

APRENDIZAJE AUTOMATICO PROFUNDO (DEEP LEARNING)



Inteligencia Artificial

- La **Inteligencia Artificial (IA)** es la inteligencia llevada a cabo por máquinas.

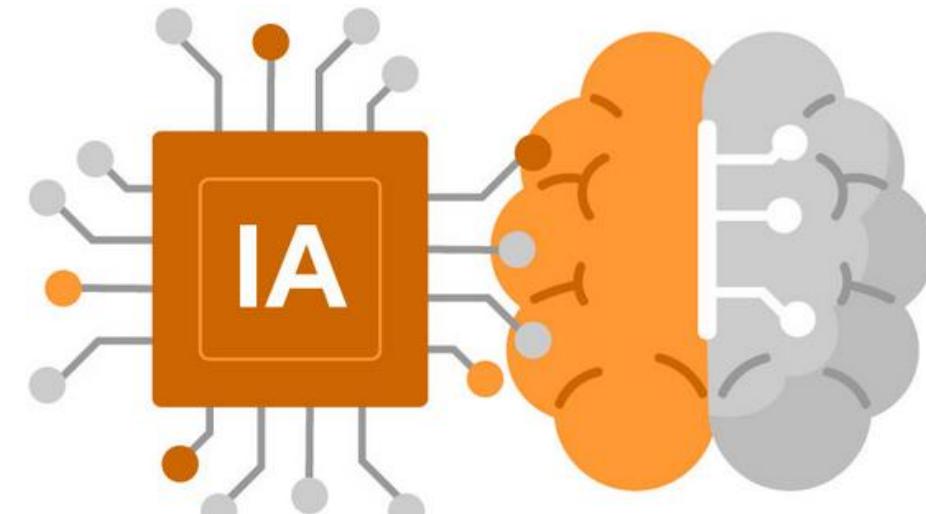
- **RAMAS**

- **DEDUCTIVA** (lógica)

- Sistemas expertos

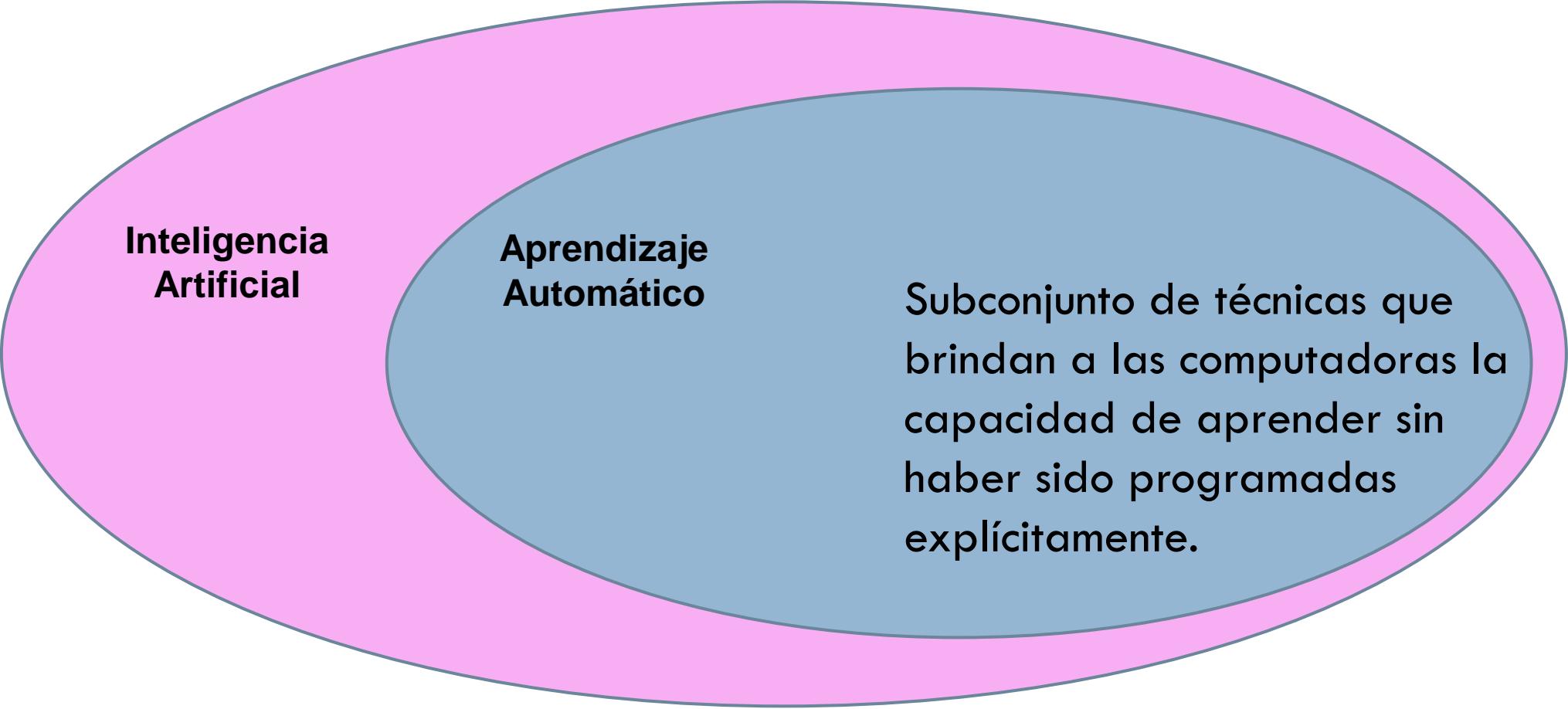
- **INDUCTIVA** (ejemplos)

- Redes Neuronales
 - Técnicas de Optimización



El aprendizaje automático
pertenece a esta rama

IA y Aprendizaje Automático



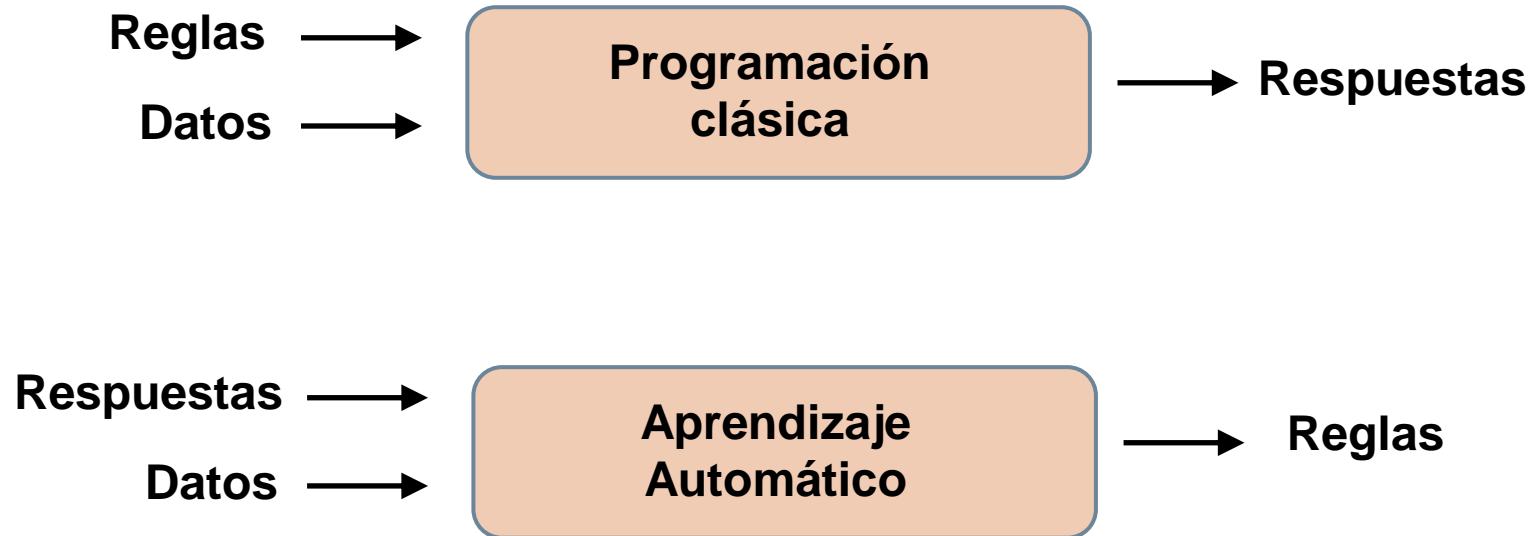
Aprendizaje Automático

Inteligencia
Artificial

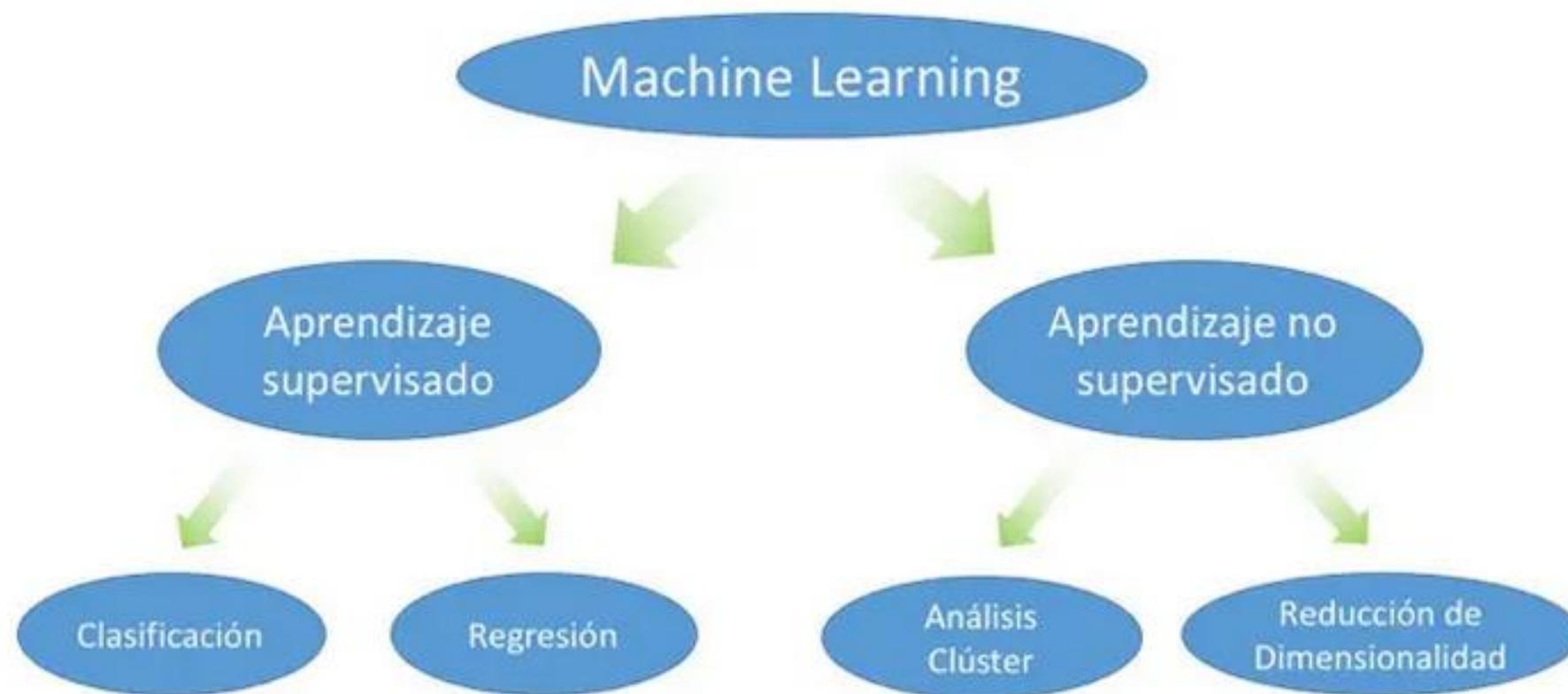
Aprendizaje
Automático

Subconjunto de técnicas que
brindan a las computadoras la
capacidad de aprender sin
haber sido programadas
explícitamente.

Programación clásica y Aprendizaje Automático



Tipos de aprendizaje



Aprendizaje no supervisado



AGRUPAMIENTO

Aprendizaje supervisado

GATO



GATO



GATO



ARBOL



ARBOL



CUADERNO



CUADERNO



CUADERNO



En este curso trabajaremos con
APRENDIZAJE SUPERVISADO

GATO

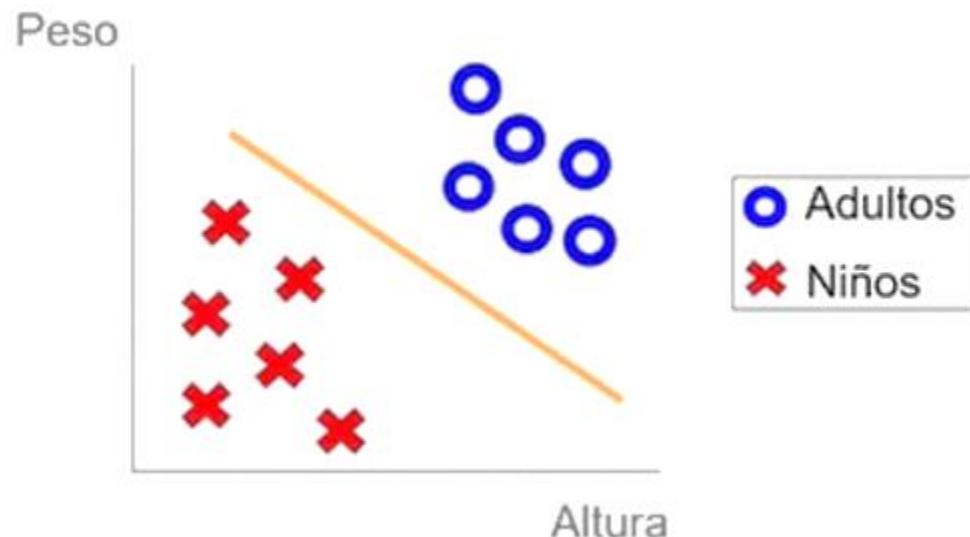


?

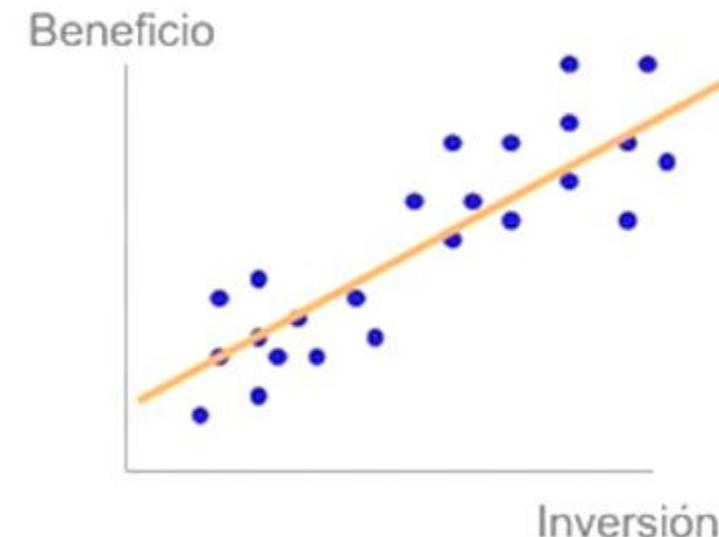
Aprendizaje Supervisado

- Según si la respuesta a predecir es **discreta** o **continua** se trata de un problema de **clasificación** o de **regresión** respectivamente.

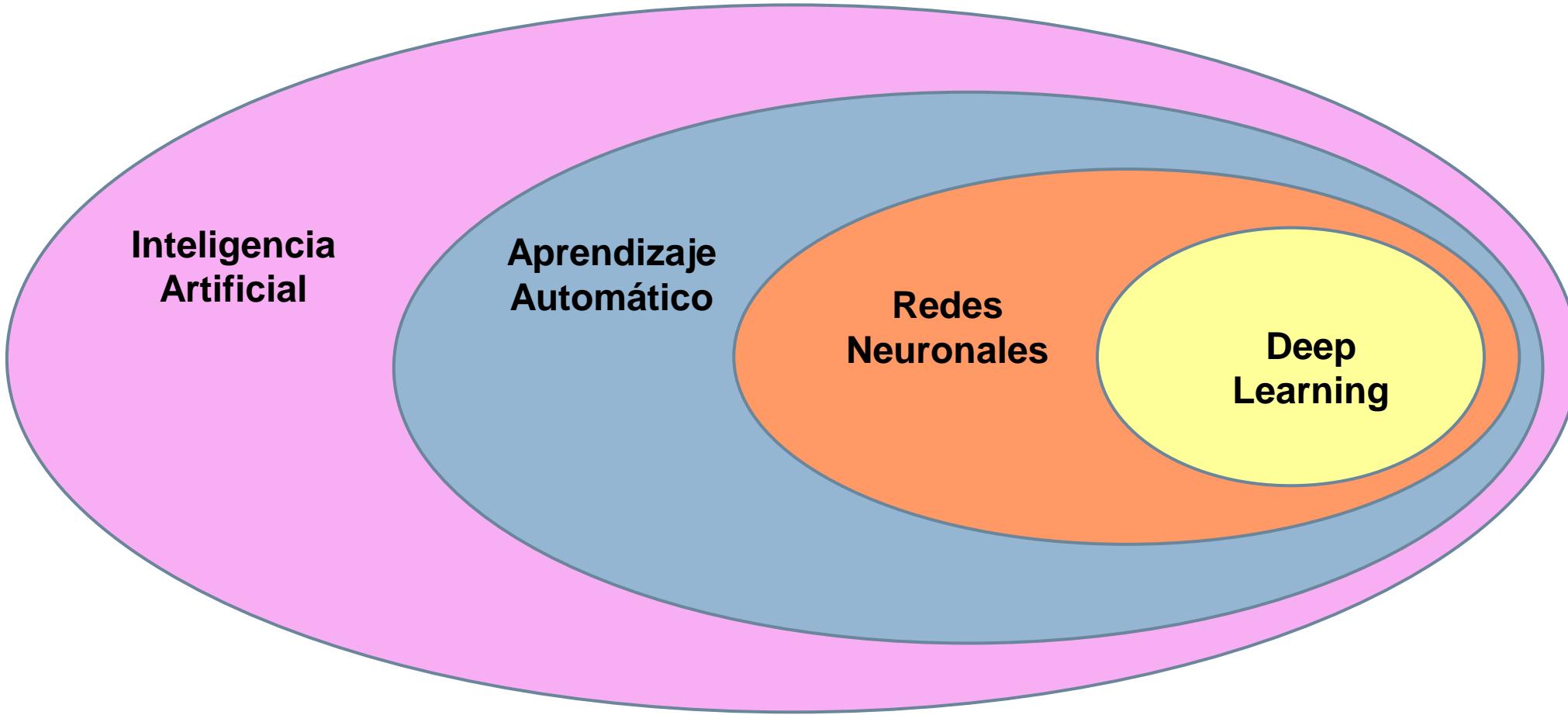
CLASIFICACION



REGRESION



Redes Neuronales y Aprendizaje Profundo



The diagram consists of four concentric ellipses. The innermost ellipse is yellow and contains the text "Deep Learning". The second ellipse out is orange and contains the text "Redes Neuronales". The third ellipse out is blue and contains the text "Aprendizaje Automático". The outermost ellipse is pink and contains the text "Inteligencia Artificial".

Inteligencia
Artificial

Aprendizaje
Automático

Redes
Neuronales

Deep
Learning

Tareas que pueden resolverse con RN

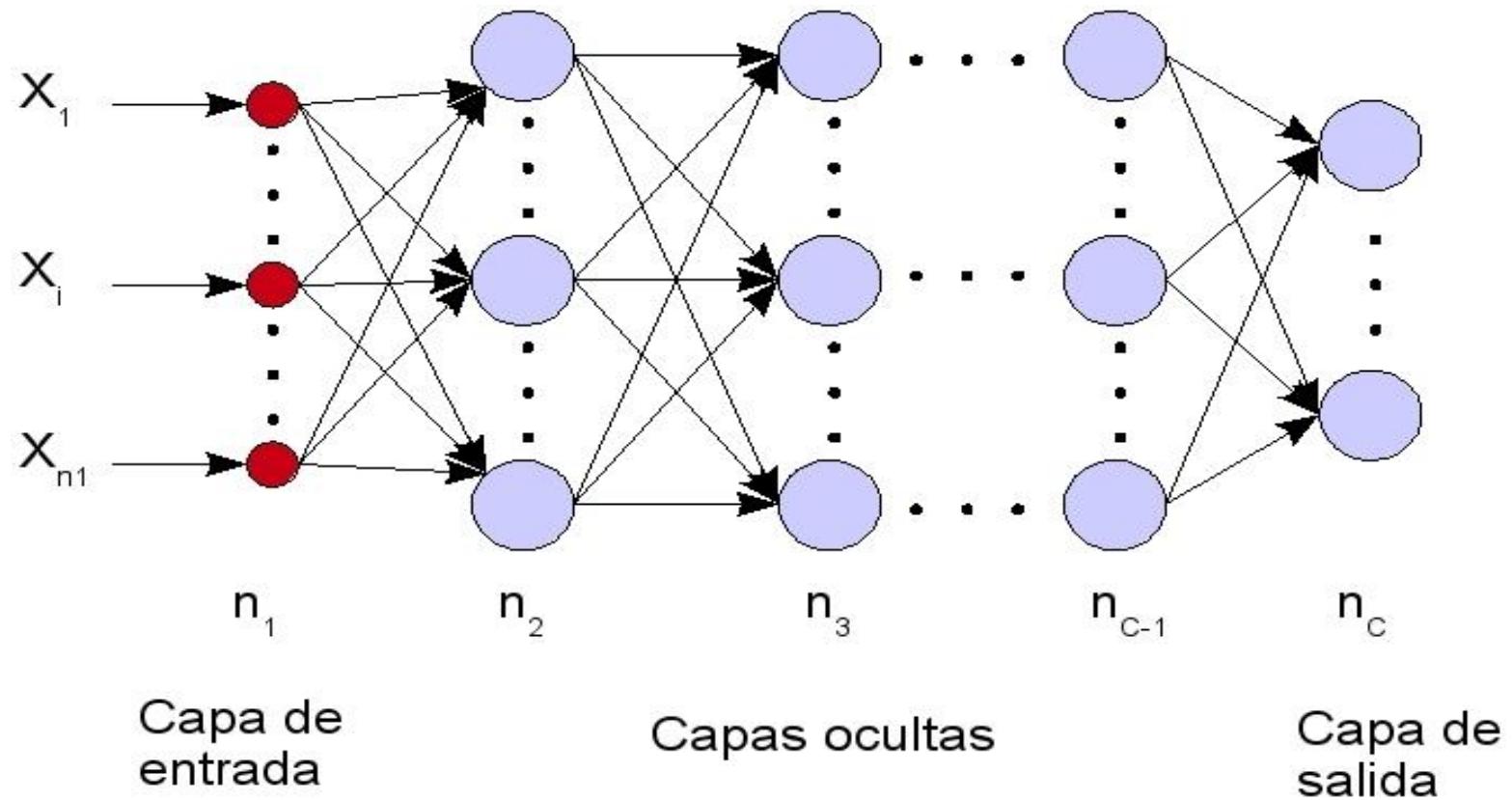
SUPERVISADO

NO SUPERVISADO

- **Predicción** de un resultado futuro a partir de los datos disponibles.
 - Predecir el nivel de seguridad de un vehículo dadas sus características.
 - Determinar si un mail recibido es spam o no.
 - Dada la historia clínica de un paciente, predecir la probabilidad de contraer cierta enfermedad.
- **Segmentación** de los datos en subgrupos con características similares
 - Agrupar clientes para determinar perfiles que ayuden a direccionar campañas de marketing.
 - Caracterizar transacciones comerciales y detectar situaciones anómalas.

Red Neuronal Artificial

25



Análisis de los datos disponibles

□ Tipos de Variables



- Cuantitativas y cualitativas

□ Descripciones estadísticas

- Medidas de tendencia central
- Medidas de dispersión

□ Gráficos

- Diagrama de barras
- Diagrama de torta
- Histograma
- Diagrama de caja
- Diagrama de dispersión

Tipos de variables

□ **Cuantitativas o numéricas**

- DISCRETAS (cant. de empleados, cant. de alumnos, etc)
- CONTINUAS (sueldo, metros cuadrados, beneficios, etc)

□ **Cualitativas o categóricas**

- NOMINALES: nombran al objeto al que se refieren sin poder establecer un orden (estado civil, raza, idioma, etc.)
- ORDINALES: se puede establecer un orden entre sus valores (alto, medio, bajo, etc)

Descripciones estadísticas básicas

- Identifican propiedades de los datos y destacan qué valores deben tratarse como ruido o valores atípicos

MEDIDAS DE TENDENCIA CENTRAL

- Media
- Mediana
- Moda
- Rango medio

MEDIDAS DE DISPERSION

- Varianza
- Desviación estndar
- Rango
- Cuartiles
- Rango Intercuartil

Descripciones estadísticas básicas

- Identifican propiedades de los datos y destacan qué valores deben tratarse como ruido o valores atípicos

