\right)^2$
- **SOFTMAX**: Para $z_1, \dots, z_n \in \mathbb{R}$, $g_M(z_i) = \frac{\exp(z_i)}{\sum_j \exp(z_j)} \Rightarrow g'_M(z_i) = g_M(z_i) (1 - g_M(z_i))$

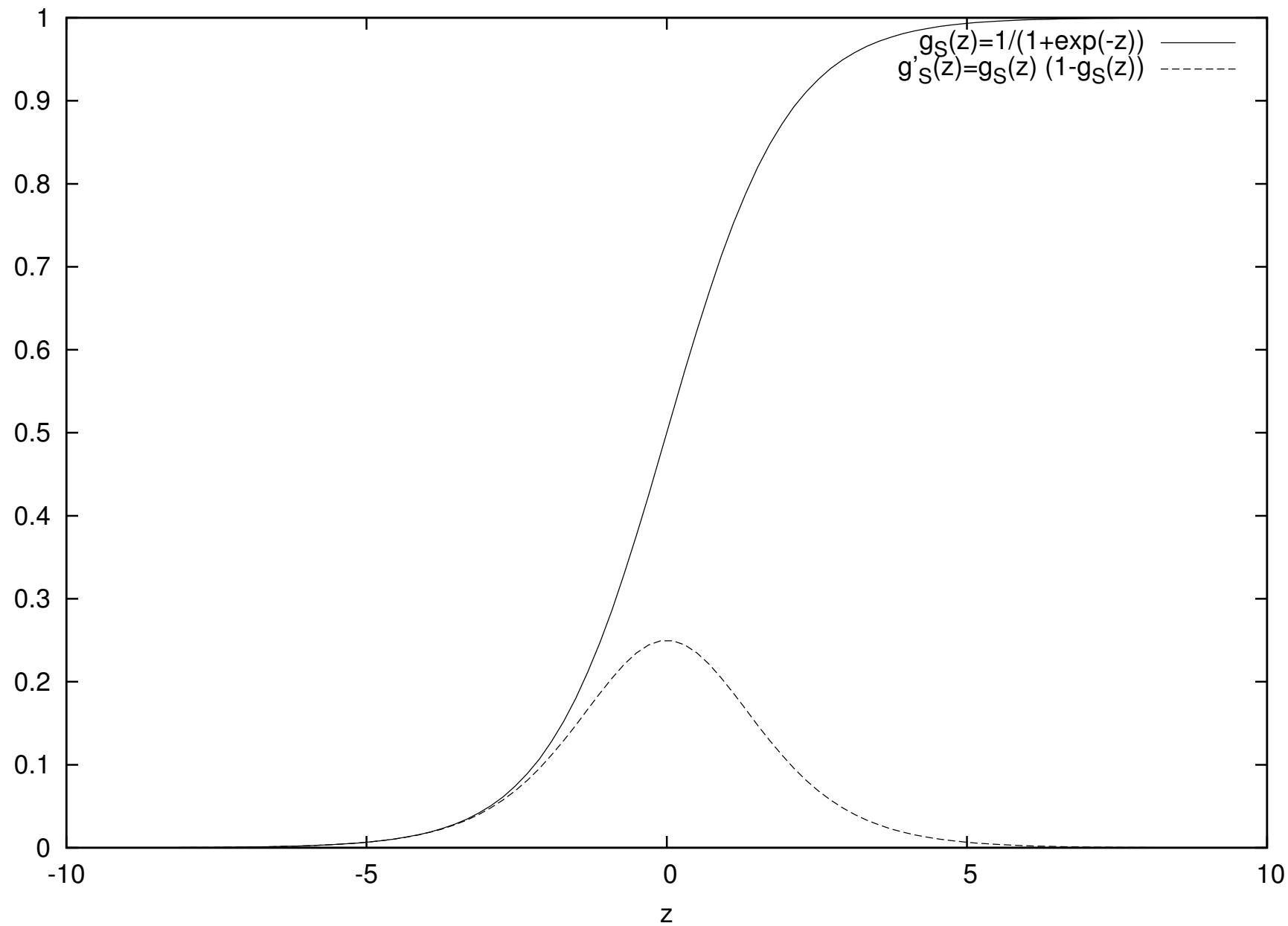
Ejercicios: Demostrar $g'_S(z) = g_S(z) (1 - g_S(z))$, $g'_T(z) = 2g_S(2z) - 1 \quad \forall z \in \mathbb{R}$

[†] sgn es la *función signo*

Funciones de activación

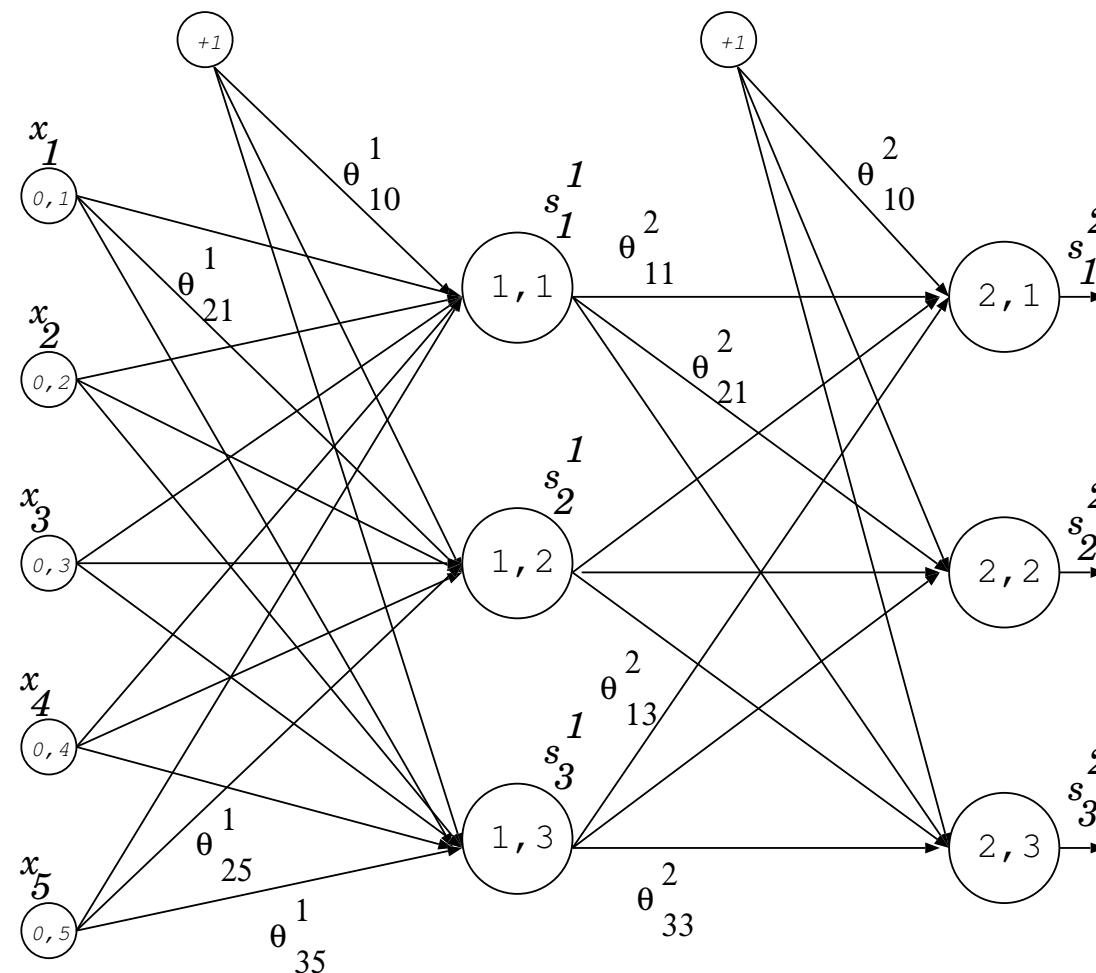


Derivada de la función de activación sigmoid



Un perceptrón de dos capas: ejemplo y notación

Topología



Dinámica

Capa de entrada

$$1 \leq i \leq M_0 \equiv d = 5$$

$$x_i \in \mathbb{R}$$

Capa oculta

$$1 \leq i \leq M_1 = 3$$

$$s_i^1(\mathbf{x}; \Theta) = g\left(\sum_{j=0}^{M_0} \theta_{ij}^1 x_j\right)$$

Capa de salida

$$1 \leq i \leq M_2 = 3$$

$$s_i^2(\mathbf{x}; \Theta) = g\left(\sum_{j=0}^{M_1} \theta_{ij}^2 s_j^1(\mathbf{x})\right)$$

Perceptrón de dos capas

- *Un perceptrón de dos capas* consiste en una combinación de FDLA agrupadas en 2 capas de tallas M_1 (capa *oculta*) y M_2 (capa de salida), más una capa de entradas de talla $M_0 = d$ (por simplicidad no se contabilizan los umbrales):

$$s_i^2(\mathbf{x}; \Theta) = g\left(\sum_{j=0}^{M_1} \theta_{ij}^2 s_j^1(\mathbf{x}; \Theta)\right) = g\left(\sum_{j=0}^{M_1} \theta_{ij}^2 g\left(\sum_{j'=0}^{M_0} \theta_{jj'}^1 x_{j'}\right)\right) \quad 1 \leq i \leq M_2$$

Los parámetros son $\Theta = [\theta_{10}^1, \dots, \theta_{M_1 M_0}^1, \theta_{10}^2, \dots, \theta_{M_2 M_1}^2] \in \mathbb{R}^D$

- *Problema:* Dado un conjunto de entrenamiento $S = \{(\mathbf{x}_1, \mathbf{t}_1), \dots, (\mathbf{x}_N, \mathbf{t}_N)\}$, con $\mathbf{x}_n \in \mathbb{R}^{M_0}$, $\mathbf{t}_n \in \mathbb{R}^{M_2}$, encontrar Θ tal que $s^2(\mathbf{x}_n; \Theta)$ aproxime lo mejor posible a $\mathbf{t}_n \forall n, 1 \leq n \leq N$.
- *En clasificación:* $M_2 \equiv C$ y las etiquetas t_n para $1 \leq n \leq N$ son de la forma

$$1 \leq c \leq C \quad t_{nc} = \begin{cases} 1 & x_n \text{ es de la clase } c \\ 0 \text{ (o } -1) & x_n \text{ no es de la clase } c \end{cases}$$

Simplificaciones de notación: $s_i^k(\mathbf{x}; \Theta) \equiv s_i^k(\mathbf{x}) \equiv s_i^k \quad \forall k, i$

El perceptrón multicapa y las funciones de activación

- Un perceptrón multicapa de dos capas define una función $\mathbb{R}^{M_0} \rightarrow \mathbb{R}^{M_2}$:

$$s_i^2(\mathbf{x}) = g\left(\sum_{j=0}^{M_1} \theta_{ij}^2 s_j^1(\mathbf{x})\right) = g\left(\sum_{j=0}^{M_1} \theta_{ij}^2 g\left(\sum_{j'=0}^{M_0} \theta_{jj'}^1 x_{j'}\right)\right) \text{ para } 1 \leq i \leq M_2$$

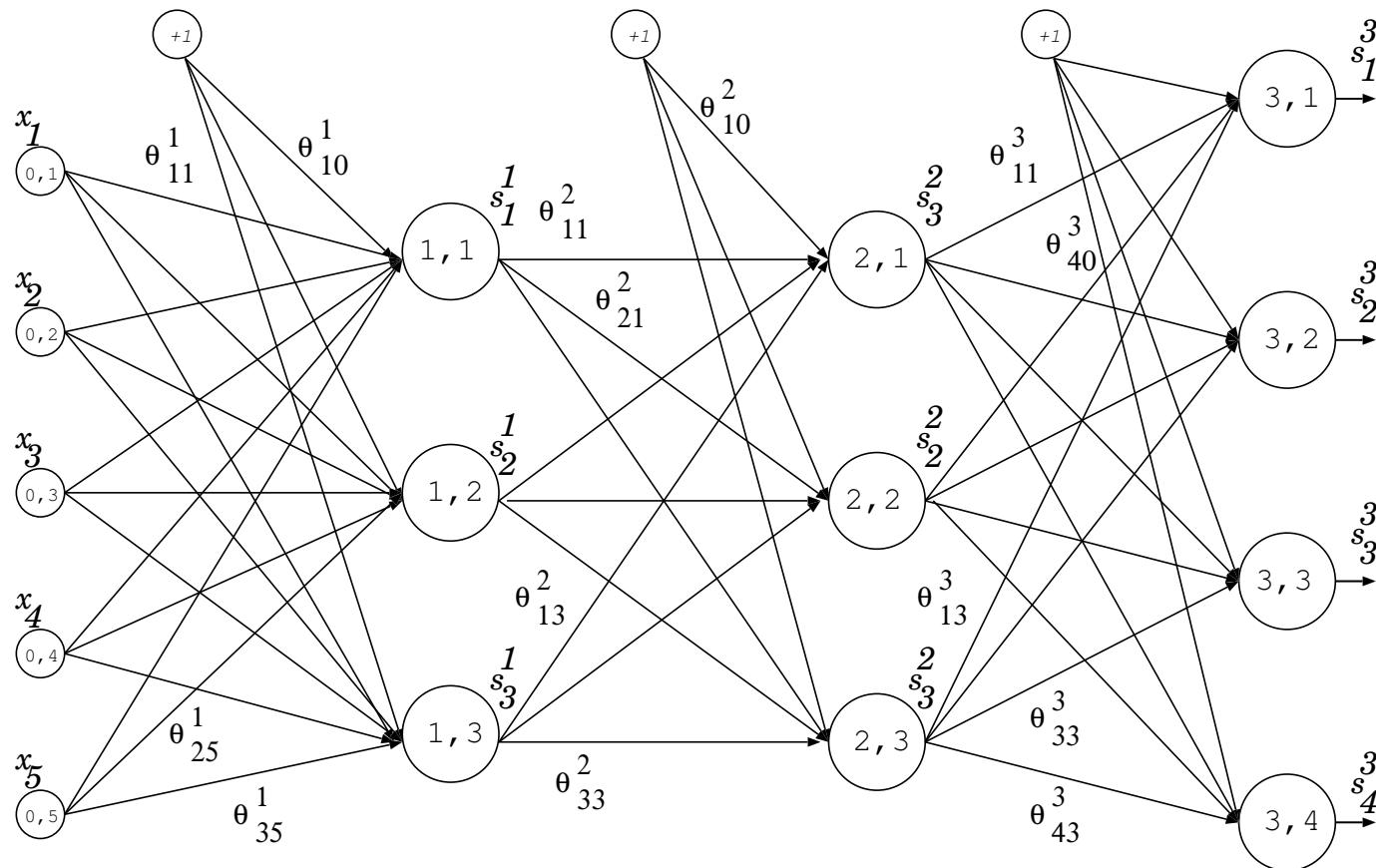
- Si todas las funciones de activación son lineales, un perceptrón multicapa define **UNA FUNCIÓN DISCRIMINANTE LINEAL**, $\phi(\mathbf{x}) = (\phi_1(\mathbf{x}), \dots, \phi_{M_2}(\mathbf{x}))^t$:

$$\phi_i(\mathbf{x}) \equiv s_i^2(\mathbf{x}) = \sum_{j=0}^{M_1} \sum_{j'=0}^{M_0} \theta_{ij}^2 \theta_{jj'}^1 x_{j'} = \sum_{j'=0}^{M_0} \left(\sum_{j=0}^{M_1} \theta_{ij}^2 \theta_{jj'}^1 \right) x_{j'} = \sum_{j'=0}^{M_0} \theta_{ij'} x_{j'}$$

- Si al menos una función de activación no es lineal (y, sin pérdida de generalidad, todas las funciones de activación de la capa de salida son lineales) un perceptrón multicapa define **UNA FUNCIÓN DISCRIMINANTE LINEAL GENERALIZADA**:

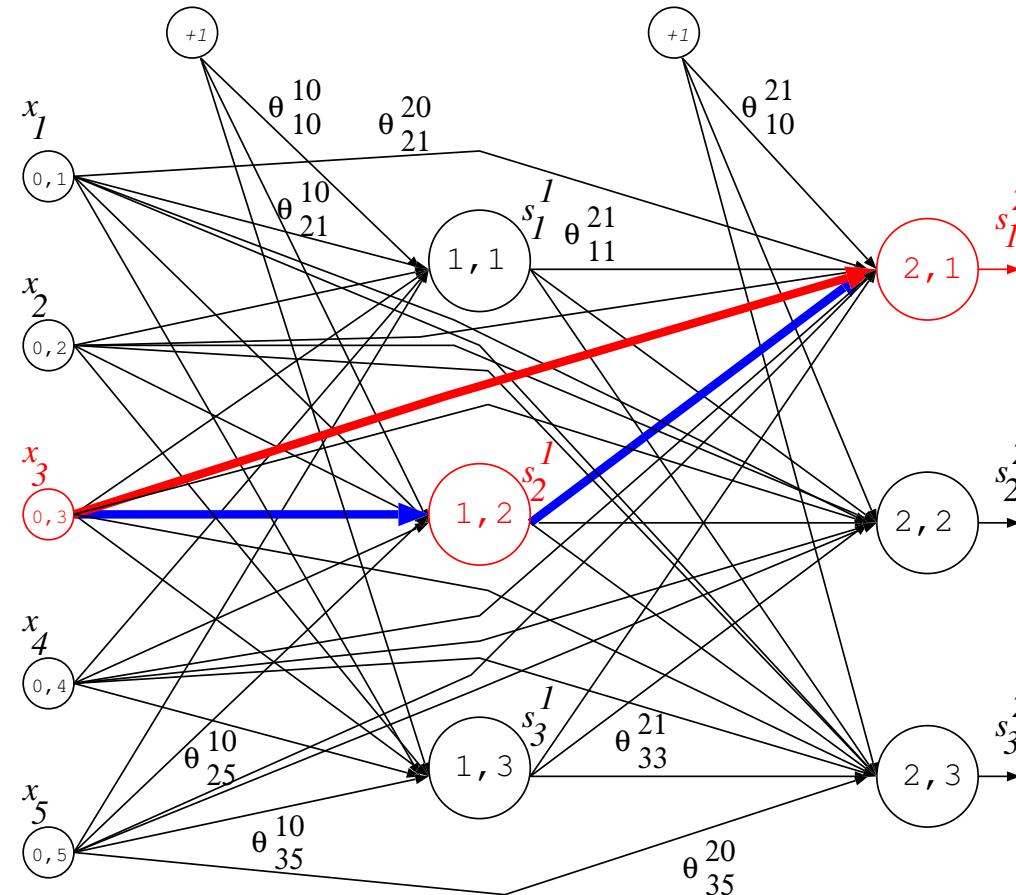
$$\phi_i(\mathbf{x}) \equiv s_i^2(\mathbf{x}) = \sum_{j=0}^{M_1} \theta_{ij}^2 g\left(\sum_{j'=0}^{M_0} \theta_{jj'}^1 x_{j'}\right) = \sum_{j=0}^{M_1} \theta_{ij}^2 \psi_j(\mathbf{x}) \text{ para } 1 \leq i \leq M_2$$

Un perceptrón de tres capas



- Primera capa oculta: $s_i^1 = g(\sum_{j=0}^{M_0} \theta_{ij}^1 x_j)$ para $1 \leq i \leq M_1$
- Segunda capa oculta: $s_i^2 = g(\sum_{j=0}^{M_1} \theta_{ij}^2 s_j^1)$ para $1 \leq i \leq M_2$
- Capa de salida: $s_i^3 = g(\sum_{j=0}^{M_2} \theta_{ij}^3 s_j^2)$ para $1 \leq i \leq M_3$

Redes hacia adelante de dos capas



- Primera capa oculta: $s_i^1 = g(\sum_{j=0}^{M_0} \theta_{ij}^{1,0} x_j)$ para $1 \leq i \leq M_1$
- Capa de salida: $s_i^2 = g(\sum_{j=0}^{M_1} \theta_{ij}^{2,1} s_j^1 + \sum_{j=0}^{M_0} \theta_{ij}^{2,0} x_j)$ para $1 \leq i \leq M_2$

El perceptrón multicapa es un caso particular

El perceptrón multicapa como regresor

Regresión de \mathbb{R}^d a $\mathbb{R}^{d'}$

(p.e. un PM de dos capas con $d \equiv M_0$ y $d' \equiv M_2$)

$$f : \mathbb{R}^{M_0} \rightarrow \mathbb{R}^{M_2} : f_i(\mathbf{x}) \equiv s_i^2(\mathbf{x}) = \sum_{j=0}^{M_1} \theta_{kj}^2 g\left(\sum_{j'=0}^{M_0} \theta_{jj'}^1 x_{j'}\right) \quad 1 \leq i \leq M_2$$

- Cualquier función se puede aproximar con precisión arbitraria mediante un perceptrón de *una* o más capas ocultas con un número de nodos suficientemente grande
- En general, para alcanzar una precisión dada, el número de nodos necesarios suele ser *mucho menor* si el número de capas ocultas es mayor o igual que *dos*
- Si se utiliza una función de activación softmax en la capa de salida, entonces el perceptrón implementa una distribución de probabilidad.

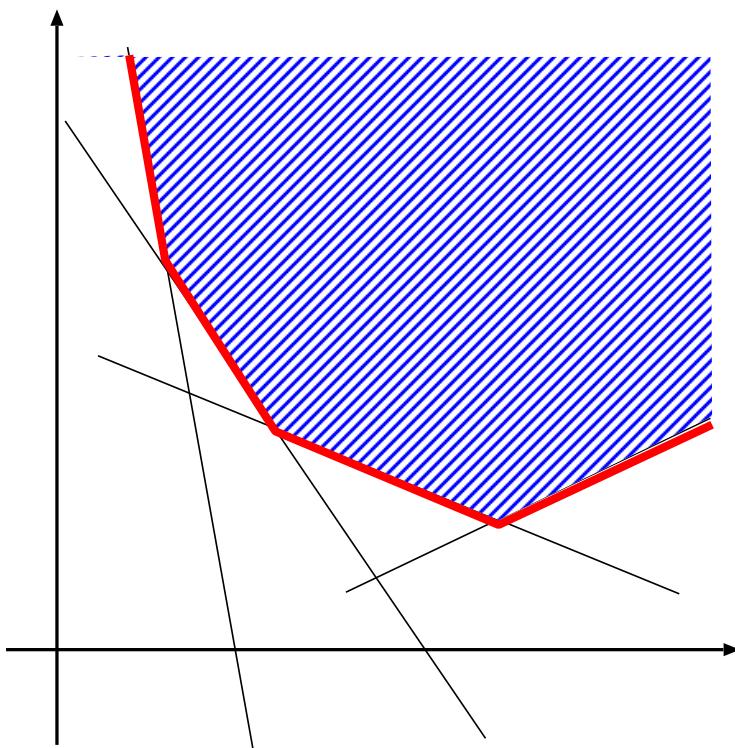
El perceptrón multicapa como clasificador

Clasificación en C clases de puntos de \mathbb{R}^d (PM con $M_0 \equiv d$, $M_2 \equiv C$)

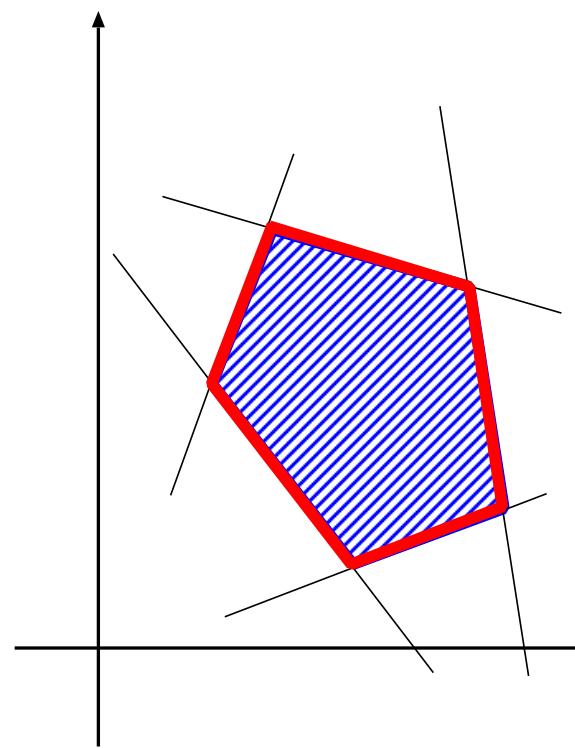
$$f : \mathbb{R}^d \rightarrow \{1, \dots, C\} : f(\mathbf{x}) = \arg \max_{1 \leq c \leq C} \phi_c(\mathbf{x}) = \arg \max_{1 \leq c \leq M_2} s_c^2(\mathbf{x})$$

- Si un conjunto de entrenamiento es linealmente separable existe un perceptrón sin capas ocultas que lo clasifica correctamente.
- Un PM con *una* capa oculta de $N - 1$ nodos puede clasificar correctamente las muestras de cualquier conjunto de entrenamiento de talla N . ¿Con qué poder de generalización?.
- Cualquier frontera de decisión basada en trozos de hiperplanos puede obtenerse mediante un PM con *una* capa oculta y un número de nodos adecuado [Huang & Lippmann, 1988], [Huang, Chen & Babri, 2000].
- En general, el número de nodos necesarios para aproximar una frontera dada suele ser *mucho menor* si el número de capas ocultas es mayor o igual que *dos*.

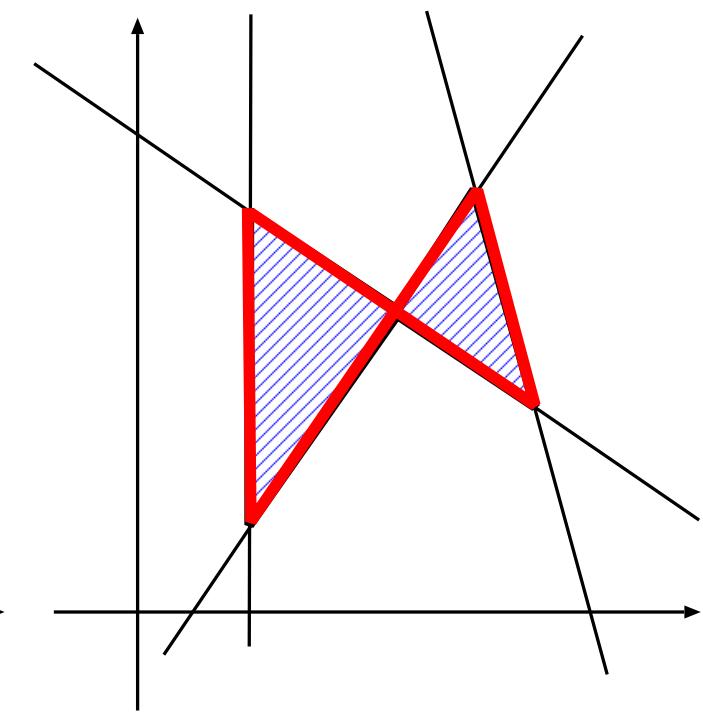
El perceptrón multicapa como clasificador



Fronteras lineales a intervalos



Regiones cerradas



Regiones no convexas

Index

- 1 Redes neuronales multicapa ▷ 2
 - 2 *Algoritmo de retropropagación del error (BackProp)* ▷ 19
 - 3 Aspectos de uso y propiedades del BackProp ▷ 34
 - 4 Introducción a las redes profundas ▷ 49
 - 5 PyTorch para redes neuronales ▷ 60
 - 6 Perceptrón con varias capas en PyTorch ▷ 63
 - 7 Redes convolucionales en PyTorch ▷ 65
 - 8 Visión por computador ▷ 71
 - 9 Modelos secuenciales ▷ 93
 - 10 ¿Que sigue? ▷ 110
 - 11 Notación ▷ 112

Aprendizaje de los pesos de un perceptrón multicapa

PROBLEMA (REGRESIÓN): dada la topología de un perceptrón multicapa con L capas y un conjunto de entrenamiento:

$$S = \{(\mathbf{x}_1, \mathbf{t}_1), \dots, (\mathbf{x}_N, \mathbf{t}_N)\}, \quad \mathbf{x}_n \in \mathbb{R}^{M_0}, \quad \mathbf{t}_n \in \mathbb{R}^{M_L}$$

obtener Θ que minimice una función objetivo o “de pérdida” (“loss”), $q_S(\Theta)$, que mida adecuadamente la discrepancia entre las salidas de la red y los valores deseados (o “targets”) dados por S :

$$q_S(\Theta) = \frac{1}{N} \sum_{n=1}^N q_n(\Theta)$$

donde $q_n(\Theta)$ es la pérdida de la muestra n -ésima.

SOLUCIÓN: descenso por gradiente (“algoritmo BACKPROP”)

$$\Delta\theta_{ij}^l = -\rho \frac{\partial q_S(\Theta)}{\partial \theta_{ij}^l} = \frac{1}{N} \sum_{n=1}^N -\rho \frac{\partial q_n(\Theta)}{\partial \theta_{ij}^l} = \frac{1}{N} \sum_{n=1}^N \Delta_n \theta_{ij}^l$$

Por tanto, para cada muestra n -ésima, hay que calcular:

$$\Delta_n \theta_{ij}^l \stackrel{\text{def}}{=} -\rho \frac{\partial q_n(\Theta)}{\partial \theta_{ij}^l}, \quad 1 \leq i \leq M_l, \quad 0 \leq j \leq M_{l-1}, \quad 1 \leq l \leq L, \quad 1 \leq n \leq N$$

Funciones objetivo o de pérdida

ERROR CUADRÁTICO: Adecuado en general para regresión y clasificación:

$$q_n(\Theta) = \frac{1}{2} \sum_{i=1}^{M_L} (t_{ni} - s_i^L(\mathbf{x}_n; \Theta))^2$$

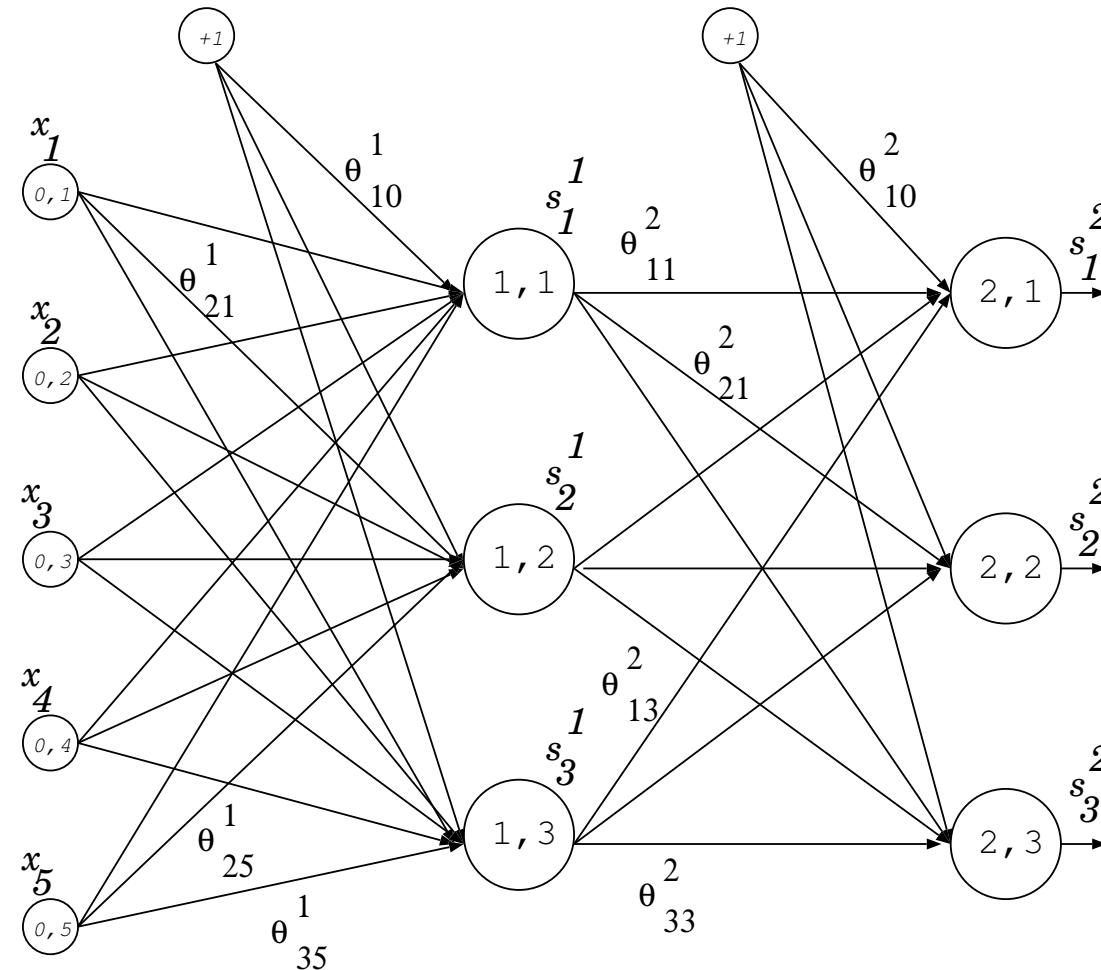
ENTROPÍA CRUZADA: Específicamente adecuada para clasificación, con activación *softmax* en la capa de salida y $M_L = C$.

$$q_n(\Theta) = - \sum_{i=1}^C t_{ni} \log s_i^L(\mathbf{x}_n; \Theta)$$

En este caso, las C salidas se consideran como aproximaciones a las probabilidades a posteriori de clase. Los valores objetivo (*targets*) de cada muestra de aprendizaje en S tienen valores en $\{0, 1\}$ y se consideran probabilidades a posteriori de clase de la distribución “empírica” dada por S .

Para simplificar, pero sin pérdida de generalidad, en lo que sigue se asume $L=2$.

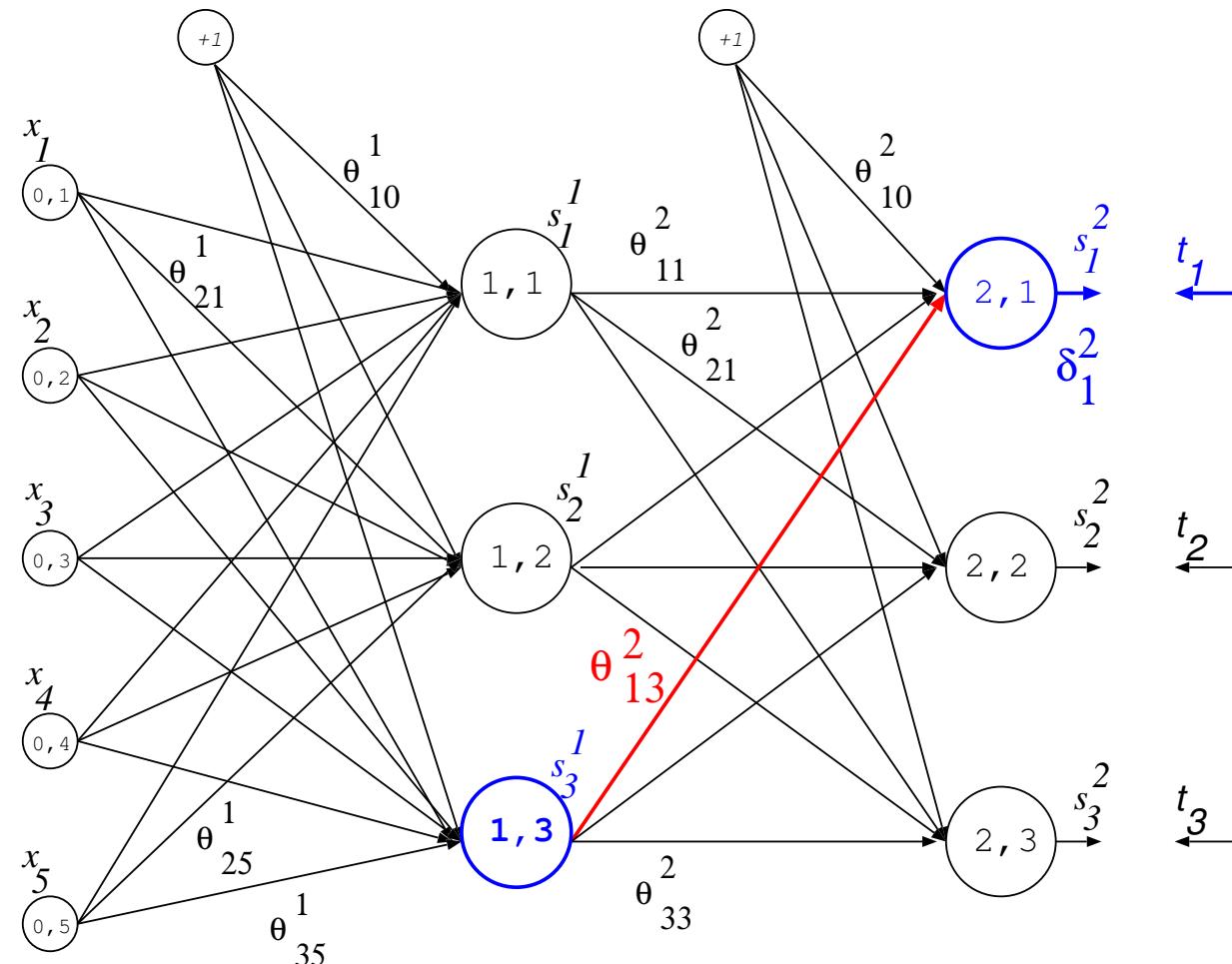
Retropropagación del error (BackProp): cálculo hacia adelante



$$s_i^1(\mathbf{x}) = g(\phi_i^1) = g\left(\sum_{j=0}^{M_0} \theta_{ij}^1 x_j\right) \quad 1 \leq i \leq M_1; \quad s_j^2 = g(\phi_j^2) = g\left(\sum_{k=0}^{M_1} \theta_{jk}^2 s_k^1(\mathbf{x})\right), \quad 1 \leq j \leq M_2$$

(para una muestra de entrenamiento genérica (\mathbf{x}, t) , y $M_0 = 5, M_1 = 3, M_2 = 3$)

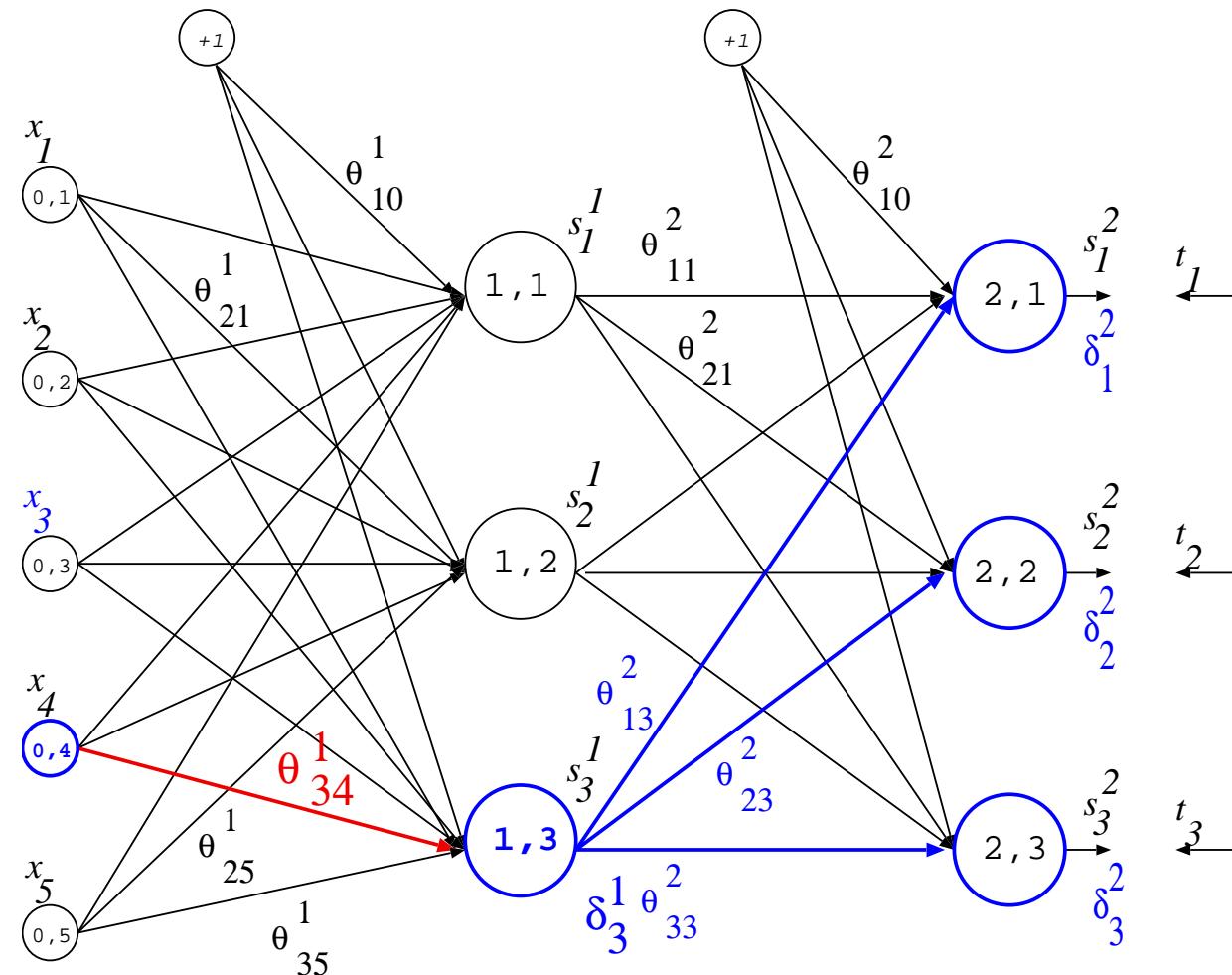
BackProp: ilustración de actualización pesos de la capa de salida



$$\Delta\theta_{13}^2 = \rho \delta_1^2 s_3^1 = \rho (t_1 - s_1^2) g'(\phi_1^2) s_3^1$$

(para la muestra (x, t) y en el caso de regresión)

BackProp: ilustración de actualización pesos de la capa oculta



$$\Delta\theta_{34}^1 = \rho \delta_3^1 x_4 = \rho (g'(\phi_3^1) \sum_{r=1}^{M_2} \delta_r^2 \theta_{r3}^2) x_4$$

(para la muestra (x, t) y en el caso de regresión)

Regla de la cadena para el cálculo de derivadas

- Función simple de otra función

$f, g : \mathbb{R} \rightarrow \mathbb{R}$:

$$\frac{d f(g(x))}{d x} = \frac{d f}{d g} \frac{d g}{d x}$$

- Función de otras dos funciones de n (o más) variables

$g_1, g_2 : \mathbb{R}^{n \geq 2} \rightarrow \mathbb{R}, \quad f : \mathbb{R}^{M \geq 2} \rightarrow \mathbb{R}$

$$\frac{\partial f(g_1(x, \dots), g_2(x, \dots))}{\partial x} = \frac{\partial f}{\partial g_1} \frac{\partial g_1}{\partial x} + \frac{\partial f}{\partial g_2} \frac{\partial g_2}{\partial x}$$

- Función de N (o más) funciones de n (o más) variables

$g_1, \dots, g_N : \mathbb{R}^{n \geq N} \rightarrow \mathbb{R}, \quad f : \mathbb{R}^{M \geq N} \rightarrow \mathbb{R}$:

$$\frac{\partial f(g_1(x, \dots), \dots, g_N(x, \dots))}{\partial x} = \sum_{i=1}^N \frac{\partial f}{\partial g_i} \frac{\partial g_i}{\partial x}$$

Derivación del algoritmo BackProp para regresión (I)

Actualización de los pesos de la capa de salida θ_{ij}^2 , para una muestra genérica $(\mathbf{x}, \mathbf{t}) \equiv (\mathbf{x}_n, \mathbf{t}_n)$:

$$q(\Theta) \equiv q_n(\Theta) = \frac{1}{2} \sum_{l=1}^{M_2} (t_l - s_l^2)^2; \quad s_l^2 = g(\phi_l^2); \quad \phi_l^2 = \sum_{m=0}^{M_1} \theta_{lm}^2 s_m^1$$

$$\begin{aligned} \frac{\partial q}{\partial \theta_{ij}^2} &= \frac{\partial q}{\partial s_i^2} \frac{\partial s_i^2}{\partial \theta_{ij}^2} = \frac{\partial q}{\partial s_i^2} \frac{d s_i^2}{d \phi_i^2} \frac{\partial \phi_i^2}{\partial \theta_{ij}^2} \\ &\quad \downarrow \quad \downarrow \quad \downarrow \\ &= -((t_i - s_i^2) g'(\phi_i^2)) s_j^1 \stackrel{\text{def}}{=} -\delta_i^2 s_j^1 \end{aligned}$$

$$\frac{\partial q}{\partial \theta_{ij}^2} = -\delta_i^2 s_j^1, \quad \delta_i^2 \stackrel{\text{def}}{=} (t_i - s_i^2) g'(\phi_i^2)$$

$$\Delta_n \theta_{ij}^2 = -\rho \frac{\partial q_n}{\partial \theta_{ij}^2} = \rho \delta_i^2 s_j^1 \quad 1 \leq i \leq M_2, \quad 0 \leq j \leq M_1$$

Derivación del algoritmo BackProp para regresión (II)

Actualización de los pesos de la capa oculta θ_{ij}^1 , para una muestra genérica $(\mathbf{x}, \mathbf{t}) \equiv (\mathbf{x}_n, \mathbf{t}_n)$:

$$q(\Theta) = \frac{1}{2} \sum_{l=1}^{M_2} (t_l - s_l^2)^2; \quad s_l^2 = g(\phi_l^2); \quad \phi_l^2 = \sum_{m=0}^{M_1} \theta_{lm}^2 s_m^1; \quad s_m^1 = g(\phi_m^1); \quad \phi_m^1 = \sum_{k=0}^{M_0} \theta_{mk}^1 x_k$$

$$\begin{aligned} \frac{\partial q}{\partial \theta_{ij}^1} &= \sum_{r=1}^{M_2} \frac{\partial q}{\partial s_r^2} \frac{\partial s_r^2}{\partial \theta_{ij}^1} = \sum_{r=1}^{M_2} \frac{\partial q}{\partial s_r^2} \frac{d s_r^2}{d \phi_r^2} \frac{\partial \phi_r^2}{\partial s_i^1} \frac{d s_i^1}{d \phi_i^1} \frac{\partial \phi_i^1}{\partial \theta_{ij}^1} \\ &\quad \downarrow \quad \downarrow \quad \downarrow \quad \downarrow \\ &= \sum_{r=1}^{M_2} -\delta_r^2 \quad \theta_{ri}^2 \ g'(\phi_i^1) \ x_j = -\left(g'(\phi_i^1) \sum_{r=1}^{M_2} \delta_r^2 \ \theta_{ri}^2\right) x_j \stackrel{\text{def}}{=} -\delta_i^1 x_j \end{aligned}$$

$$\frac{\partial q}{\partial \theta_{ij}^1} = -\delta_i^1 x_j, \quad \delta_i^1 \stackrel{\text{def}}{=} g'(\phi_i^1) \sum_{r=1}^{M_2} \delta_r^2 \ \theta_{ri}^2$$

$$\Delta_n \theta_{ij}^1 = -\rho \frac{\partial q_n}{\partial \theta_{ij}^1} = \rho \delta_i^1 x_j \quad 1 \leq i \leq M_1, \ 0 \leq j \leq M_0$$

Ecuaciones BackProp

- Actualización de los pesos de la capa de salida: ($1 \leq i \leq M_2$, $0 \leq j \leq M_1$):

$$\Delta\theta_{ij}^2 = -\rho \frac{\partial q_S(\Theta)}{\partial \theta_{ij}^2} = \frac{\rho}{N} \sum_{n=1}^N \delta_i^2(\mathbf{x}_n) s_j^1(\mathbf{x}_n)$$

$$\delta_i^2(\mathbf{x}_n) = (t_{ni} - s_i^2(\mathbf{x}_n)) g'(\phi_i^2(\mathbf{x}_n)) \text{ con } \phi_i^2(\mathbf{x}_n) = \sum_{j=0}^{M_1} \theta_{ij}^2 s_j^1(\mathbf{x}_n)$$

- Actualización de los pesos de la capa oculta ($1 \leq i \leq M_1$, $0 \leq j \leq M_0$):

$$\Delta\theta_{ij}^1 = -\rho \frac{\partial q_S(\Theta)}{\partial \theta_{ij}^1} = \frac{\rho}{N} \sum_{n=1}^N \delta_i^1(\mathbf{x}_n) x_{nj}$$

$$\delta_i^1(\mathbf{x}_n) = \left(\sum_{r=1}^{M_2} \delta_r^2(\mathbf{x}_n) \theta_{ri}^2 \right) g'(\phi_i^1(\mathbf{x}_n)) \text{ con } \phi_i^1(\mathbf{x}_n) = \sum_{j=0}^{M_0} \theta_{ij}^1 x_{nj}$$

Ejercicio: derivar las ecuaciones BackProp para el caso de clasificación.