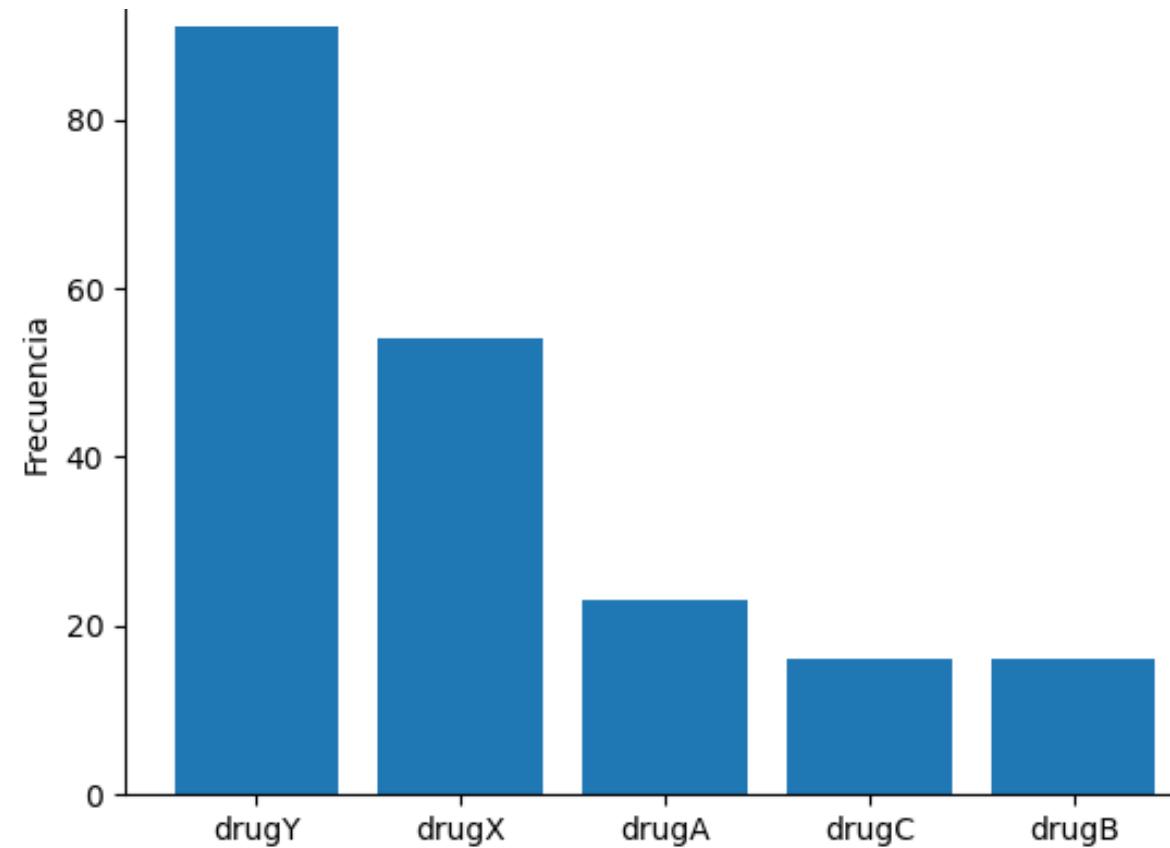
MEDIDAS DE TENDENCIA CENTRAL

- Media
- Mediana
- Moda
- Rango medio

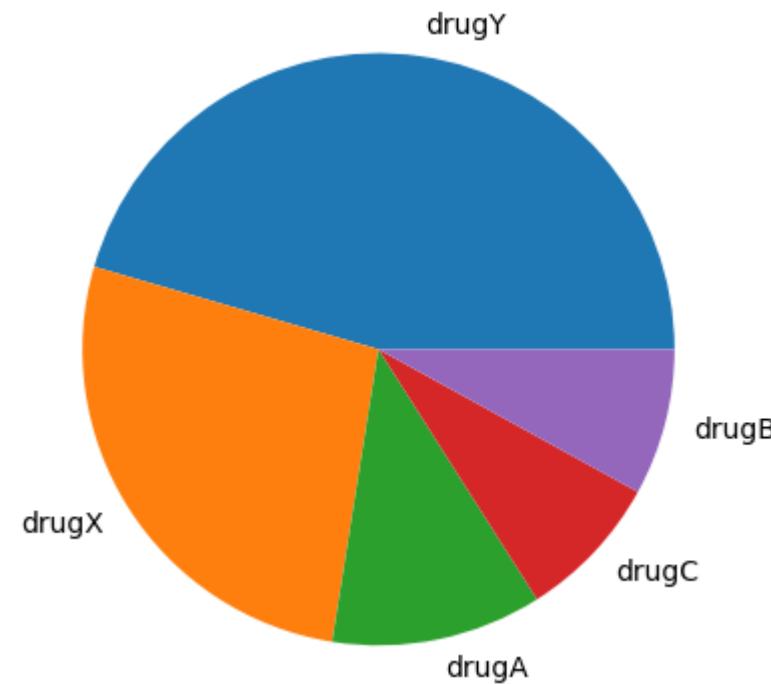
MEDIDAS DE DISPERSION

- Varianza
- Desviación estndar
- Rango
- Cuartiles
- Rango Intercuartil

Atributo Drug - Diagrama de barras

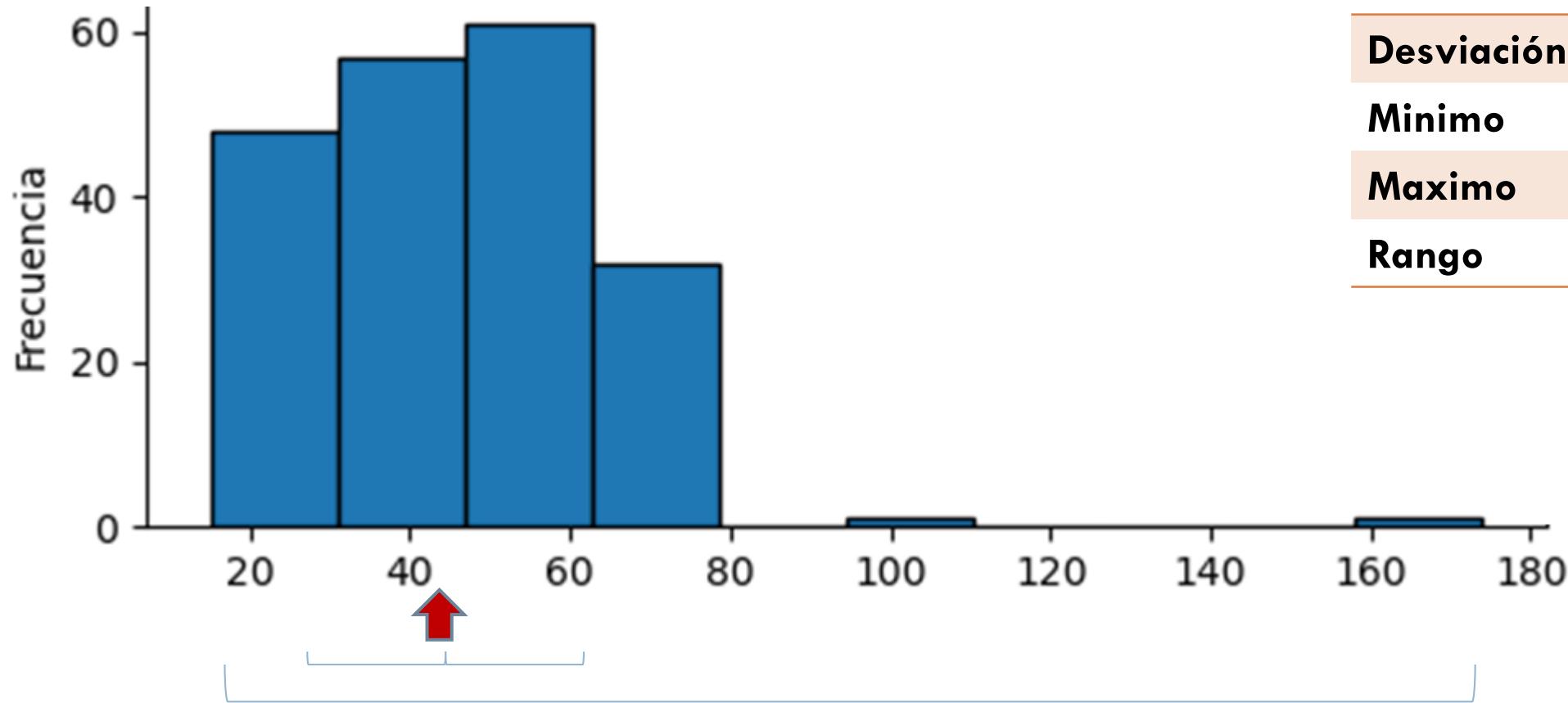


Atributo Drug - Gráfico de Torta



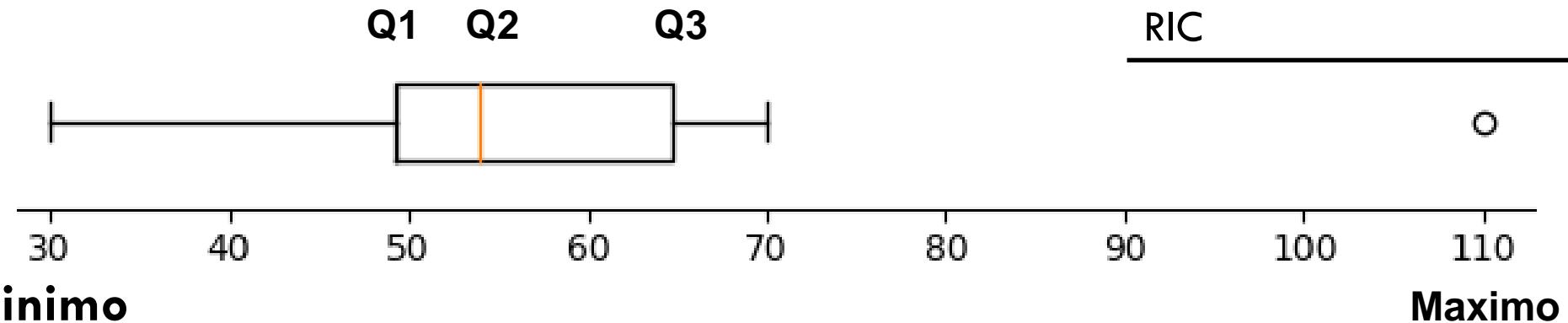
Atributo AGE – Histograma

(Atributo AGE del archivo Drug5_atipicos.CSV)



Media	44.965
Desviación	19.145
Mínimo	15
Máximo	174
Rango	159

Diagrama de caja - Ejemplo



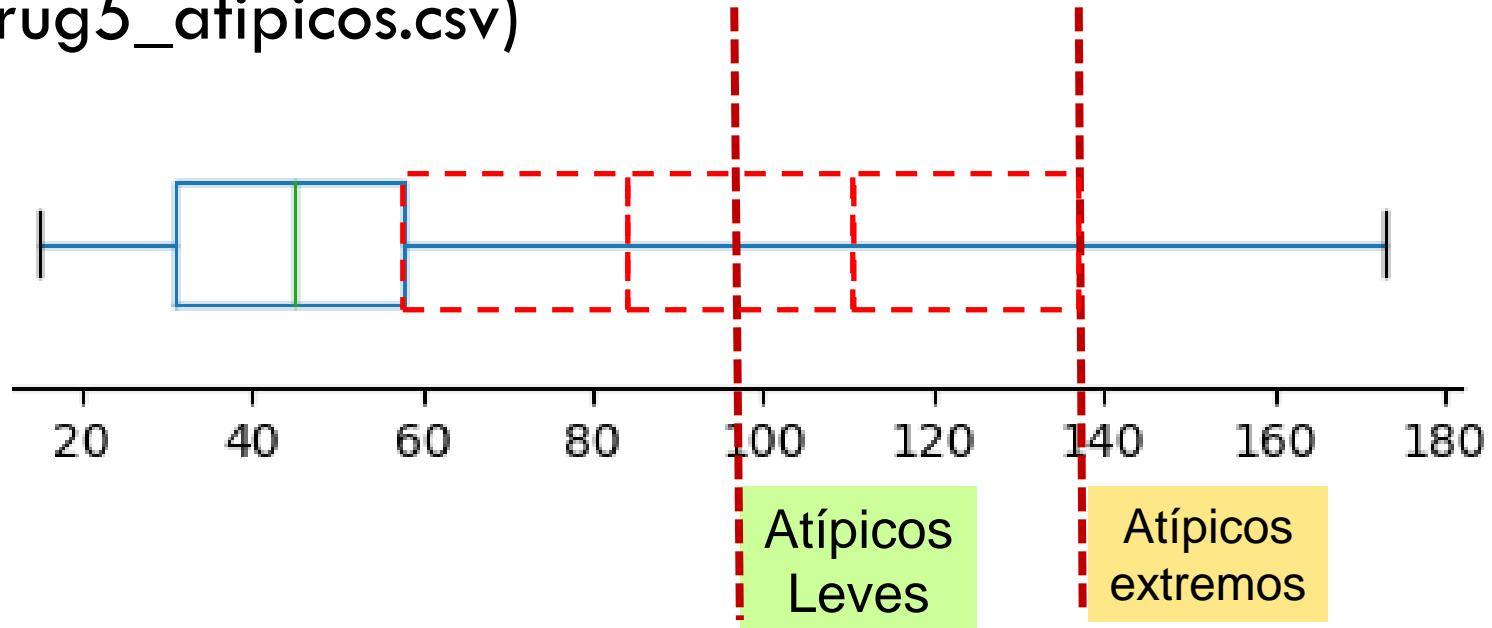
- Se consideran **valores atípicos leves** a los que se encuentran a $1.5 * \text{RIC}$ más allá de los límites de la caja y **atípicos extremos** a los que están más allá de $3 * \text{RIC}$.

Determine si hay valores atípicos y si son leves o extremos

Diagrama de caja (en construcción)

□ Atributo AGE (archivo Drug5_atipicos.csv)

Minimo	15
Q1	31
Q2	45
Q3	58
Maximo	174

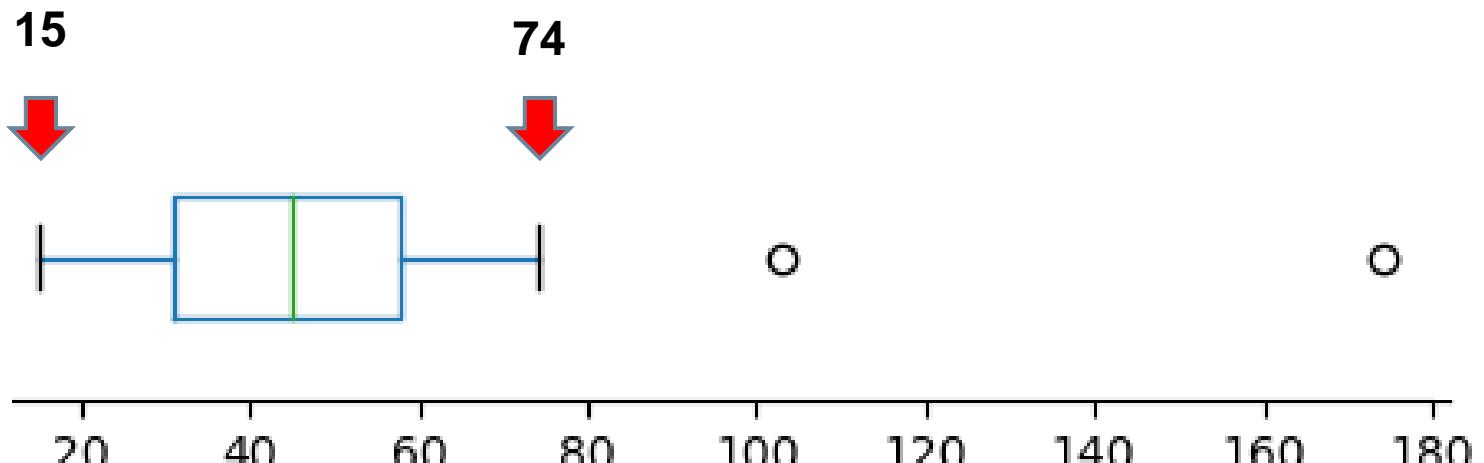


RIC	$Q3 - Q1 = 58 - 31 = 27$
Lim.Inf	$Q1 - 1.5 * RIC = 31 - 1.5 * 27 = -9.5$
Lim.Sup	$Q3 + 1.5 * RIC = 58 + 1.5 * 27 = 98.5$

Diagrama de caja

□ Atributo AGE

Minimo	15
Q1	31
Q2	45
Q3	58
Maximo	174



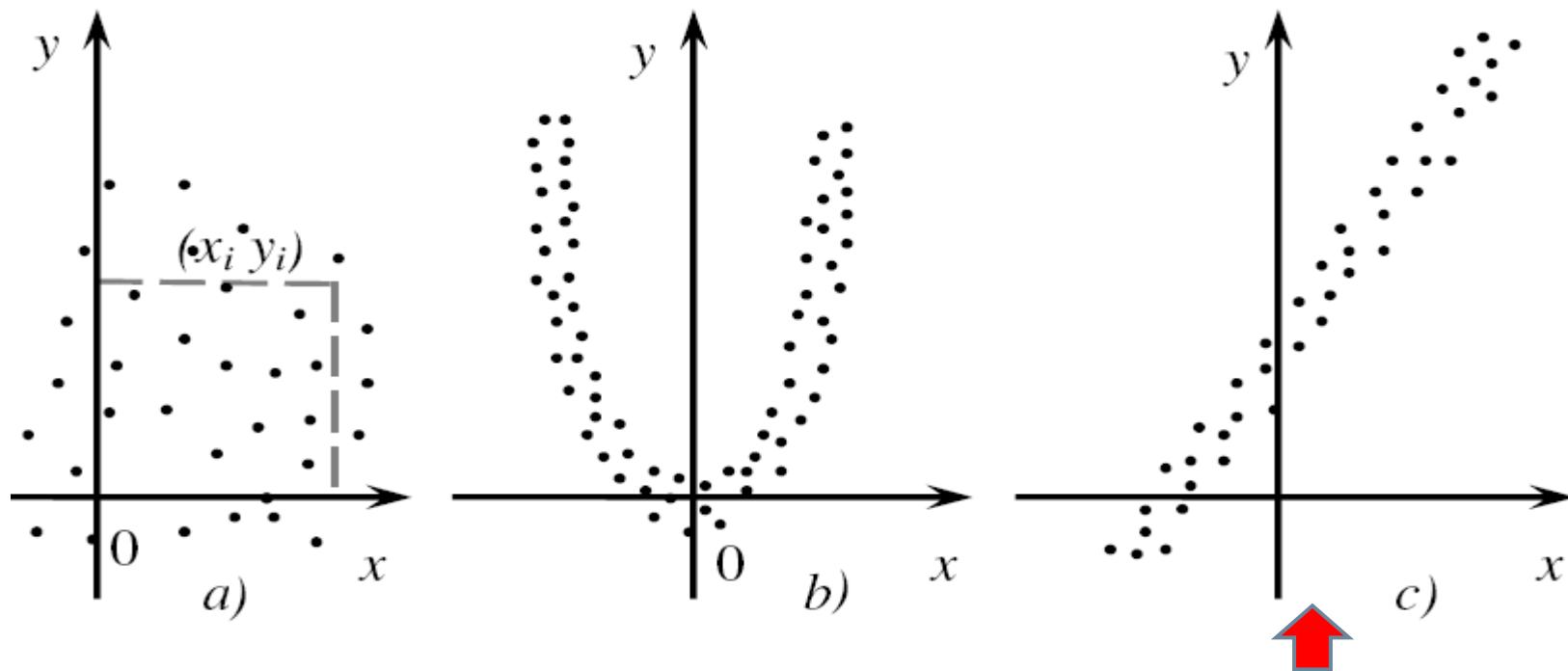
RIC	$Q3 - Q1 = 27$
Lim.Inf	$Q1 - 1.5 * RIC = -9.5$
Lim.Sup	$Q3 + 1.5 * RIC = 98.5$

Los bigotes indican el rango de los valores de la muestra comprendidos en el intervalo

$$[Q1 - 1.5 * RIC ; Q3 + 1.5 * RIC] = [-9.5, 98.5]$$

Diagrama de Dispersion

- Consiste en dibujar pares de valores (x_i, y_i) medidos de la v.a. (X,Y) en un sistema de coordenadas



Entre X e Y existe una **relación lineal**. Este es el tipo de relación que nos interesa

Relación entre atributos numéricicos

- Al momento de construir un modelo resulta de interés saber si dos atributos numéricicos se encuentran linealmente relacionados o no. Para ello se usa el **coeficiente de correlación lineal**.

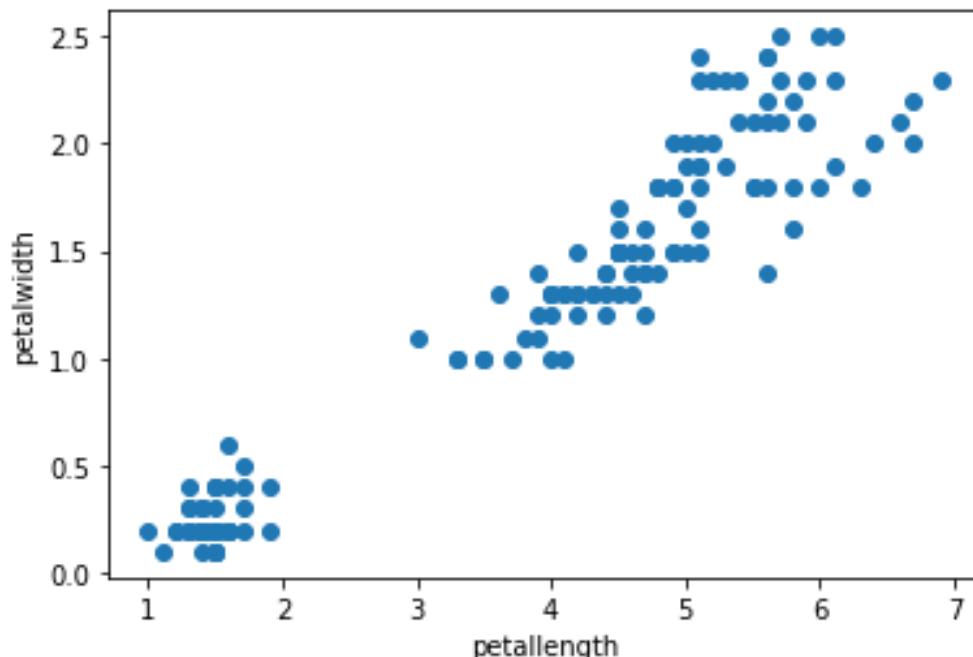


Diagrama de dispersión entre la longitud y el ancho del pétalo de una flor.

Coeficiente de correlación lineal

INTERPRETACION

- Si $0.5 \leq \text{abs}(\text{Corr}(A,B)) < 0.8$ se dice que A y B tienen una correlación lineal débil.
- Si $\text{abs}(\text{Corr}(A,B)) \geq 0.8$ se dice que A y B tienen una correlación lineal fuerte
- Si $\text{abs}(\text{Corr}(A,B)) < 0.5$ se dice que A y B no están correlacionados linealmente. Esto NO implica que son independientes, sólo que entre ambos no hay una correlación lineal.

MEDIA

- La **MEDIA** es el promedio de los valores del atributo. Dicho atributo debe ser numérico.

$$\bar{X} = \frac{\sum_{i=1}^N x_i}{N}$$

N es la cantidad de valores a promediar

- Ejemplo

30 36 47 50 52 52 56 60 63 70 70 110

$$\bar{X} = \frac{30 + 36 + 47 + 50 + 52 + 52 + 56 + 60 + 63 + 70 + 70 + 110}{12} = \frac{696}{12} = 58$$

MEDIANA

- Divide a los valores del atributo en dos partes iguales de manera que los anteriores son todos menores que él y los siguientes son mayores.
- Antes de calcularla deben **ordenarse los valores** del atributo.
- Ejemplo: atributo numérico con una **cantidad impar** de valores



$$\tilde{X} = x_{(N+1)/2} = 56$$

MEDIANA

- Divide a los valores del atributo en dos partes iguales de manera que los anteriores son todos menores que él y los siguientes son mayores.
- Antes de calcularla deben **ordenarse los valores** del atributo.
- Ejemplo: atributo numérico con una **cantidad par** de valores

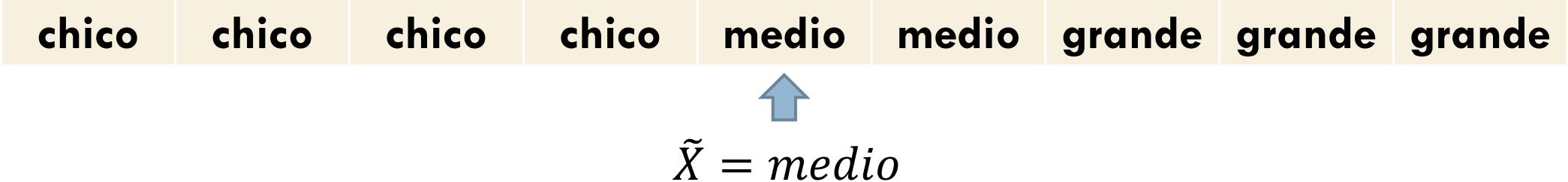
30 36 47 50 52 52 56 60 63 70 70 110



$$\tilde{X} = \frac{x_{N/2} + x_{(N+1)/2}}{2} = \frac{52 + 56}{2} = 54$$

MEDIANA

- También puede calcularse sobre **atributos ordinales**. En tal caso, el resultado será o bien el valor que divide al conjunto en dos partes iguales o bien se dirá que “la mediana está entre los valores ...”.
- Antes de calcularla deben **ordenarse los valores** del atributo.
- Ejemplo: atributo ordinal con una **cantidad impar** de valores



MODA

- La moda es el valor que aparece con mayor frecuencia. Por lo tanto, puede determinarse para atributos cualitativos y cuantitativos.
- Es posible que la mayor frecuencia corresponda a varios valores diferentes, lo que da lugar a más de una MODA.
- Los conjuntos de datos con uno, dos o tres modas se denominan unimodal, bimodal y trimodal, respectivamente.
- En general, un conjunto de datos con dos o más modas es multimodal.
- Si cada valor de los datos ocurre sólo una vez, entonces no hay moda.

MODA

- La moda es el valor que aparece con mayor frecuencia. Por lo tanto, puede determinarse para atributos cualitativos y cuantitativos.
- Ejemplo: atributo numérico

30 36 47 50 52 52 56 60 63 70 70 110

- Hay 2 modas y sus valores son 52 y 70
- Ejemplo: atributo nominal

español inglés chino inglés chino chino

- La moda es “chino” por ser el valor que aparece más veces

RANGO MEDIO

- El rango medio es fácil de calcular y también puede utilizarse para evaluar la tendencia central de un conjunto de datos numéricos.
- Es la media de los valores máximo y mínimo del conjunto.
- Ejemplo

30 36 47 50 52 52 56 60 63 70 70 110

$$\text{rango medio} = \frac{\text{maximo} + \text{minimo}}{2} = \frac{110 + 30}{2} = \frac{140}{2} = 70$$

VARIANZA Y DESVIACION ESTANDARD

- La varianza mide la dispersión de los datos con respecto a la media.
- Valores bajos indican que las observaciones de los datos tienden a estar muy cerca de la media, mientras que valores altos indican que los datos están muy dispersos.

$$\sigma^2 = \frac{1}{N} \sum_{i=1}^N (x_i - \bar{x})^2 = \left(\frac{1}{N} \sum_{i=1}^N x_i^2 \right) - \bar{x}^2$$

- La desviación estándar σ es la raíz cuadrada de la varianza

VARIANZA Y DESVIACION ESTANDARD

□ Ejemplo

30 36 47 50 52 52 56 60 63 70 70 110

VARIANZA POBLACIONAL

$$\sigma^2 = \left(\frac{1}{N} \sum_{i=1}^N x_i^2 \right) - \bar{x}^2 = \frac{1}{12} (30^2 + 36^2 + \dots + 110^2) - 58^2 \approx 379.17$$

$$\sigma \approx \sqrt{379.17} \approx 19.47$$

VARIANZA Y DESVIACION MUESTRAL

□ Ejemplo

30 36 47 50 52 52 56 60 63 70 70 110

VARIANZA MUESTRAL

$$S^2 = \left(\frac{1}{N-1} \sum_{i=1}^N x_i^2 \right) - \bar{x}^2 = \frac{1}{11} (30^2 + 36^2 + \dots + 110^2) - 58^2 \approx 413.64$$

$$S \approx \sqrt{413.64} \approx 20.34$$

RANGO

- El rango de un conjunto de valores numéricos es la diferencia entre los valores máximo y mínimo de dicho conjunto.
- Ejemplo

30	36	47	50	52	52	56	60	63	70	70	110
----	----	----	----	----	----	----	----	----	----	----	-----

$$\text{rango} = \text{maximo} - \text{minimo} = 110 - 30 = 80$$

Cuantiles, Cuartiles y Percentiles

- Los cuantiles son valores que dividen un conjunto numérico ordenado en partes iguales. Es decir que determinan intervalos que comprenden el mismo número de valores.
- Los cuantiles más usados son los siguientes:
 - CUARTILES: dividen la distribución en cuatro partes.
 - DECILES: dividen la distribución en diez partes.
 - Centiles o PERCENTILES: dividen la distribución en cien partes.
 - *El percentil es una medida de posición usada en estadística que indica, una vez ordenados los datos de menor a mayor, el valor de la variable por debajo del cual se encuentra un porcentaje dado de observaciones en un grupo.*

CUARTILES

- Los cuartiles suelen representarse como Q1, Q2 y Q3.
- El 2do. cuartil o Q2 coincide con la MEDIANA.
- Para hallar las posiciones de Q1 y Q3 usaremos $(N+1)/4$ y $3(N+1)/4$ respectivamente, siendo N la cantidad de valores disponibles.
 - ▣ Si no hay parte decimal, se toma directamente el elemento.
 - ▣ Si la posición corresponde a un número con parte decimal entre el elemento i y el $i+1$, se determinar un factor realizando una **interpolación lineal**.

El cuartil será:

$$Q = x_i + (x_{i+1} - x_i) * factor$$

CUARTILES

- Ejemplo:

30	36	47	50	52	52	56	60	63	70	70	110
----	----	----	----	----	----	----	----	----	----	----	-----

- La ubicación de Q1 es $(N+1)/4$, es decir, $(12+1)/4=13/4=3.25$
- Como no es un número entero calculamos su valor entre el 3ro y el 4to elemento.

$$Q_1 = x_3 + (x_4 - x_3) * \underbrace{factor}$$



CUARTILES – cálculo del factor

Ubicación de Q1

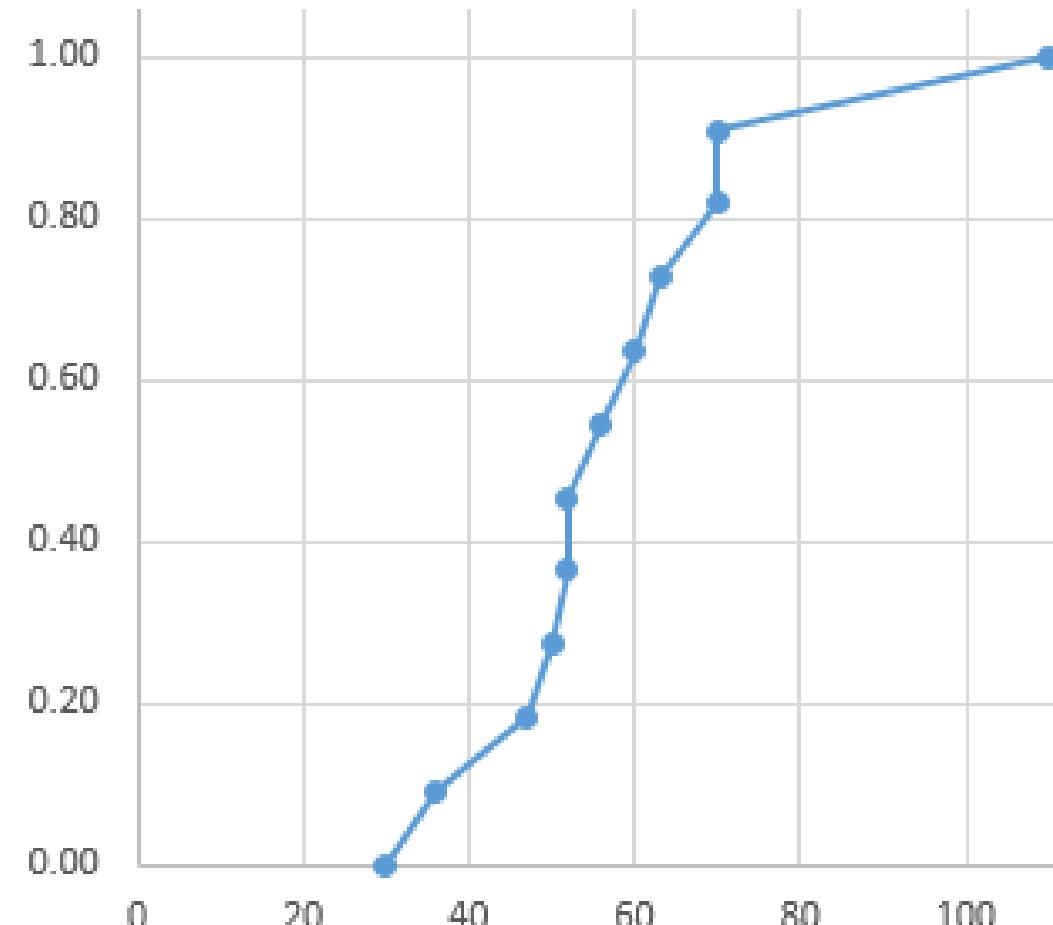
$$(N + 1)/4 = 13/4 = 3.25$$

Q1 

X	F_i
30	0.00
36	0.09
47	0.18
50	0.27
52	0.36
52	0.45
56	0.55
60	0.64
63	0.73
70	0.82
70	0.91
110	1.00

$$N = 12$$

$$F_i = \frac{i - 1}{N - 1}$$

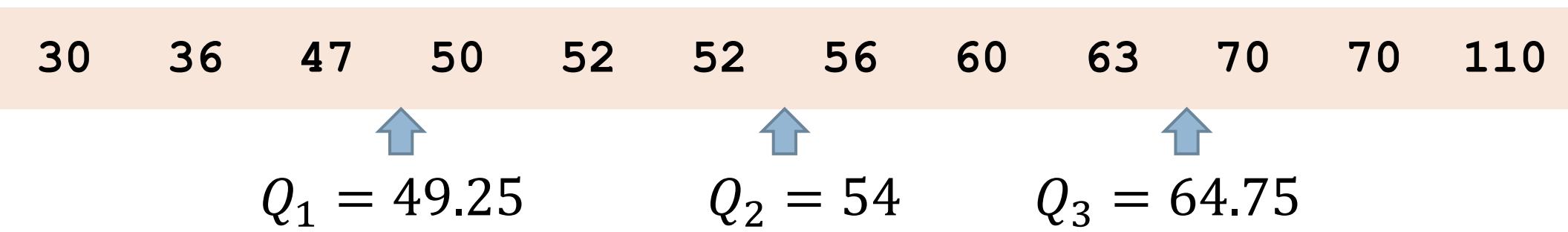


RANGO INTERCUARTIL

- La distancia entre Q_1 y Q_3 es una medida sencilla de dispersión que da el rango cubierto por la mitad de los datos.
- Esta distancia se denomina **rango intercuartil (RIC)** y se define como

$$RIC = Q_3 - Q_1$$

- Ejemplo:



$$RIC = Q_3 - Q_1 = 64.75 - 49.25 = 15.50$$

Valor atípico o fuera de rango

- Los valores de la muestra que pertenezcan a alguno de estos intervalos

$$[Q1 - 3 \cdot RIC ; Q1 - 1.5 \cdot RIC] \text{ o } [Q3 + 1.5 \cdot RIC ; Q3 + 3 \cdot RIC]$$

serán considerados **valores fuera de rango leves**.

- Los valores de la muestra inferiores a $Q1 - 3 \cdot RIC$ o superiores a $Q3 + 3 \cdot RIC$ serán considerados **valores fuera de rango extremos**.

Coeficiente de correlación lineal

- Dados dos atributos X e Y el coeficiente de correlación lineal entre ellos se calcula de la siguiente forma

$$\text{Corr}(X, Y) = \frac{\text{Cov}(X, Y)}{\sigma_X \sigma_Y}$$

siendo $\text{Cov}(X, Y)$ la covarianza entre X e Y y σ_X y σ_Y los desvíos de cada variable.

Covarianza y desvío estándar

- Dadas dos variables X y Y

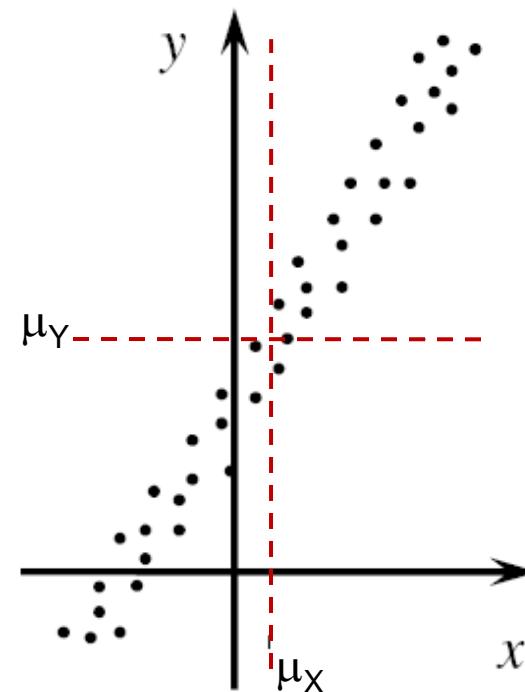
$$Cov(X, Y) = \left[\sum_{i=1}^N (x_i - \mu_X)(y_i - \mu_Y) \right] / N$$

$$\sigma_X = \sqrt{\left[\sum_{I=1}^N (x_i - \mu_X)^2 \right] / N}$$

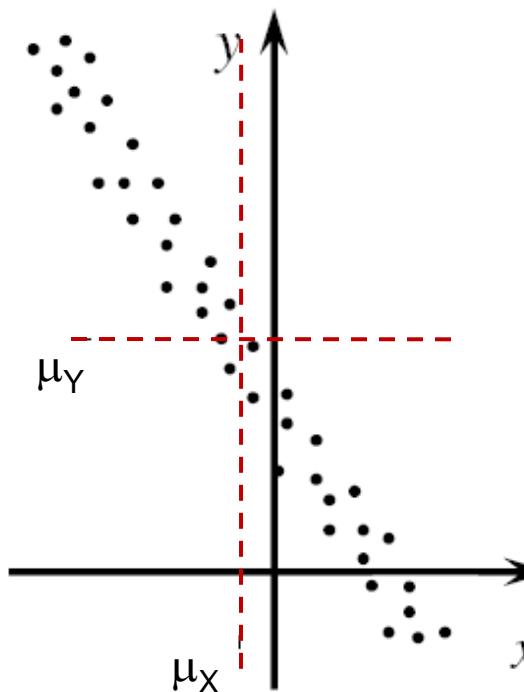
Covarianza

$$Cov(X, Y) = \left[\sum_{i=1}^N (x_i - \mu_X)(y_i - \mu_Y) \right] / N$$

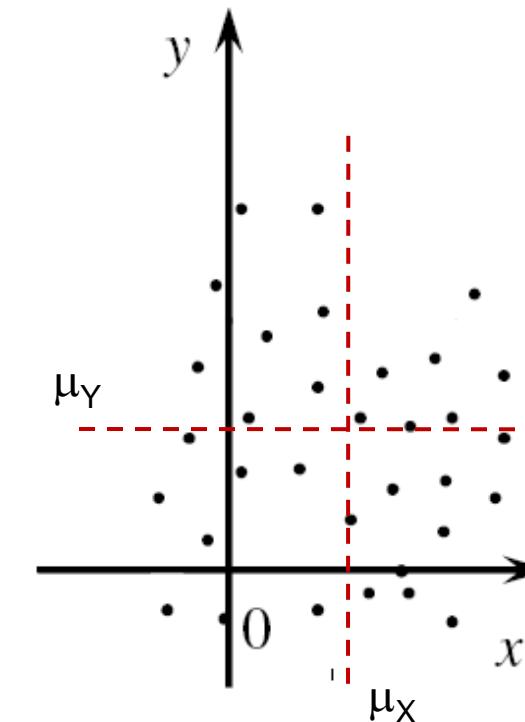
- La **covarianza** es un valor que indica el grado de variación conjunta de dos **variables aleatorias** respecto a sus medias.



Covarianza Positiva



Covarianza Negativa



Covarianza cercana a cero

Fase de Preparación de los Datos

- La información almacenada siempre tiene
 - Datos faltantes ←
 - Valores extremos
 - Inconsistencias
 - Ruido
- Tareas a realizar
 - Limpieza (ej: resolver outliers e inconsistencias)
 - Transformación (ej: numerización)

Limpieza - Valores faltantes

- Qué hacer con los valores faltantes?
 - Ignorar la tupla.
 - Rellenar la tupla manualmente.
 - Usar una constante global para llenar el valor faltante.
 - Utilizar el valor de la media u otra medida de centralidad para llenar el valor.
 - Utilizar el valor de la media u otra medida de centralidad de los objetos que pertenecen la misma clase.
 - Utilizar alguna técnica de Aprendizaje Automático para calcular el valor más probable.

Transformación de atributos

□ DISCRETIZACION

- Algunos algoritmos de minería de datos sólo operan con atributos cualitativos. La discretización convierte los atributos numéricos en ordinales.

□ NUMERIZACION

- Es el proceso contrario a la discretización. Convierte atributos cualitativos en numéricos.

□ NORMALIZACION

- Permite expresar los valores de los atributos sin utilizar las unidades de medida originales facilitando su comparación y uso conjunto.

Discretización

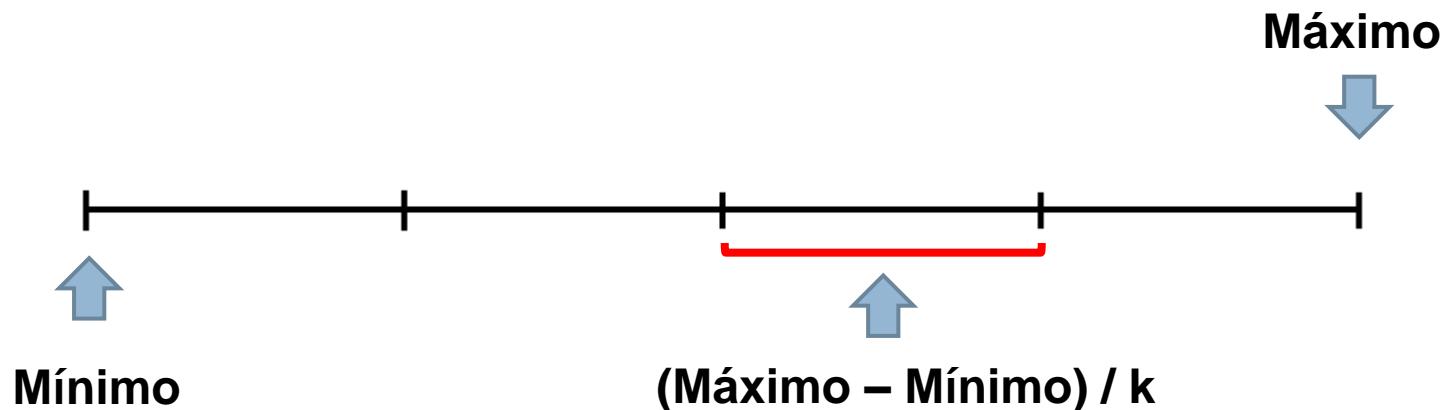
- Convierte un valor numérico en un nominal ordenado (que representa un intervalo o "**bin**")
- **Ejemplo:** Podemos transformar
 - la edad de la persona en categorías: [0,12] niño, (12-21) joven, [21,65] adulto y >65 anciano.
 - La calificación de un alumno en: [4,10] aprobado o [0,4) desaprobado

Discretización

- Puede discretizarse en un número fijo de intervalos. El ancho del intervalo se calcula
 - ▣ Dividiendo el rango en partes iguales
 - ▣ Dividiendo la cantidad de ejemplos en partes iguales (igual frecuencia)
 - ▣ Indicando los límites de cada intervalo en forma manual.
- Averigüe por otras variantes de discretización

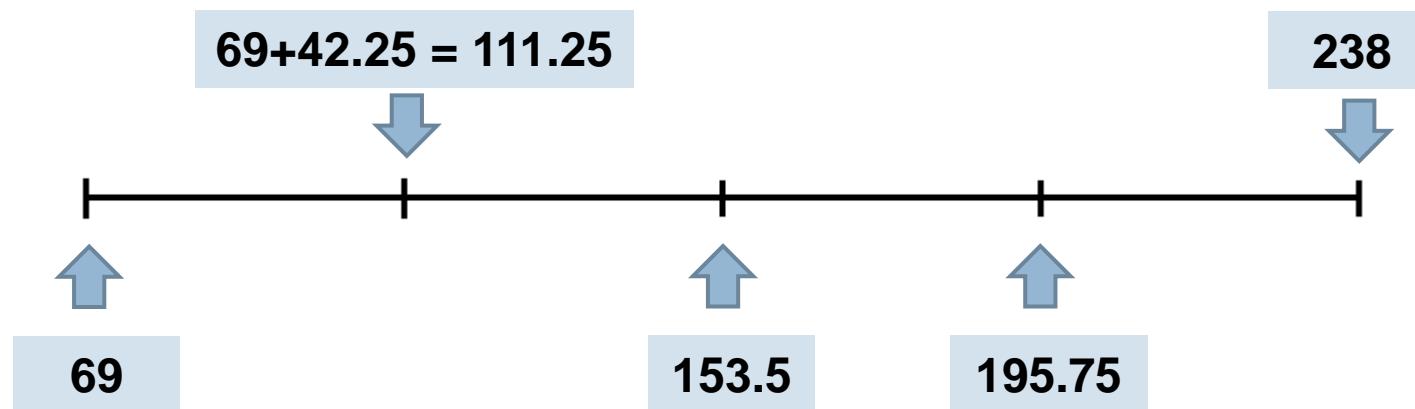
Discretización por rango

- El objetivo es dividir el rango del atributo (intervalo entre el máximo y el mínimo) en una cierta cantidad k de partes iguales.
- Los valores comprendidos en una misma parte serán asociados al mismo valor ordinal.
- Ejemplo: k=4



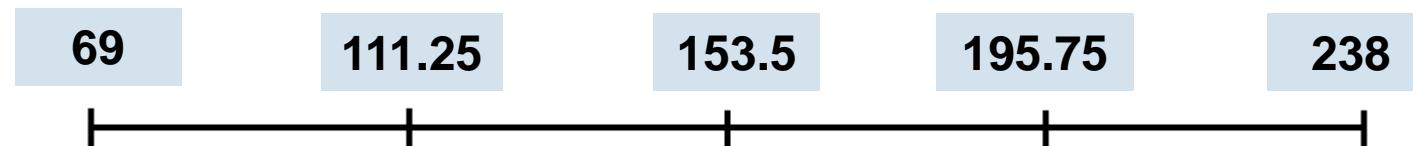
Discretización por rango

- **Ejemplo: Discretizar el atributo DURATION en 4 intervalos de igual longitud**
 - DURATION toma valores entre 69 y 238 minutos. Si dividimos el rango en 4 partes iguales, cada una tendría una longitud de $(238-69)/4 = 42.25$



Discretización por rango

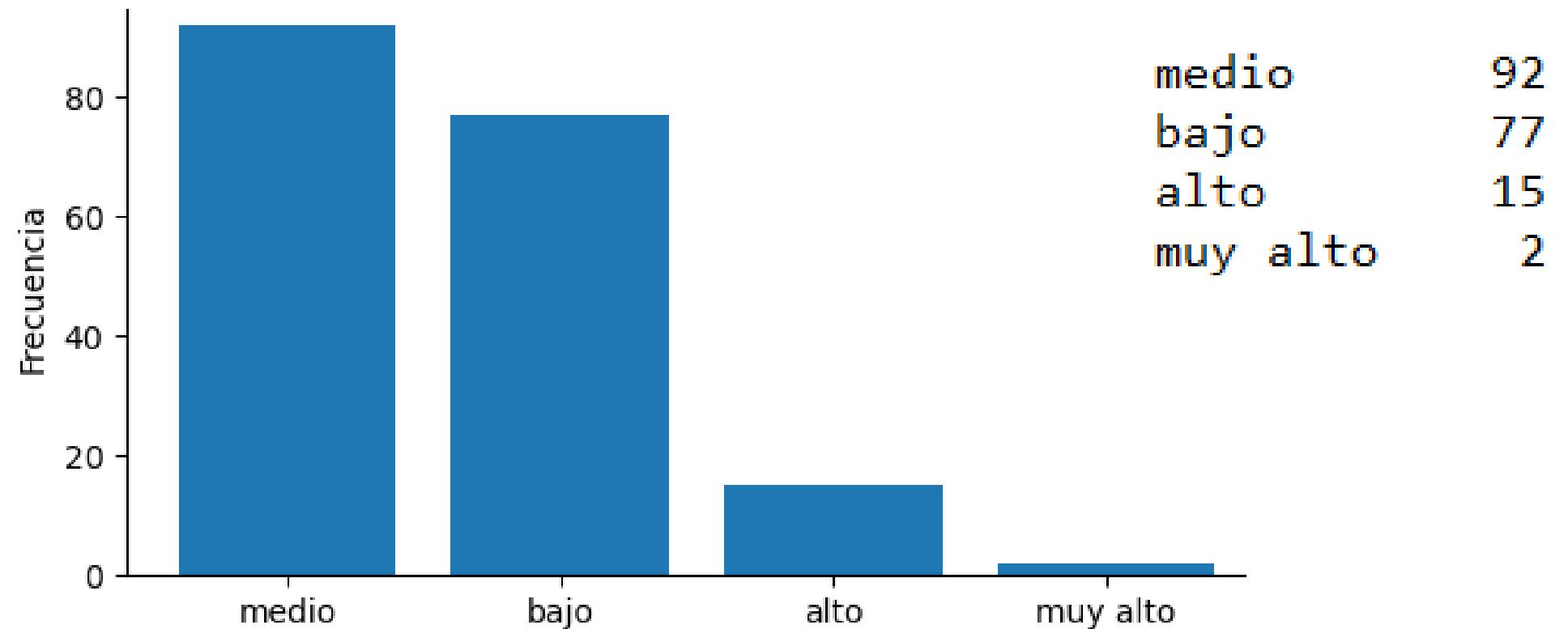
- **Ejemplo: Discretizar el atributo DURATION en 4 intervalos de igual longitud**



Valor	Intervalo	Frecuencia
Rango1	$[-\infty - 111.25]$	77
Rango2	$(111.25 - 153.5]$	92
Rango3	$(153.5 - 195.75]$	15
Rango4	$(195.75 - \infty]$	2

Discretización por rango

□ DURATION discretizado en 4 intervalos de igual longitud



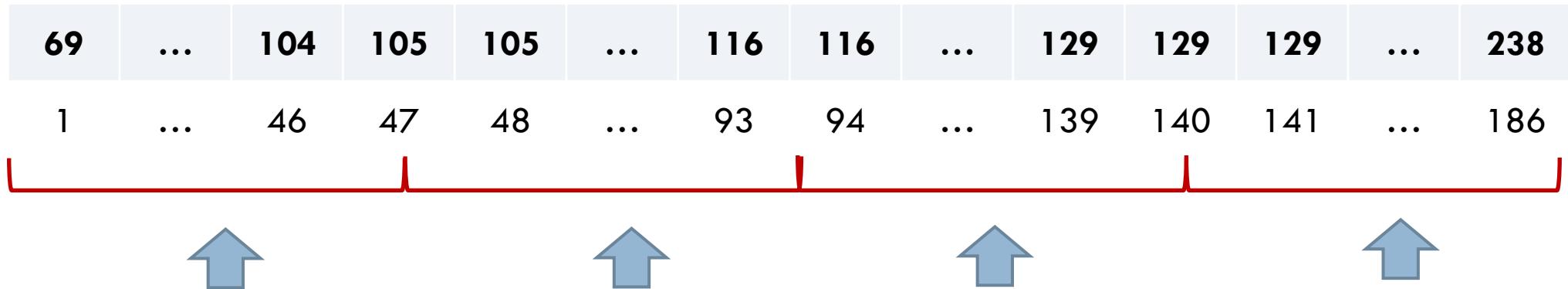
Discretización por frecuencia

- El objetivo es dividir los valores del atributo numérico en k partes con la misma cantidad de valores en cada una de ellas.
- El atributo debe tener al menos k valores diferentes.
- Si hay valores numéricos repetidos los valores ordinales no tendrán la misma frecuencia.

Discretización por frecuencia

- **Ejemplo: Discretizar el atributo DURATION en 4 intervalos de igual frecuencia**

- DURATION tiene 186 valores entre 69 y 238 minutos. Luego de ordenar los valores, los dividimos en k partes con igual cantidad de elementos



Cada intervalo tiene $N/K = 186/4 = 46.5$ elementos

Discretización por frecuencia

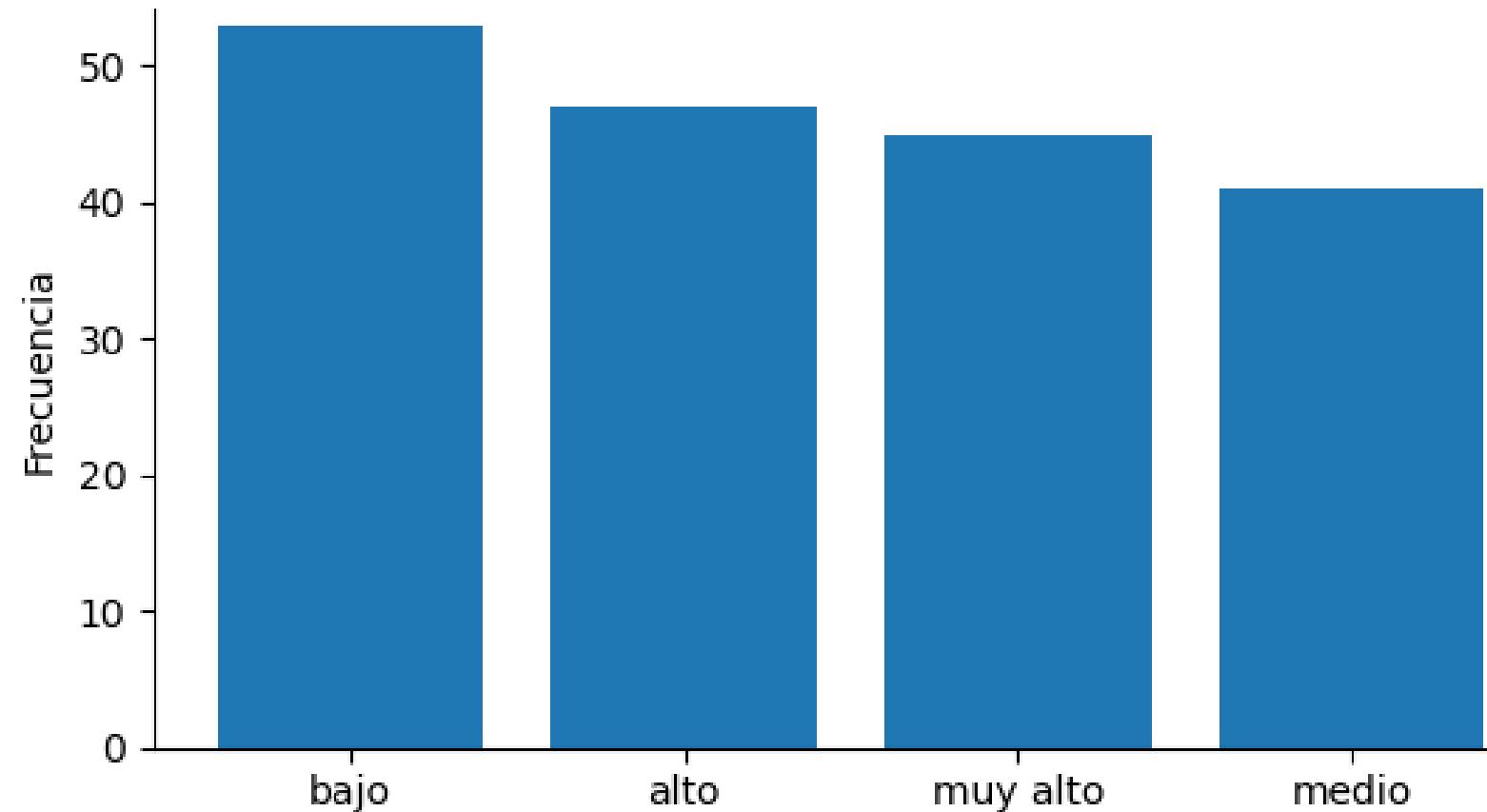
- Ejemplo: Discretizar el atributo DURATION en 4 intervalos de igual frecuencia



Valor	Intervalo	Frecuencia
range1	$[-\infty - 105]$	53
range2	$(105 - 116]$	47
range3	$(116 - 129]$	45
range4	$(129 - \infty]$	41

Discretización por frecuencia

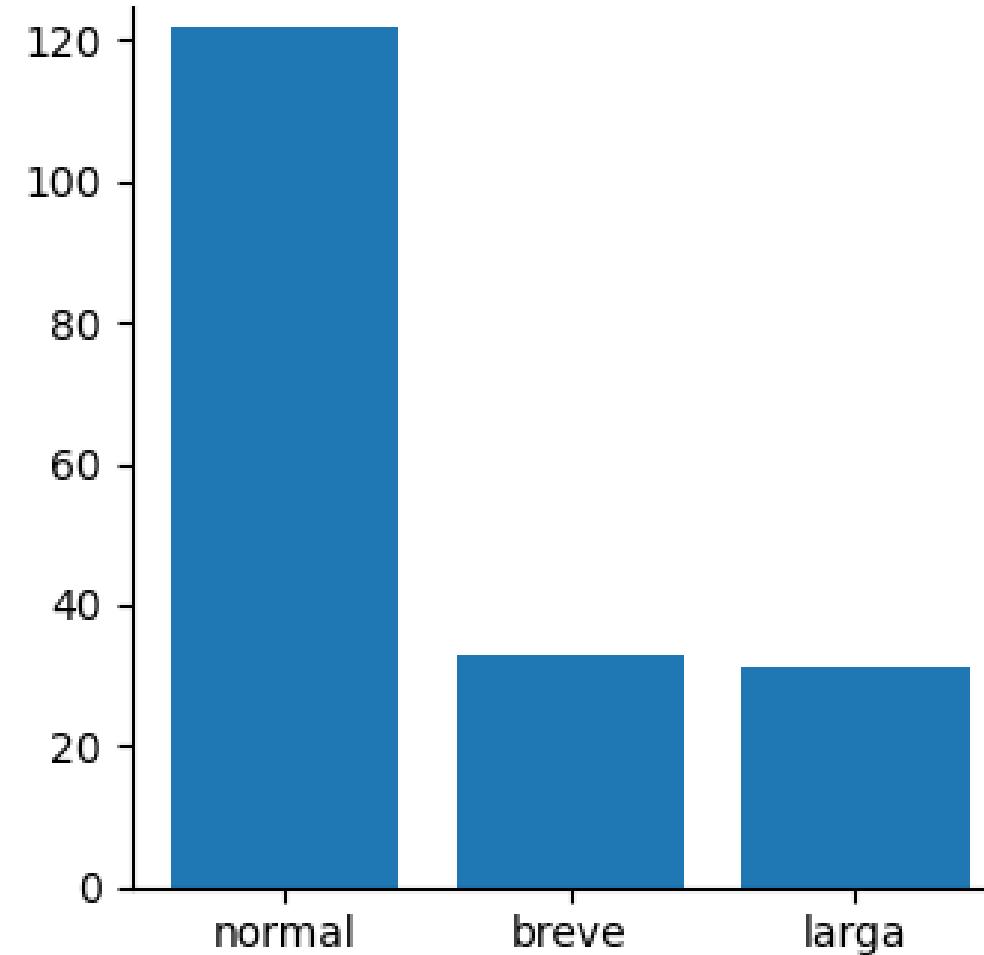
- DURATION discretizado en 4 intervalos de igual frecuencia



Discretización especificada por el usuario

- Si DURATION <= 100, BREVE
- Si (DURATION> 100) y
(DURATION<=136), NORMAL
- Si (DURATION>136), LARGA

Discretizacion.ipynb



Transformación de atributos

□ DISCRETIZACION

- Algunos algoritmos de minería de datos sólo operan con atributos cualitativos. La discretización convierte los atributos numéricos en ordinales.

□ NUMERIZACION

- Es el proceso contrario a la discretización. Convierte atributos cualitativos en numéricos.

□ NORMALIZACION

- Permite expresar los valores de los atributos sin utilizar las unidades de medida originales facilitando su comparación y uso conjunto.

Numerización

- En ocasiones los atributos nominales u ordinales deben convertirse en números.
- Para los nominales suele utilizarse una representación binaria y para los ordinales suele utilizarse una representación entera.
- Es importante considerar que si se numeran en forma correlativa los valores de un atributo nominal se agrega un orden que originalmente no está presente en la información disponible.

Premios2020.csv

Ejemplo

◆ El archivo **Premios2020.csv** contiene 186 premios otorgados entre 1928 y 2020

Year	Age	Actor	Sex	Film	nominations	rating	duration	genre1	genre2	release	synopsis
1928	44	Emil Jannings	M	The Last Command	2.0	8.0	88	Drama	History	4	A former Imperial Russian general and cousin o...
1928	22	Laura Gainor (aka Janet Gaynor)	F	Sunrise	5.0	7.8	110	Drama	Romance	12	A street cleaner saves a young womans life and...
1929	37	Mary Pickford	F	Coquette	1.0	7.3	76	Drama	Romance	4	A flirtatious southern belle is compromised wi...
1929	38	Warner Baxter	M	In Old Arizona	5.0	5.8	95	Romance	Western	1	A charming happy-go-lucky bandit in old Arizon...
1930	62	George Arliss	M	Disraeli	3.0	6.5	90	Biography	Drama	11	Prime Minister of Great Britain Benjamin Disra...



Numerizacion.ipynb

Para las variables ordinales podemos utilizar la numerización de entero único

Numerización Binaria (dummy)

- La numerización binaria reemplaza al atributo nominal por tantos atributos numéricos binarios como valores distintos pueda tomar.
- Las denominaciones de estos nuevos atributos surgen de igualar el nombre original con cada uno de los posibles valores.
- Para un mismo ejemplo sólo uno de estos nuevos atributos tendrá valor 1 y el resto 0.

Ejemplo

◆ El archivo **Premios2020.csv** contiene 186 premios otorgados

Year	Age	Actor	Sex	Film	nominations	rating	duration	genre1	genre2	release	synopsis
1928	44	Emil Jannings	M	The Last Command	2.0	8.0	88	Drama	History	April	A former Imperial Russian general and cousin o...
1928	22	Laura Gainor (aka Janet Gaynor)	F	Sunrise	5.0	7.8	110	Drama	Romance	December	A street cleaner saves a young womans life and...
1929	37	Mary Pickford	F	Coquette	1.0	7.3	76	Drama	Romance	April	A flirtatious southern belle is compromised wi...
1929	38	Warner Baxter	M	In Old Arizona	5.0	5.8	95	Romance	Western	January	A charming happy-go-lucky bandit in old Arizon...
1930	62	George Arliss	M	Disraeli	3.0	6.5	90	Biography	Drama	November	Prime Minister of Great Britain Benjamin Disra...



Las variables nominales deben numerizarse utilizando una representación binaria

Numerizacion.ipynb

Transformación de atributos

□ DISCRETIZACION

- Algunos algoritmos de minería de datos sólo operan con atributos cualitativos. La discretización convierte los atributos numéricos en ordinales.

□ NUMERIZACION

- Es el proceso contrario a la discretización. Convierte atributos cualitativos en numéricos.

□ NORMALIZACION

- Permite expresar los valores de los atributos sin utilizar las unidades de medida originales facilitando su comparación y uso conjunto.

Normalización

- Se aplica según el modelo que se va a construir.
- La más común es la **normalización lineal uniforme**

$$X' = \frac{X - X_{min}}{X_{max} - X_{min}}$$

- Es muy sensible a valores fuera de rango (outliers).
- Si se recortan los extremos se obtiene valor negativos y/o mayores a 1.