Ecuaciones BackProp para perceptrones de tres capas

- Actualización de los pesos de la capa de salida ($1 \leq i \leq M_3, 0 \leq j \leq M_2$)

$$\Delta\theta_{ij}^3 = -\rho \frac{\partial q_S(\Theta)}{\partial \theta_{ij}^3} = \frac{\rho}{N} \sum_{n=1}^N \delta_i^3(\mathbf{x}_n) s_j^2(\mathbf{x}_n) \quad \delta_i^3(\mathbf{x}_n) = \left(t_{ni} - s_i^3(\mathbf{x}_n) \right) g'(\phi_i^3(\mathbf{x}_n))$$

- Actualización de los pesos de la segunda capa oculta ($1 \leq i \leq M_2, 0 \leq j \leq M_1$)

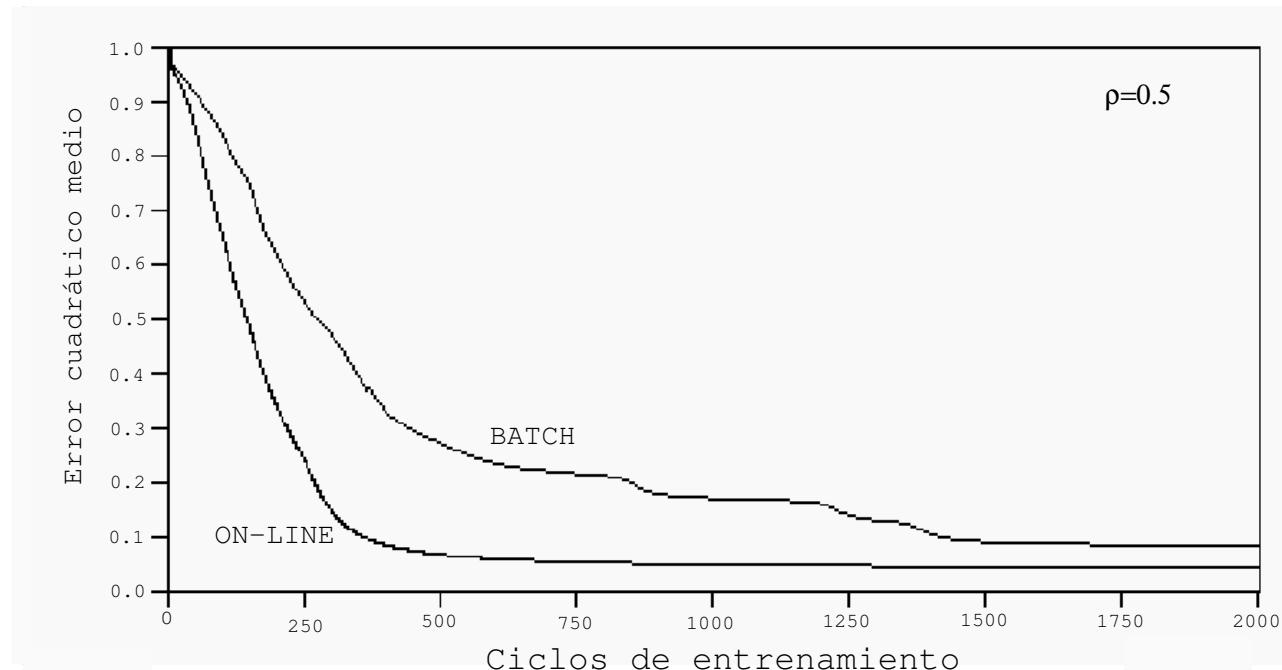
$$\Delta\theta_{ij}^2 = -\rho \frac{\partial q_S(\Theta)}{\partial \theta_{ij}^2} = \frac{\rho}{N} \sum_{n=1}^N \delta_i^2(\mathbf{x}_n) s_j^1(\mathbf{x}_n) \quad \delta_i^2(\mathbf{x}_n) = \left(\sum_{r=1}^{M_3} \delta_r^3(\mathbf{x}_n) \theta_{ri}^3 \right) g'(\phi_i^2(\mathbf{x}_n))$$

- Actualización de los pesos de la primera capa oculta ($1 \leq i \leq M_1, 0 \leq j \leq M_0$)

$$\Delta\theta_{ij}^1 = -\rho \frac{\partial q_S(\Theta)}{\partial \theta_{ij}^1} = \frac{\rho}{N} \sum_{n=1}^N \delta_i^1(\mathbf{x}_n) x_{nj} \quad \delta_i^1(\mathbf{x}_n) = \left(\sum_{r=1}^{M_2} \delta_r^2(\mathbf{x}_n) \theta_{ri}^2 \right) g'(\phi_i^1(\mathbf{x}_n))$$

BackProp incremental o “batch”

- BackProp “batch”: en cada iteración se procesan las N muestras de entrenamiento (“epoch”) y los pesos del PM se actualizan una sola vez.
- BackProp “*mini-batch*”: el conjunto de entrenamiento se divide en B bloques, en cada uno se procesan las muestras que están contenidas y luego se actualizan los pesos del PM. Por tanto en un epoch los pesos se actualizan N/B veces.
- BackProp “*incremental*”: en cada iteración se procesa solo una muestra de entrenamiento (aleatoria) y se actualizan los pesos del PM. Por tanto en un epoch los pesos se actualizan N veces.



Algoritmo BACKPROP

Entrada: Topología, pesos iniciales θ_{ij}^l , $1 \leq l \leq L$, $1 \leq i \leq M_l$, $0 \leq j \leq M_{l-1}$, factor de aprendizaje ρ , condiciones de convergencia, N datos de entrenamiento S

Salidas: Pesos de las conexiones que minimizan el error cuadrático medio de S

Mientras no se cumplan las condiciones de convergencia

Para $1 \leq l \leq L$, $1 \leq i \leq M_l$, $0 \leq j \leq M_{l-1}$, inicializar $\Delta\theta_{ij}^l = 0$

Para cada muestra de entrenamiento $(x, t) \in S$

 Desde la capa de entrada a la de salida ($l = 0, \dots, L$):

 Para $1 \leq i \leq M_l$ si $l = 0$ entonces $s_i^0 = x_i$ sino calcular ϕ_i^l y $s_i^l = g(\phi_i^l)$

 Desde la capa de salida a la de entrada ($l = L, \dots, 1$),

 Para cada nodo ($1 \leq i \leq M_l$)

 Calcular $\delta_i^l = \begin{cases} g'(\phi_i^l) (t_{ni} - s_i^L) & \text{si } l == L \\ g'(\phi_i^l) (\sum_r \delta_r^{l+1} \theta_{ri}^{l+1}) & \text{en otro caso} \end{cases}$

 Para cada peso θ_{ij}^l ($0 \leq j \leq M_{l-1}$) calcular: $\Delta\theta_{ij}^l = \Delta\theta_{ij}^l + \rho \delta_i^l s_j^{l-1}$

 Para $1 \leq l \leq L$, $1 \leq i \leq M_l$, $0 \leq j \leq M_{l-1}$, actualizar pesos: $\theta_{ij}^l = \theta_{ij}^l + \frac{1}{N} \Delta\theta_{ij}^l$

Coste computacional por cada iteración *mientras*: $O(N D)$, $N = |S|$, D = número de pesos

DEMO: <http://playground.tensorflow.org/>

Algoritmo BACKPROP (“incremental”)

Entrada: Topología, pesos iniciales θ_{ij}^l , $1 \leq l \leq L$, $1 \leq i \leq M_l$, $0 \leq j \leq M_{l-1}$, factor de aprendizaje ρ , condiciones de convergencia, N datos de entrenamiento S

Salidas: Pesos de las conexiones que minimizan el error cuadrático medio de S

Mientras no se cumplan las condiciones de convergencia

Para cada muestra de entrenamiento $(x, t) \in S$ (en orden aleatorio)

Desde la capa de entrada a la de salida ($l = 0, \dots, L$):

Para $1 \leq i \leq M_l$ si $l = 0$ entonces $s_i^0 = x_i$ sino calcular ϕ_i^l y $s_i^l = g(\phi_i^l)$

Desde la capa de salida a la de entrada ($l = L, \dots, 1$),

Para cada nodo ($1 \leq i \leq M_l$)

Calcular $\delta_i^l = \begin{cases} g'(\phi_i^l) (t_{ni} - s_i^L) & \text{si } l == L \\ g'(\phi_i^l) (\sum_r \delta_r^{l+1} \theta_{ri}^{l+1}) & \text{en otro caso} \end{cases}$

Para cada peso θ_{ij}^l ($0 \leq j \leq M_{l-1}$) calcular: $\Delta\theta_{ij}^l = \rho \delta_i^l s_j^{l-1}$

Para $1 \leq l \leq L$, $1 \leq i \leq M_l$, $0 \leq j \leq M_{l-1}$, actualizar pesos: $\theta_{ij}^l = \theta_{ij}^l + \frac{1}{N} \Delta\theta_{ij}^l$

Coste computacional por cada iteración *mientras*: $O(N D)$, $N = |S|$, D = número de pesos

Algoritmo BACKPROP (“on-line”)

Entrada: Topología, pesos iniciales θ_{ij}^l , $1 \leq l \leq L$, $1 \leq i \leq M_l$, $0 \leq j \leq M_{l-1}$, factor de aprendizaje ρ , dato de entrenamiento (x, t)

Salidas: Pesos de las conexiones actualizados mediante (x, t)

Desde la capa de entrada a la de salida ($l = 0, \dots, L$):

Para $1 \leq i \leq M_l$ si $l = 0$ entonces $s_i^0 = x_i$ sino calcular ϕ_i^l y $s_i^l = g(\phi_i^l)$

Desde la capa de salida a la de entrada ($l = L, \dots, 1$),

Para cada nodo ($1 \leq i \leq M_l$)

Calcular $\delta_i^l = \begin{cases} g'(\phi_i^l) (t_{ni} - s_i^L) & \text{si } l == L \\ g'(\phi_i^l) (\sum_r \delta_r^{l+1} \theta_{ri}^{l+1}) & \text{en otro caso} \end{cases}$

Para cada peso θ_{ij}^l ($0 \leq j \leq M_{l-1}$) calcular: $\Delta\theta_{ij}^l = \rho \delta_i^l s_j^{l-1}$

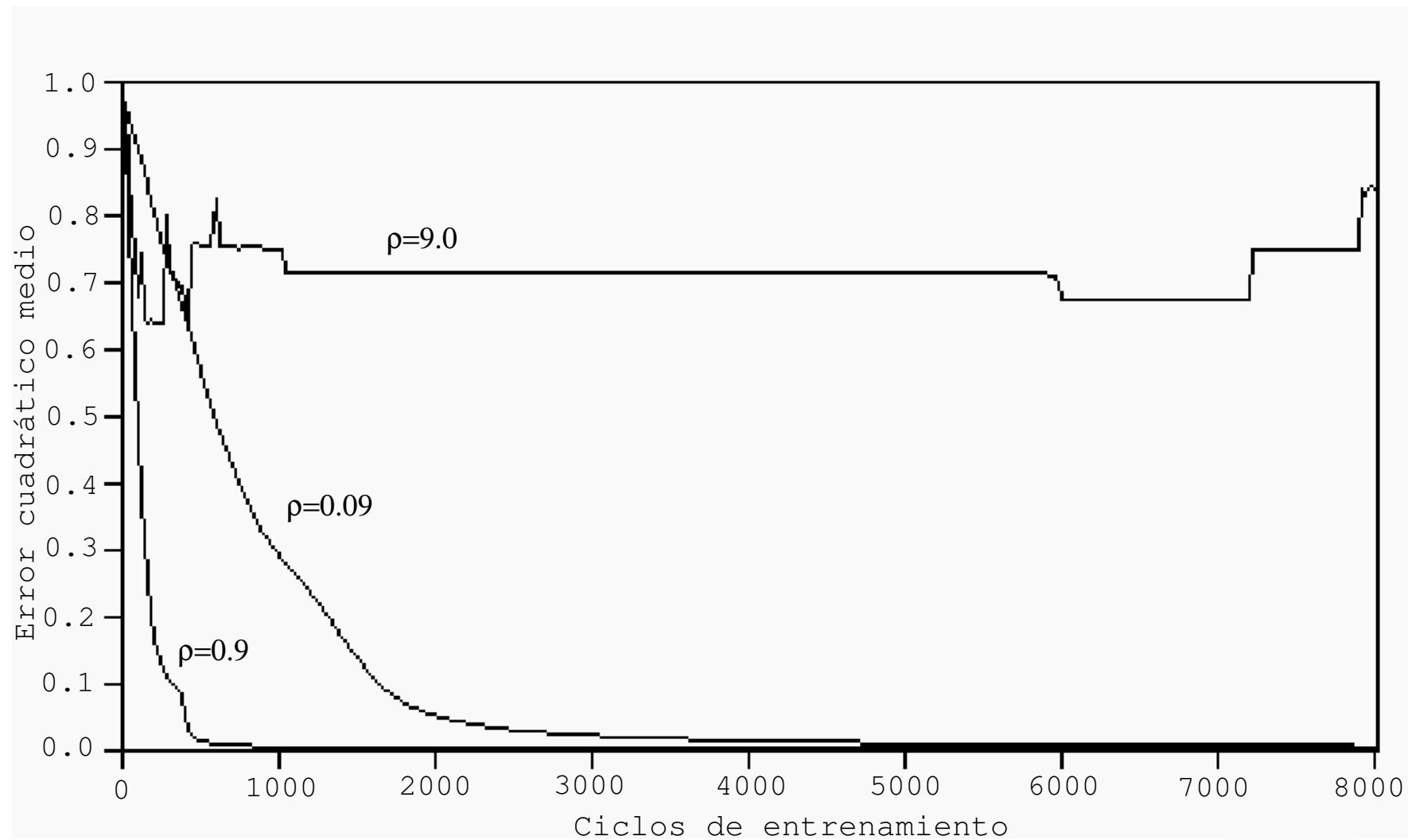
Para $1 \leq l \leq L$, $1 \leq i \leq M_l$, $0 \leq j \leq M_{l-1}$, actualizar pesos: $\theta_{ij}^l = \theta_{ij}^l + \Delta\theta_{ij}^l$

Coste computacional por cada muestra procesada: $O(D)$, $D = \text{número de pesos}$

Index

- 1 Redes neuronales multicapa ▷ 2
- 2 Algoritmo de retropropagación del error (BackProp) ▷ 19
 - 3 *Aspectos de uso y propiedades del BackProp* ▷ 34
- 4 Introducción a las redes profundas ▷ 49
- 5 PyTorch para redes neuronales ▷ 60
- 6 Perceptrón con varias capas en PyTorch ▷ 63
- 7 Redes convolucionales en PyTorch ▷ 65
- 8 Visión por computador ▷ 71
- 9 Modelos secuenciales ▷ 93
- 10 ¿Que sigue? ▷ 110
- 11 Notación ▷ 112

Selección del factor de aprendizaje

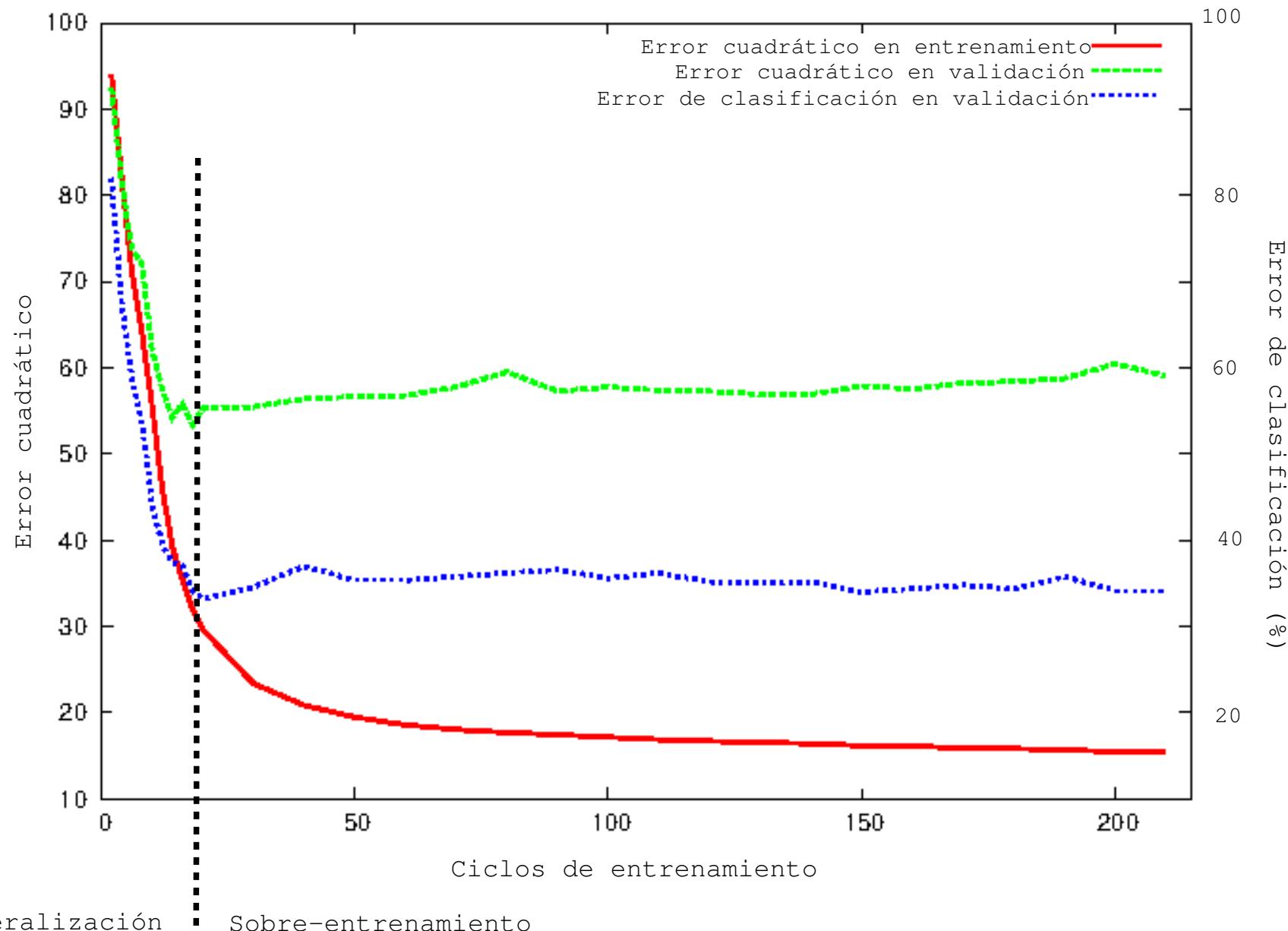


<http://playground.tensorflow.org/>

Algoritmos de optimización

- **SDG** (stochastic gradient descent).
- **SGD with momentum**
- **Adagrad** (Adaptive Gradient)
- **Adadelta** (an extension of Adagrad)
- **ADAM** (Adaptive Moment Estimation)
- NAG, RMSProp, AdaMax, Nadam, ...

Condiciones de convergencia



Algunos problemas y soluciones con el BackProp

- El problema de la anulación o explosión del gradiente en el caso de muchas capas:
 - Uso de funciones de activación como Relu, LeakyRelu/PreLU, Maxout, ELU, ...
 - Regularización.
 - Drop-out: Durante el entrenamiento se seleccionan aleatoriamente un subconjunto de nodos y no son utilizados en una iteración.
 - Conexiones residuales.
 - Normalización a nivel de batch y/o de capa.
 - Gradiente recortado (clipping).
- Evitar “malos” mínimos locales:
 - Barajar los datos de entrenamiento.
 - Aprender primero las muestras más “fáciles” (Curriculum learning).
 - Regularización.
 - Añadir ruido durante el entrenamiento.

Funciones de activación (1)

- Maxout: $f(z) = \max(w_1 z + b_1, w_2 z + b_2), \quad z \in \mathbb{R}$

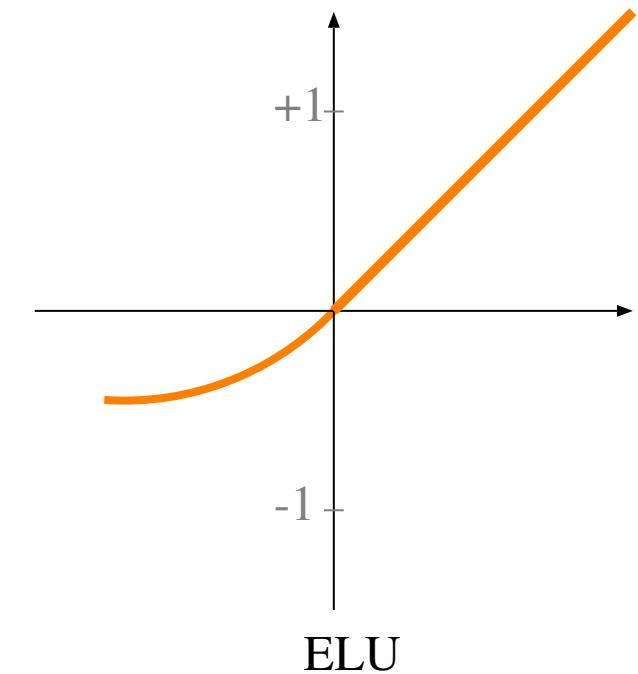
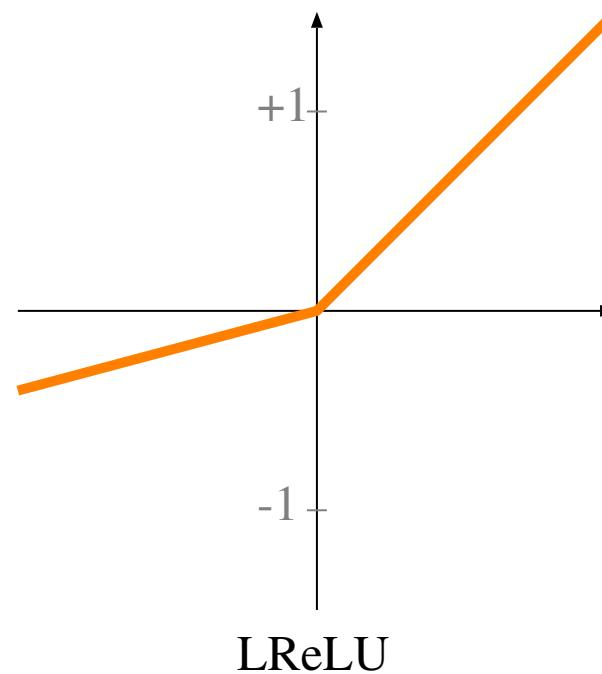
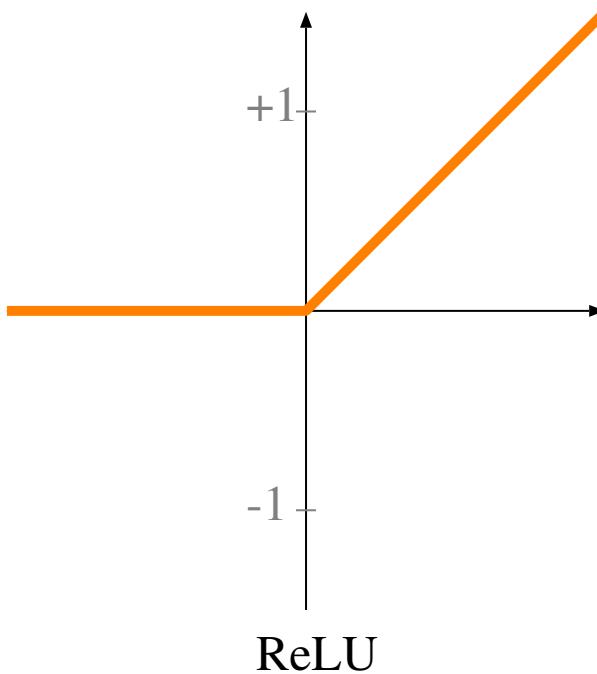
Casos particulares relevantes:

- Relu (Rectified Linear Unit): $f(z) = \max(0, z), \quad z \in \mathbb{R}$
- PreLU (Parametric Rectified Linear Unit): $f(z) = \max(\alpha z, z), \quad z \in \mathbb{R}$

- ELU (Exponential Linear Units) : $f(z) = \begin{cases} z & z \geq 0 \\ \alpha(e^z - 1) & z < 0 \end{cases}, \quad z \in \mathbb{R}$

- Softmax: Para z_1, \dots, z_I con $z_i \in \mathbb{R}$ $1 \leq i \leq I$, $f(z_i) = \frac{e^{z_i}}{\sum_j e^{z_j}}$.