Normalización

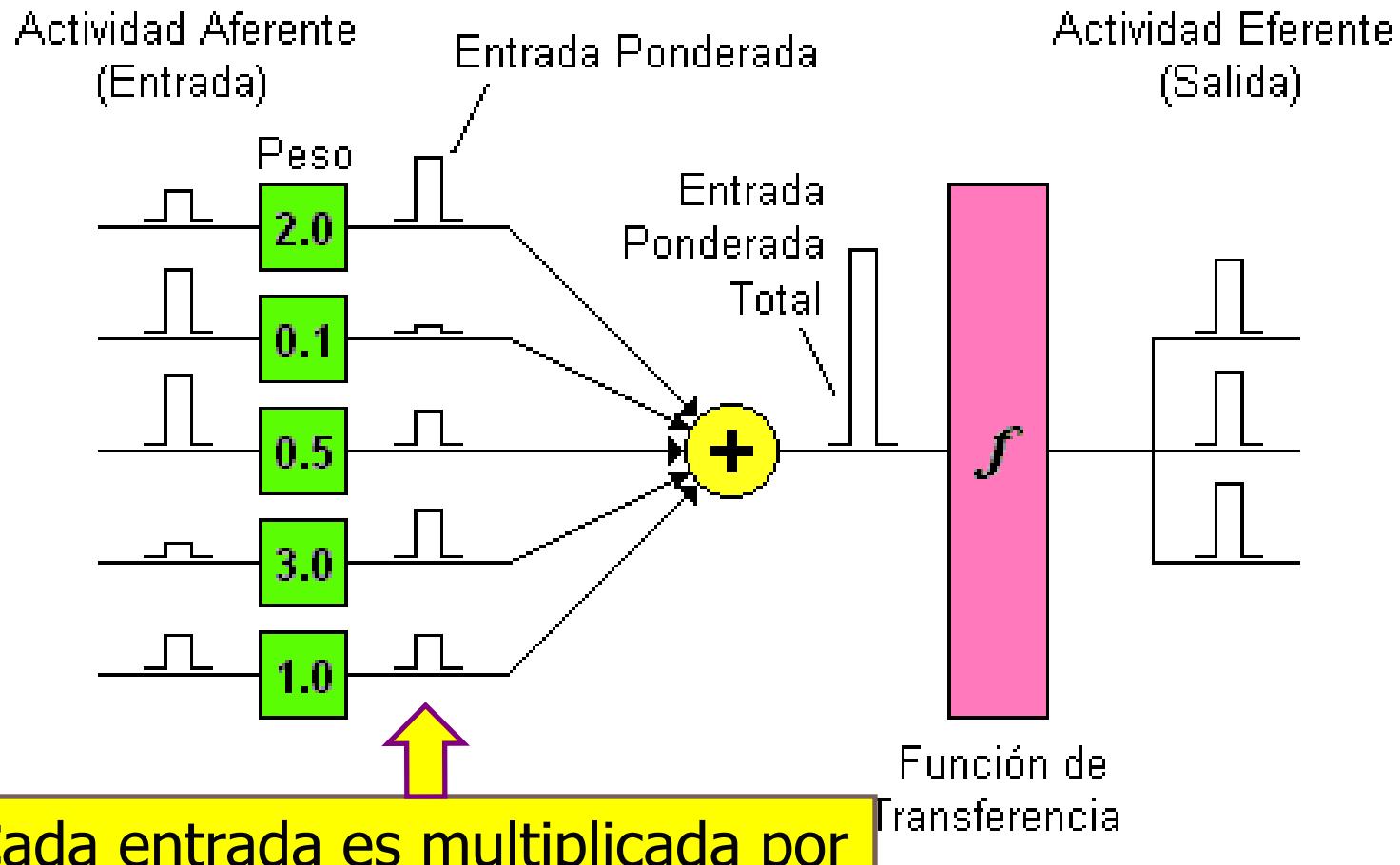
- Existen otras transformaciones. Por ejemplo, si los datos tienen distribución normal se pueden **tipificar**

$$X' = \frac{X - \text{media}(X)}{\text{desviacion}(X)}$$

- De esta forma los datos se distribuyen normalmente alrededor de 0 con desviación 1.

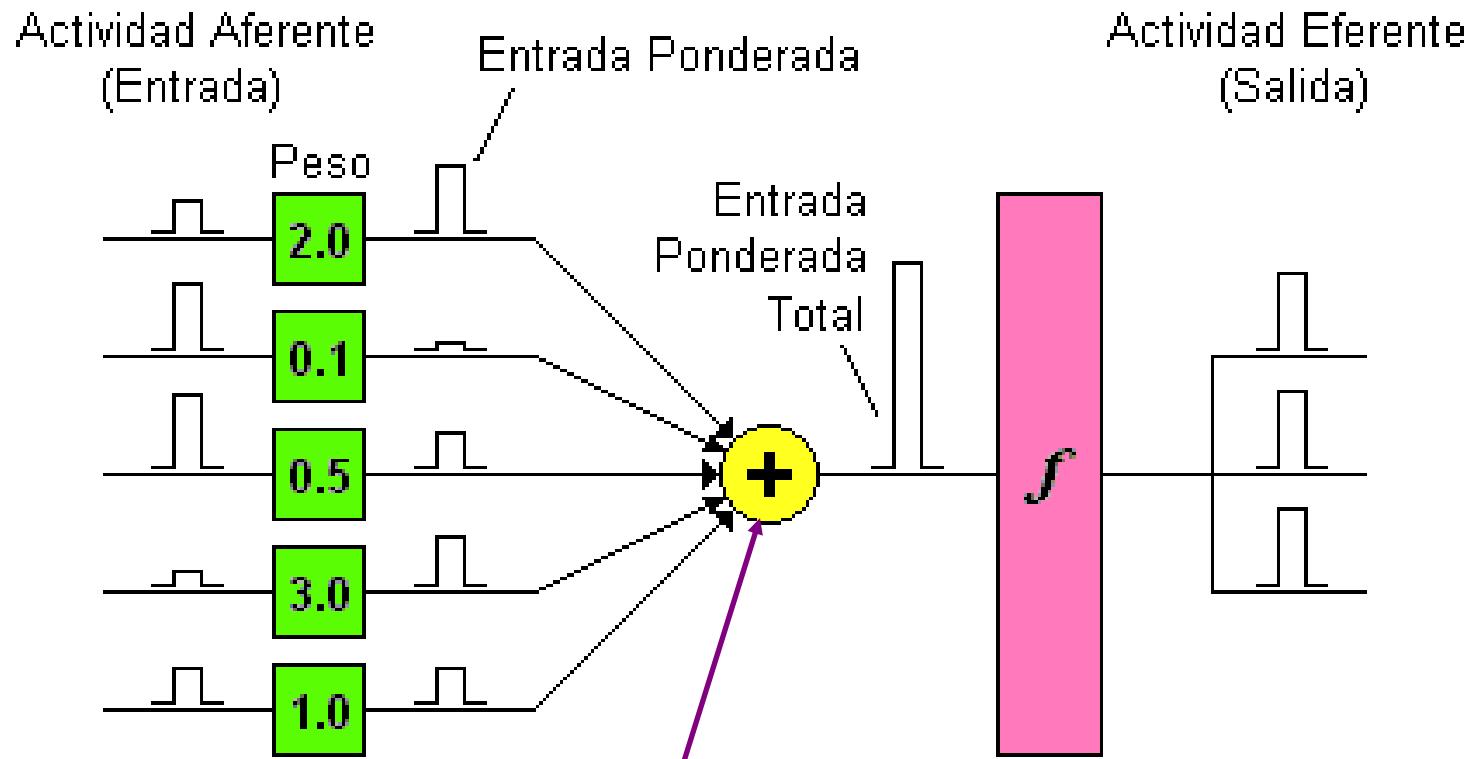
Neurona Artificial

3



Neurona Artificial

4

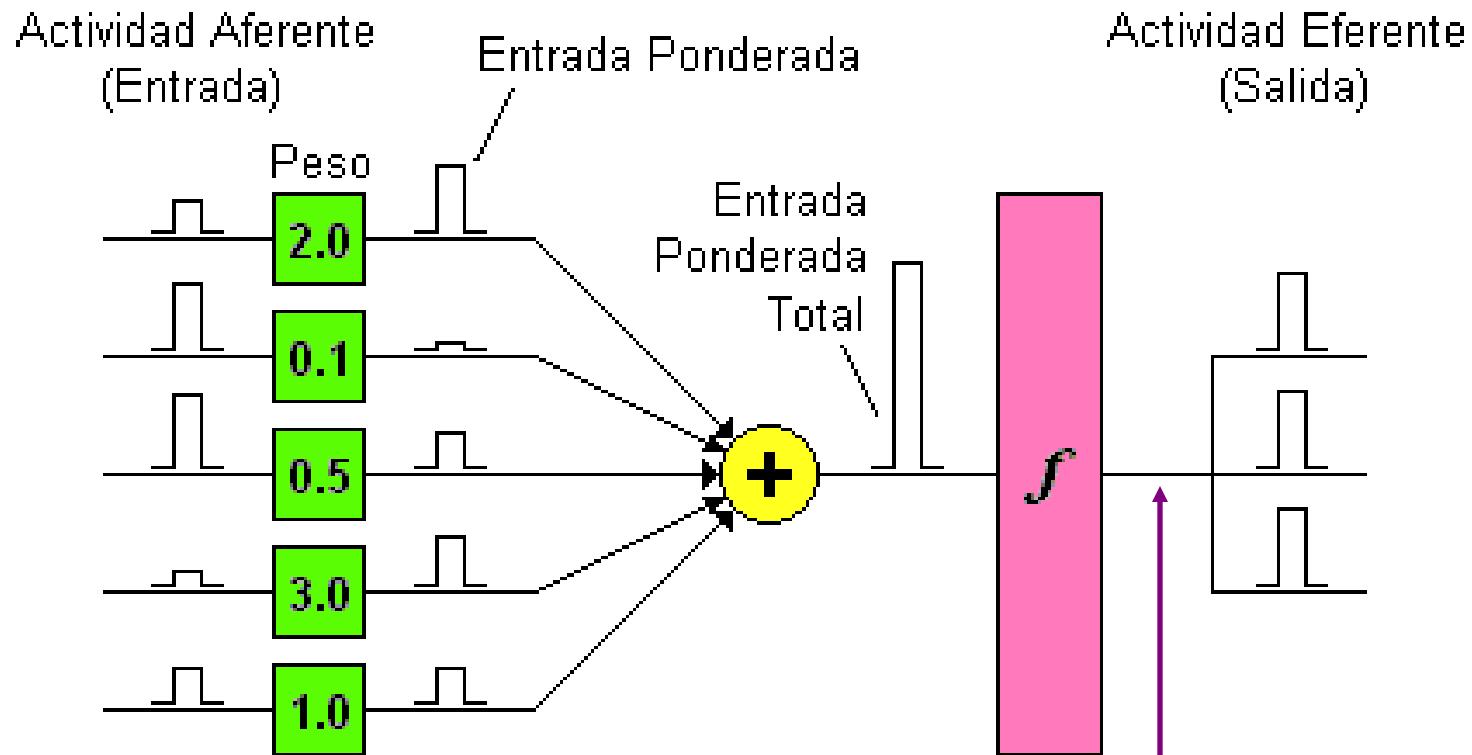


Cada neurona calcula su entrada neta como:

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

Neurona Artificial

5



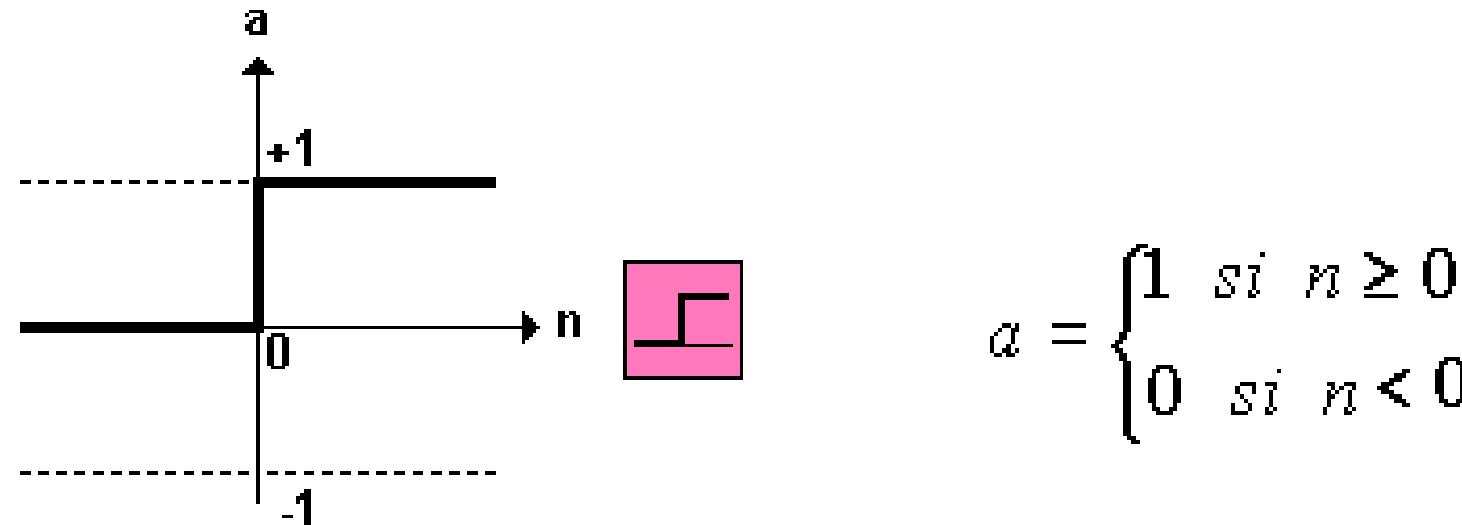
El valor de salida (único) se obtiene como

$$y = f(neta)$$

Funciones de Transferencia

6

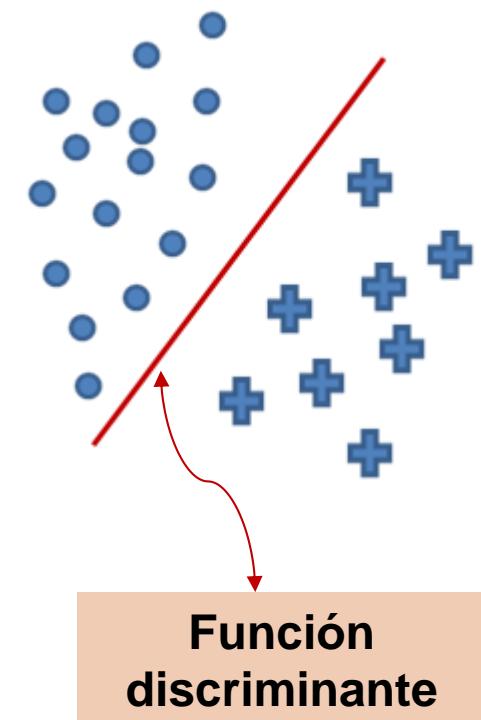
□ Función umbral binaria



Esta función crea neuronas que clasifican las entradas en dos categorías diferentes.

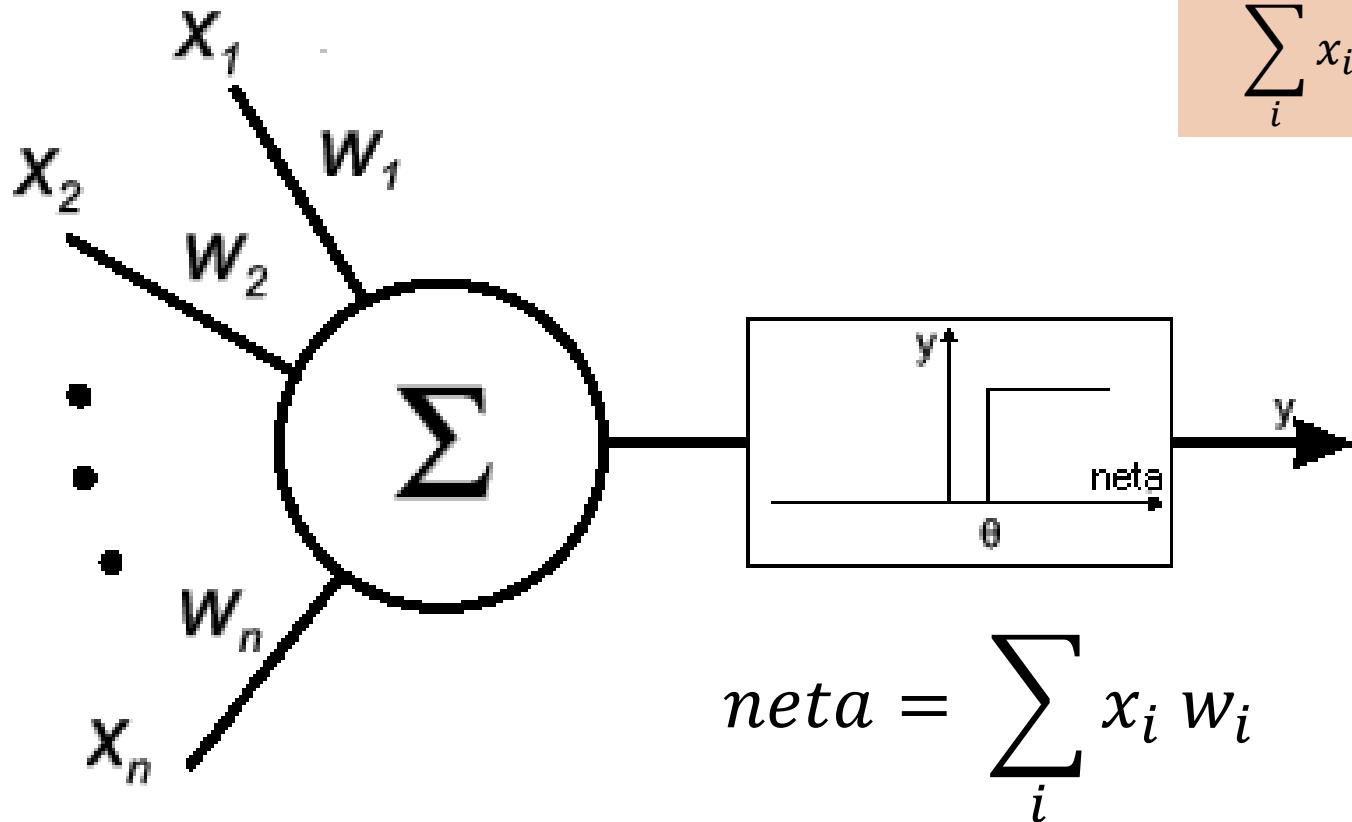
La primera RN – El Perceptrón

- Está formada por una única neurona.
- Utiliza aprendizaje supervisado.
- Su regla de aprendizaje es una modificación de la propuesta por Hebb.
- Se adapta teniendo en cuenta el error entre la salida que da la red y la salida esperada.
- Representa una única función discriminante que separa **linealmente** los ejemplos en **dos** clases.

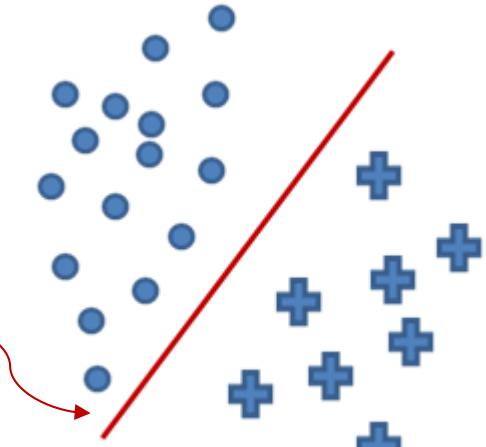


Perceptrón

8



$$\sum_i x_i w_i = \theta$$

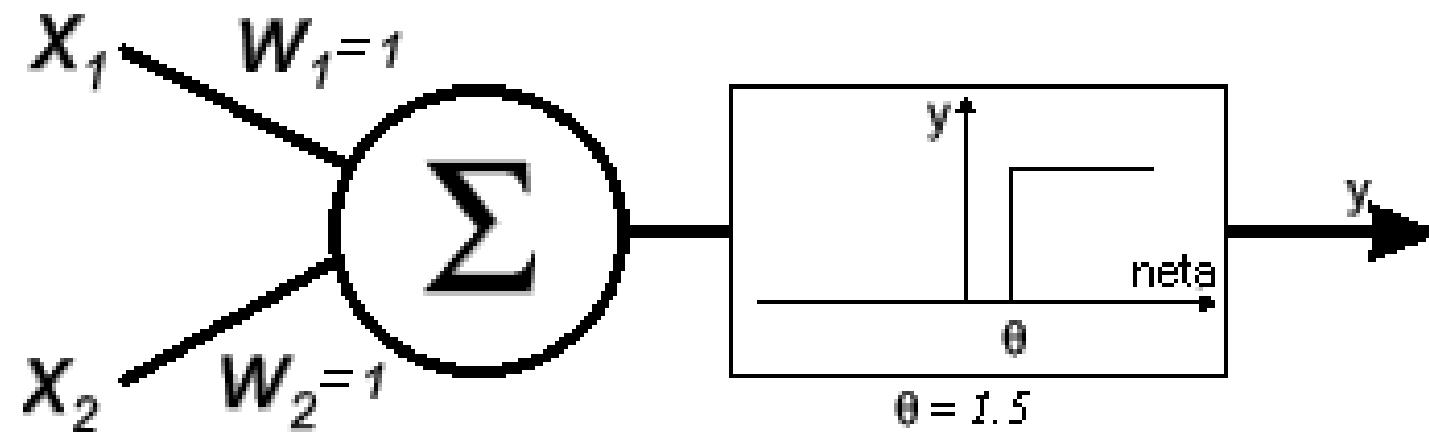


$$y = \begin{cases} 1 & \text{si } neta \geq \theta \\ 0 & \text{si } neta < \theta \end{cases}$$

Ejemplo

9

- Verifique si la siguiente red neuronal se comporta como la función lógica AND



AND

10

$$\theta = 1.5$$

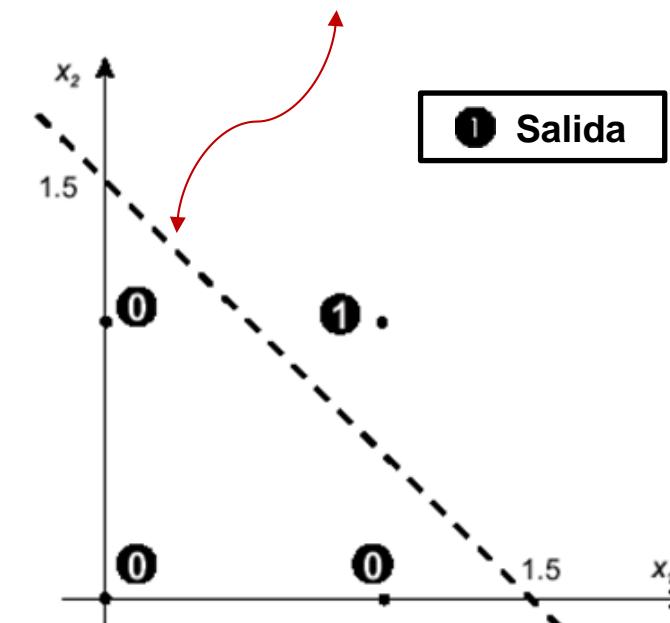
x_1	x_2	neta	salida
0	0	0	0
0	1	1	0
1	0	1	0
1	1	2	1

Graficar la función discriminante
(recta)

Función discriminante

$$x_1 w_1 + x_2 w_2 = \theta$$

$$x_1 + x_2 = 1.5$$



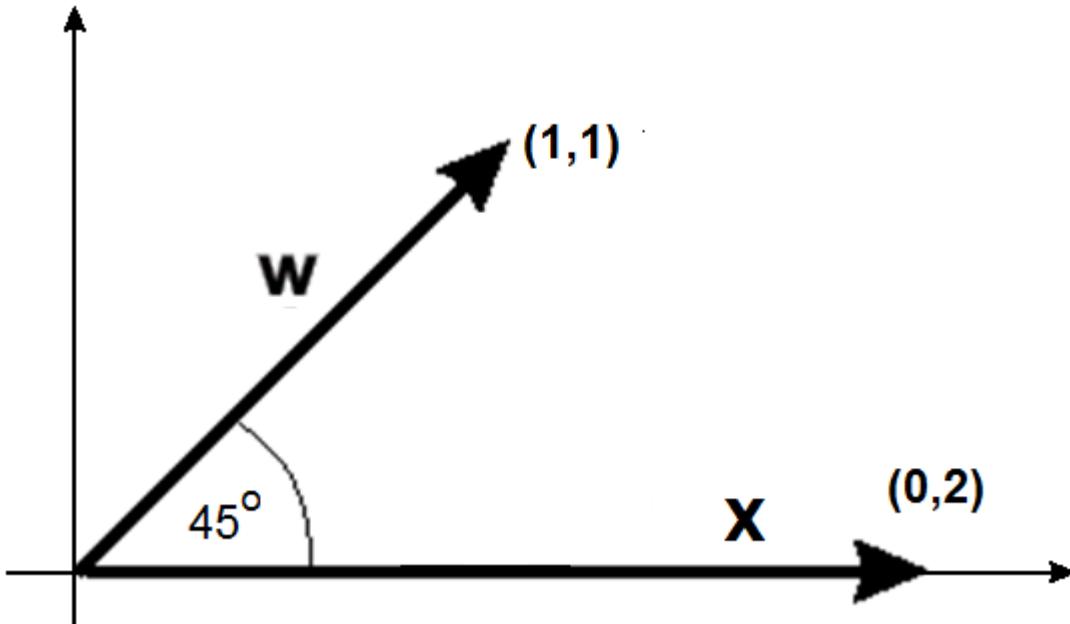
Entrenamiento del perceptrón

11

- Se busca una estrategia iterativa que permita adaptar los valores de las conexiones a medida que se presentan los datos de entrada.
- Ver que el estímulo de entrada se corresponde con el producto interior de los vectores X y W .

Producto interior

12

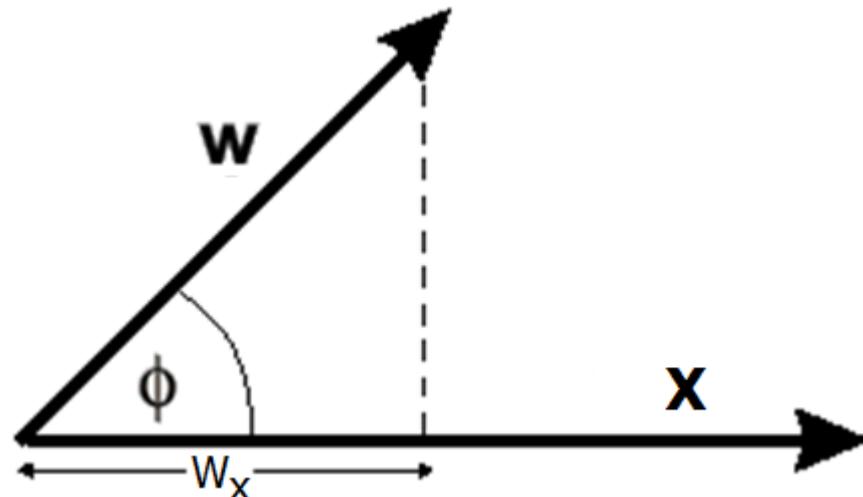


$$w \cdot x = \|w\| \cdot \|x\| \cdot \cos(\phi)$$

$$w \cdot x = \sum_{i=1}^n w_i x_i$$

Vector de proyección

13

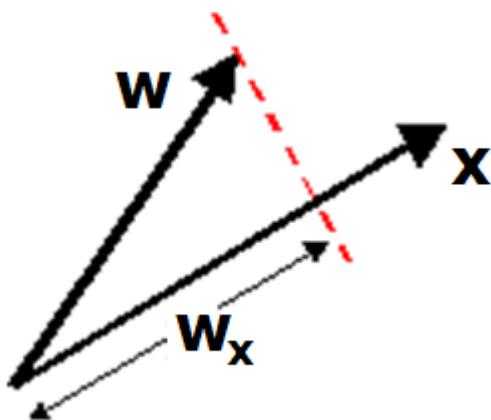


$$w_x = \| w \| \cdot \cos(\phi)$$

$$w_x \cdot \| x \| = W \cdot X$$

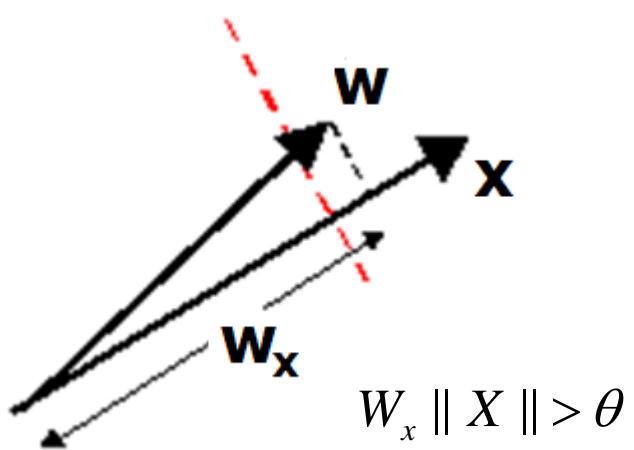
Uso del vector de proyección

14

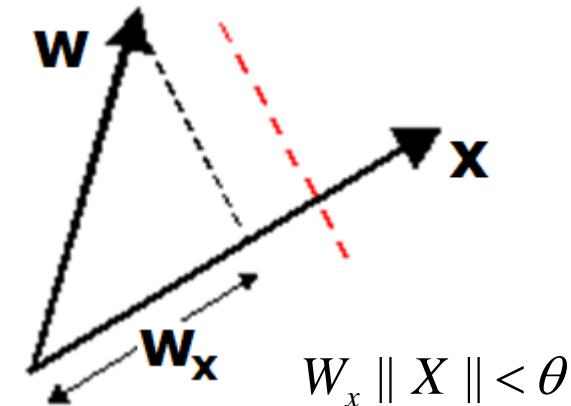


$$W_x \parallel X \parallel = \theta$$

$$W.X = \theta$$



$$W.X > \theta$$



$$W.X < \theta$$

Entrenamiento del Perceptrón

15

- Inicializar los pesos de las conexiones con valores random (vector W)
- □ Mientras no se clasifiquen todos los ejemplos correctamente
 - Ingresar un ejemplo a la red.
 - Si fue clasificado incorrectamente
 - Si esperaba obtener $W \cdot X > \theta$ y no lo logró, "acerque" el vector W al vector X.
 - Si esperaba obtener $W \cdot X < \theta$ y no lo logró, "aleje" el vector W al vector X.

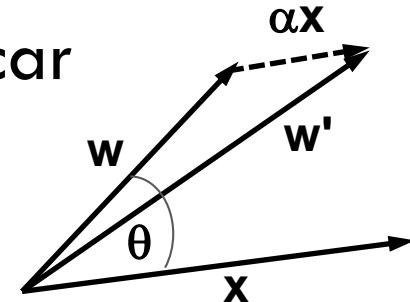
Aprendizaje supervisado

Ajuste del vector de pesos

16

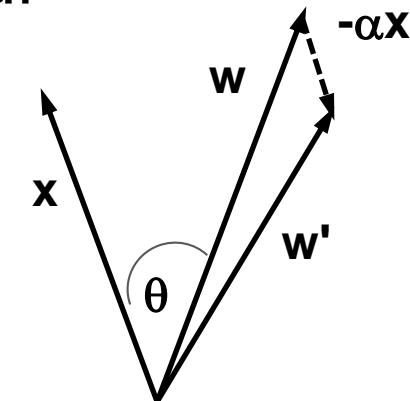
- Si $W \cdot X < \theta$ no es el valor esperado entonces acercar W a X de la siguiente forma

$$w' = w + \alpha x$$



- Si $W \cdot X > \theta$ no es el valor esperado entonces alejar W a X de la siguiente forma

$$w' = w - \alpha x$$



La velocidad de aprendizaje α es un valor real perteneciente a $(0,1]$

Ajuste del vector de pesos

17

- La salida del perceptrón es

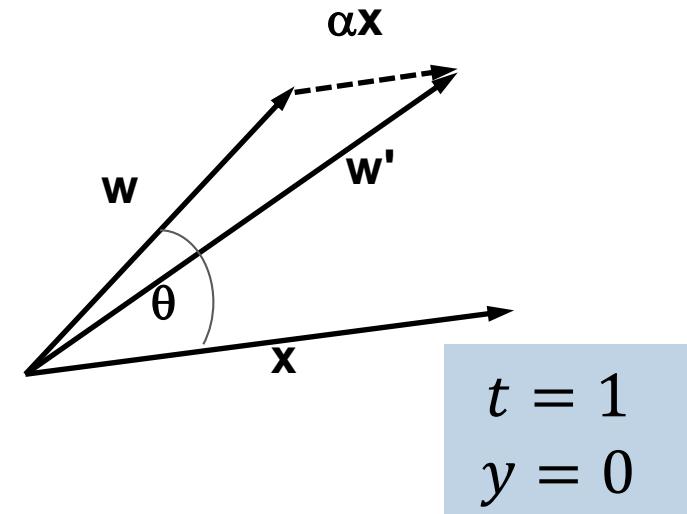
$$y = \begin{cases} 1 & \text{si } W \cdot X \geq \theta \\ 0 & \text{si } W \cdot X < \theta \end{cases}$$

- La actualización de los pesos puede calcularse como

$$w_{\text{nuevo}} = w + \alpha (t - y) x$$

donde

- t es valor esperado
- y es valor obtenido



Ajuste del vector de pesos

18

- La salida del perceptrón es

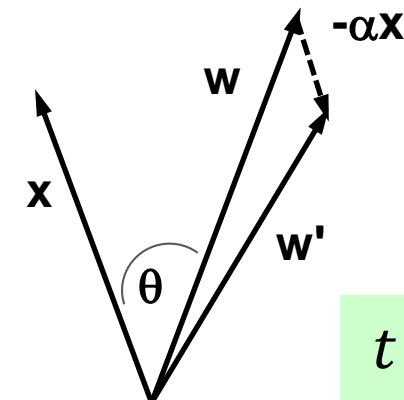
$$y = \begin{cases} 1 & \text{si } W \cdot X \geq \theta \\ 0 & \text{si } W \cdot X < \theta \end{cases}$$

- La actualización de los pesos puede calcularse como

$$w_{\text{nuevo}} = w + \alpha (t - y) x$$

donde

- t es valor esperado
- y es valor obtenido

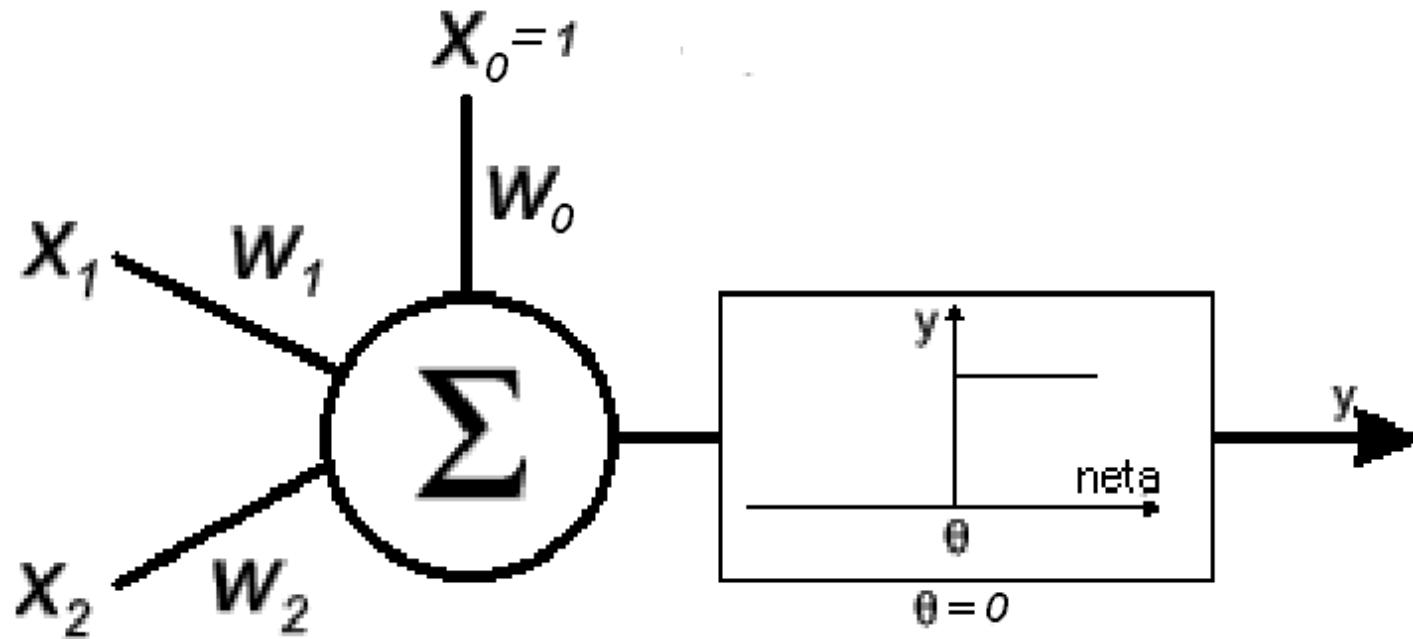


$$\begin{aligned} t &= 0 \\ y &= 1 \end{aligned}$$

Entrenamiento del perceptrón

- Seleccionar el valor de α y θ
- Inicializar los pesos de las conexiones con valores random (vector W)
- Mientras no se clasifiquen todos los ejemplos correctamente
 - Ingresar un ejemplo a la red.
 - Si fue clasificado incorrectamente
 - $W_{\text{nuevo}} = W + \alpha (t - y) x$

Perceptrón



$$neta = \sum_i x_i w_i$$

$$y = \begin{cases} 1 & \text{si } neta \geq 0 \\ 0 & \text{si } neta < 0 \end{cases}$$

Entrenamiento del perceptrón

- Seleccionar el valor de α
- Inicializar los pesos de las conexiones con valores random (vector W y el bias b)
- Mientras no se clasifiquen todos los ejemplos correctamente
 - Ingresar uno a uno los ejemplos a la red.
 - Para cada ejemplo incorrectamente clasificado
 - $W_{nuevo} = W + \alpha (t - y) X$
 - $b_{nuevo} = b + \alpha (t - y)$

COMPLETAR
PERCEPTRON_AND_enClase.ipynb

ClassPerceptron.py

```
ppn = Perceptron(alpha=0.1, n_iter=30, draw=1, title=['X1', 'X2'],
                  random_state=1)
```

- Parámetros de entrada

- **alpha**: valor en el intervalo (0, 1] que representa la velocidad de aprendizaje.
- **n_iter**: máxima cantidad de iteraciones a realizar.
- **draw**: valor distinto de 0 si se desea ver el gráfico y 0 si no. Sólo si es 2D.
- **title**: lista con los nombres de los ejes para el gráfico. Se usa sólo si **draw** no es cero.
- **random_state**: None si los pesos se inicializan en forma aleatoria, un valor entero para fijar la semilla

ClassPerceptron.py

```
ppn = Perceptron(alpha=0.1, n_iter=30)  
ppn.fit(X, T)
```

- Parámetros de entrada
 - **X** : arreglo de NxM donde N es la cantidad de ejemplos y M la cantidad de atributos.
 - **T** : arreglo de N elementos siendo N la cantidad de ejemplos
- Retorna
 - **w_** : arreglo de M elementos siendo M la cantidad de atributos de entrada
 - **b_** : valor numérico continuo correspondiente al bias.
 - **errors_**: errores cometidos en cada iteración.

ClassPerceptron.py

$Y = \text{ppn.predict}(X)$

- Parámetros de entrada
 - X : arreglo de $N \times M$ donde N es la cantidad de ejemplos y M la cantidad de atributos.
- Retorna: un arreglo con el resultado de aplicar el perceptrón entrenado previamente con `fit()` a la matriz de ejemplos X .
 - Y : arreglo de N elementos siendo N la cantidad de ejemplos

Evaluación del modelo

- Matriz de confusión
- Métricas
 - Accuracy
 - Precisión
 - Recall
 - F1-score
 - AUC, Curva ROC

Matriz de Confusión

	Predice Clase 1	Predice Clase 2	Recall
True Clase 1	A	B	$A/(A+B)$
True Clase 2	C	D	$D/(C+D)$
Precision	$A/(A+C)$	$D/(B+D)$	$(A+D)/(A+B+C+D)$

accuracy

- Los **aciertos** del modelo están sobre la **diagonal** de la matriz.
- **Precision:** la proporción de **predicciones correctas** sobre **una clase**.
- **Recall:** la proporción de **ejemplos** de **una clase** que son **correctamente clasificados**.
- **Accuracy:** la performance general del modelo, sobre **todas las clases**. Es la cantidad de **aciertos** sobre el **total** de ejemplos.

Sklearn.metrics.confusion_matrix

```
Y_train = [0, 1, 2, 3, 0, 1, 2, 3, 0, 1, 2, 3]  
Y_pred = [0, 2, 1, 3, 0, 1, 2, 0, 0, 1, 2, 3]
```

```
MM = metrics.confusion_matrix(Y_train,Y_pred)  
print("Matriz de confusión:\n%s" % MM)
```

Matriz de confusión:

	0	1	2	3
0	3	0	0	0
1	0	2	1	0
2	0	1	2	0
3	1	0	0	2

PREDICE

Esperaba obtener 1
como respuesta pero
la red respondió 2

Sklearn.metrics.classification_report

```
Y_train = [0, 1, 2, 3, 0, 1, 2, 3, 0, 1, 2, 3]
Y_pred = [0, 2, 1, 3, 0, 1, 2, 0, 0, 1, 2, 3]
report = metrics.classification_report(Y_train,Y_pred)
print("Resultado de la clasificación:\n%s" % report)
```

Resultado de la clasificación:

	precision	recall	f1-score	support
0	0.75	1.00	0.86	3
1	0.67	0.67	0.67	3
2	0.67	0.67	0.67	3
3	1.00	0.67	0.80	3

accuracy			0.75	12
macro avg	0.77	0.75	0.75	12
weighted avg	0.77	0.75	0.75	12



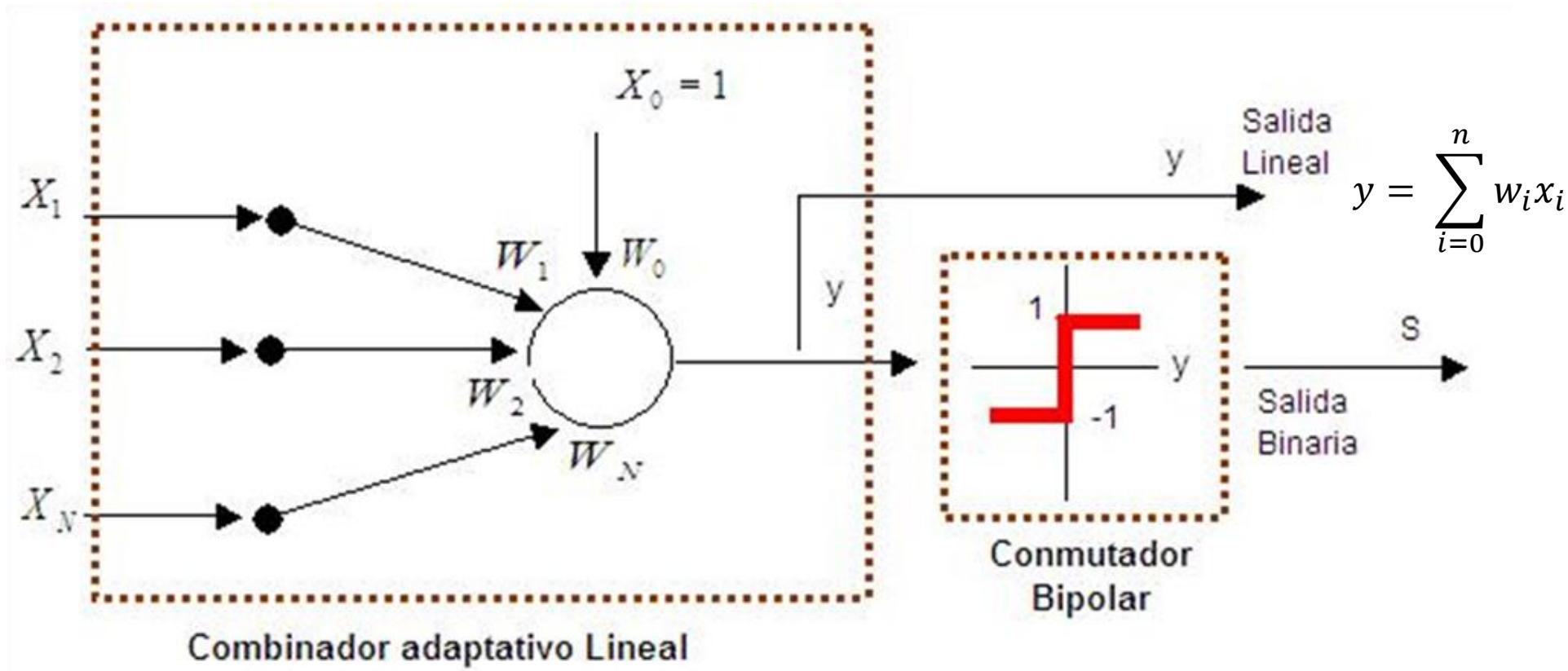
F1-score

$$F1 = 2 * \frac{precision * recall}{precision + recall}$$

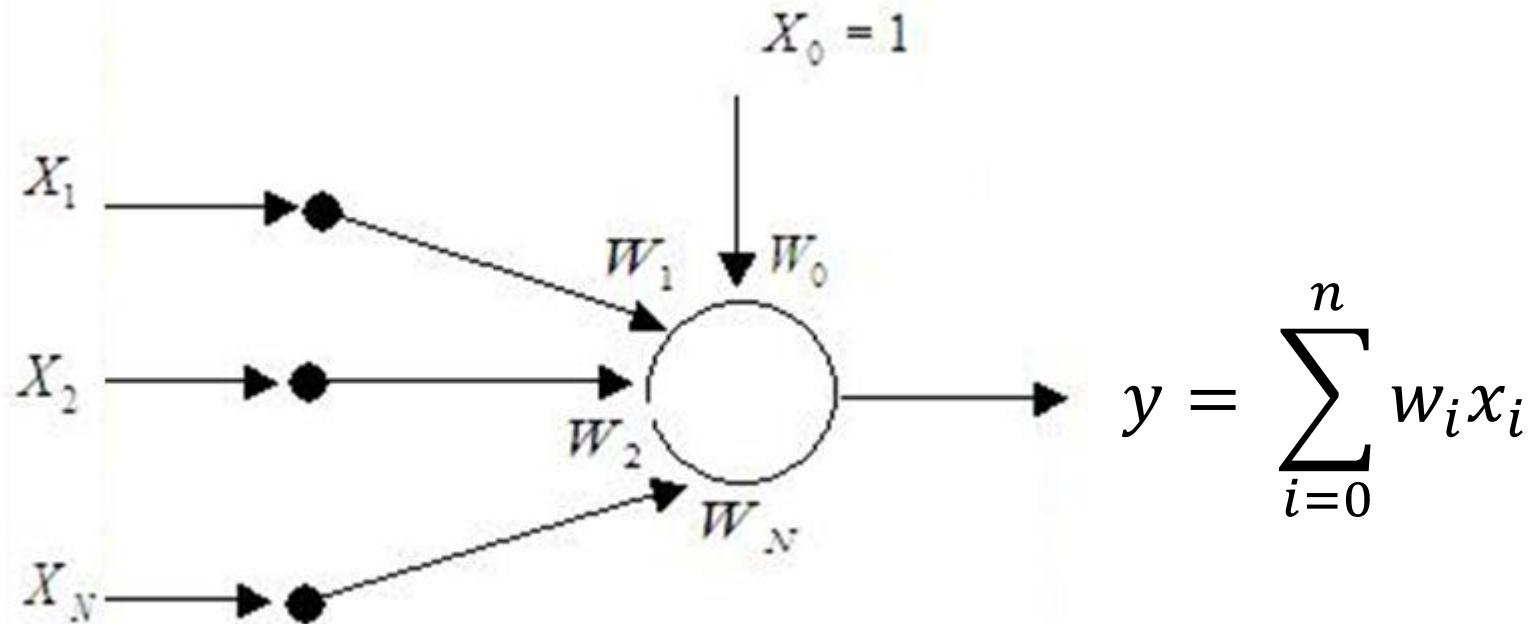
Red ADALINE (ADaptive LINear Element)

- Red neuronal formada por una sola neurona desarrollada por Widrow en 1960 al mismo tiempo que Rosenblatt trabajaba en el modelo del Perceptrón.
 - Adapta los pesos de las conexiones **teniendo en cuenta el error** cometido al responder con respecto al valor esperado.
 - Utiliza el **algoritmo LMS** (Least Mean Square) o **regla delta** para determinar los pesos de los arcos a fin de minimizar el error cuadrático medio.
-

Red ADALINE



Combinador Adaptativo Lineal



Combinador Lineal

□ Busca minimizar

$$\xi = \langle \varepsilon_k^2 \rangle = \frac{1}{L} \left[\sum_{k=1}^L \left(d_k - \sum_{i=0}^N x_{ik} w_i \right)^2 \right]$$

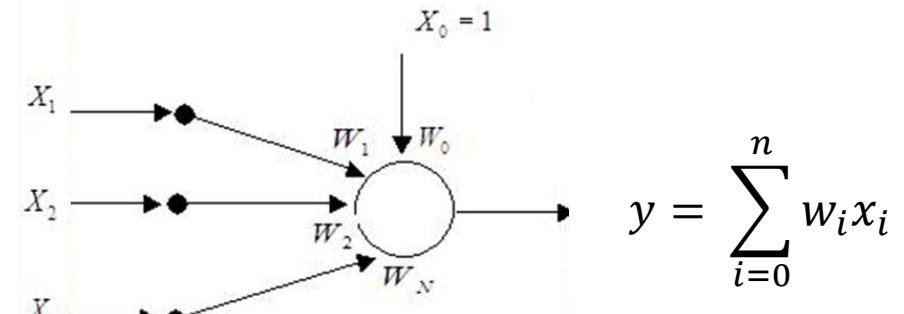
donde

L : Cantidad de ejemplos

N : Cantidad de neuronas de entrada

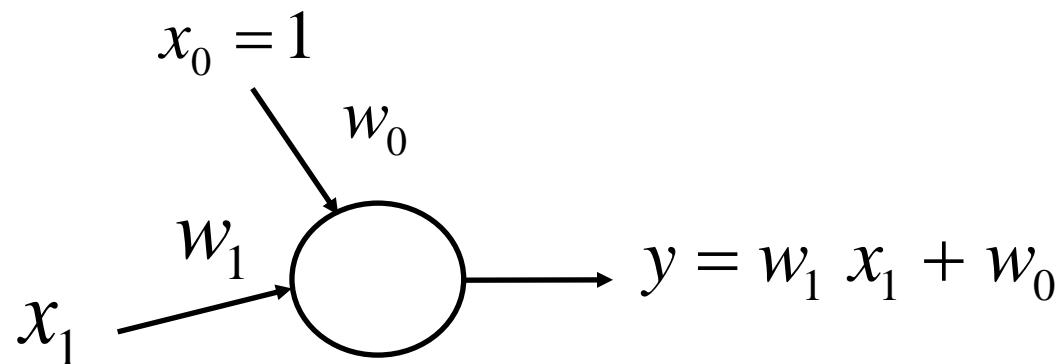
$\langle \rangle$ representa promedio

d_k : salida esperada para el ejemplo x_k



Ejemplo: Entrene un combinador lineal utilizando los siguientes ejemplos: (2,3), (1,1), (-1,-3).

- El combinador lineal será de la forma



- Se busca determinar los valores de w_0 y w_1 que minimicen el error cuadrático medio

$$\xi = \frac{1}{3} \sum_{k=1}^3 (d_k - y_k)^2 = \frac{1}{3} \sum_{k=1}^3 \left(d_k - \sum_{i=0}^1 w_i x_{ik} \right)^2 = \frac{1}{3} \sum_{k=1}^3 (d_k - (w_1 x_k + w_0))^2$$

Ejemplo: Función de error a minimizar para los ejemplos:
 $(2,3), (1,1), (-1,-3)$

□ Se busca minimizar

$$\xi = \frac{1}{3} \sum_{k=1}^3 (d_k - y_k)^2 = \frac{1}{3} \sum_{k=1}^3 (d_k - (w_1 x_k + w_o))^2$$

$$\xi = \frac{1}{3} \left((3 - (w_1 2 + w_o))^2 + (1 - (w_1 + w_o))^2 + (-3 - (w_1(-1) + w_o))^2 \right)$$

$$\xi = \frac{1}{3} ((3 - 2w_1 - w_0)^2 + (1 - w_1 - w_o)^2 + (-3 + w_1 - w_0)^2)$$

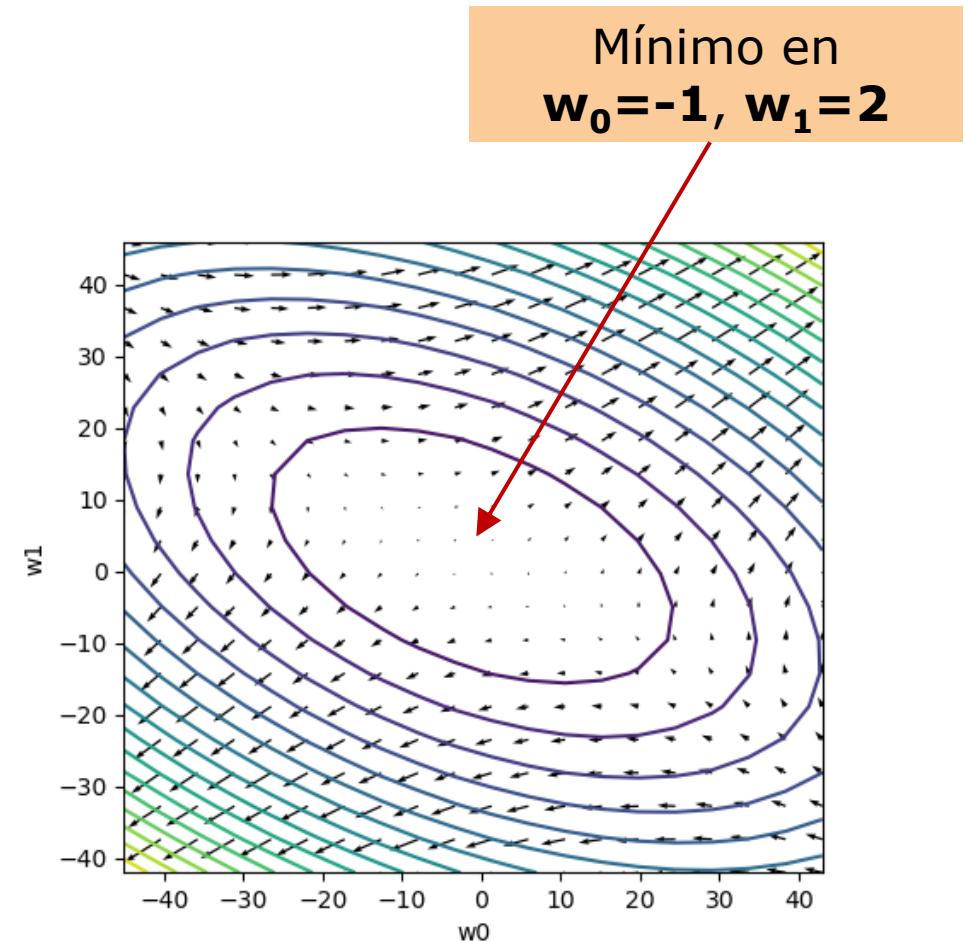
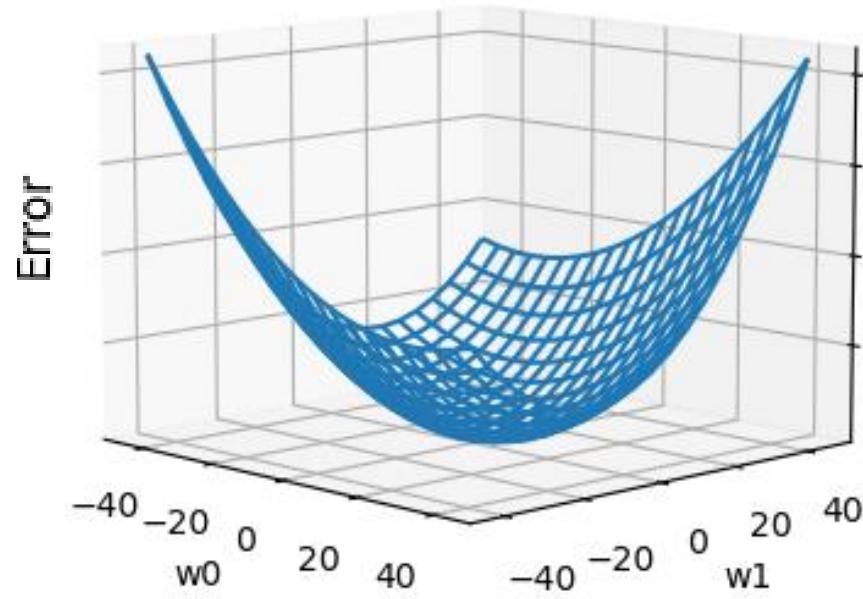
$$\xi = \frac{1}{3} (19 - 20w_1 - 2w_0 + 6w_1^2 + 4w_1 w_o + 3w_0^2)$$

Ejemplo: Función de error a minimizar para los ejemplos:

(2,3), (1,1), (-1,-3)

- Se busca minimizar la siguiente función

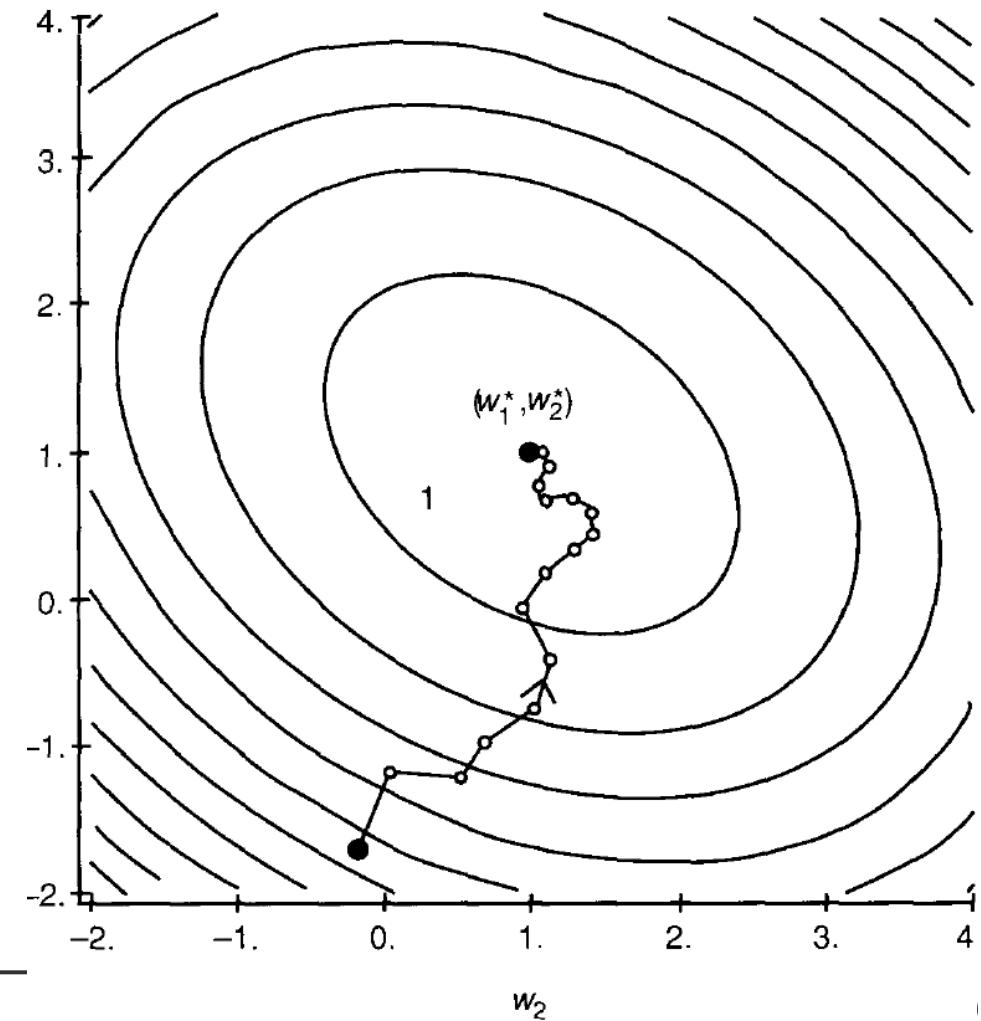
$$\xi = \frac{1}{3}(19 - 20w_1 - 2w_0 + 6w_1^2 + 4w_1w_0 + 3w_0^2)$$



Entrenamiento del Combinador Lineal

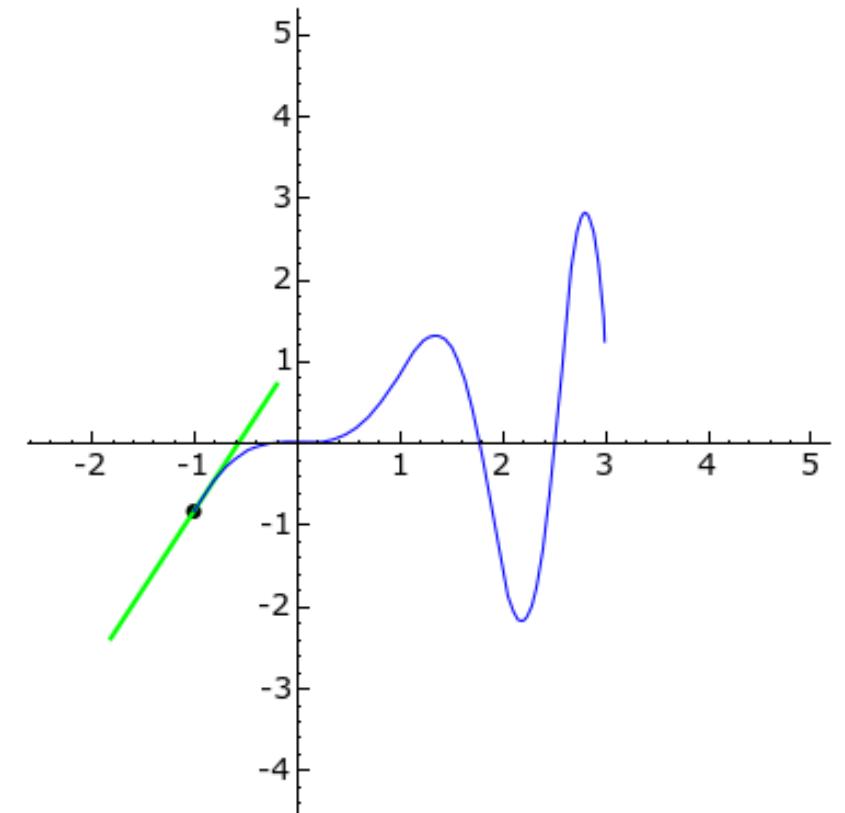
- Se busca un método de entrenamiento que, a partir de los datos de entrada, permita calcular el vector W.