Funciones de activación (2)



Normalización de las entradas

- Parálisis de la red: para valores grandes de z la derivada $g'(z)$ es muy pequeña y por tanto, los incrementos de los pesos son muy pequeños. Una forma de disminuir este efecto es **normalizar el rango de entrada**.

$$S = \{\mathbf{x}_1, \dots, \mathbf{x}_N\} \subset \mathbb{R}^d \Rightarrow \begin{cases} \mu_j = \frac{1}{N} \sum_{i=1}^N x_{ij} & 1 \leq j \leq d \\ \sigma_j^2 = \frac{1}{N-1} \sum_{i=1}^N (x_{ij} - \mu_j)^2 & 1 \leq j \leq d \end{cases}$$

$$\forall \mathbf{x} \in \mathbb{R}^d, \hat{\mathbf{x}} : \hat{x}_j = \frac{x_j - \mu_j}{\sqrt{\sigma_j^2 + \epsilon}} \Rightarrow \begin{cases} \hat{\mu}_j = 0 & \text{for } 1 \leq j \leq d \\ \hat{\sigma}_j = 1 & \end{cases}$$

donde ϵ es una constante pequeña para evitar división por cero.

Normalización a nivel de capa

- En la capa k se dispone de las correspondiente salida $s^k \in \mathbb{R}^{M_k}$

$$\mu_L = \frac{1}{M_k} \sum_{i=1}^{M_k} s_i^k$$

$$\sigma_L^2 = \frac{1}{M_k - 1} \sum_{i=1}^{M_k} (s_i^k - \mu_L)^2$$

$$\forall s_i^k \in \mathbb{R}^{M_k}, \hat{s}_i^k := \frac{s_i^k - \mu_L}{\sqrt{\sigma_L^2 + \epsilon}} \text{ for } 1 \leq i \leq M_k$$

	s_1^k	s_2^k	s_3^k	s_4^k	s_5^k	s_6^k	s_7^k	s_8^k
x_1	○	○	○	○	○	○	○	○
x_2	○	○	○	○	○	○	○	○
x_3	○	○	○	○	○	○	○	○
x_4	○	○	○	○	○	○	○	○
x_5	○	○	○	○	○	○	○	○

Normalización a nivel de minibatch

- En la capa k se dispone de las correspondientes salidas para un minibatch (subconjunto de S) de tamaño b : $B = \{s_1^k, \dots, s_b^k\} \subset \mathbb{R}^{M_k}$

$$\mu_B = \frac{1}{b} \sum_{i=1}^b s_i^k$$

$$\sigma_B^2 = \frac{1}{b-1} \| s_i^k - \mu_B \|^2$$

$$\forall s^k \in B, \quad \bar{s}^k = \frac{s^k - \mu_B}{\sqrt{\sigma_B^2 + \epsilon}} \rightarrow \hat{s}^k = \gamma \bar{s}^k + \beta$$

	s_1^k	s_2^k	s_3^k	s_4^k	s_5^k	s_6^k	s_7^k	s_8^k
x_1	○	○	○	○	○	○	○	○
x_2	○	○	○	○	○	○	○	○
x_3	○	○	○	○	○	○	○	○
x_4	○	○	○	○	○	○	○	○
x_5	○	○	○	○	○	○	○	○

Recorte del gradiente y Regularización

Problema: La actualización de los pesos durante el entrenamiento puede provocar un desbordamiento numérico, a menudo denominado "gradiente explosivo" [Brownlee 2019]. Esto puede provocar una "parálisis de la red".

Soluciones:

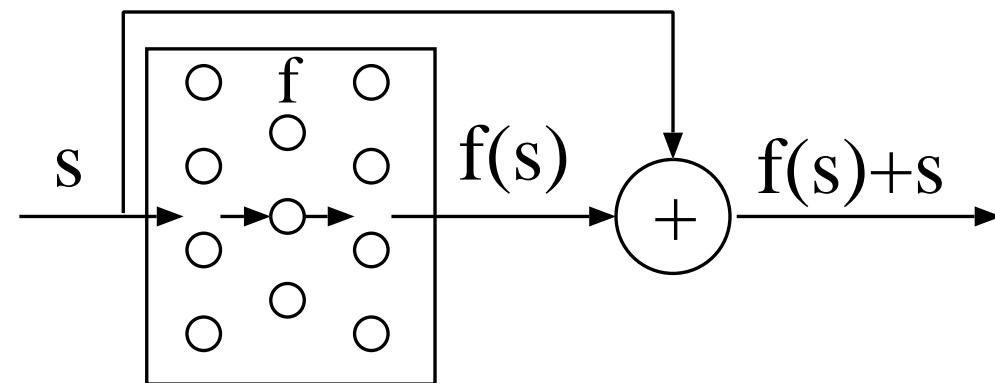
- Gradiente escalado: normalizar el vector de gradiente de manera que su norma (magnitud) sea igual a un valor definido, como 1.0.
- Recorte de gradiente: forzar los valores del gradiente (por elementos) a un valor mínimo o máximo si el gradiente excede un rango prefijado.
- Regularización: añadir a la función objetivo a minimizar un término relacionado con la suma de magnitudes de los pesos:

$$-\text{Regularización } L_2 : q_S(\Theta) + \frac{\lambda}{2} \sum_{l,i,j} (\theta_{ij}^l)^2$$

$$-\text{Regularización } L_1 : q_S(\Theta) + \lambda \sum_{l,i,j} |\theta_{ij}^l|$$

Conexión residual y drop-out

- Conexión residual: Si una serie de capas de una red neuronal implementa una función $s \rightarrow f(s)$, y asumiendo que s y $f(s)$ son de la misma dimensionalidad, un “conexión residual” permite implementar $f(s) + s$



- Drop-out: Durante el entrenamiento se seleccionan aleatoriamente un subconjunto de nodos y no son utilizados en una iteración.

Aumento de datos artificiales

- Aumentar el tamaño con muestras sintéticas a partir de reales mediante transformaciones.
 - En imágenes:
 - * Giros horizontales.
 - * Rotación.
 - * Recortes.
 - * Transformaciones de color.
 - * Añadiendo ruido.
 - En texto:
 - * Sustituyendo palabras.
 - * Insertando palabras.
 - * Borrando palabras.

Propiedades del BackProp

- Convergencia: teorema general del descenso por gradiente (Tema 3)
- Elección del factor de aprendizaje: Tema 3 (Adadelta, Adam, ...)
- Coste computacional: $O(N D)$ en cada iteración
- *En condiciones límites, las salidas de un perceptrón entrenado para minimizar el error cuadrático medio de las muestras de entrenamiento de un problema de clasificación aproximan la distribución a-posteriori subyacente en las muestras de entrenamiento.*

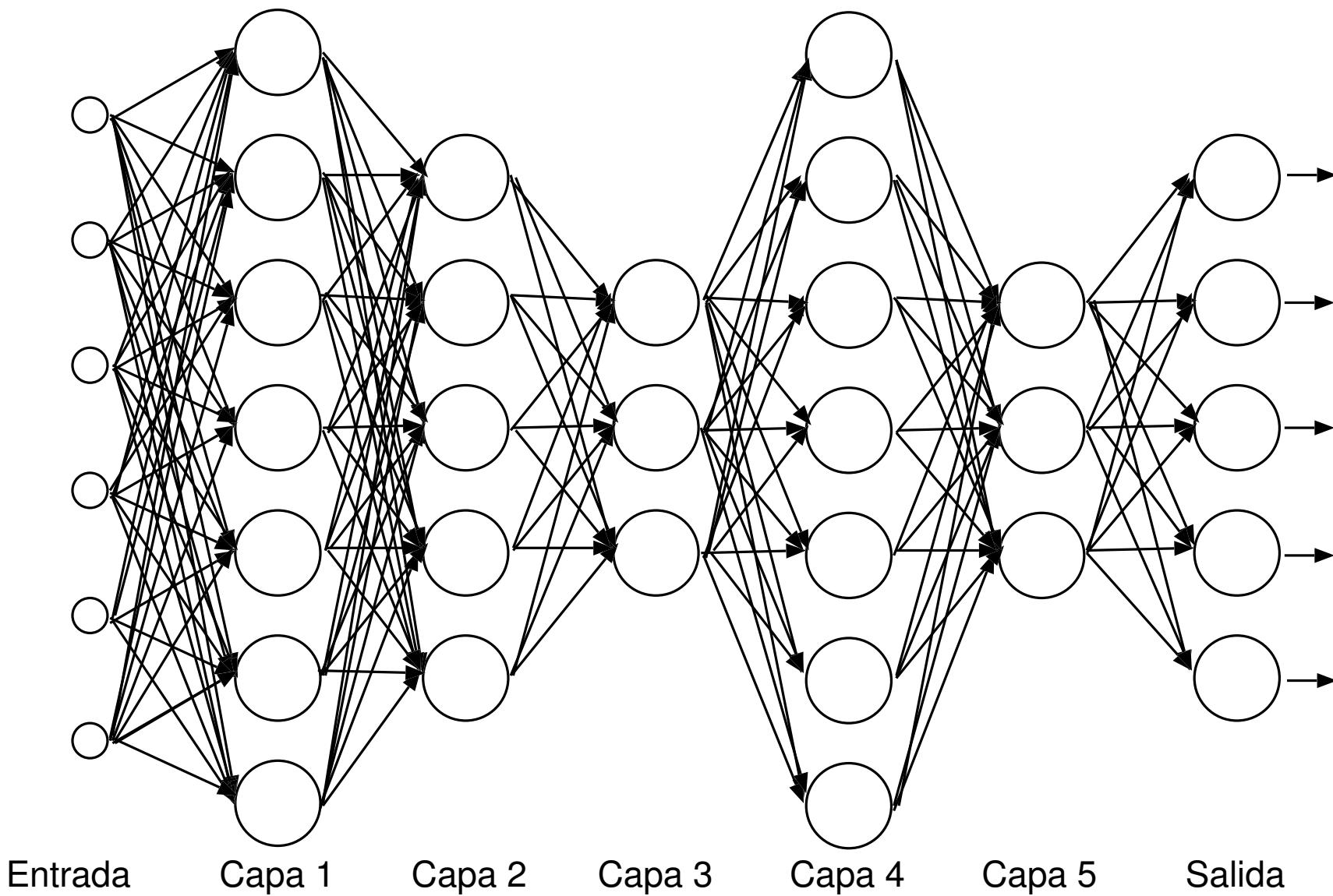
Reducción del tamaño de las redes neuronales

- MÉTODOS DE PODA:
 - Poda de pesos basadas en:
 - * RELEVANCIA
 - * CASTIGO
- Quantización de los parámetros.
- Entrenamiento de redes pequeñas a partir de datos generados por redes grandes (Knowledge distillation)

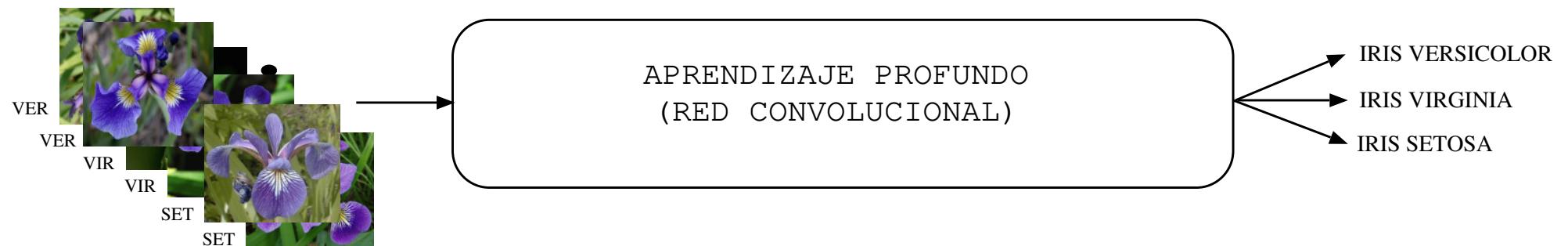
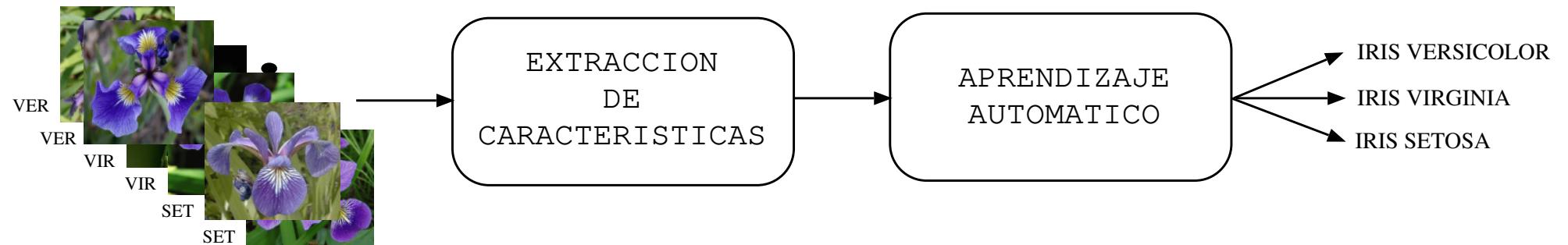
Index

- 1 Redes neuronales multicapa ▷ 2
- 2 Algoritmo de retropropagación del error (BackProp) ▷ 19
- 3 Aspectos de uso y propiedades del BackProp ▷ 34
- 4 *Introducción a las redes profundas* ▷ 49
- 5 PyTorch para redes neuronales ▷ 60
- 6 Perceptrón con varias capas en PyTorch ▷ 63
- 7 Redes convolucionales en PyTorch ▷ 65
- 8 Visión por computador ▷ 71
- 9 Modelos secuenciales ▷ 93
- 10 ¿Que sigue? ▷ 110
- 11 Notación ▷ 112

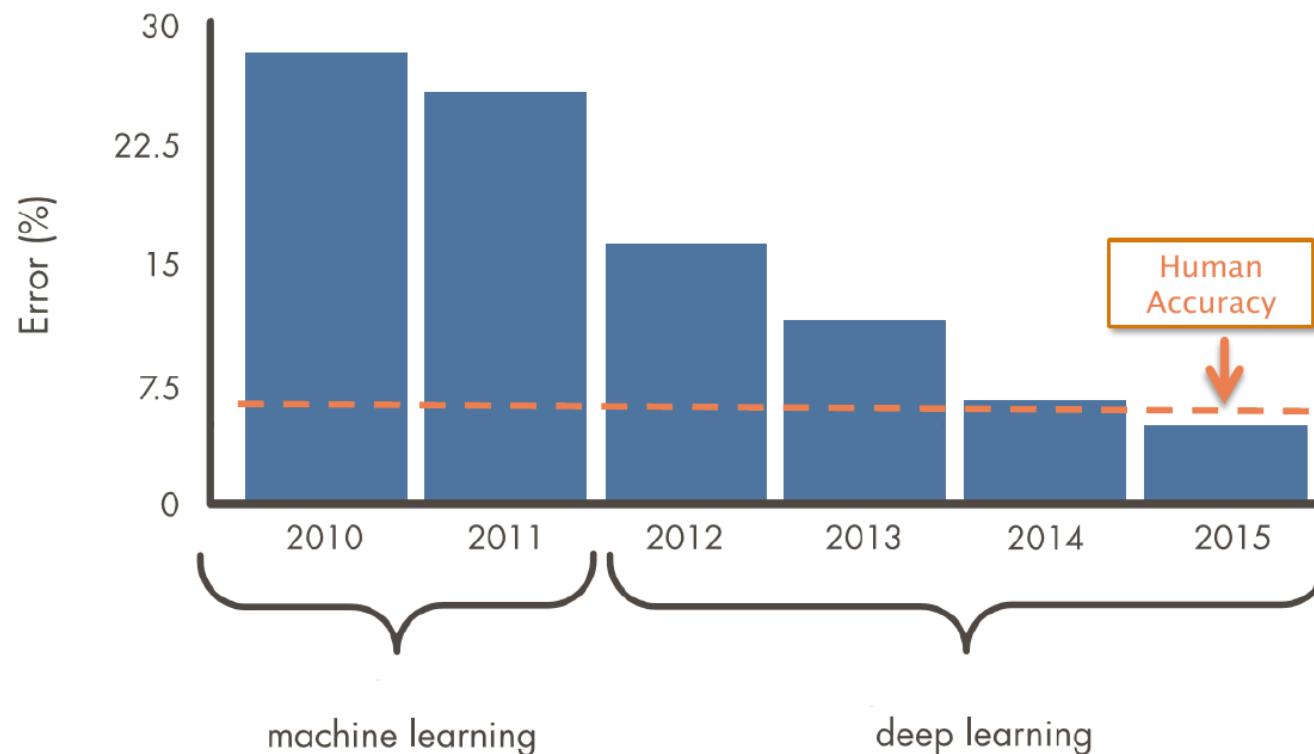
Concepto de red profunda



Aprendizaje profundo [Daly 2017]

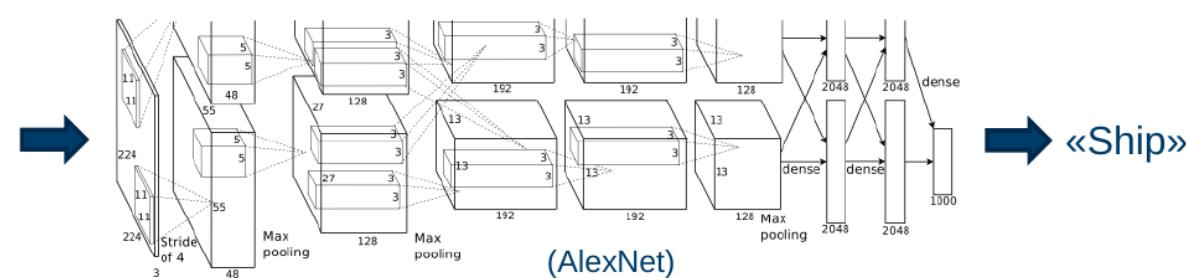
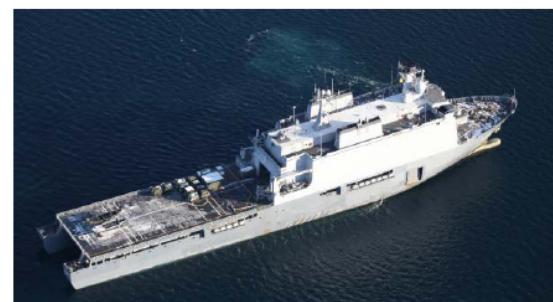


Aprendizaje profundo [Daly 2017]



Source: ILSVRC Top-5 Error on ImageNet

Aprendizaje profundo [Dyrdal 2019]



Millions of images

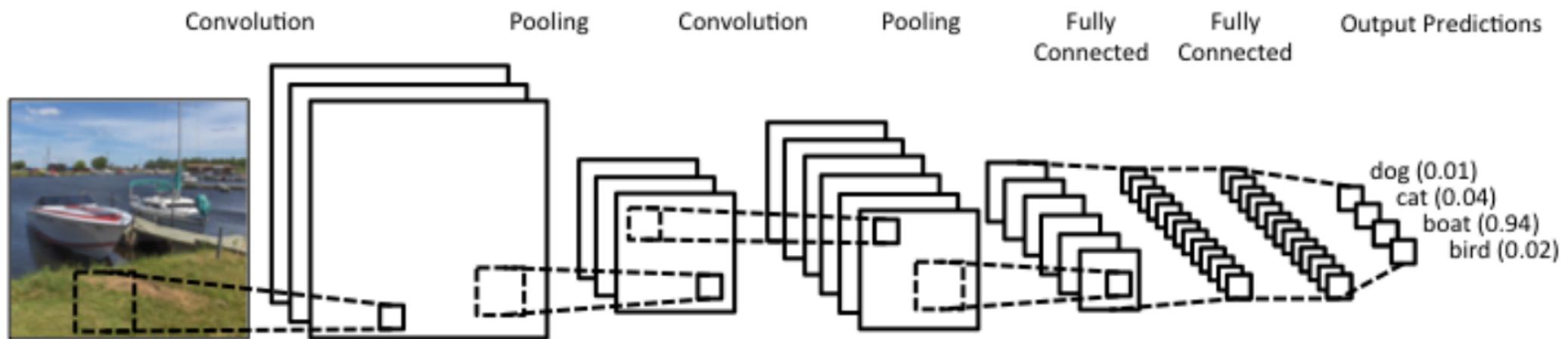
Millions of parameters

Thousands of classes

Casos importantes de redes profundas

- Redes recurrentes.
 - Red recurrente simple.
 - Red de Elman: recurrente + perceptrón en la salida.
 - Redes recurrentes de segundo orden.
 - Long Short-Term Memory (LSTM)
 - Gated Recurrent Units (GRU)
- Redes dinámicas no recurrentes.
 - Redes convolucionales.
 - Transformer (para texto e imágenes).
 - Redes preentrenadas (BERT, GPT, XML, ...).