- Conceptos relacionados
 - Derivada
 - Derivada parcial
 - Vector gradiente

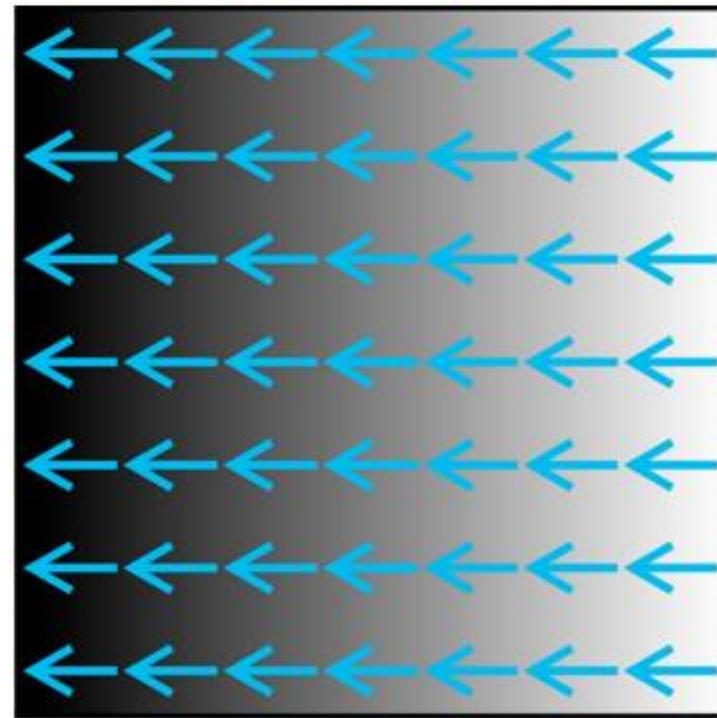
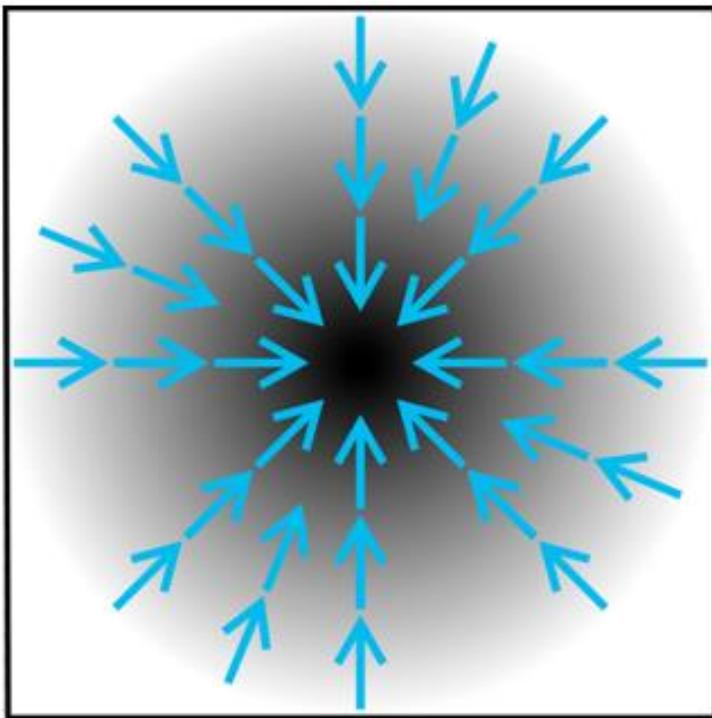


Derivada de una función

- Es una medida de la rapidez con la que cambia el valor de la función.
- Evaluada en un punto se corresponde con la pendiente de la recta tangente a la gráfica de la función en dicho punto.



Gradiente



En las dos imágenes, los valores de la función se representan en blanco y negro. El negro representa valores más altos y su gradiente correspondiente se representa con flechas azules.

Minimización de funciones usando el gradiente

- Dada una función continua

- Tomar un punto dentro del dominio de la función.
- Calcular el vector gradiente de la función en ese punto.
- Sumarle al punto anterior una fracción del gradiente negativo (para ir hacia el mínimo).
- Repetir los dos pasos anteriores hasta que la diferencia entre evaluaciones consecutivas de la función sea inferior a una cierta cota.

Gradiente

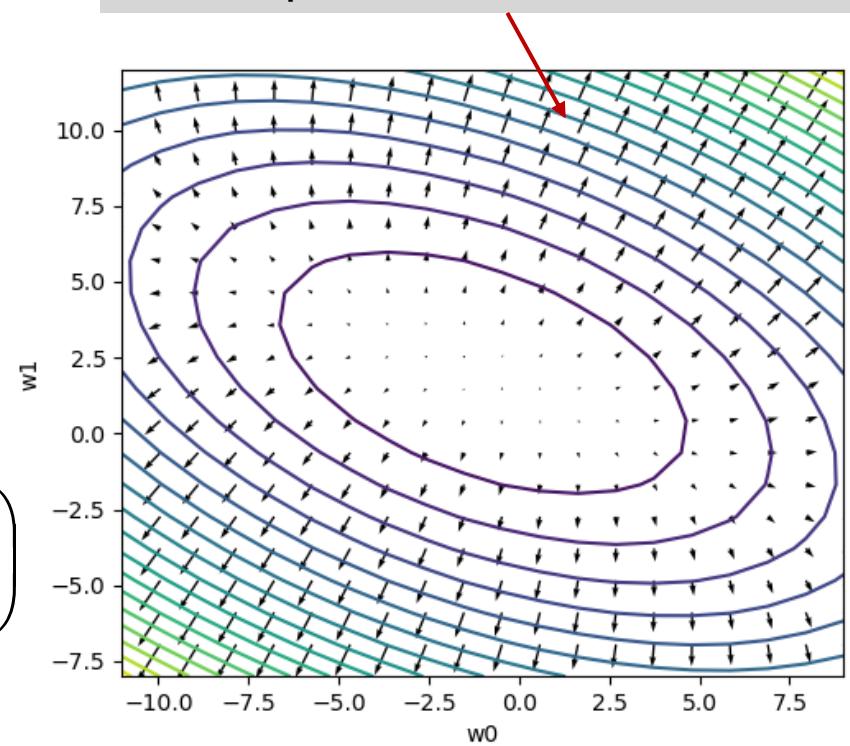
$$\xi = \frac{1}{3} (19 - 20w_1 - 2w_0 + 6w_1^2 + 4w_1w_0 + 3w_0^2)$$

$$\frac{\partial \xi}{\partial w_0} = \frac{1}{3} (-2 + 4w_1 + 6w_0)$$

$$\frac{\partial \xi}{\partial w_1} = \frac{1}{3} (-20 + 12w_1 + 4w_0)$$

$$\nabla \xi = \left(\frac{\partial \xi}{\partial w_0}, \frac{\partial \xi}{\partial w_1} \right) = \left(\frac{-2 + 4w_1 + 6w_0}{3}, \frac{-20 + 12w_1 + 4w_0}{3} \right)$$

Vector numérico que resulta de evaluar las derivadas parciales en un punto dado de la función

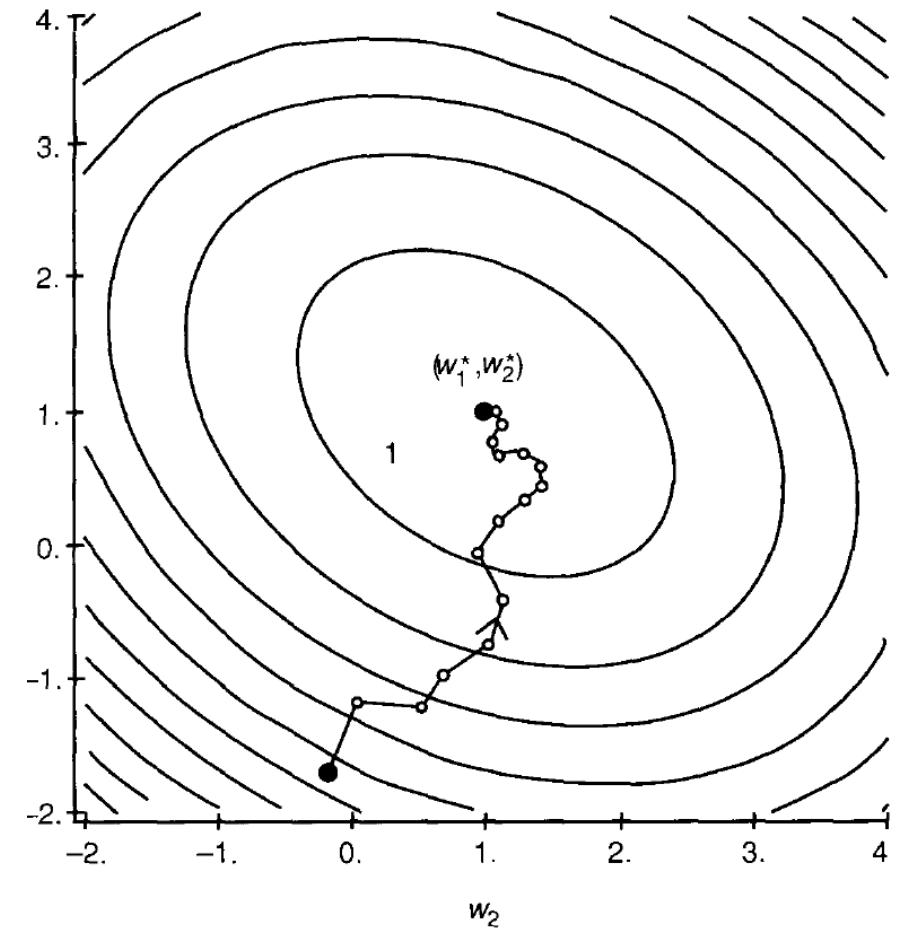


Técnica del descenso del gradiente estocástico

$$w_{t+1} = w_t - \alpha \nabla \xi(w_t)$$

□ se utiliza

$$\xi = \langle \varepsilon_k^2 \rangle \approx \varepsilon_k^2 = \left(d_k - \sum_{i=0}^N x_{ik} w_i \right)^2$$



Técnica del descenso del gradiente estocástico

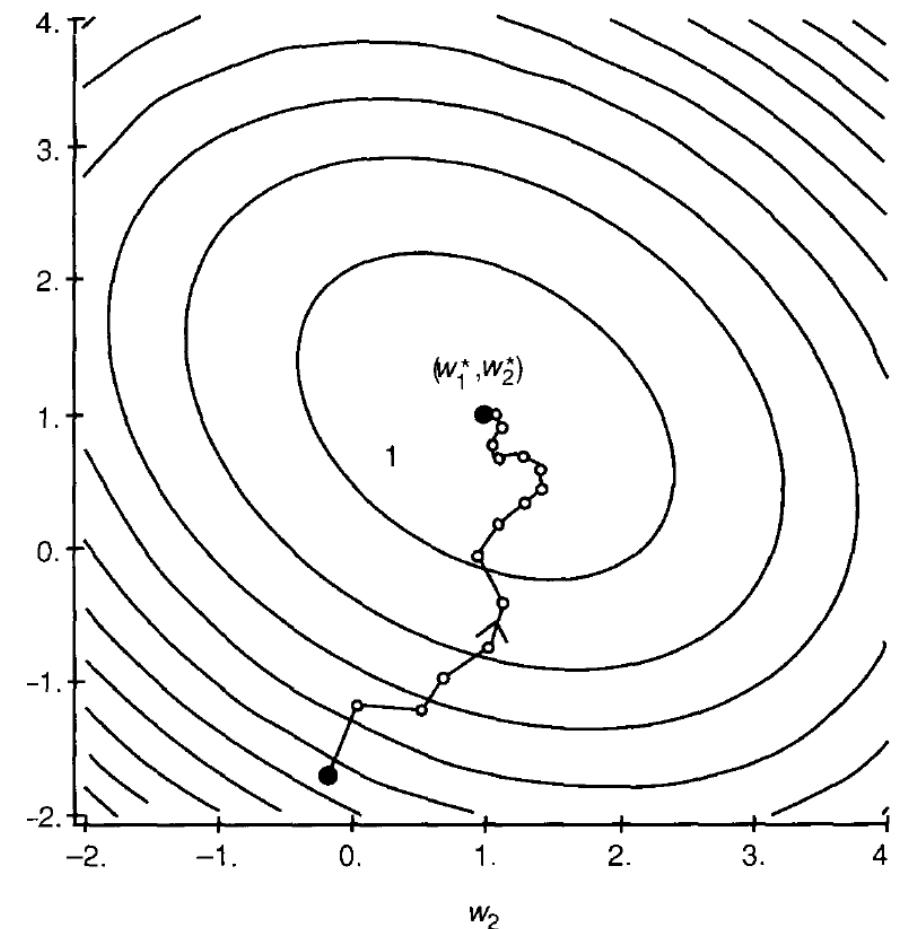
$$w_{t+1} = w_t - \alpha \nabla \xi(w_t)$$

□ se utiliza

$$\xi = \langle \varepsilon_k^2 \rangle \approx \varepsilon_k^2 = \left(d_k - \sum_{i=0}^N x_{ik} w_i \right)^2$$

□ Veamos que

$$\nabla \varepsilon_k^2 = -2 \varepsilon_k x_k$$



Gradiente del error en el ejemplo x_k

$$\nabla \varepsilon_k^2(t) = \frac{\partial \varepsilon_k^2}{\partial w} = \left[\frac{\partial (d_k - y_k)^2}{\partial w_0}; \dots; \frac{\partial (d_k - y_k)^2}{\partial w_n} \right]$$

$$\nabla \varepsilon_k^2(t) = \frac{\partial \varepsilon_k^2}{\partial w} = \left[-2(d_k - y_k) \frac{\partial y_k}{\partial w_0}; \dots; -2(d_k - y_k) \frac{\partial y_k}{\partial w_n} \right]$$

$$\nabla \varepsilon_k^2(t) = \frac{\partial \varepsilon_k^2}{\partial w} = -2e_k \left[\frac{\partial y_k}{\partial w_0}; \dots; \frac{\partial y_k}{\partial w_n} \right]$$

$$\nabla \varepsilon_k^2(t) = \frac{\partial \varepsilon_k^2}{\partial w} = -2e_k [x_{0k}, x_{1k}, \dots, x_{nk}] = -2e_k x_k$$

Entrenamiento del combinador lineal

- Para cada vector de entrada

- Aplicar el vector de entrada, x_k
 - Calcular el gradiente utilizando

$$\nabla \langle \varepsilon_k^2 \rangle \approx \nabla \varepsilon_k^2 = -2\varepsilon_k x_k = -2(d_k - y_k)x_k$$

- Actualizar el vector de pesos

$$w_{t+1} = w_t + 2\alpha(d_k - y_k)x_k$$

- Repetir todo hasta que el error sea aceptable
-

```

import numpy as np
X = [2, 1, -1] # (2,3), (1,1), (-1,-3)
Y = [3, 1, -3]
[w0, w1] = np.random.uniform(-50, 50, size=2)

alfa = 0.1
MAX_ITE = 5000
COTA = 10e-06

ite = 0
E_ant = 0
E = 1
while ((ite<MAX_ITE) and (np.abs(E_ant - E) > COTA)):
    E_ant=E
    sumaError = 0

    for p in range(len(X)):
        salida = w1 * X[p] + w0
        Error = Y[p]-salida

        grad_w0 = -2*Error
        grad_w1 = -2*Error*X[p]

        w0 = w0 - alfa * grad_w0
        w1 = w1 - alfa * grad_w1

        sumaError = sumaError + Error**2

    E = sumaError / len(X)
    ite = ite + 1
print ("ite= %d      w0= %8.5f      w1=%8.5f      E=% .8f" % (ite,w0,w1,E))

```

Termina o bien porque realizó la máxima cantidad de intentos o porque el valor absoluto de la diferencia entre dos valores consecutivos de la función es inferior a cierta cota

```

import numpy as np
x = [2, 1, -1] # (2,3), (1,1), (-1,-3)
Y = [3, 1, -3]
[w0, w1] = np.random.uniform(-50, 50, size=2)

alfa = 0.1
MAX_ITE = 5000
COTA = 10e-06

ite = 0
E_ant = 0
E = 1
while ((ite<MAX_ITE) and (np.abs(E_ant - E) > COTA)):
    E_ant=E
    sumaError = 0

    for p in range(len(X)):
        salida = w1 * X[p] + w0
        Error = Y[p]-salida
        grad_w0 = -2*Error
        grad_w1 = -2*Error*X[p]
        w0 = w0 - alfa * grad_w0
        w1 = w1 - alfa * grad_w1
        sumaError = sumaError + Error**2

    E = sumaError / len(X)
    ite = ite + 1
print ("ite= %d      w0= %8.5f      w1=%8.5f      E=% .8f" % (ite,w0,w1,E))

```

Calculamos la salida del combinador lineal y evaluamos el error cometido

```

import numpy as np
X = [2, 1, -1] # (2,3), (1,1), (-1,-3)
Y = [3, 1, -3]
[w0, w1] = np.random.uniform(-50, 50, size=2)

alfa = 0.1
MAX_ITE = 5000
COTA = 10e-06

ite = 0
E_ant = 0
E = 1
while ((ite<MAX_ITE) and (np.abs(E_ant - E) > COTA)):
    E_ant=E
    sumaError = 0

    for p in range(len(X)):
        salida = w1 * X[p] + w0
        Error = Y[p]-salida
        grad_w0 = -2*Error
        grad_w1 = -2*Error*X[p]
        w0 = w0 - alfa * grad_w0
        w1 = w1 - alfa * grad_w1
        sumaError = sumaError + Error**2

    E = sumaError / len(X)
    ite = ite + 1
print ("ite= %d      w0= %8.5f      w1=%8.5f      E=% .8f" % (ite,w0,w1,E))

```

Calculamos el vector gradiente

```

import numpy as np
X = [2, 1, -1] # (2,3), (1,1), (-1,-3)
Y = [3, 1, -3]
[w0, w1] = np.random.uniform(-50, 50, size=2)

alfa = 0.1
MAX_ITE = 5000
COTA = 10e-06

ite = 0
E_ant = 0
E = 1
while ((ite<MAX_ITE) and (np.abs(E_ant - E) > COTA)):
    E_ant=E
    sumaError = 0

    for p in range(len(X)):
        salida = w1 * X[p] + w0
        Error = Y[p]-salida

        grad_w0 = -2*Error
        grad_w1 = -2*Error*X[p]
        w0 = w0 - alfa * grad_w0
        w1 = w1 - alfa * grad_w1 ] ]
        sumaError = sumaError + Error**2

    E = sumaError / len(X)
    ite = ite + 1
    print ("ite= %d      w0= %8.5f      w1=%8.5f      E=% .8f" % (ite,w0,w1,E))

```

Actualizamos los pesos en la dirección del gradiente negativo

```

import numpy as np
X = [2, 1, -1] # (2,3), (1,1), (-1,-3)
Y = [3, 1, -3]
[w0, w1] = np.random.uniform(-50, 50, size=2)

alfa = 0.1
MAX ITE = 5000
COTA = 10e-06

ite = 0
E_ant = 0
E = 1
while ((ite<MAX ITE) and (np.abs(E_ant - E) > COTA)):
    E_ant=E
    sumaError = 0

    for p in range(len(X)):
        salida = w1 * X[p] + w0
        Error = Y[p]-salida

        grad_w0 = -2*Error
        grad_w1 = -2*Error*X[p]

        w0 = w0 - alfa * grad_w0
        w1 = w1 - alfa * grad_w1

        sumaError = sumaError + Error**2

    E = sumaError / len(X)
    ite = ite + 1
print ("ite= %d      w0= %8.5f      w1=%8.5f      E=% .8f" % (ite,w0,w1,E))

```

Acumulamos el cuadrado de los errores cometidos



```

import numpy as np
X = [2, 1, -1] # (2,3), (1,1), (-1,-3)
Y = [3, 1, -3]
[w0, w1] = np.random.uniform(-50, 50, size=2)

alfa = 0.1
MAX_ITE = 5000
COTA = 10e-06

ite = 0
E_ant = 0
E = 1
while ((ite<MAX_ITE) and (np.abs(E_ant - E) > COTA)):
    E_ant=E
    sumaError = 0

    for p in range(len(X)):
        salida = w1 * X[p] + w0
        Error = Y[p]-salida

        grad_w0 = -2*Error
        grad_w1 = -2*Error*X[p]

        w0 = w0 - alfa * grad_w0
        w1 = w1 - alfa * grad_w1

        sumaError = sumaError + Error**2

    E = sumaError / len(X) ← Dividimos por la cantidad de ejemplos para obtener el ECM
    ite = ite + 1
    print ("ite= %d      w0= %8.5f      w1=%8.5f      E=% .8f" % (ite,w0,w1,E))

```

```

import numpy as np

X = [2, 1, -1] # (2,3), (1,1), (-1,-3)
Y = [3, 1, -3]

[w0, w1] = np.random.uniform(-50, 50, size=2)

alfa = 0.1
MAX_ITE = 5000
COTA = 10e-06

ite = 0
E_ant = 0
E = 1

while ((ite<MAX_ITE) and (np.abs(E_ant - E) > COTA)):
    E_ant=E
    sumaError = 0

    for p in range(len(X)):
        salida = w1 * X[p] + w0
        Error = Y[p]-salida

        grad_w0 = -2*Error
        grad_w1 = -2*Error*X[p]

        w0 = w0 - alfa * grad_w0
        w1 = w1 - alfa * grad_w1

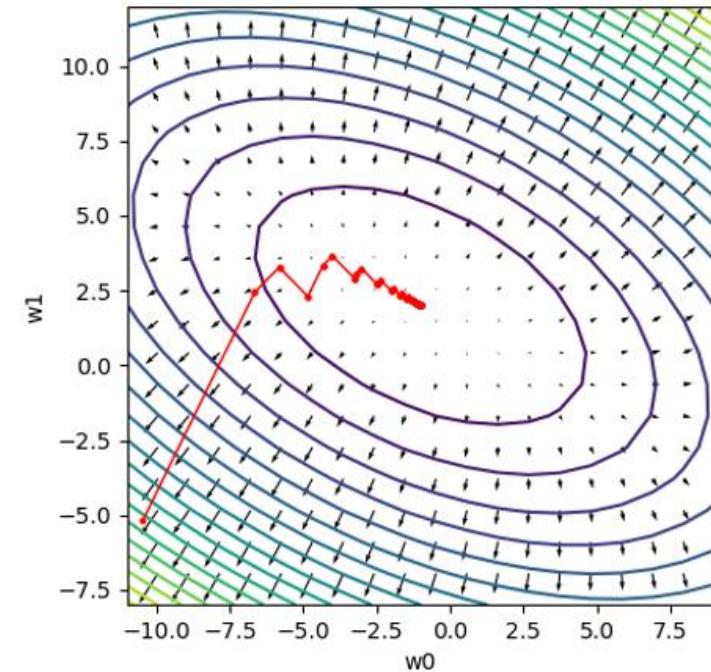
        sumaError = sumaError + Error**2

    E = sumaError / len(X)
    ite = ite + 1

print ("ite= %d      w0= %8.5f      w1=%8.5f      E=% .8f" % (ite,w0,w1,E))

```

gradienteFuncion4_CombinadorLineal.py



Note que el vector W se actualiza con cada ejemplo que ingresa a la red

ClassNeuronaLineal.py

```
nl = NeuronaLineal(alpha=0.01, n_iter=50, cotaE=10E-07,  
                    random_state=None, draw=0, title=['X1','X2'])
```

□ Parámetros de entrada

- **alpha**: valor en el intervalo (0, 1] que representa la velocidad de aprendizaje.
- **n_iter**: máxima cantidad de iteraciones a realizar.
- **cotaE**: termina si la diferencia entre dos errores consecutivos es menor que este valor.
- **random_state**: None si los pesos se inicializan en forma aleatoria, un valor entero para fijar la semilla
- **draw**: valor distinto de 0 si se desea ver el gráfico y 0 si no. Sólo si es 2D.
- **title**: lista con los nombres de los ejes para el gráfico. Se usa sólo si **draw** no es cero.

ClassNeuronaLineal.py

```
nl = NeuronaLineal(alpha=0.01, n_iter=50)  
nl.fit(X, T)
```

□ Parámetros de entrada

- **X** : arreglo de NxM donde N es la cantidad de ejemplos y M la cantidad de atributos.
- **T** : arreglo de N elementos siendo N la cantidad de ejemplos

□ Retorna

- **w_** : arreglo de M elementos siendo M la cantidad de atributos de entrada
- **b_** : valor numérico continuo correspondiente al bias.
- **errors_**: errores cometidos en cada iteración.

ClassNeuronaLineal.py

Y = nl.predict(X)

□ Parámetros de entrada

- **X** : arreglo de NxM donde N es la cantidad de ejemplos y M la cantidad de atributos.

□ Retorna: un arreglo con el resultado de aplicar el combinador lineal entrenado previamente con fit() a la matriz de ejemplos X.