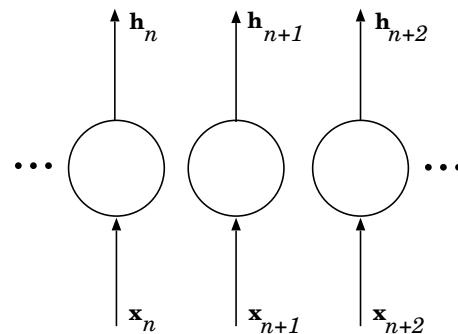
Ejemplo de red convolucional



Redes dinámicas

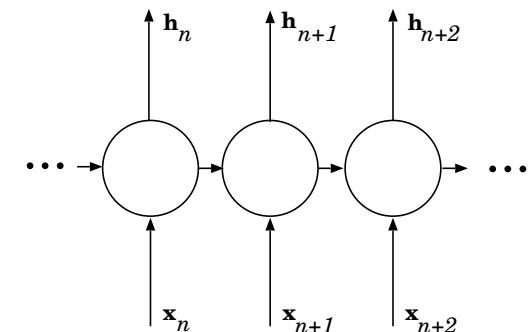
- Redes diseñadas para tratar secuencias $x_n \in \mathbb{R}^{d_X}$ y producir secuencias $y_n \in \mathbb{R}^{d_Y}$, $n = 1, 2, \dots$
- Un perceptrón multicapa puede ser usado para generar la secuencia de salida:

$$\mathbf{y}_n = f(W\mathbf{x}_n). \quad n = 1, 2, \dots$$

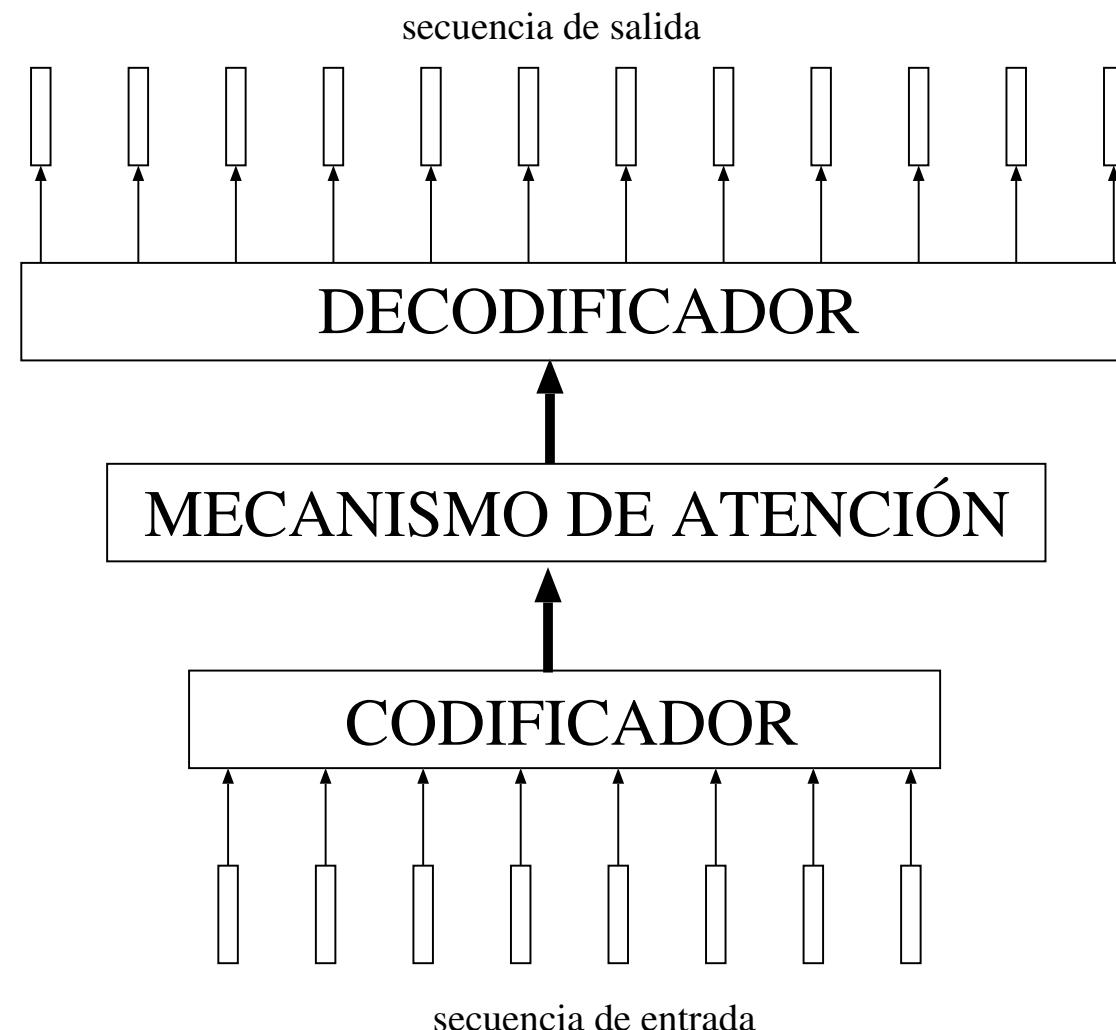


- Una red recurrente tiene en consideración lo que ha generado anteriormente:

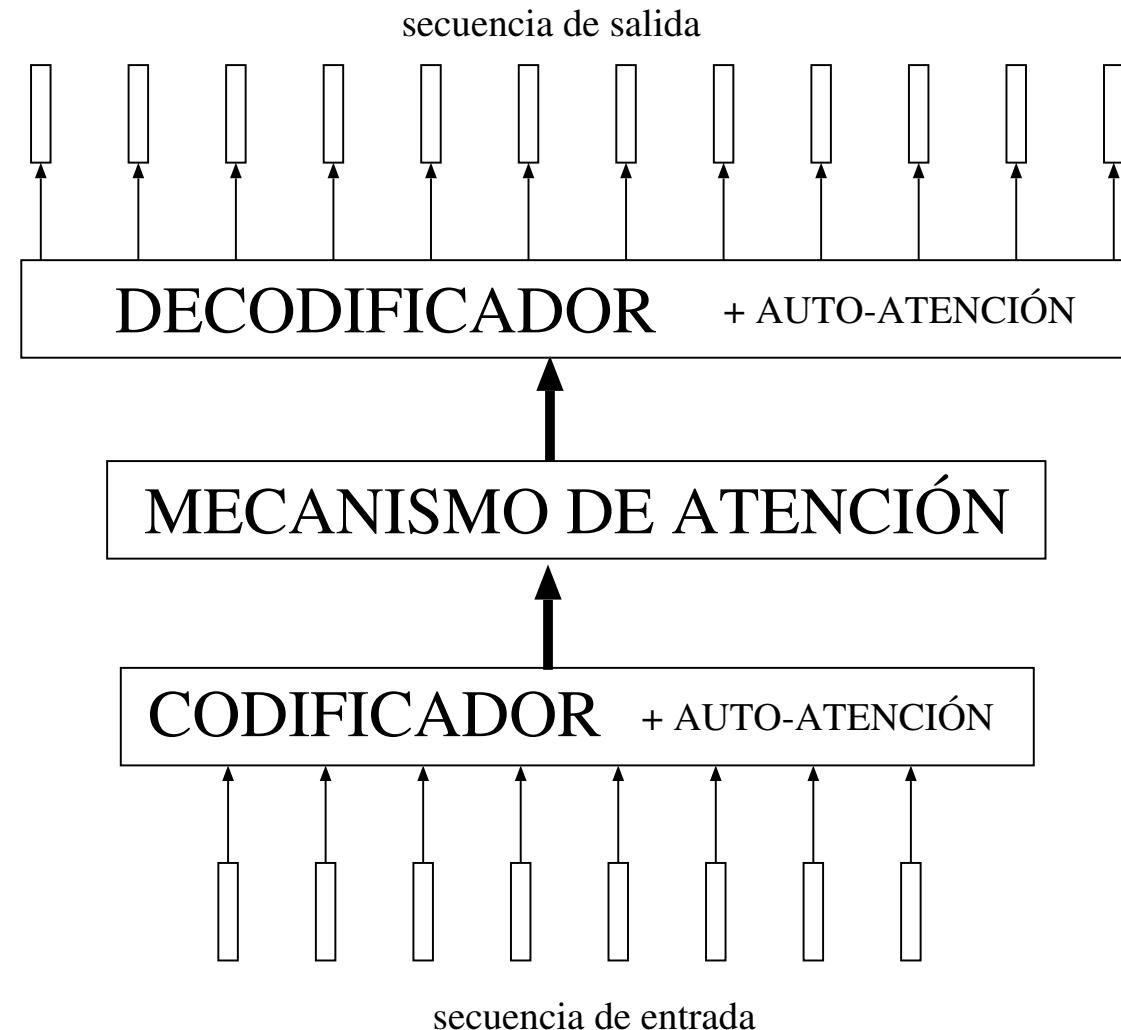
$$\mathbf{y}_n = f(W^Y \mathbf{y}_{n-1} + W^X \mathbf{x}_n) \quad n = 1, 2, \dots$$



Arquitectura condificador-decodificador



Arquitectura totalmente con modelos de atención



Aprendizaje de redes dinámicas

- Al desplegar una red dinámica para una secuencia de entrada se obtiene una red no recurrente de tantas capas como tamaño tenga la secuencia de entrada.
- Entrenamiento mediante “Back-propagation through time”: aplicación del algoritmos de propagación del error convencional sobre la red desplegada.
- Uso de los mismos algoritmo de optimización sobre los grafos de computación generados por los toolkits convencionales.
- Inferencia con redes dinámicas: voraz o búsqueda en árbol (con poda).

Index

- 1 Redes neuronales multicapa ▷ 2
- 2 Algoritmo de retropropagación del error (BackProp) ▷ 19
- 3 Aspectos de uso y propiedades del BackProp ▷ 34
- 4 Introducción a las redes profundas ▷ 49
- 5 *PyTorch para redes neuronales* ▷ 60
- 6 Perceptrón con varias capas en PyTorch ▷ 63
- 7 Redes convolucionales en PyTorch ▷ 65
- 8 Visión por computador ▷ 71
- 9 Modelos secuenciales ▷ 93
- 10 ¿Que sigue? ▷ 110
- 11 Notación ▷ 112

Toolkits

La red neuronal como un grafo de computación que soporta la propagación del gradiente:

- TensorFlow - Google Brain - Python, C/C++
- PyTorch - Facebook - Python
- Caffe - UC Berkeley / Caffe2 Facebook, Python, MATLAB
- Higher level interfaces e.g. Keras for TensorFlow.
- CUDA/GPU/CLOUD support.

Características de PyTorch

- Es una librería para Python.
- Cálculo automático del gradiente.
- Uso de GPUs.
- Módulos y optimizadores.

Notebook [URL aquí].

https://colab.research.google.com/drive/1Aw8G2EkOWP1XB_xTfBkHNgzI_NPcefgQ?usp=sharing

Index

- 1 Redes neuronales multicapa ▷ 2
- 2 Algoritmo de retropropagación del error (BackProp) ▷ 19
- 3 Aspectos de uso y propiedades del BackProp ▷ 34
- 4 Introducción a las redes profundas ▷ 49
- 5 PyTorch para redes neuronales ▷ 60
 - 6 *Perceptrón con varias capas en PyTorch* ▷ 63
- 7 Redes convolucionales en PyTorch ▷ 65
- 8 Visión por computador ▷ 71
- 9 Modelos secuenciales ▷ 93
- 10 ¿Que sigue? ▷ 110
- 11 Notación ▷ 112

Perceptrón con varias capas en PyTorch

- Cargamos los datos con “torchvision”.
- Definimos el modelo usando “nn.module”.
- Definimos la función de pérdida, el optimizador y entrenamos.
- Evaluamos.

Celda del Notebook . . . https://colab.research.google.com/drive/1Aw8G2EkOWPlXB_xTfBkHNgzI_NPcefQ#scrollTo=1fMxDT2kRTew&line=21&uniqifier=1

Index

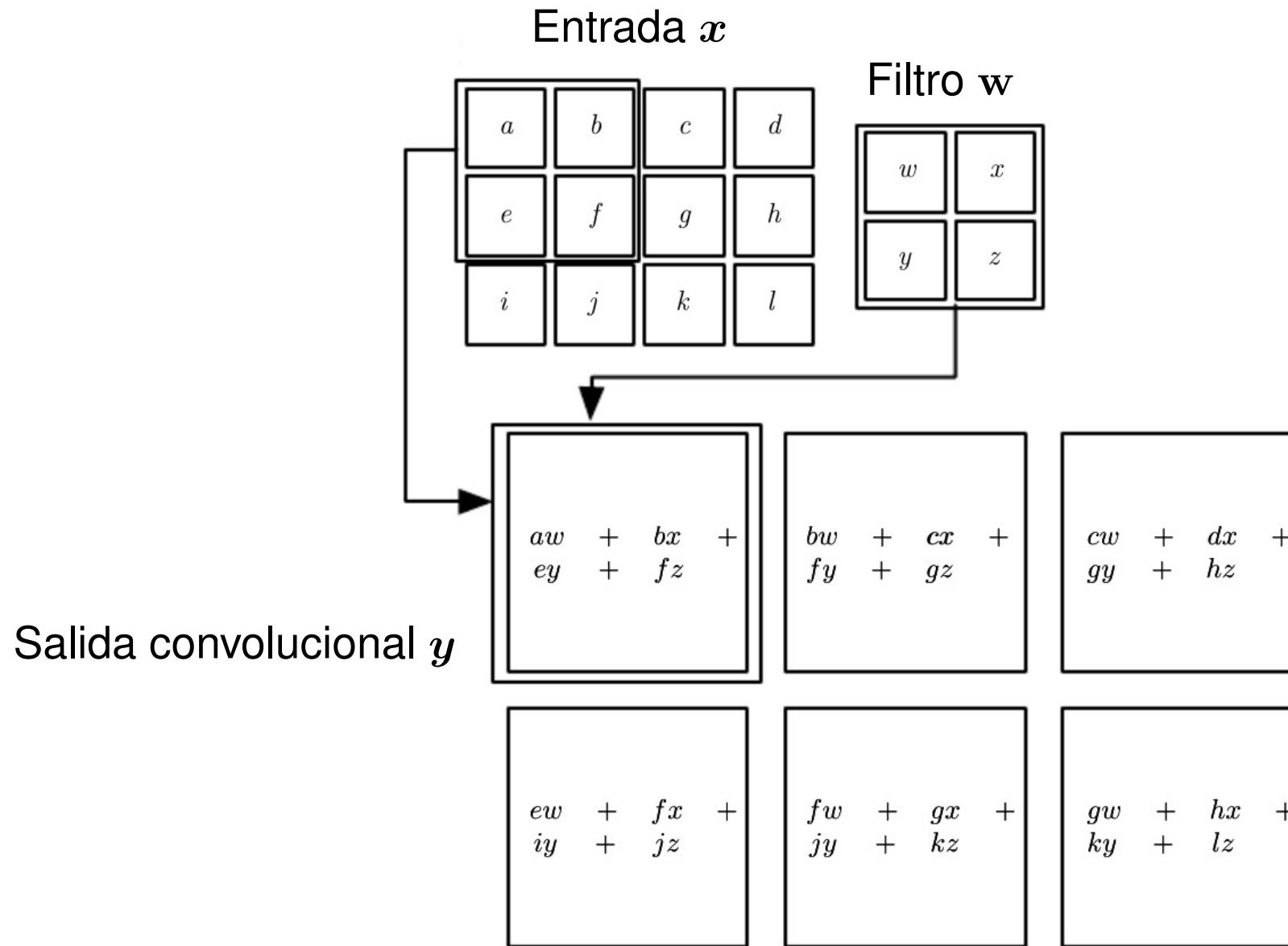
- 1 Redes neuronales multicapa ▷ 2
- 2 Algoritmo de retropropagación del error (BackProp) ▷ 19
- 3 Aspectos de uso y propiedades del BackProp ▷ 34
- 4 Introducción a las redes profundas ▷ 49
- 5 PyTorch para redes neuronales ▷ 60
- 6 Perceptrón con varias capas en PyTorch ▷ 63
 - 7 *Redes convolucionales en PyTorch* ▷ 65
- 8 Visión por computador ▷ 71
- 9 Modelos secuenciales ▷ 93
- 10 ¿Que sigue? ▷ 110
- 11 Notación ▷ 112

Redes convolucionales

- Permiten trabajar de forma más eficaz con imágenes y otros tipos de datos estructurados.
- Tamaño filtro o kernel $M \times N$.
- Paso del filtro (Stride) horizontal y vertical.
- Relleno a ceros u otro valor (Padding) si el filtro se sale de la imagen.

Celda del Notebook . https://colab.research.google.com/drive/1Aw8G2EkOWP1XB_xTfBkHNgzI_NPcefgQ#scrollTo=c1Fj5psNXK0t

Convolución [Goodfellow 2016]

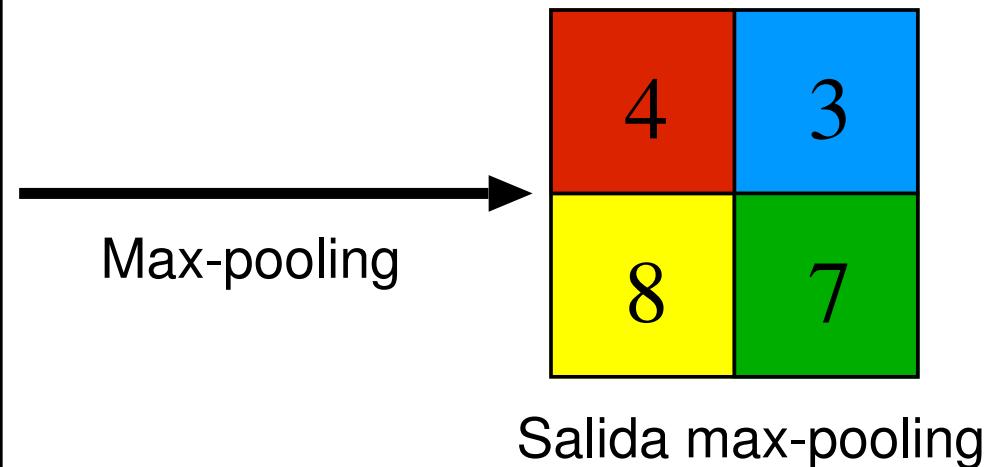


Pooling

- Valor máximo
- Valor medio

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

Salida convolucional



Max-pooling

Salida max-pooling

Redes convolucionales

- Capa 1 convolucional + función de activación ReLU
- Capa 1 pooling
- ...
- Capa L convolucional + función de activación ReLU
- Capa L pooling
- Una red totalmente conectada.
- Salida softmax

Redes convolucionales vs full-connected

- Una red totalmente conectada, como un MLP, para una imagen puede ser tener un número enorme de pesos. Estamos conectando todos los píxeles de una imagen con todos los pesos de la capa de entrada, sin tener en cuenta su estructura.
- Las redes convolucionales están basadas en filtros, redes pequeñas que se desplazan por la imagen. Sabemos que la imagen tiene una estructura, y que los píxeles más cercanos probablemente tengan más que ver entre ellos que los píxeles más lejanos. El kernel de una convolución aprovecha esta característica.

Index

- 1 Redes neuronales multicapa ▷ 2
- 2 Algoritmo de retropropagación del error (BackProp) ▷ 19
- 3 Aspectos de uso y propiedades del BackProp ▷ 34
- 4 Introducción a las redes profundas ▷ 49
- 5 PyTorch para redes neuronales ▷ 60
- 6 Perceptrón con varias capas en PyTorch ▷ 63
- 7 Redes convolucionales en PyTorch ▷ 65
- 8 *Visión por computador* ▷ 71
- 9 Modelos secuenciales ▷ 93
- 10 ¿Que sigue? ▷ 110
- 11 Notación ▷ 112

Visión por computador

- Campo trata de reconocer y entender imágenes y escenas.
- También engloba aspectos como reconocimiento de imágenes, generación, super-resolución, etc.
- Las redes convolucionales han sido ampliamente utilizadas a lo largo de la historia en este campo.

Clasificación de imágenes

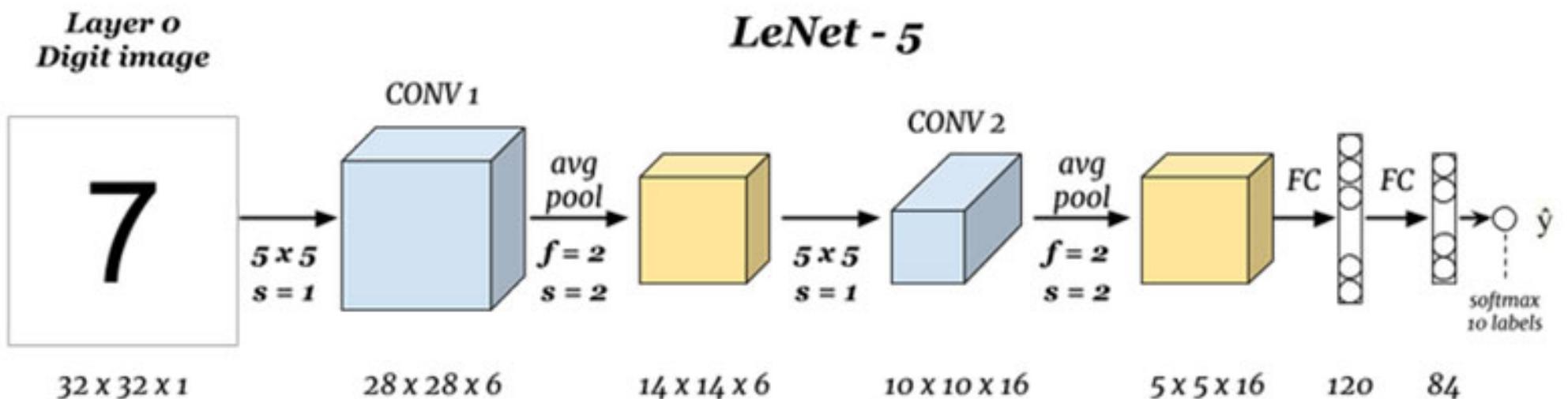
Clasificar cada imagen por su contenido. Por ejemplo,
¿contienen estas imágenes abejas o hormigas?



Veremos algunos ejemplos en:

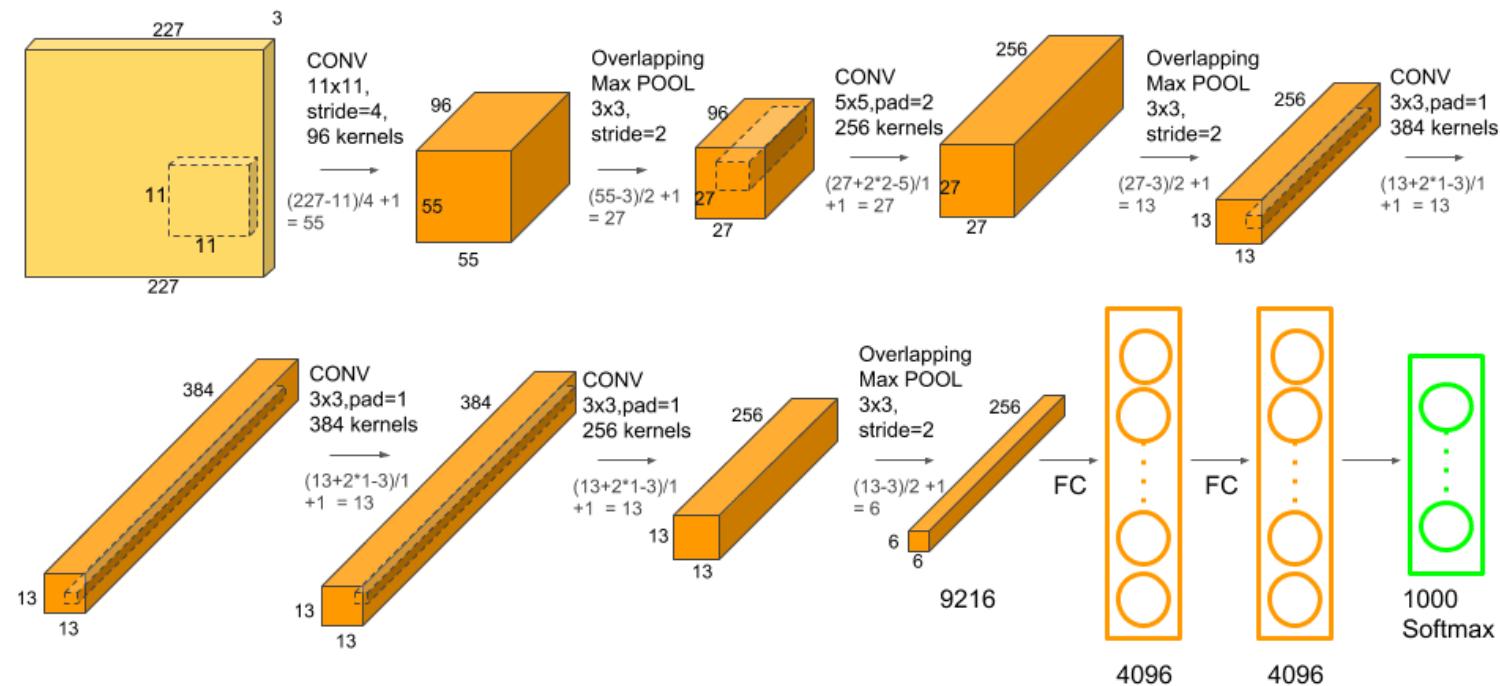
https://colab.research.google.com/drive/1VL6q_26sYD3kYAmndCfjuKGPMokXOhY?usp=sharing

Redes clásicas - LeNet-5



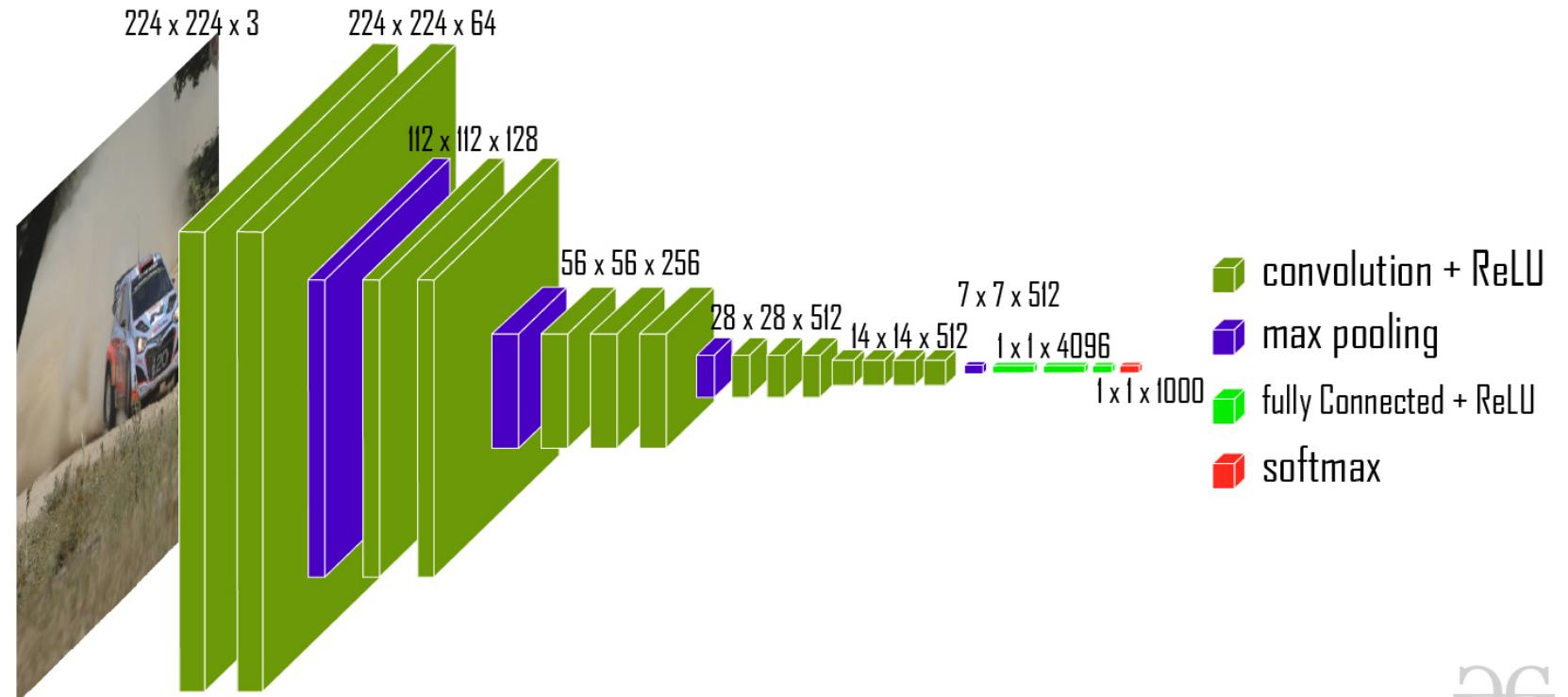
- Alrededor de 60.000 parámetros entrenables.
- Disminuye el el ancho y alto del tensor en cada bloque pero aumenta el número de canales.
- Uso de Sigmoid y Tanh como función de activación.

Redes clásicas - AlexNet



- Alrededor de 60 millones de parámetros entrenables.
- Similar a LeNet pero mucho más grande.
- Utiliza ReLU como función de activación.
- De los primeros modelos en utilizar dos GPUs (2012).

Redes clásicas - VGG-16



- Alrededor de 138 millones de parámetros entrenables.
- Bloques de convoluciones con kernels de 3×3 y stride 1.
- MaxPooling de 2×2 y stride 2.
- Primera red profunda con 13 capas convolucionales. Cada capa convolucional doblando el número de filtros de la anterior.



Arquitecturas ResNets

Al entrenar redes muy profundas el gradiente se desvanece.

$$s^l \rightarrow \text{Linear}_1(s^l) \rightarrow \text{ReLU}(\phi^{l+1}) \rightarrow \text{Linear}(s^{l+1}) \rightarrow \text{ReLU}(\phi^{l+2}) \rightarrow (s^{l+2})$$

$$\phi^{l+1} = \Theta^{l+1} s^l \rightarrow s^{l+1} = f_{relu}(\phi^{l+1}) \rightarrow \phi^{l+2} = \Theta^{l+2} s^{l+1} \rightarrow s^{l+2} = f_{relu}(\phi^{l+2})$$

En las arquitecturas ResNet se añade un "atajo", comúnmente llamados *Skip connection* en la literatura.

$$\phi^{l+1} = \Theta^{l+1} s^l \rightarrow s^{l+1} = f_{relu}(\phi^{l+1}) \rightarrow \phi^{l+2} = \Theta^{l+2} s^{l+1} \rightarrow s^{l+2} = f_{relu}(\phi^{l+2} + s^l)$$

$$s^l \rightarrow \text{Linear}_1(s^l) \rightarrow \text{ReLU}(\phi^{l+1}) \rightarrow \text{Linear}(s^{l+1}) \rightarrow \text{ReLU}(\phi^{l+2} + s^l) \rightarrow (s^{l+2})$$

Esto nos permitirá entrenar redes mucho más profundas.

Arquitecturas ResNets

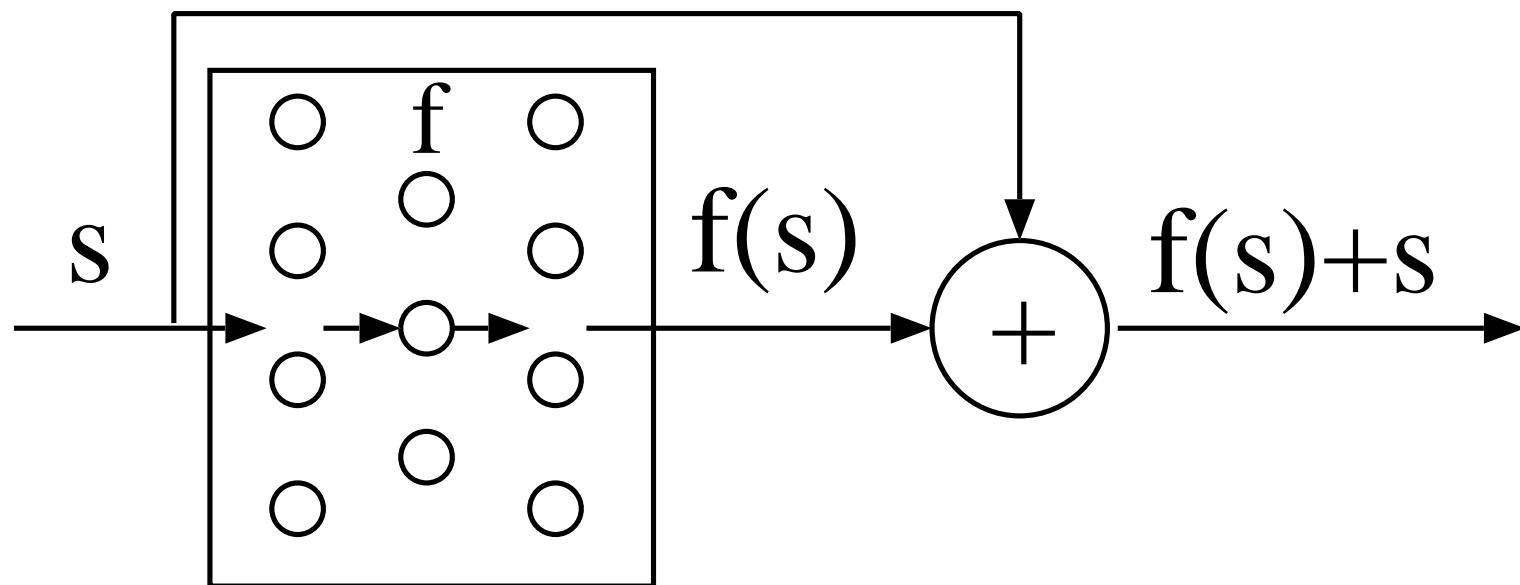


Figure 1: Bloque residual.

Arquitecturas ResNets

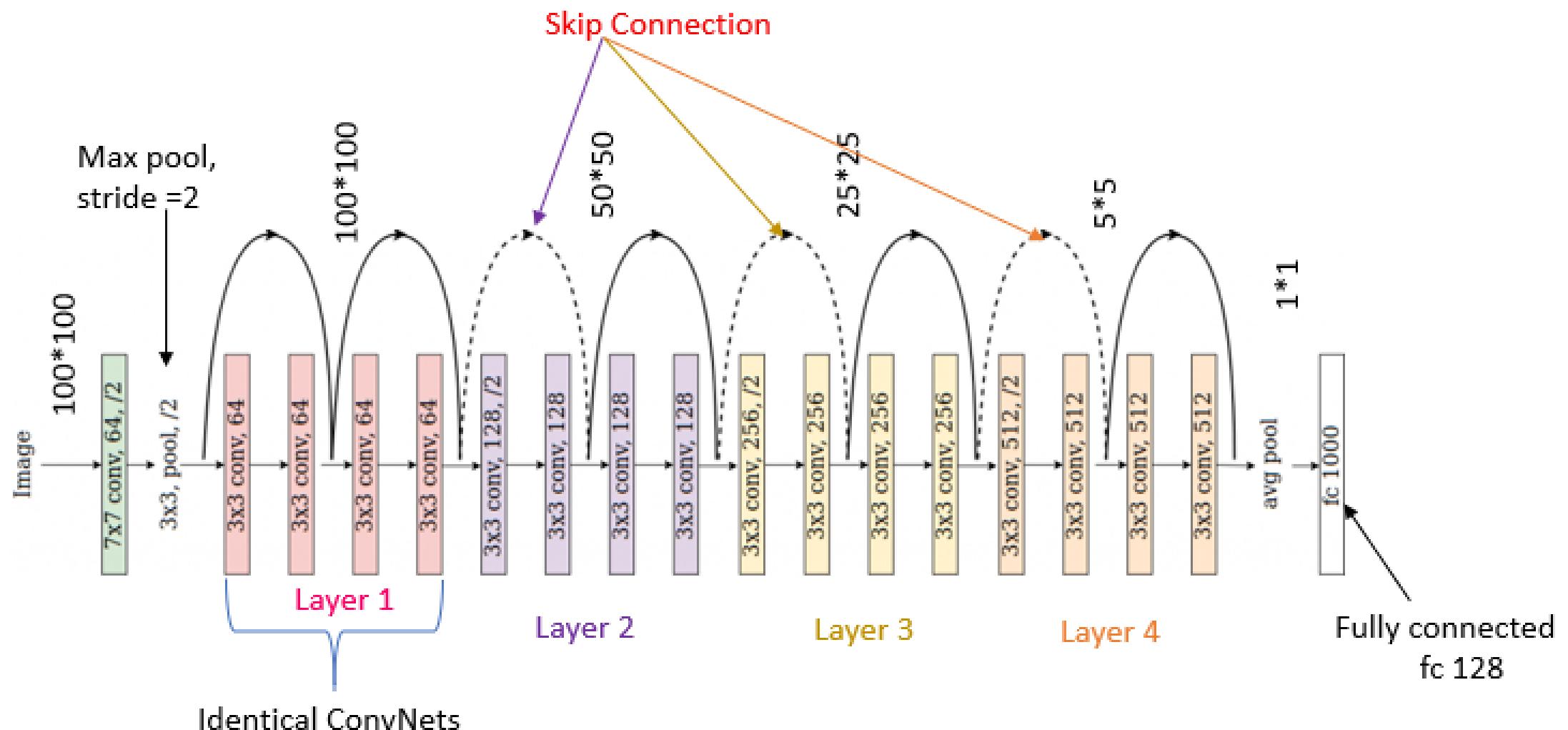
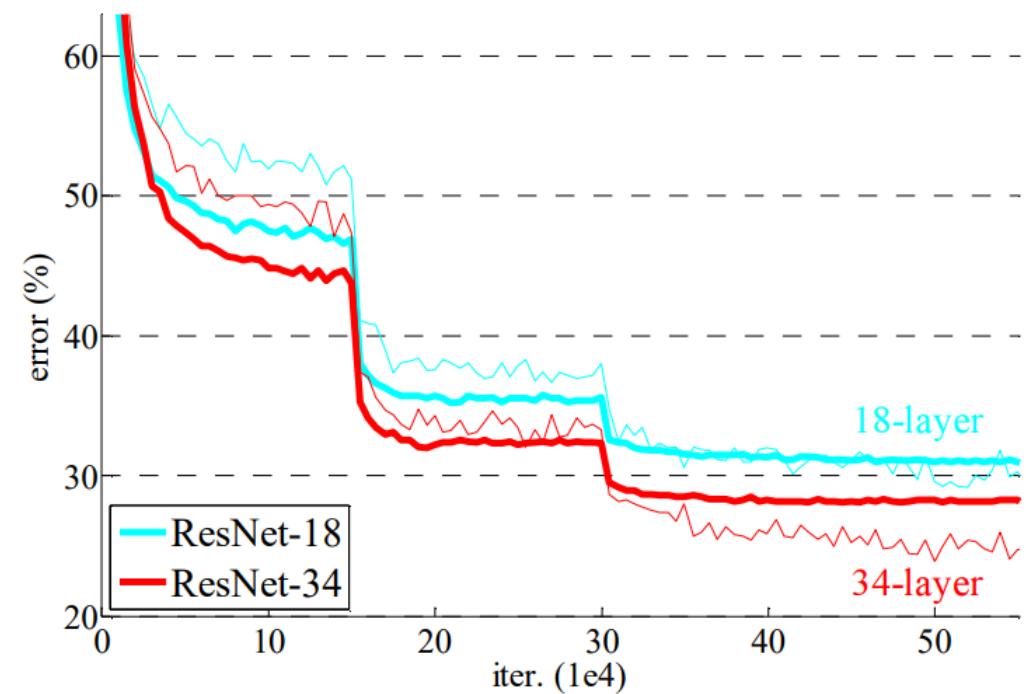
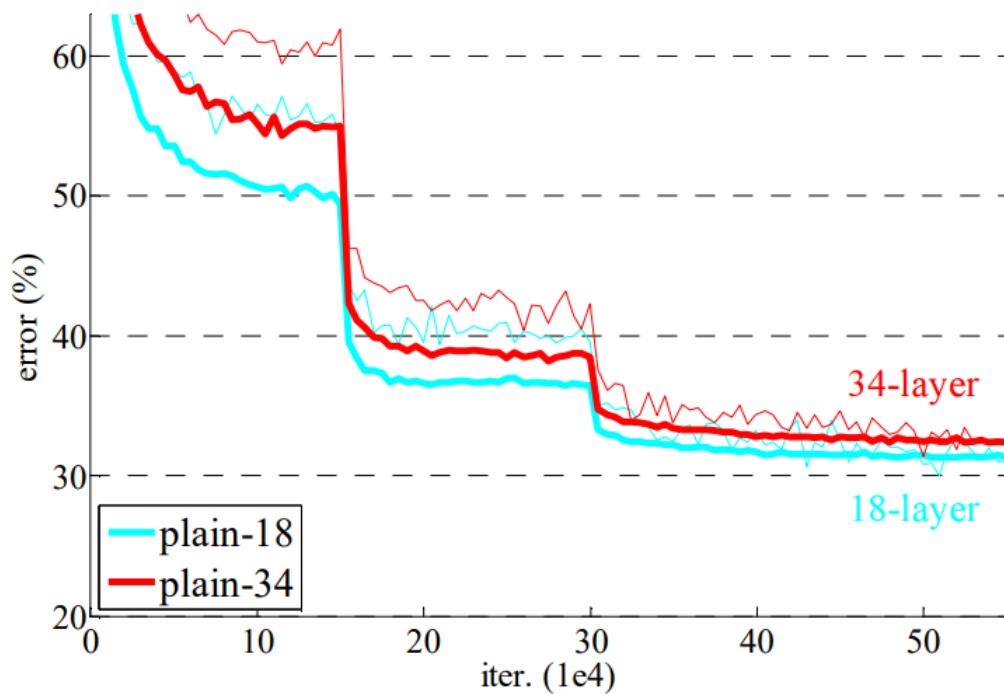


Figure 2: Arquitectura ResNet-18 con alrededor de 11 millones de parámetros entrenables.

Arquitecturas ResNets



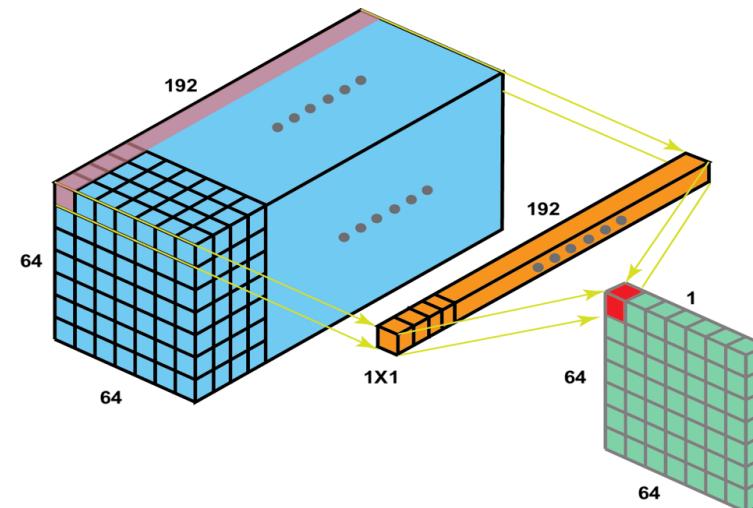
Las líneas más delgadas indican el error en la partición de entrenamiento mientras que las líneas más marcadas son las de la partición de validación. Todo entrenado con ImageNet.

Izquierdo: redes neuronales "planas" de 18 y 34 capas convolucionales.

Derecha: Las mismas redes pero con conexiones residuales.

Estas últimas no tienen parámetros extra respecto a su contraparte de la izquierda.

Convoluciones 1x1

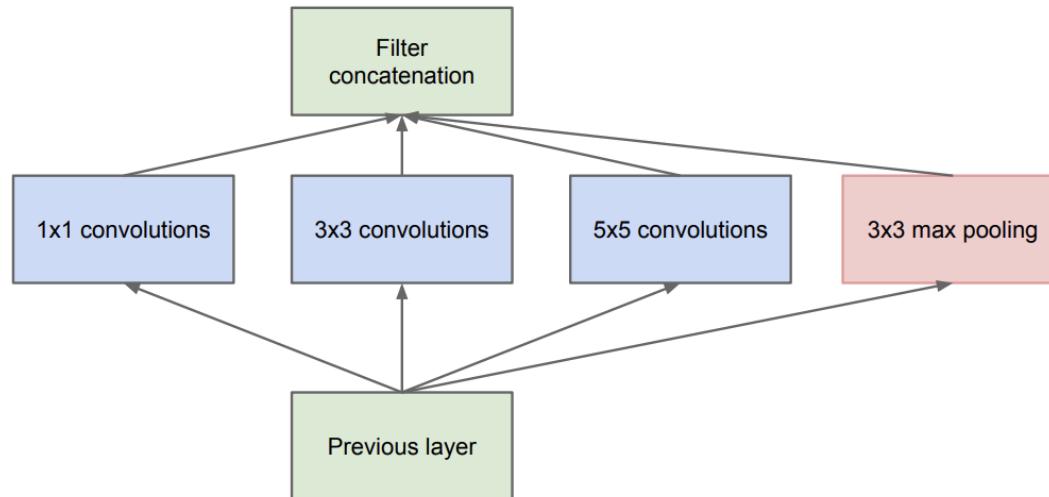


- Como hemos visto, la tendencia iba a hacer redes convolucionales más complejas, pesadas y obteniendo más filtros a cada paso.
- Podemos usar convoluciones con un kernel 1×1 para reducir la dimensionalidad.
- Este kernel tratará de juntar muchos de los canales añadiendo, además, otra capa no lineal. Esto nos permitirá aprender funciones más complicadas.

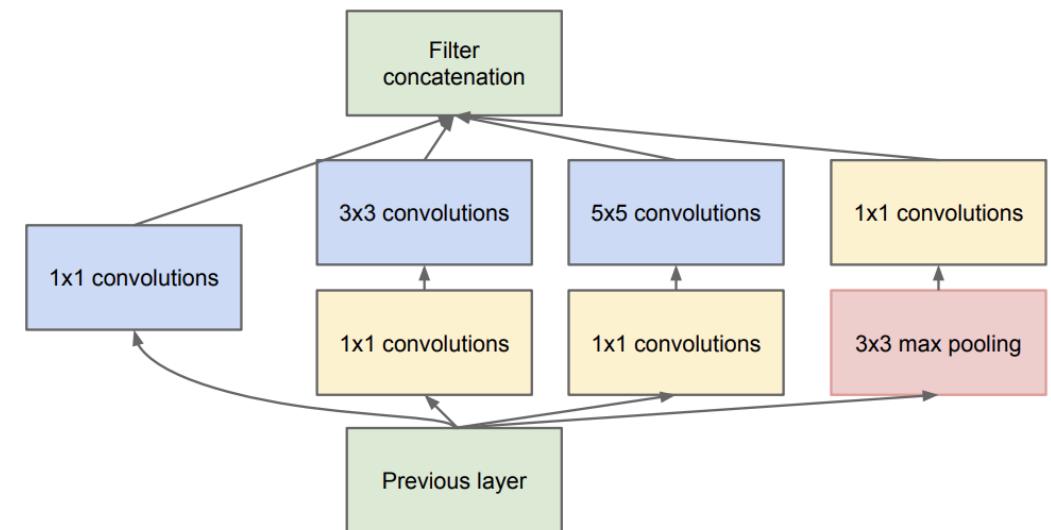
Arquitecturas Inception

- Hasta ahora vemos que se usan diferentes tamaños de kernels.
- Cada tamaño tiene en cuenta un contexto más grande o pequeño y es una decisión más a tomar a la hora de crear una red neuronal convolucional.
- En el modelo *Inception* se implementaron los bloques *inception*, en los cuales se aplican diferentes convoluciones de diferentes tamaños de kernel y después se concatenan todos.
- No obstante, esto aumenta el número de parámetros, sobre todo al usar kernels ligeramente grandes. En otra versión se aplicaron primero kernels 1×1 para reducir primero la dimensionalidad.