- **Y** : arreglo de N elementos siendo N la cantidad de ejemplos

Descenso de gradiente estocástico

- Se busca minimizar el ECM para el ejemplo actual

$$E = (y - \hat{y})^2 = (y - (w_1 \cdot x_1 + w_2 \cdot x_2 + b))^2$$

- Debemos calcular el gradiente para saber cómo modificar los pesos

$$\frac{\partial E}{\partial w_1} = -2(y - \hat{y}) \frac{\partial(w_1 \cdot x_1 + w_2 \cdot x_2 + b)}{\partial w_1} = -2(y - \hat{y})x_1$$

$$\frac{\partial E}{\partial w_2} = -2(y - \hat{y}) \frac{\partial(w_1 \cdot x_1 + w_2 \cdot x_2 + b)}{\partial w_2} = -2(y - \hat{y})x_2$$

$$\frac{\partial E}{\partial b} = -2(y - \hat{y}) \frac{\partial(w_1 \cdot x_1 + w_2 \cdot x_2 + b)}{\partial b} = -2(y - \hat{y})$$

- La forma gral. es

$$\frac{\partial E}{\partial w_i} = -2(y - \hat{y})x_i$$

Descenso de gradiente estocástico

- Se busca minimizar el ECM para el ejemplo actual

$$E = (y - \hat{y})^2 = (y - (w_1 \cdot x_1 + w_2 \cdot x_2 + b))^2$$

- Debemos calcular el gradiente para saber cómo modificar los pesos
- Luego

$$w_1 = w_1 - \alpha \frac{\partial E}{\partial w_1}$$

- La forma gral. es

$$\frac{\partial E}{\partial w_i} = -2(y - \hat{y})x_i$$

$$w_1 = w_1 + 2\alpha \cdot (y - \hat{y}) \cdot x_1 = -8.50 + 2\alpha * 94.76 * 14.96 = -8.47$$



Descenso de gradiente estocástico

- Se busca minimizar el ECM para el ejemplo actual

$$E = (y - \hat{y})^2 = (y - (w_1 \cdot x_1 + w_2 \cdot x_2 + b))^2$$

- Debemos calcular el gradiente para saber cómo modificar los pesos
- Luego

$$w_1 = w_1 - \alpha \frac{\partial E}{\partial w_1}$$

$$w_1 = -8.47$$

$$w_2 = w_2 - \alpha \frac{\partial E}{\partial w_2}$$

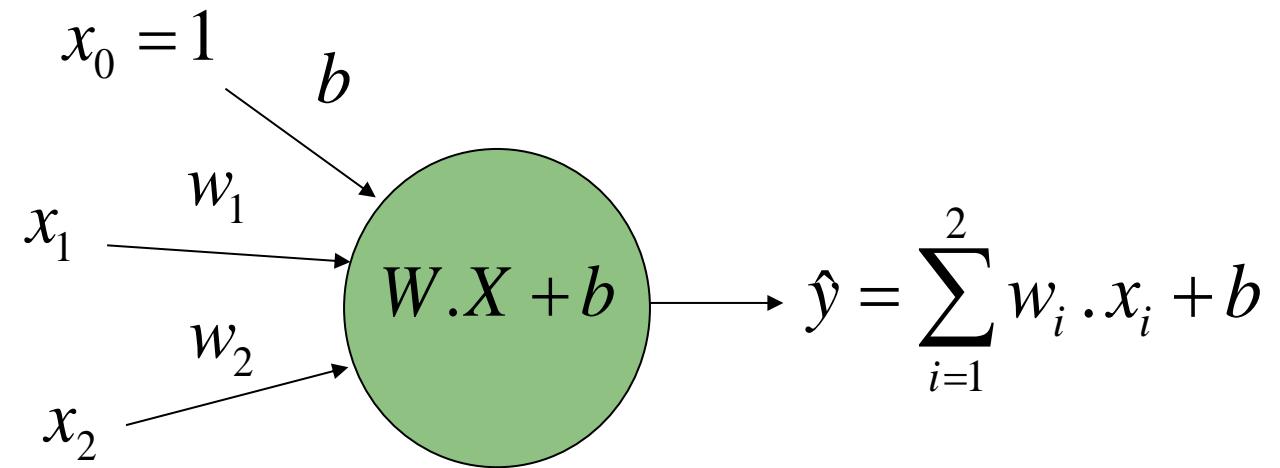
$$w_2 = 7.76$$

$$b = b - \alpha \frac{\partial E}{\partial b}$$

$$b = 174.80$$

Predicción de la Producción de Energía (PE)

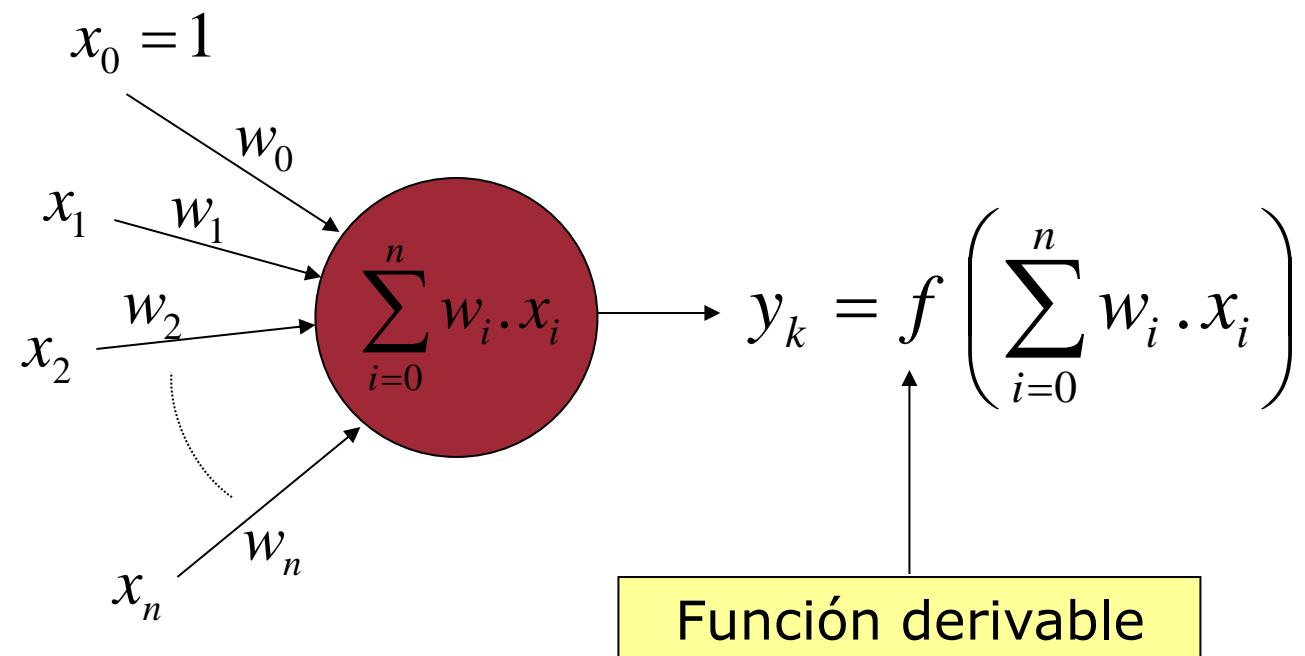
X	Y
[14.96, 41.76],	[463.26,
[25.18, 62.96],	444.37,
[5.11, 39.4],	488.56,
...,	...,
[31.32, 74.33],	429.57,
[24.48, 69.45],	435.74,
[21.6 , 62.52]]	453.28]



$$\underbrace{\begin{pmatrix} -8.50 & 7.68 \end{pmatrix}}_w * \underbrace{\begin{pmatrix} 14.96 \\ 41.73 \end{pmatrix}}_b + 174.80 \rightarrow \hat{y} = 368.49$$
$$y = 463.26$$

Error cometido en este ejemplo → $Error = (y - \hat{y}) = 94.76$

Neurona General



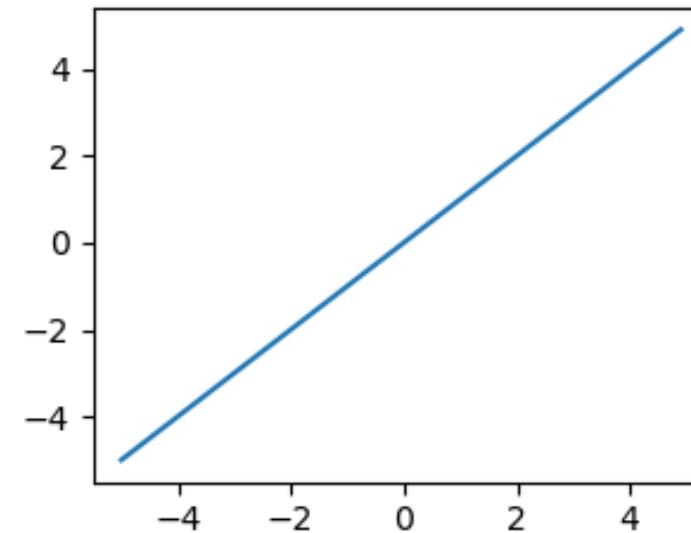
Función de Salida LINEAL

$$f(x) = x \quad f'(x) = 1$$

desde Python

```
import numpy as np
import grafica as gr
from matplotlib import pyplot as plt

x = np.array(range(-50,50,1))/10.0
y = gr.evaluar('purelin', x)
plt.plot(x,y, '-')
```



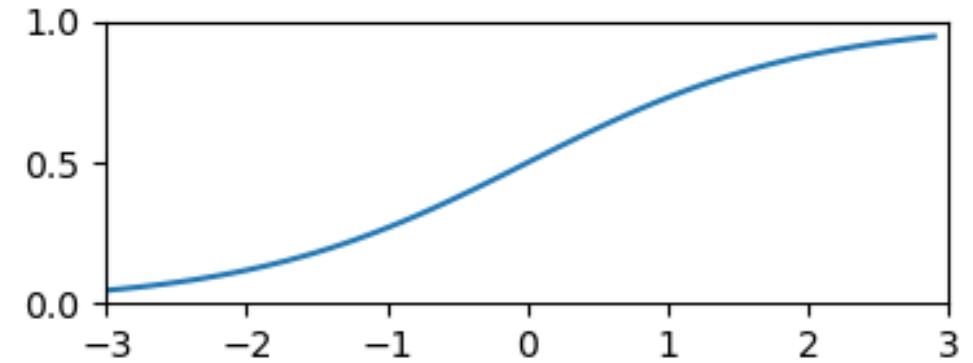
Función SIGMOIDE $\in (0,1)$

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

desde Python

```
import numpy as np
import grafica as gr
from matplotlib import pyplot as plt

x = np.array(range(-30,30,1))/10.0
y = gr.evaluar('logsig', x)
plt.plot(x,y, '-')
plt.axis([-3, 3, 0, 1])
plt.show()
```



Función SIGMOIDE $\in (-1,1)$

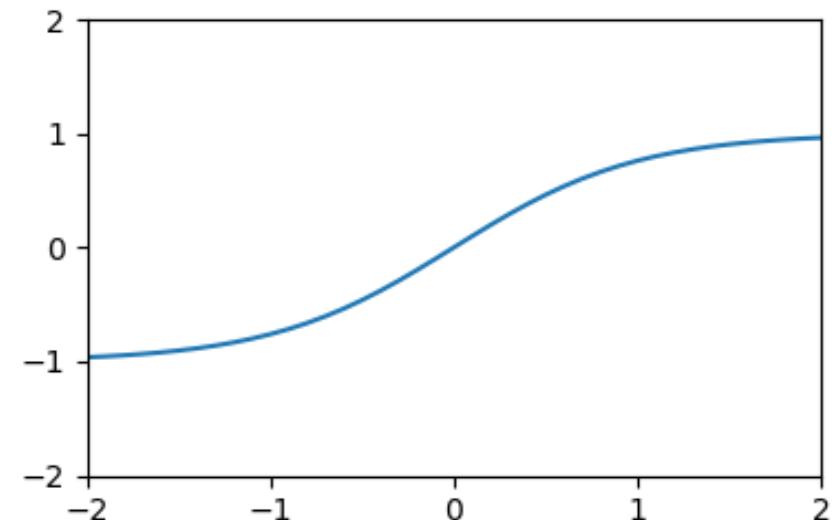
$$f(x) = \frac{2}{1+e^{-2x}} - 1$$

$$f'(x) = 1 - f(x)^2$$

desde Python

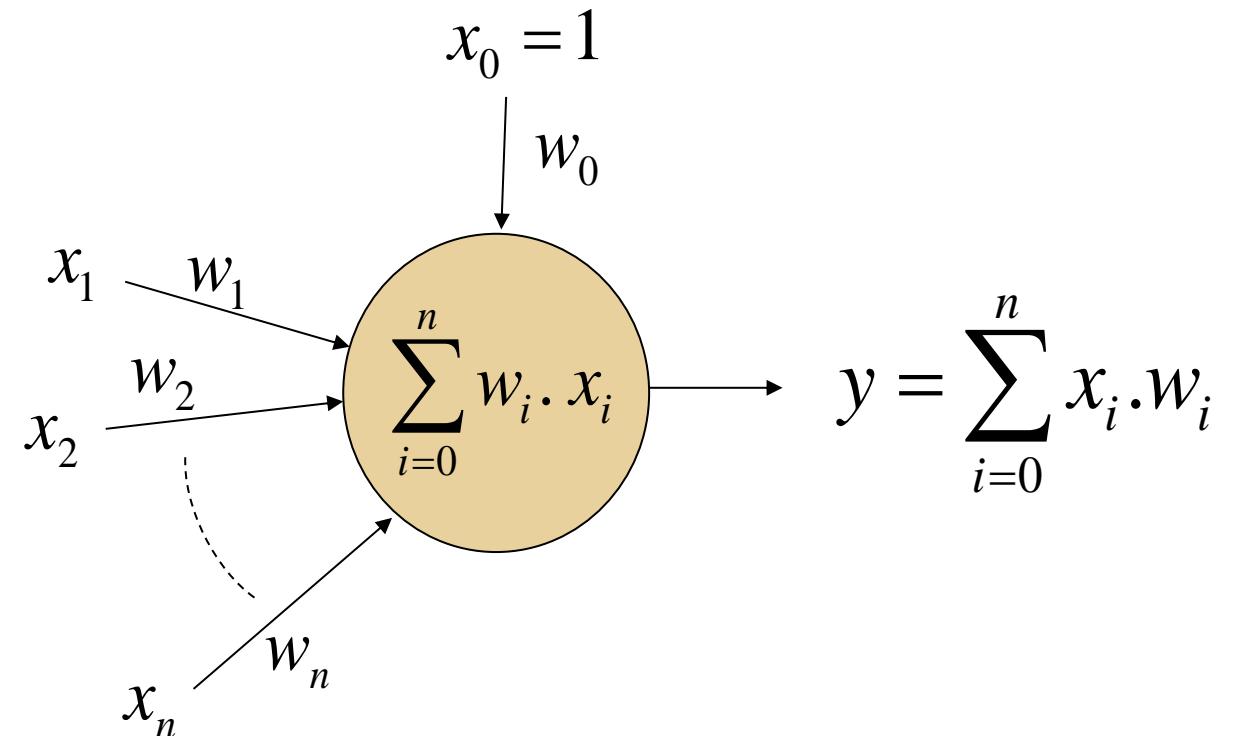
```
import numpy as np
import grafica as gr
from matplotlib import pyplot as plt

x = np.array(range(-30,30,1))/10.0
y = gr.evaluar('tansig', x)
plt.plot(x,y, '-')
plt.axis([-2, 2, -2, 2])
plt.show()
```



Combinador Lineal

- Resuelve un problema de **Regresión Lineal**
- Función de Error
 - Error cuadrático medio
- Técnica de optimización
 - Descenso de gradiente estocástico



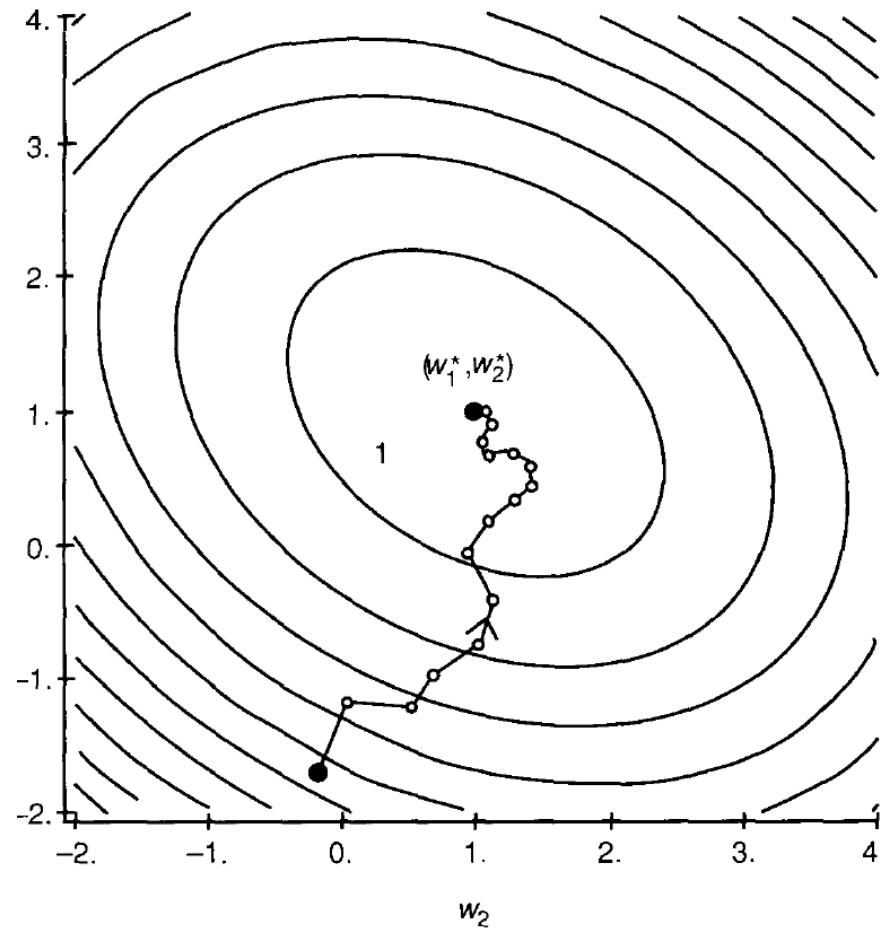
Técnica del descenso del gradiente estocástico

$$w(t+1) = w(t) + \Delta w(t)$$

$$w(t+1) = w(t) - \mu \nabla \xi(w(t))$$

□ se utiliza

$$\xi = \langle \varepsilon_k^2 \rangle \approx \varepsilon_k^2 = (d_k - \sum_{i=0}^N x_{ik} w_i)^2$$



Técnica del descenso del gradiente estocástico

$$w(t+1) = w(t) + \Delta w(t)$$

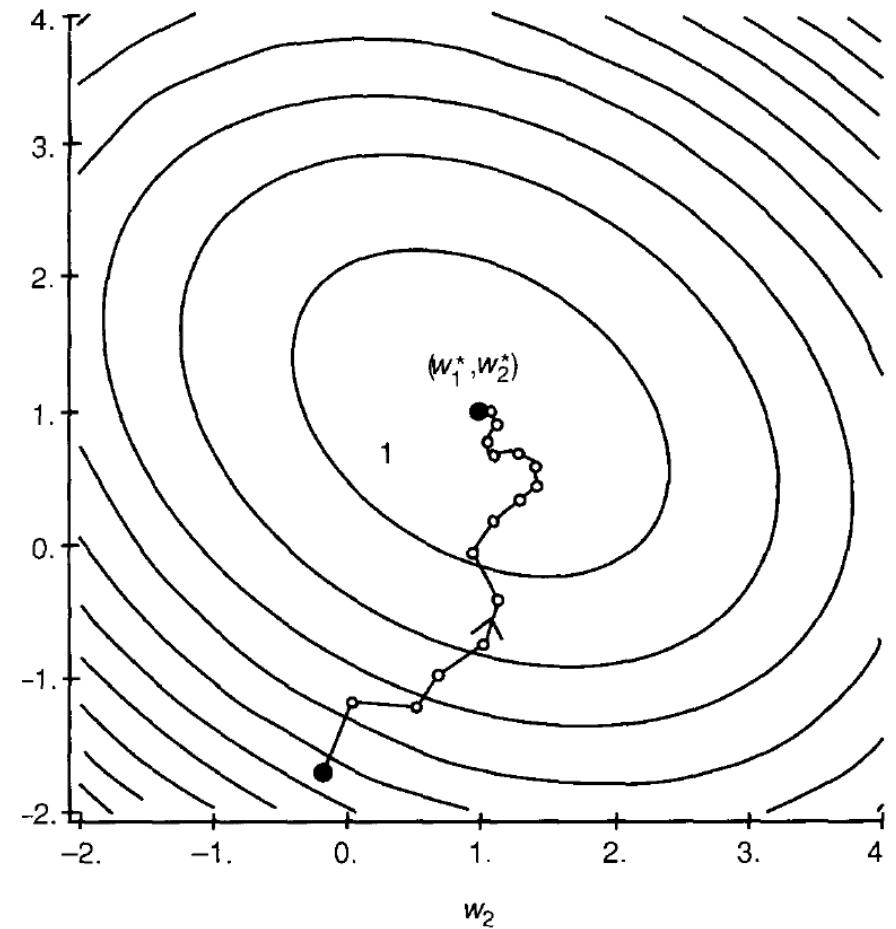
$$w(t+1) = w(t) - \mu \nabla \xi(w(t))$$

□ se utiliza

$$\xi = \langle \varepsilon_k^2 \rangle \approx \varepsilon_k^2 = (d_k - \sum_{i=0}^N x_{ik} w_i)^2$$

□ veamos que

$$\nabla \varepsilon_k^2(t) = -2 \varepsilon_k(t) x_k$$



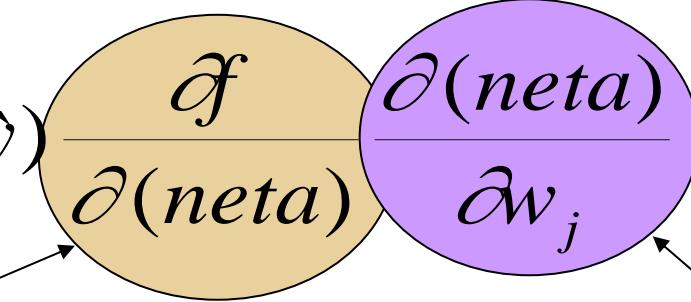
Entrenamiento de una neurona no lineal

- Seleccionar el valor de α
- Inicializar los pesos W y b con valores random
- Mientras (la variación del ECM sea mayor a la cota prefijada)
 - Para cada ejemplo
 - Ingresar el ejemplo a la red.
 - Calcular el error $\varepsilon = (y - \hat{y})$ y $\frac{\partial \varepsilon^2}{\partial w} = \frac{\partial(y-\hat{y})^2}{\partial w}$
 - Actualizar los pesos de la red

$$w_i = w_i - \alpha \frac{\partial \varepsilon^2}{\partial w_i}$$

¿Cómo sería la derivada del error si la neurona no es lineal?

$$\frac{\partial \mathcal{E}^2}{\partial w} = \left[\frac{\partial(y - \hat{y})^2}{\partial w_0}; \dots; \frac{\partial(y - \hat{y})^2}{\partial w_n} \right]$$

$$\frac{\partial \mathcal{E}^2}{\partial w_j} = -2(y - \hat{y}) \frac{\partial(neta)}{\partial w_j}$$


$$\frac{\partial(neta)}{\partial w_j} = \frac{\partial(\sum_{i=0}^n w_i x_i)}{\partial w_j} = x_j$$

$$\frac{\partial \mathcal{E}^2}{\partial w_j} = -2(y - \hat{y}) f'(neta) x_j$$

Entrenamiento de una neurona no lineal

- Seleccionar el valor de α
- Inicializar los pesos W y b con valores random
- Mientras (la variación del ECM sea mayor a la cota prefijada)
 - Para cada ejemplo
 - Ingresar el ejemplo a la red.
 - Calcular $\frac{\partial \varepsilon^2}{\partial w_i} = -2 * \varepsilon * f'(neta) * x_i$
 - Actualizar los pesos de la red

$$w_i = w_i - \alpha \frac{\partial \varepsilon}{\partial w_i} = w_i + 2\alpha * \varepsilon * f'(neta) * x_i$$

```

import numpy as np
X = np.array([[0,0], [0,1],[1,0],[1,1]])
T = np.array([0,0,0,1])

W = np.random.uniform(-0.5, 0.5,size=2)
b = np.random.uniform(-0.5, 0.5)

alfa = 0.1
MAX_ITE = 5000
COTA = 10e-06

ite = 0
E_ant = 0
E = 1

while ((ite<MAX_ITE) and (np.abs(E_ant - E) > COTA)):
    E_ant=E
    sumaError = 0

    for p in range(len(X)):
        neta = np.dot(W,X[p,:])+b

        Y = 1/(1+np.exp(-neta))
        deriv = Y * (1-Y)

        Error = T[p]-Y

        W = W + alfa * Error * deriv * X[p,:]
        b = b + alfa * Error * deriv

        sumaError = sumaError + Error**2

    E = sumaError / len(X)
    ite = ite + 1

print ("ite= %d      w0= %8.5f      w1=%8.5f      E=% .8f" % (ite,W[0],W[1],E))

```

Termina o bien porque realizó la máxima cantidad de intentos o porque el valor absoluto de la diferencia entre dos valores consecutivos de la función es inferior a cierta cota

```

import numpy as np
X = np.array([[0,0], [0,1],[1,0],[1,1]])
T = np.array([0,0,0,1])

W = np.random.uniform(-0.5, 0.5,size=2)
b = np.random.uniform(-0.5, 0.5)

alfa = 0.1
MAX_ITE = 5000
COTA = 10e-06

ite = 0
E_ant = 0
E = 1

while ((ite<MAX_ITE) and (np.abs(E_ant - E) > COTA)):
    E_ant=E
    sumaError = 0

    for p in range(len(X)):
        neta = np.dot(W,X[p,:])+b
        Y = 1/(1+np.exp(-neta))
        deriv = Y * (1-Y)

        Error = T[p]-Y

        W = W + alfa * Error * deriv * X[p,:]
        b = b + alfa * Error * deriv

        sumaError = sumaError + Error**2

    E = sumaError / len(X)
    ite = ite + 1

print ("ite= %d      w0= %8.5f      w1=%8.5f      E=% .8f" % (ite,W[0],W[1],E))

```

Calculamos la salida del combinador lineal y evaluamos el error cometido

```

import numpy as np
X = np.array([[0,0], [0,1],[1,0],[1,1]])
T = np.array([0,0,0,1])

W = np.random.uniform(-0.5, 0.5,size=2)
b = np.random.uniform(-0.5, 0.5)

alfa = 0.1
MAX_ITE = 5000
COTA = 10e-06

ite = 0
E_ant = 0
E = 1

while ((ite<MAX_ITE) and (np.abs(E_ant - E) > COTA)):
    E_ant=E
    sumaError = 0

    for p in range(len(X)):
        neta = np.dot(W,X[p,:])+b

        Y = 1/(1+np.exp(-neta))
        deriv = Y * (1-Y)
        Error = T[p]-Y
        W = W + alfa * Error * deriv * X[p,:]
        b = b + alfa * Error * deriv
        sumaError = sumaError + Error**2

    E = sumaError / len(X)
    ite = ite + 1

print ("ite= %d      w0= %8.5f      w1=%8.5f      E=% .8f" % (ite,W[0],W[1],E))

```

Son parte del vector gradiente

```

import numpy as np
X = np.array([[0,0], [0,1],[1,0],[1,1]])
T = np.array([0,0,0,1])

W = np.random.uniform(-0.5, 0.5,size=2)
b = np.random.uniform(-0.5, 0.5)

alfa = 0.1
MAX_ITE = 5000
COTA = 10e-06

ite = 0
E_ant = 0
E = 1

while ((ite<MAX_ITE) and (np.abs(E_ant - E) > COTA)):
    E_ant=E
    sumaError = 0

    for p in range(len(X)):
        neta = np.dot(W,X[p,:])+b

        Y = 1/(1+np.exp(-neta))
        deriv = Y * (1-Y)

        Error = T[p]-Y
        W = W + alfa * Error * deriv * X[p,:]
        b = b + alfa * Error * deriv
        sumaError = sumaError + Error**2

    E = sumaError / len(X)
    ite = ite + 1

print ("ite= %d      w0= %8.5f      w1=%8.5f      E=% .8f" % (ite,W[0],W[1],E))

```

Actualizamos los pesos en la dirección del gradiente negativo

```

import numpy as np
X = np.array([[0,0], [0,1],[1,0],[1,1]])
T = np.array([0,0,0,1])

W = np.random.uniform(-0.5, 0.5,size=2)
b = np.random.uniform(-0.5, 0.5)

alfa = 0.1
MAX_ITE = 5000
COTA = 10e-06

ite = 0
E_ant = 0
E = 1

while ((ite<MAX_ITE) and (np.abs(E_ant - E) > COTA)):
    E_ant=E
    sumaError = 0

    for p in range(len(X)):
        neta = np.dot(W,X[p,:])+b

        Y = 1/(1+np.exp(-neta))
        deriv = Y * (1-Y)

        Error = T[p]-Y

        W = W + alfa * Error * deriv * X[p,:]
        b = b + alfa * Error * deriv

        sumaError = sumaError + Error**2

    E = sumaError / len(X)
    ite = ite + 1

print ("ite= %d      w0= %8.5f      w1=%8.5f      E=% .8f" % (ite,W[0],W[1],E))

```

Acumulamos el cuadrado de los errores cometidos

```

import numpy as np
X = np.array([[0,0], [0,1],[1,0],[1,1]])
T = np.array([0,0,0,1])

W = np.random.uniform(-0.5, 0.5,size=2)
b = np.random.uniform(-0.5, 0.5)

alfa = 0.1
MAX_ITE = 5000
COTA = 10e-06

ite = 0
E_ant = 0
E = 1

while ((ite<MAX_ITE) and (np.abs(E_ant - E) > COTA)):
    E_ant=E
    sumaError = 0

    for p in range(len(X)):
        neta = np.dot(W,X[p,:])+b
        Y = 1/(1+np.exp(-neta))
        deriv = Y * (1-Y)

        Error = T[p]-Y

        W = W + alfa * Error * deriv * X[p,:]
        b = b + alfa * Error * deriv

        sumaError = sumaError + Error**2

    E = sumaError / len(X) ← Dividimos por la cantidad de ejemplos para obtener el ECM
    ite = ite + 1

print ("ite= %d      w0= %8.5f      w1=%8.5f      E=% .8f" % (ite,W[0],W[1],E))

```

ClassNeuronaGral.py

```
nn = NeuronaGradiente(alpha=0.01, n_iter=50, cotaE=10E-07, FUN='sigmoid',
random_state=None, draw=0, title=['X1','X2'])
```

□ Parámetros de entrada

- **alpha**: valor en el intervalo (0, 1] que representa la velocidad de aprendizaje.
- **n_iter**: máxima cantidad de iteraciones a realizar.
- **cotaE**: termina si la diferencia entre dos errores consecutivos es menor que este valor.
- **FUN**: función de activación – ‘sigmoid’, ‘tanh’, ‘purelin’.
- **random_state**: None si los pesos se inicializan en forma aleatoria, un valor entero para fijar la semilla
- **draw**: valor distinto de 0 si se desea ver el gráfico y 0 si no. Sólo si es 2D.
- **title**: lista con los nombres de los ejes para el gráfico. Se usa sólo si **draw** no es cero.

ClassNeuronaGral.py

```
nn = NeuronaGradiente(alpha=0.01, n_iter=50, cotaE=10E-07, FUN='sigmoid',
random_state=None, draw=0, title=['X1','X2'])

nn.fit(X, T)
```

□ Parámetros de entrada

- **X** : arreglo de NxM donde N es la cantidad de ejemplos y M la cantidad de atributos.
- **T** : arreglo de N elementos siendo N la cantidad de ejemplos

□ Retorna

- **w_** : arreglo de M elementos siendo M la cantidad de atributos de entrada
- **b_** : valor numérico continuo correspondiente al bias.
- **errors_**: errores cometidos en cada iteración.