Arquitecturas Inception - Bloques

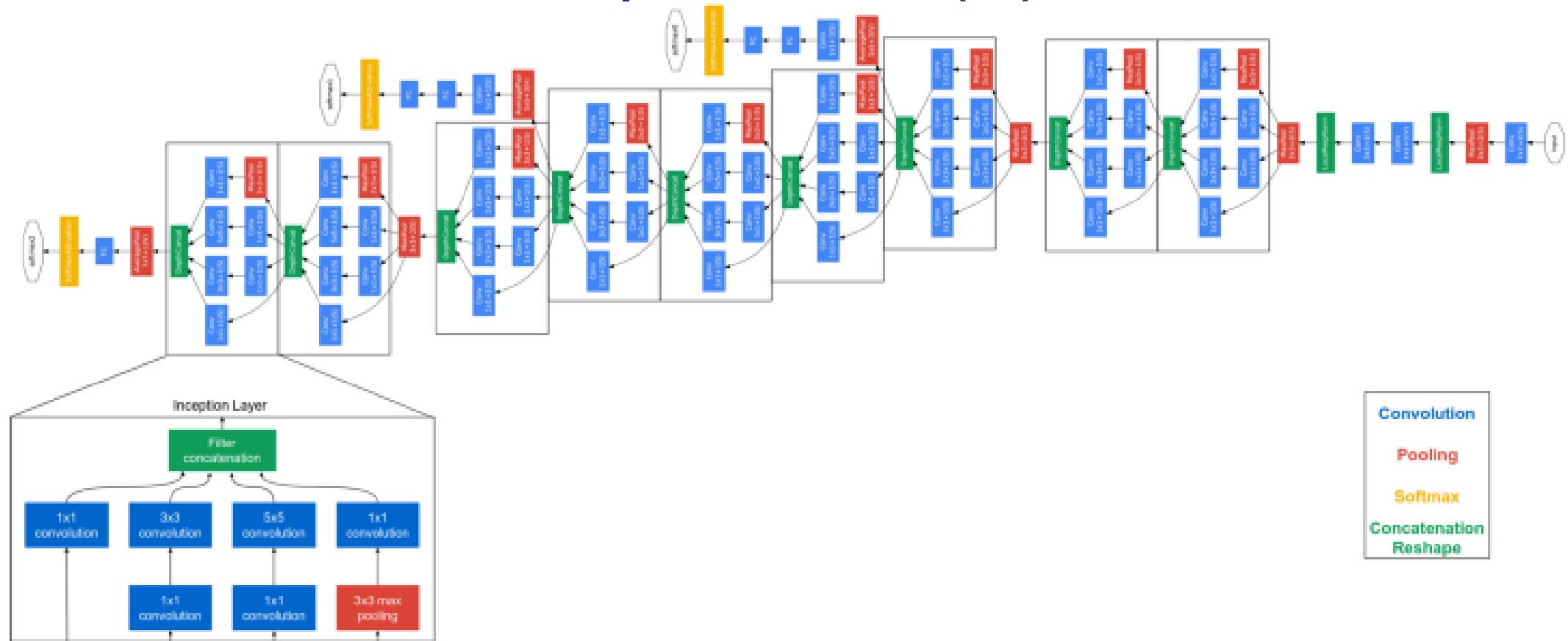


(a) Inception module, naïve version



(b) Inception module with dimension reductions

Inception network (v1)

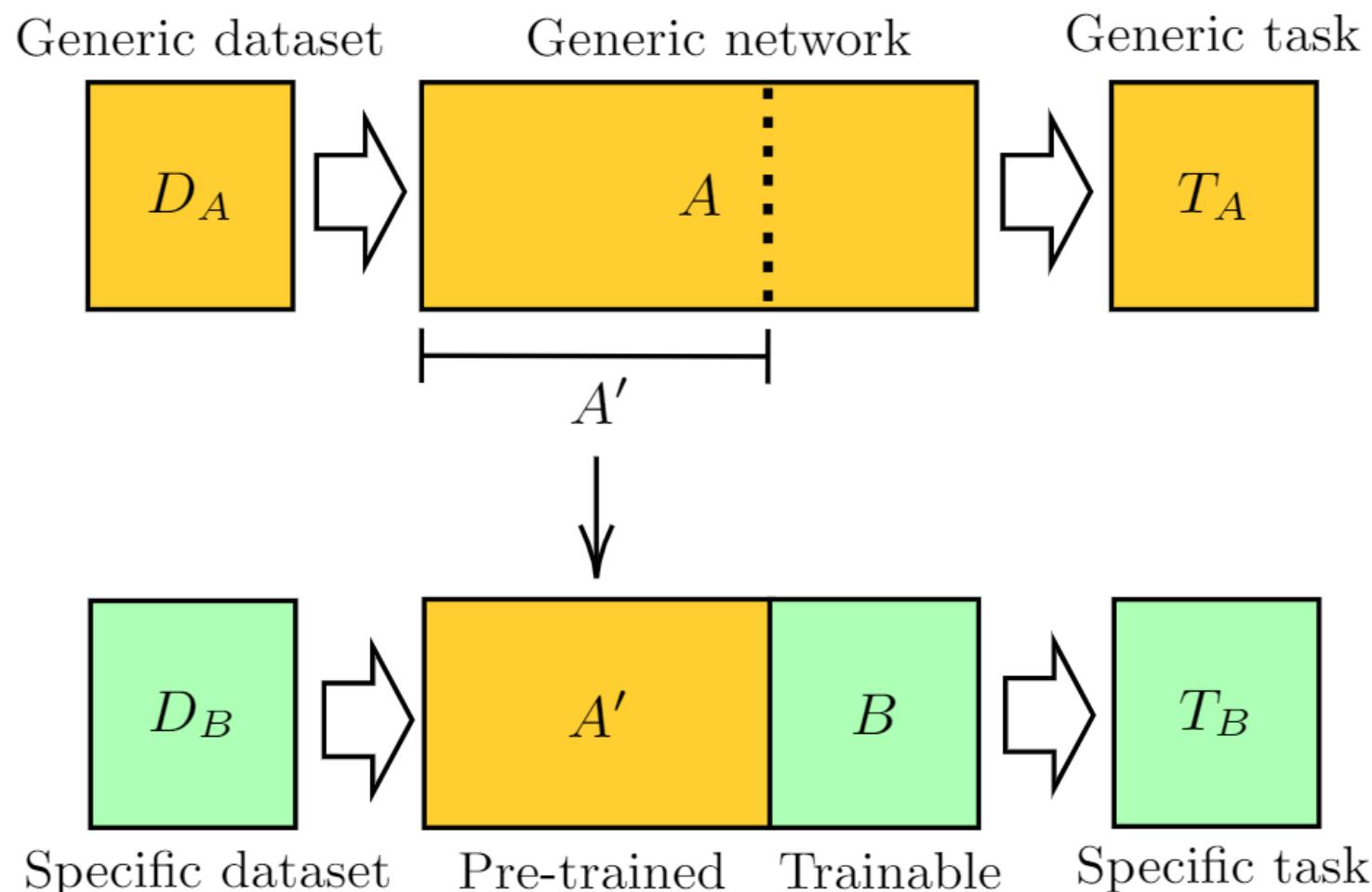


- Alrededor de 10 millones de parámetros.
- Funciones de pérdida auxiliares ponderadas en diferentes partes de la red.

Transferencia del aprendizaje

- Entrenar un modelo de zero puede ser muy costoso.
- Podemos tener muy pocas muestras de aprendizaje y muchos parámetros a entrenar.
- Con la transferencia de aprendizaje (Transfer learning) podemos aprovechar los pesos de un modelo ya entrenado en una tarea similar para nuestro problema.

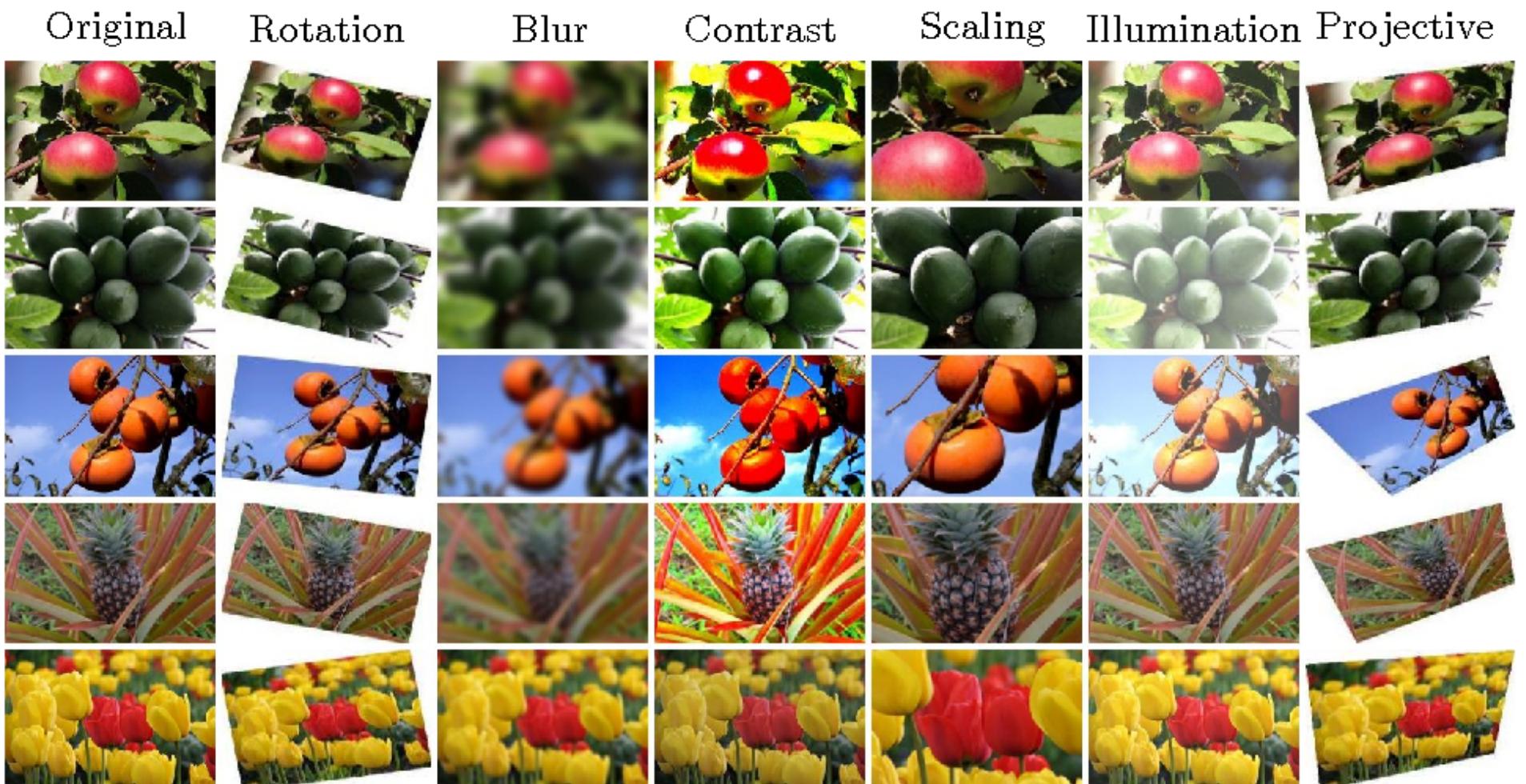
Transferencia del aprendizaje



Data Augmentation

- Se trata de crear datos nuevos para entrenamiento a partir de los que ya tenemos.
- Sobre una imagen se pueden aplicar distorsiones, recortes, giros y un largo etc.
- Se aplican solo sobre el conjunto de entrenamiento y evitan que el modelo sobre entrene y se aprenda los datos de memoria (no generalice).

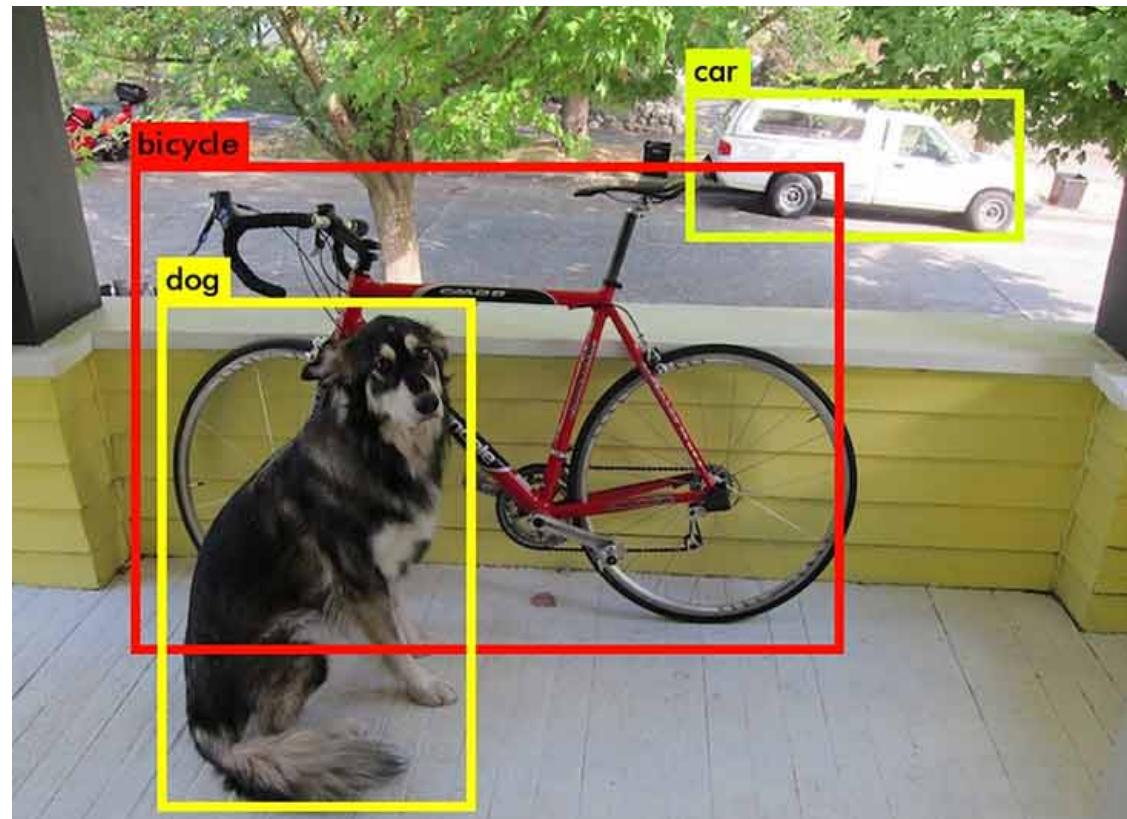
Data Augmentation



Detección de objetos

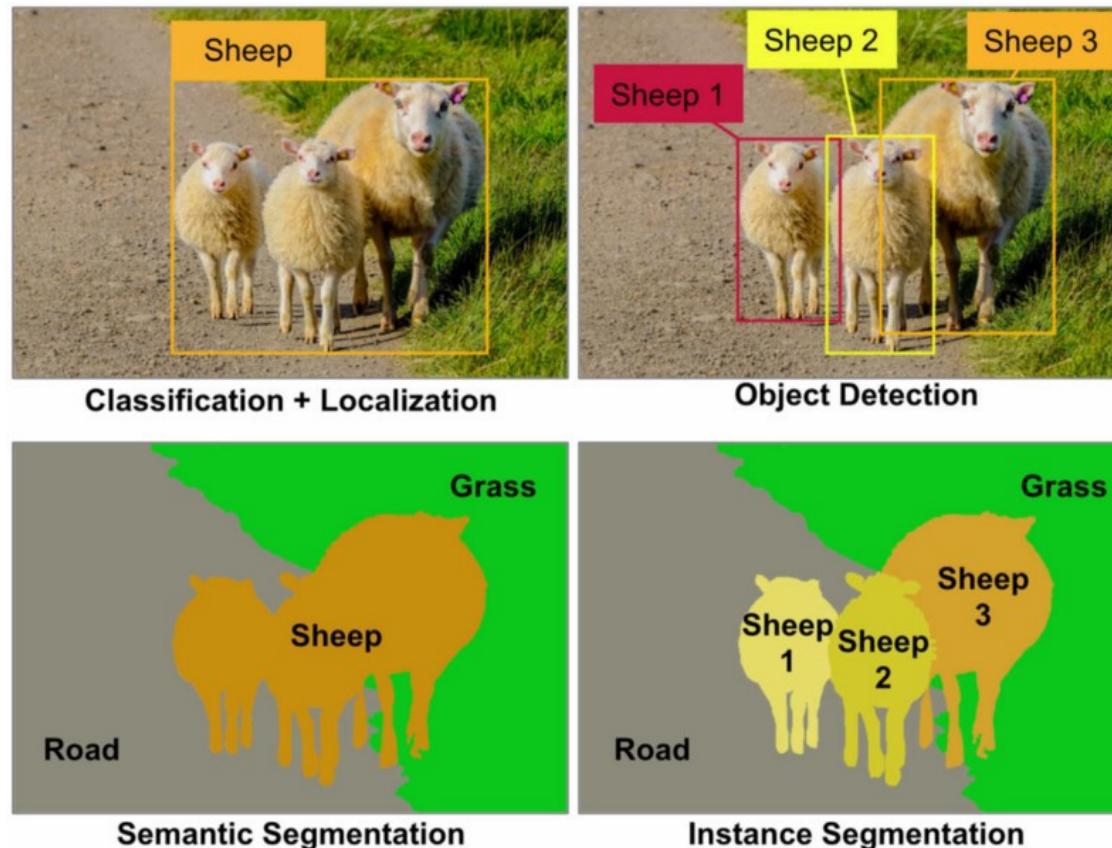
- Localizar e identificar objetos en una imagen.
- Usado históricamente en diferentes tareas como la detección de rostros, localización de vehículos, contador de peatones, sistemas de seguridad, conducción autónoma, etc.
- Queremos saber que objetos hay en la imagen y donde están. Por lo tanto, tendremos que buscar unas coordenadas de los objetos y clasificarlos.

Detección de objetos



Segmentación

- Clasificación píxel a píxel.
- Podemos hacerlo sobre toda la imagen o sobre una instancia encontrada en el paso anterior.



Detección de objetos y segmentación - MaskRCNN

- Uno de los modelos más utilizados es MaskRCNN

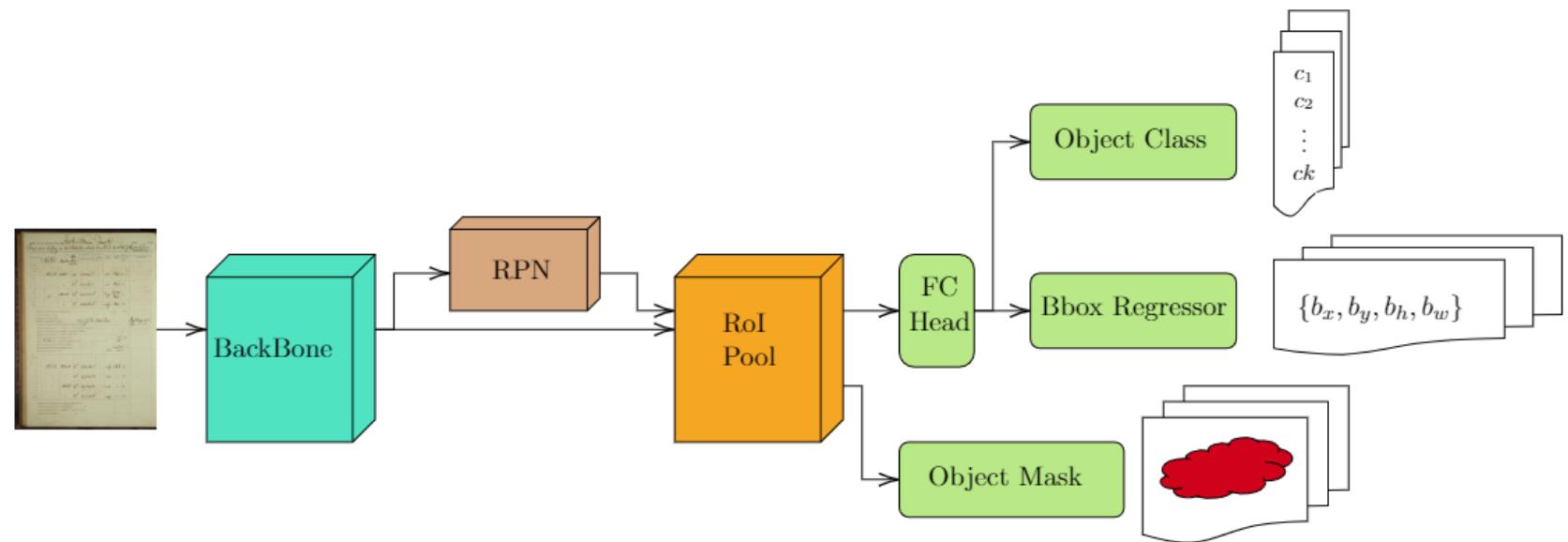


Figure 3: Diagrama básico de la arquitectura basada en Mask-RCNN.

https://colab.research.google.com/drive/1u3g0QmCLUrjllKCETKyGO_IMUfc4Z5pl?usp=sharing

Index

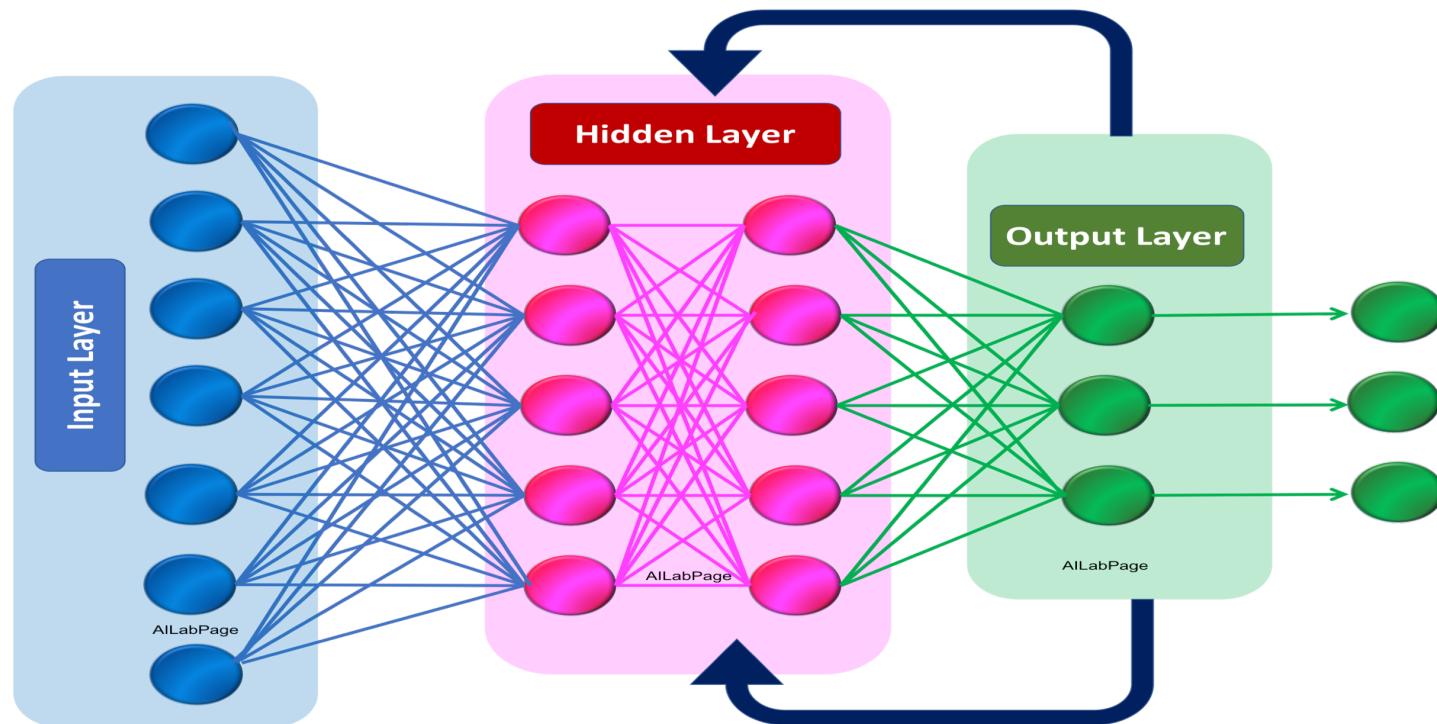
- 1 Redes neuronales multicapa ▷ 2
- 2 Algoritmo de retropropagación del error (BackProp) ▷ 19
- 3 Aspectos de uso y propiedades del BackProp ▷ 34
- 4 Introducción a las redes profundas ▷ 49
- 5 PyTorch para redes neuronales ▷ 60
- 6 Perceptrón con varias capas en PyTorch ▷ 63
- 7 Redes convolucionales en PyTorch ▷ 65
- 8 Visión por computador ▷ 71
- 9 *Modelos secuenciales* ▷ 93
- 10 ¿Que sigue? ▷ 110
- 11 Notación ▷ 112

Secuencias de datos

- Reconocimiento del habla
- Clasificación de textos
- Traducción automática
- Tratamiento de vídeos
- Reconocimiento de entidades (NER)
- Reconocimiento de texto (HTR)

Redes recurrentes (RNN)

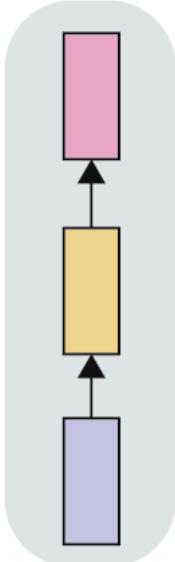
Recurrent Neural Networks



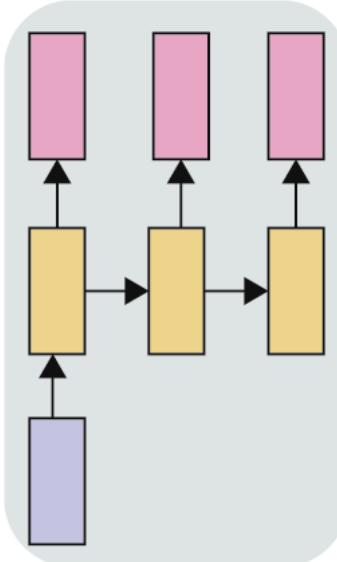
Tener un contexto de la información vista anteriormente
Se entranan con retropropagación del gradiente en el tiempo
(BPTT)

Tipos de RNN

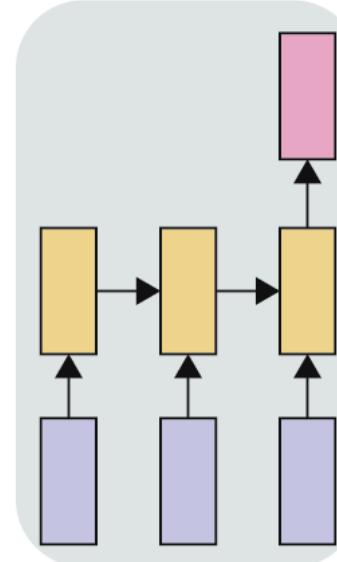
one to one



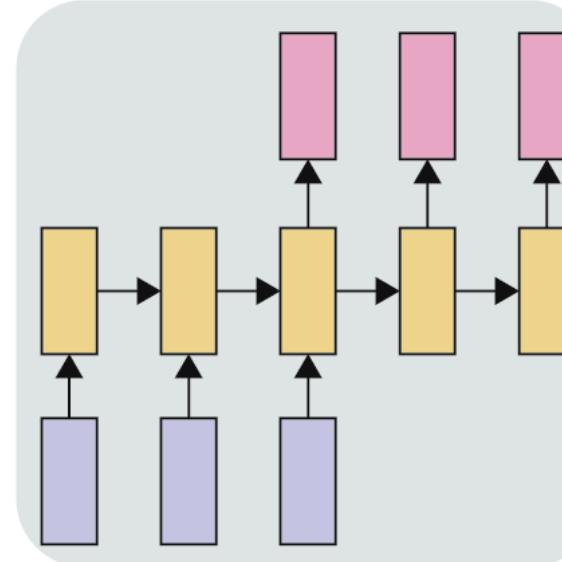
one to many



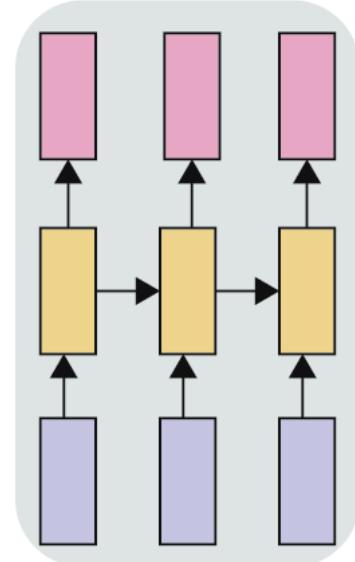
many to one



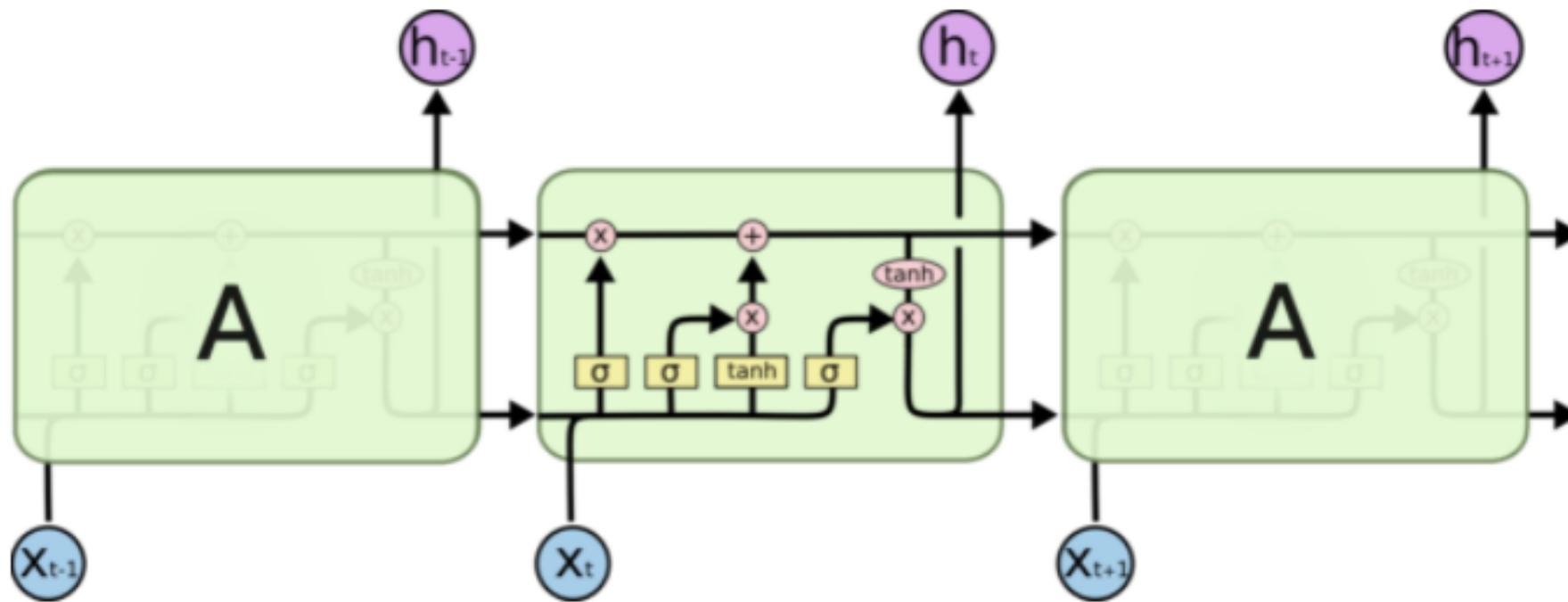
many to many



many to many



Long Short Term Memory - LSTM

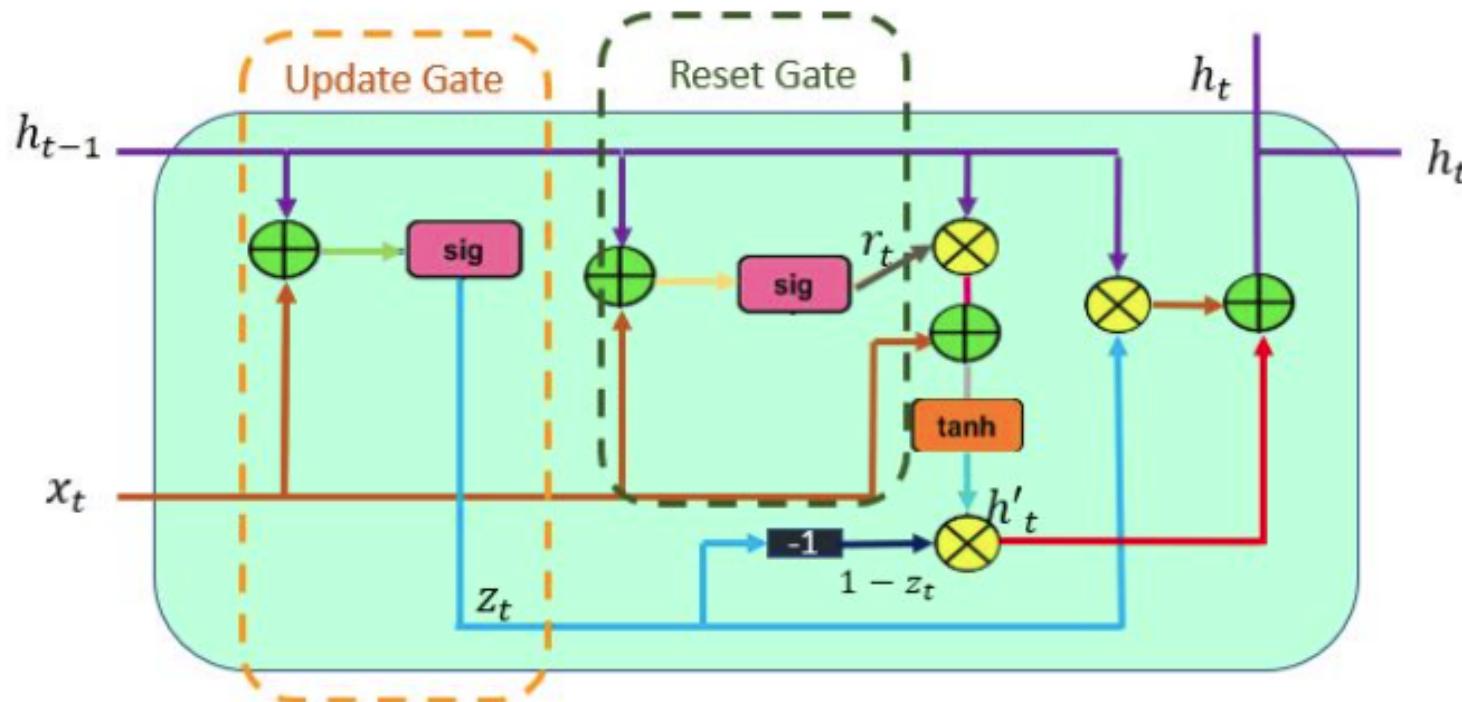


- Diseñadas específicamente para evitar el problema de la memoria a largo plazo.
- Toma tres entradas.
- La LSTM tiene unas "puertas", o *gates* en inglés, para decidir si la información es útil o ha de ser descartada, actuando como filtros.

LSTM - Gates

- **Input gate:** Decide la información que se guardará. Trabaja solo con la información entrante y la salida de la celda anterior.
- **Forget gate:** Decide que información se quedará o se tratará de olvidar. Esto se hace multiplicando por un vector generado a partir de la entrada actual y la salida de la celda anterior.
- **Output gate:** Usará la entrada actual, la salida de la celda anterior y un nuevo vector para calcular la salida de la celda y pasarla en el siguiente paso. Esta salida se puede entender como el *estado oculto* de la neurona.

Gated Recurrent Unit - GRU



- Incorpora dos puertas llamadas *Update Gate* y *Reset Gate*

GRU - Gates

- **Update gate:** Es responsable de determinar la cantidad de información previa que necesita para el siguiente estado. Puede llegar a decidir pasar toda la información, evitando así el problema desvanecimiento del gradiente.
- **Reset gate:** Se encarga de decidir si la información que proviene del estado anterior es importante o no.

LSTM vs GRU

- Las LSTM tienen tres "puertas", mientras que las GRU tienen dos.
- GRU no tiene ninguna memoria interna mientras que la LSTM tiene la *output gate*.
- GRU es más simple que LSTM por lo que necesitará menos tiempo y menos datos de entrenamiento.
- Las LSTM pueden llegar a recordar secuencias más largas que las GRU, por lo que son mejores para trabajar con relaciones a más distancia.

Notebook RNN

- RNN para trabajar con secuencias de caracteres para clasificar palabras: https://colab.research.google.com/drive/18zastAVMZ_6nIvMUK2tQa2D1h7GPOf6E?usp=sharing
- El mismo notebook pero utilizando LSTM: <https://colab.research.google.com/drive/1cWK7zmJUT0pHS17MUFB0zzs-p440udpo?usp=sharing>

Procesamiento del lenguaje natural

- El procesamiento del lenguaje natural (NLP, por sus siglas en inglés) aborda, de forma automática, el lenguaje natural como el texto o el habla.
- En el campo del NLP se encuentran muy diversas tareas, como pueden ser: reconocimiento del habla, segmentación de textos, etiquetado gramatical, reconocimiento de entidades nombradas (NER), clasificación de textos, etc.

Procesamiento del lenguaje natural - Representación del texto

La representación del texto es el primer problema que deberemos afrontar. Existen multitud de formas de representar el texto.

En otras ocasiones habréis utilizado un sistema de representación basado en bolsas de palabras (bag of words, BoW).

A continuación, veremos como podemos representarlo aprovechando su contexto local.

- ***One-hot***
- ***Word Embedding***

NLP - Codificación One-hot

- Tamaño máximo de vocabulario $|v|$.
- Se crea un vector de 0s de tamaño $|v|$ para cada palabra, poniendo un 1 en la posición de esa palabra respecto al vocabulario.
- *Las palabras parecidas no estarán cerca en ningún espacio vectorial. Es poco representativo del texto.*
- Usa mucha memoria.

$$V = \{caballo, casa, bicicleta, perro\}$$

$$v(caballo) = [1, 0, 0, 0]$$

$$v(casa) = [0, 1, 0, 0]$$

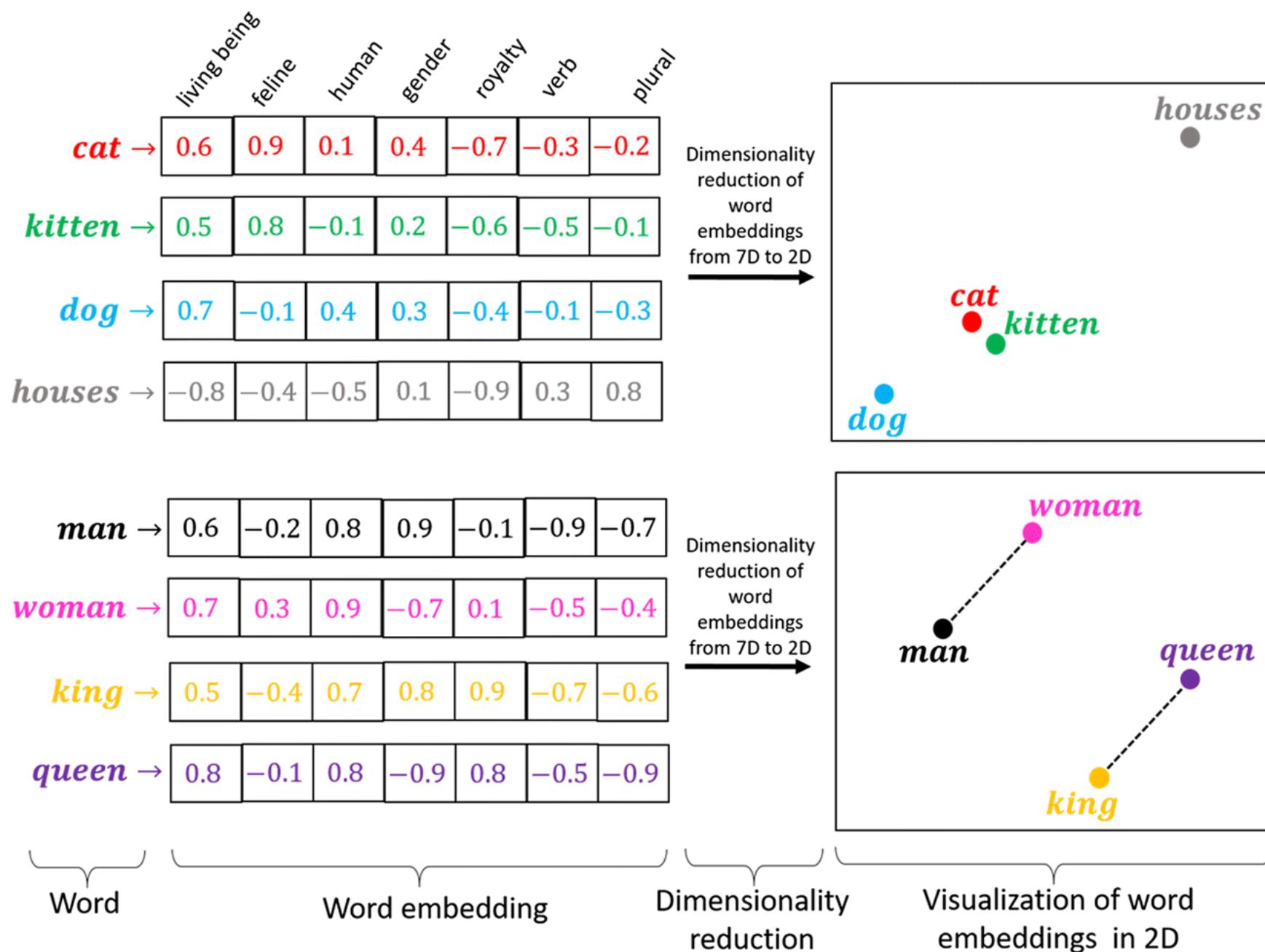
$$v(bicicleta) = [0, 0, 1, 0]$$

$$v(perro) = [0, 0, 0, 1]$$

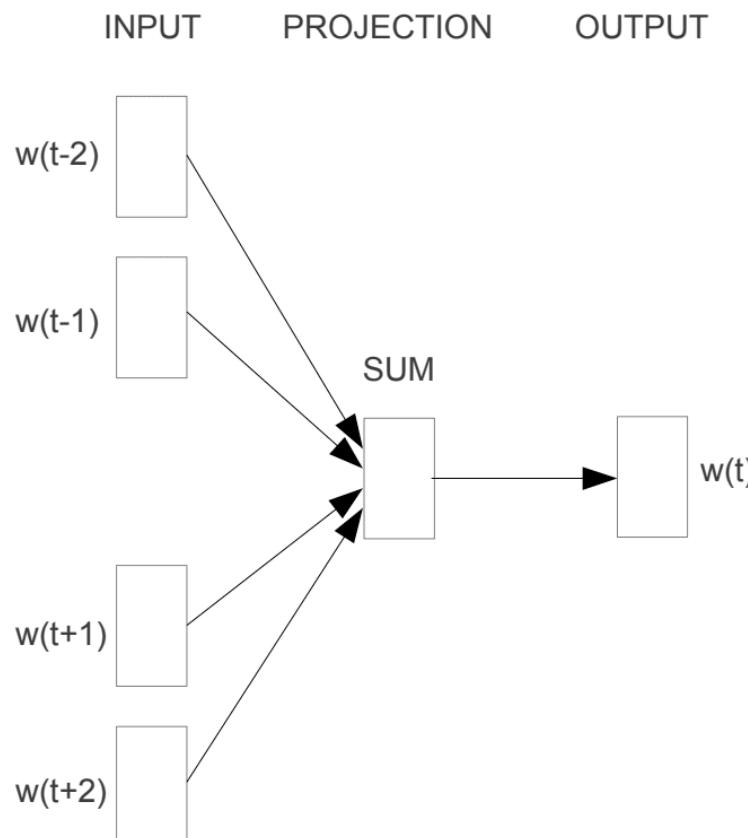
NLP - Word Embedding

- Tamaño máximo de vocabulario $|v|$ puede ser mucho mayor.
- Tamaño del vector de representación mucho menor respecto a *one-hot*.
- Las palabras con significado parecido deberían mantenerse cerca en alguna dimensión.

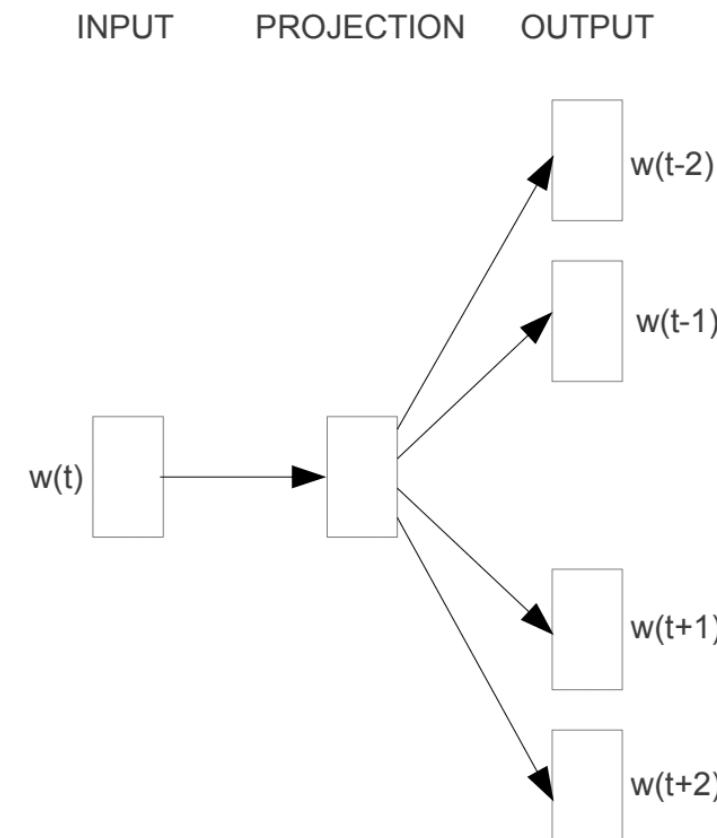
NLP - Word Embedding



Algoritmos Word Embedding - Word2Vec



CBOW



Skip-gram

Notebooks - Clasificación de textos

- Clasificación de textos con *torchtext* y *EmbeddingBag*
<https://colab.research.google.com/drive/1mxE6k1AO06HEmm6lSvYRCurgONWV5Zzw?usp=sharing>
- Clasificación de textos con *Word Embeddings* y *LSTM*:
<https://colab.research.google.com/drive/1tv1Nq6msObJx9F8E2OKgtIbvpEqyKIy4?usp=sharing>

Index

- 1 Redes neuronales multicapa ▷ 2
- 2 Algoritmo de retropropagación del error (BackProp) ▷ 19
- 3 Aspectos de uso y propiedades del BackProp ▷ 34
- 4 Introducción a las redes profundas ▷ 49
- 5 PyTorch para redes neuronales ▷ 60
- 6 Perceptrón con varias capas en PyTorch ▷ 63
- 7 Redes convolucionales en PyTorch ▷ 65
- 8 Visión por computador ▷ 71
- 9 Modelos secuenciales ▷ 93
- 10 *¿Que sigue?* ▷ 110
- 11 Notación ▷ 112

¿Qué sigue?

- Modelos de atención
- Modelos generativos (GAN) y autoencoders (VAE)
- Modelos basados en grafos
- Transformers
- y mucho más...

Index

- 1 Redes neuronales multicapa ▷ 2
- 2 Algoritmo de retropropagación del error (BackProp) ▷ 19
- 3 Aspectos de uso y propiedades del BackProp ▷ 34
- 4 Introducción a las redes profundas ▷ 49
- 5 PyTorch para redes neuronales ▷ 60
- 6 Perceptrón con varias capas en PyTorch ▷ 63
- 7 Redes convolucionales en PyTorch ▷ 65
- 8 Visión por computador ▷ 71
- 9 Modelos secuenciales ▷ 93
- 10 ¿Que sigue? ▷ 110
- 11 Notación ▷ 112

Notación

- **Funciones discriminantes lineales:** $\phi(\mathbf{x}; \Theta) = \Theta^t \mathbf{x}$ para una entrada \mathbf{x} y parámetros Θ compuestos por vector de pesos y umbral (θ, θ_0)
- **Funciones discriminantes lineales con activación:** $g \circ \phi(\mathbf{x}; \theta)$ para una entrada \mathbf{x} , parámetros (θ, θ_0) y g una función de activación. g' es la derivada de la función de activación g
- **Función de activación sigmoid:** $g_S(z)$
- **Salida del nodo i en la capa k :** s_i^k en perceptrones multicapa y redes hacia adelante
- **Pesos de la conexión** que va del nodo j de la capa $k - 1$ al nodo i de la capa k en un perceptrón multicapa: θ_{ij}^k . Θ es un vector de talla D formado por todos los pesos θ_{ij}^k . Pesos de la conexión que va del nodo j de la capa k' al nodo i de la capa k en una red hacia adelante: $\theta_{ij}^{k',k}$
- **Conjunto de N muestras de entrenamiento:** $S = \{(\mathbf{x}_1, \mathbf{t}_1), \dots, (\mathbf{x}_N, \mathbf{t}_N)\}$ con $\mathbf{x}_n \in \mathbb{R}^{M_0}$ y $\mathbf{t}_n \in \mathbb{R}^{M_K}$, siendo K el número de capas y M_k el número de nodos de la capa k
- **Función a minimizar** en el entrenamiento de un perceptrón multicapa: $q_S(\Theta) \in \mathbb{R}$
- **Clasificador** en $C \equiv M_K$ clases de puntos de $\mathbb{R}^d \equiv \mathbb{R}^{M_0}$: $f : \mathbb{R}^{M_0} \rightarrow \{1, \dots, M_K\}$
- **Error en el nodo i de la capa k para la muestra \mathbf{x}_n :** $\delta_i^k(\mathbf{x}_n)$
- **Incremento del peso** que va del nodo j en la capa $k - 1$ al nodo i en la capa k : $\Delta\theta_{ij}^k$
- **Factor de aprendizaje, momentum y factor de regularización:** ρ , ν y λ
- **Media y desviación típica:** μ y σ