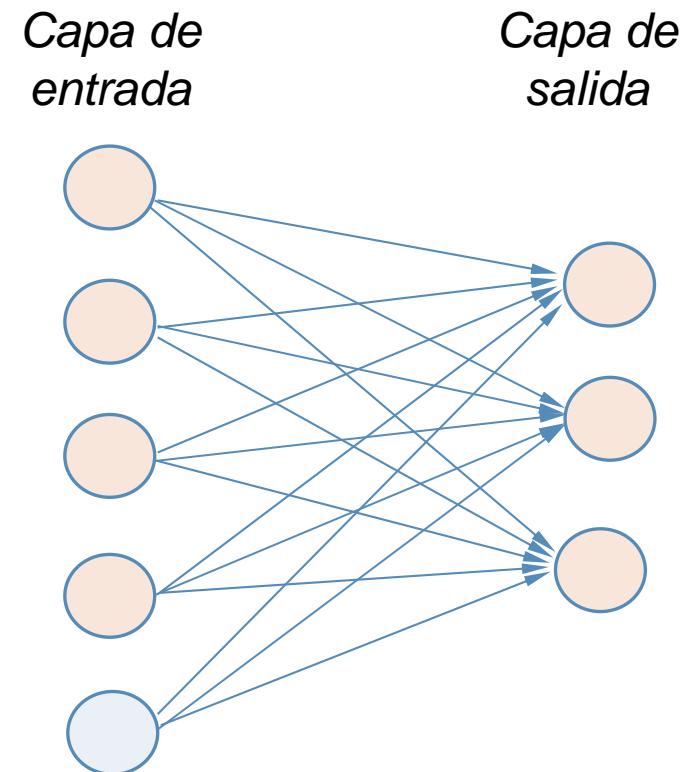
ClassNeuronaGral.py

Y = nn.predict(X)

- Parámetros de entrada
 - X : arreglo de NxM donde N es la cantidad de ejemplos y M la cantidad de atributos.
- Retorna: un arreglo con el resultado de aplicar la neurona general entrenada previamente con fit() a la matriz de ejemplos X.
 - Y : arreglo de N elementos siendo N la cantidad de ejemplos

Clasificación con más de 2 clases

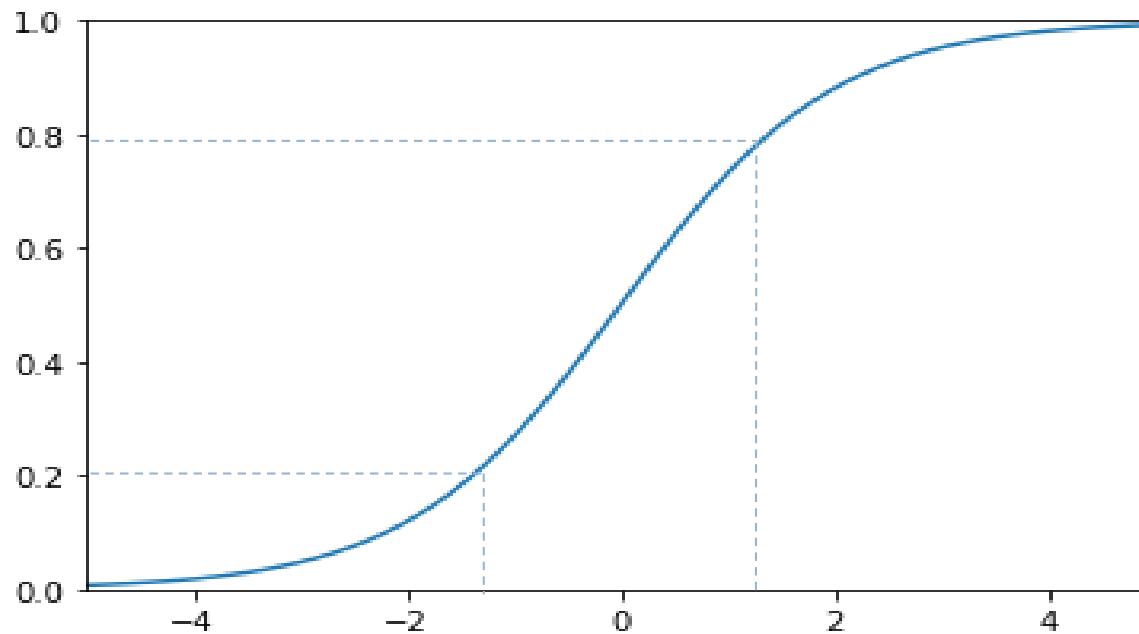
- Pueden utilizarse varias neuronas no lineales para resolver un problema de clasificación con más de 2 clases.
- Cada neurona de la capa de salida buscará responder por un valor de clase distinto.
- El error de la capa será la suma de los errores de las neuronas que la forman.



Función sigmoid()

X	$f(X)$
-5.00	0.01
-4.00	0.02
-3.00	0.05
-2.00	0.12
-1.39	0.20
-1.00	0.27
0.00	0.50
1.00	0.73
1.39	0.80
2.00	0.88
3.00	0.95
4.00	0.98
5.00	0.99

$$f(x) = \frac{1}{1 + \exp(-x)}$$



Funciones de costo

- Error cuadrático medio

$$C = \frac{1}{n} \sum_n (t - y)^2 = \frac{1}{n} \sum_n (t - f(\text{neta}))^2$$

- Entropía cruzada binaria

$$C = -\frac{1}{n} \sum_n [t \ln y + (1 - t) \ln(1 - y)]$$

Entropía cruzada binaria

- Es una función de costo que puede usarse con neuronas con función de activación sigmoide entre 0 y 1

$$C = -\frac{1}{n} \sum_n [t \ln y + (1 - t) \ln(1 - y)]$$

donde

- t es el valor binario esperado
- $y = 1/(1 + e^{-\sum x_i w_i})$ es la salida de la neurona

- Ver que es una función de costo
 - $C > 0$
 - C tiende a 0 (cero) a medida que la neurona aprende la salida deseada.

Derivada de la entropía cruzada binaria

$$C = -\frac{1}{n} \sum_n [t \ln y + (1 - t) \ln(1 - y)]$$

$$\frac{\partial C}{\partial w_j} = -\frac{1}{n} \sum_n \left(\frac{t}{y} - \frac{1-t}{1-y} \right) \frac{\partial y}{\partial w_j}$$

f(neta)

$$\frac{\partial C}{\partial w_j} = -\frac{1}{n} \sum_n \left(\frac{t}{f(neta)} - \frac{1-t}{1-f(neta)} \right) \frac{\partial f(neta)}{\partial w_j}$$

Derivada de la entropía cruzada binaria

$$C = -\frac{1}{n} \sum_n [t \ln y + (1 - t) \ln(1 - y)]$$

$$\frac{\partial C}{\partial w_j} = -\frac{1}{n} \sum_n \left(\frac{t}{f(neta)} - \frac{1 - t}{1 - f(neta)} \right) \frac{\partial f(neta)}{\partial w_j}$$

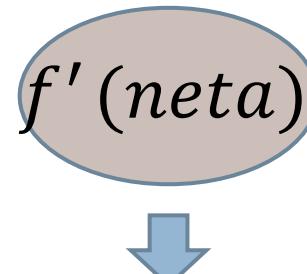


$$f'(neta)x_j$$

$$\frac{t - f(neta)}{f(neta)(1 - f(neta))}$$

Derivada de la entropía cruzada binaria

$$C = -\frac{1}{n} \sum_n [t \ln y + (1 - t) \ln(1 - y)]$$

$$\frac{\partial C}{\partial w_j} = -\frac{1}{n} \sum_n \left(\frac{t - f(\text{neta})}{f(\text{neta})(1 - f(\text{neta}))} \right) f'(\text{neta}) x_j$$


Si $f(\text{neta}) = \frac{1}{1+e^{-\text{neta}}}$, $f'(\text{neta}) = f(\text{neta})(1 - f(\text{neta}))$

Derivada de la entropía cruzada binaria

$$C = -\frac{1}{n} \sum_n [t \ln y + (1 - t) \ln(1 - y)]$$

$$\frac{\partial C}{\partial w_j} = -\frac{1}{n} \sum_n \left(\frac{t - f(\text{neta})}{f(\text{neta})(1 - f(\text{neta}))} \right) f'(\text{neta}) x_j$$

$$\frac{\partial C}{\partial w_j} = -\frac{1}{n} \sum_n (t - f(\text{neta})) x_j$$

Derivada de la entropía cruzada binaria

$$C = -\frac{1}{n} \sum_n [t \ln y + (1 - t) \ln(1 - y)]$$

$$\frac{\partial C}{\partial w_j} = -\frac{1}{n} \sum_n \left(\frac{t - f(\text{neta})}{f(\text{neta})(1 - f(\text{neta}))} \right) f'(\text{neta}) x_j$$

$$\frac{\partial C}{\partial w_j} = -\frac{1}{n} \sum_n (t - y) x_j$$

Derivada de la entropía cruzada binaria

$$C = -\frac{1}{n} \sum_n [t \ln y + (1 - t) \ln(1 - y)]$$

$$\frac{\partial C}{\partial w_j} = -\frac{1}{n} \sum_n (t - y) x_j$$

- Si utilizamos descenso de gradiente estocástico sólo se mide el error cometido en el ejemplo k

***Gradiente de la función de costo ECM
calculado sobre el k-ésimo ejemplo***

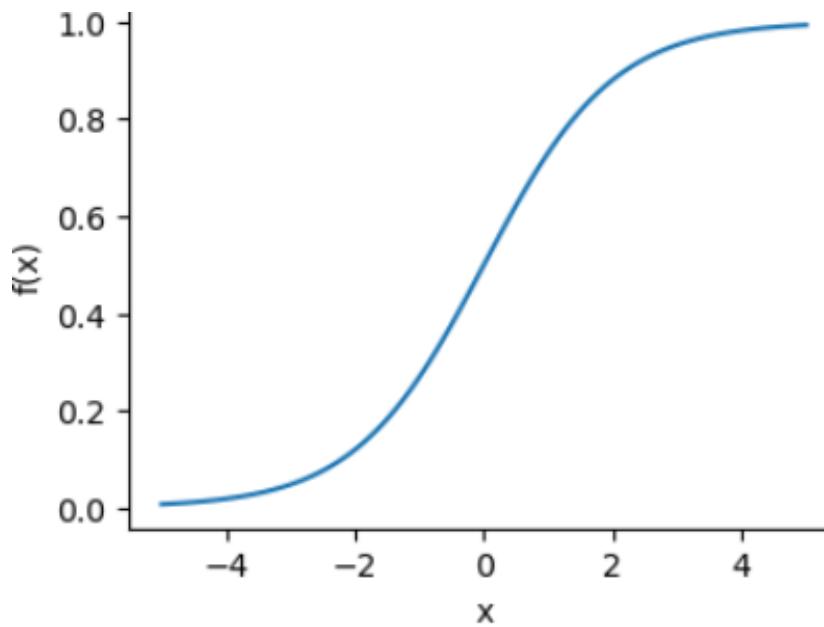
$$\frac{\partial C_k}{\partial w_j} = -(t_k - y_k) x_j$$

$$\frac{\partial ECM_k}{\partial w_j} = -(t_k - y_k) f'(neta) x_j$$

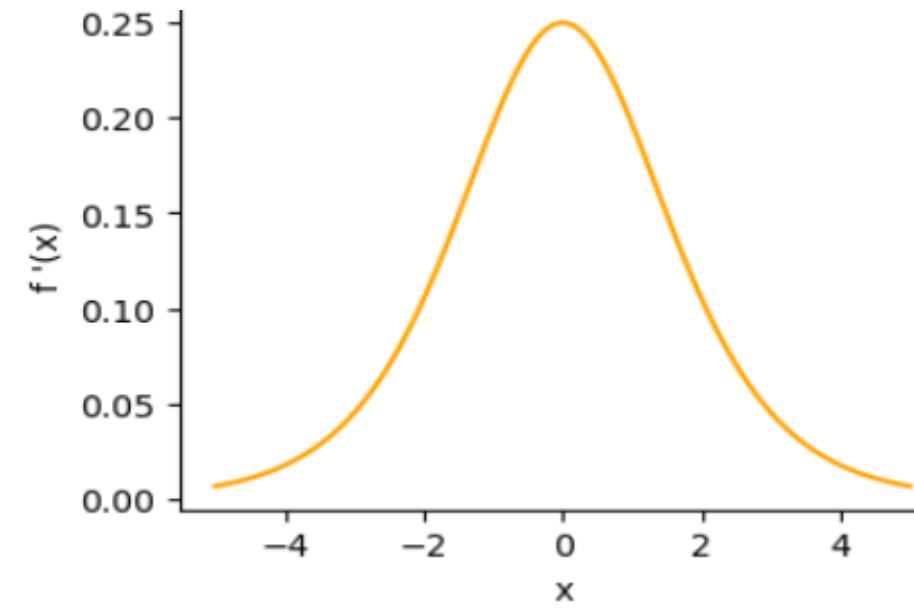
¿Qué valores toma $f'(neta)$ cuando se trata de la función sigmoide entre 0 y 1?

Función sigmoide

$$f(x) = \frac{1}{1 + \exp(-x)}$$

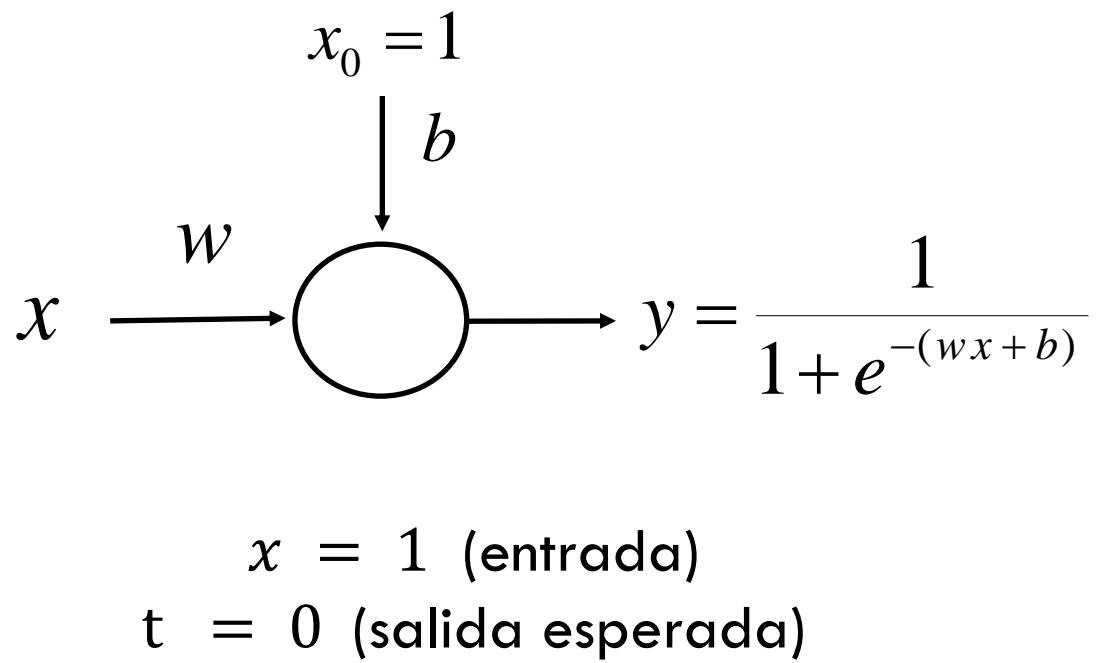


$$f'(x) = f(x) * (1 - f(x))$$



Ejemplo: Entrene una neurona con función de activación sigmoide entre 0 y 1 para que reciba un 1 y responda 0

- Usando como Función de Costo el **Error Cuadrático Medio (ECM)**



Función de costo (para 1 ejemplo)

$$C = \frac{(t - y)^2}{2}$$

$$\frac{\partial C}{\partial w} = -(t - y) [y(1 - y)] x$$

$f'(neta)$

```
x = 1  
T = 0
```

```
W = 0.6  
b = 0.9
```

```
MAX_ITE, alfa = 2000, 0.25
```

```
ite = 0
```

```
C_ant, Costo = 0, 1
```

```
while (ite<MAX_ITE) and (np.abs(C_ant-Costo)>10e-05):
```

```
    C_ant = Costo
```

```
    neta = W * X + b
```

```
    y = 1.0/(1+np.exp(-neta))
```

```
    Error = T - y
```

```
    Costo = (Error**2)/2
```

```
    gradiente_W = - Error * (y * (1-y)) * X
```

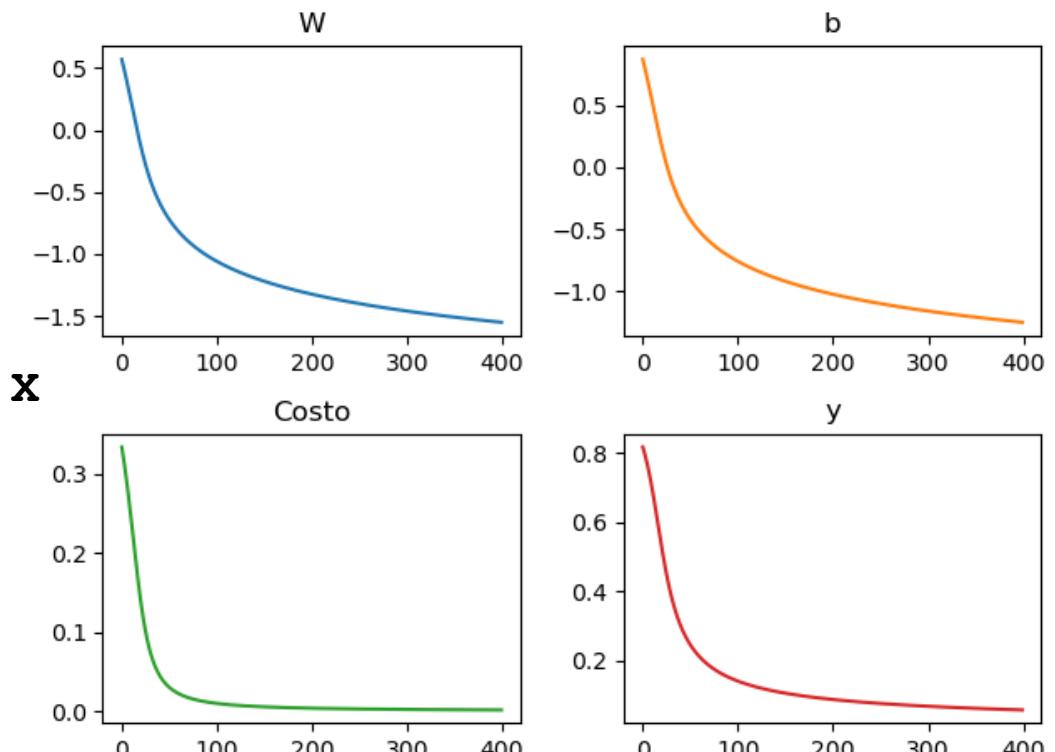
```
    gradiente_b = - Error * (y * (1-y))
```

```
    W = W - alfa * gradiente_W
```

```
    b = b - alfa * gradiente_b
```

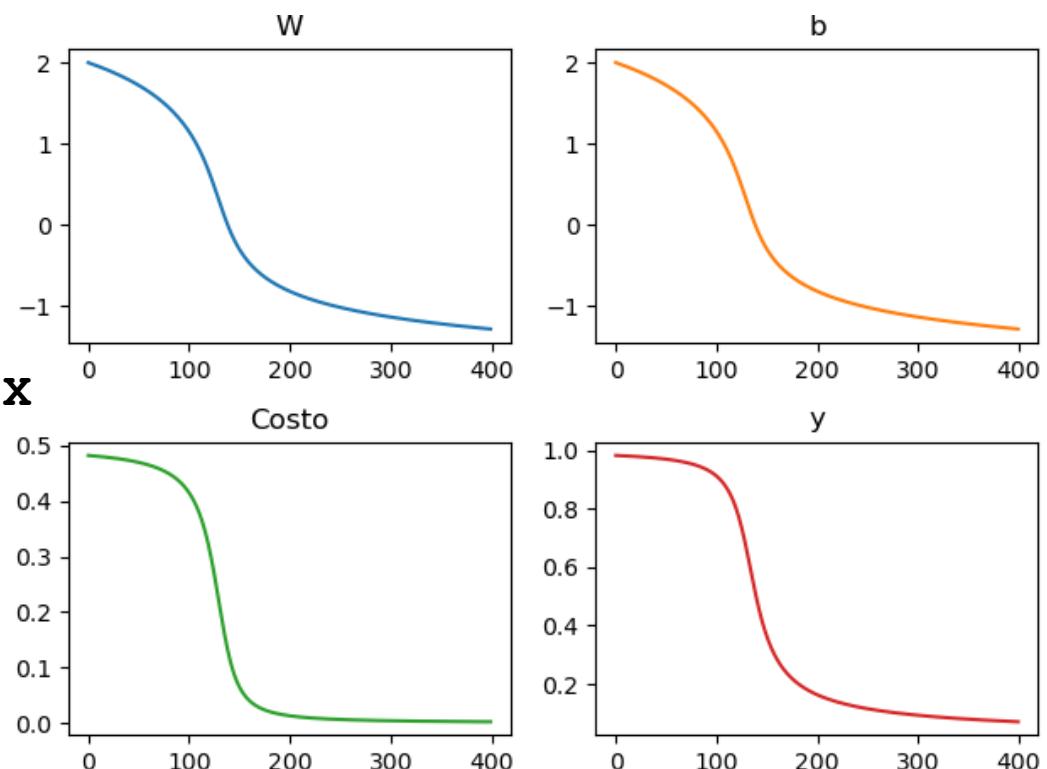
```
    ite = ite + 1
```

NeuronaGral_1Ej.ipynb



```
x = 1
T = 0
W = 2
b = 2
```

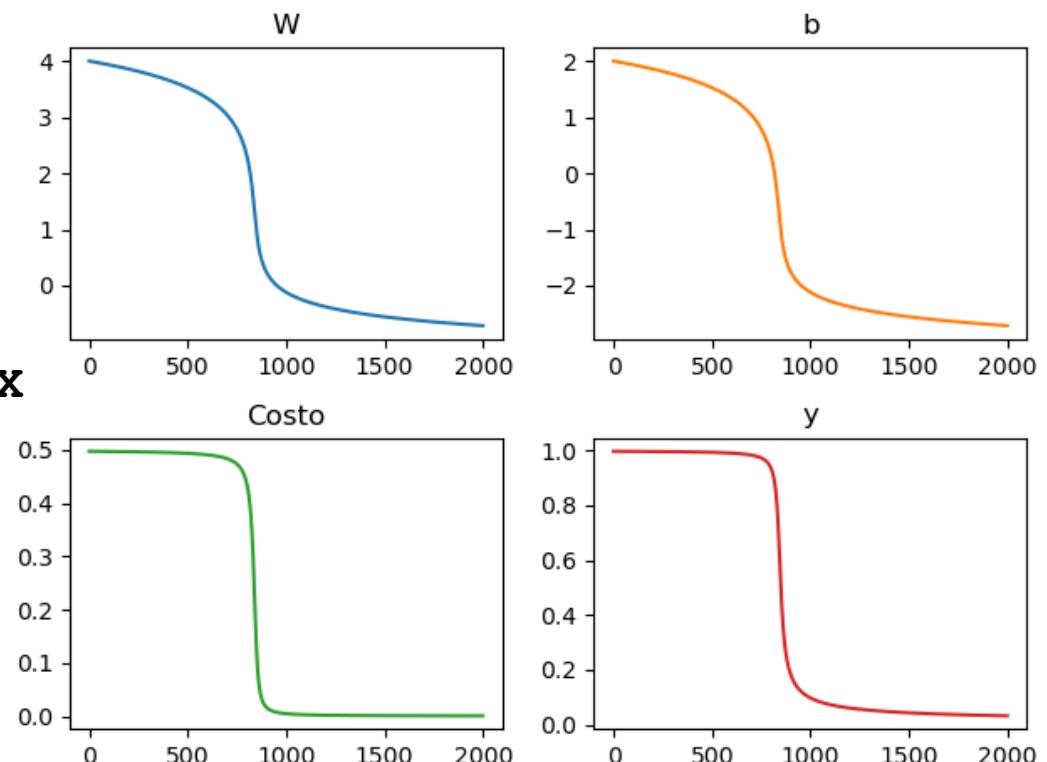
```
MAX_ITE, alfa = 2000, 0.25
ite = 0
C_ant, Costo = 0, 1
while (ite<MAX_ITE) and (np.abs(C_ant-Costo)>10e-05):
    C_ant = Costo
    neta = W * X + b
    y = 1.0/(1+np.exp(-neta))
    Error = T - y
    Costo = (Error**2)/2
    gradiente_W = - Error * (y * (1-y)) * X
    gradiente_b = - Error * (y * (1-y))
    W = W - alfa * gradiente_W
    b = b - alfa * gradiente_b
    ite = ite + 1
```



```
x = 1  
T = 0  
W = 4  
b = 2
```

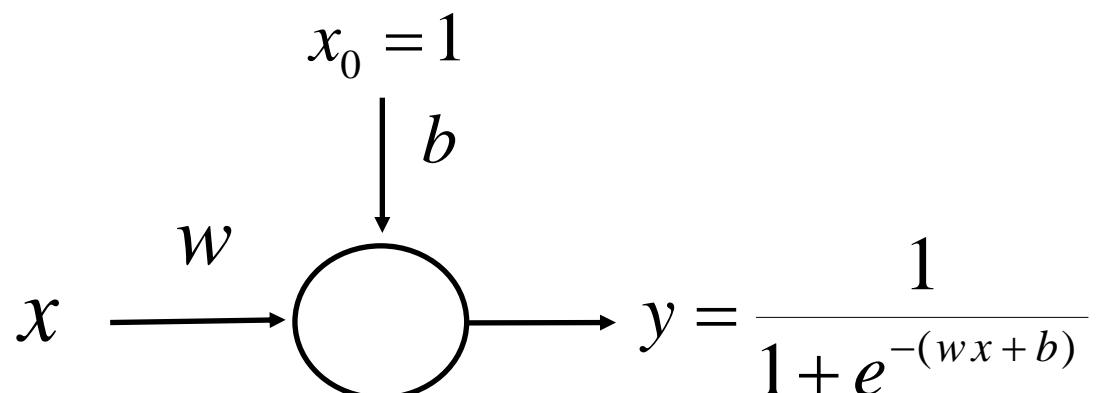
NeuronaGral_1Ej.ipynb

```
MAX_ITE, alfa = 2000, 0.25  
ite = 0  
C_ant, Costo = 0, 1  
while (ite<MAX_ITE) and (np.abs(C_ant-Costo)>10e-05):  
    C_ant = Costo  
  
    neta = W * x + b  
    y = 1.0/(1+np.exp(-neta))  
    Error = T - y  
    Costo = (Error**2)/2  
  
    gradiente_W = - Error * (y * (1-y)) * x  
    gradiente_b = - Error * (y * (1-y))  
  
    W = W - alfa * gradiente_W  
    b = b - alfa * gradiente_b  
  
    ite = ite + 1
```



Ejemplo: Entrene una neurona con función de activación sigmoide entre 0 y 1 para que reciba un 1 y responda 0

□ Función de Costo: **Entropía Cruzada Binaria (EC_binaria)**



$$x = 1 \text{ (entrada)}$$
$$t = 0 \text{ (salida esperada)}$$

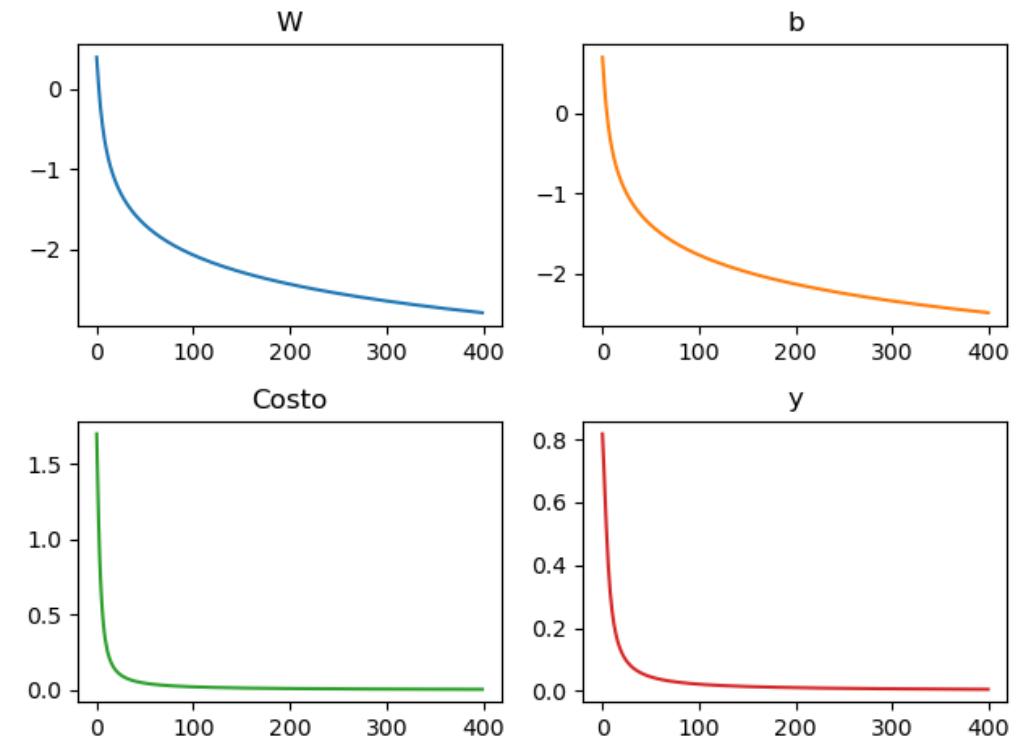
Función de costo (para 1 ejemplo)

$$C = -(t \ln y + (1 - t) \ln(1 - y))$$

$$\frac{\partial C}{\partial w} = -(t - y) x$$

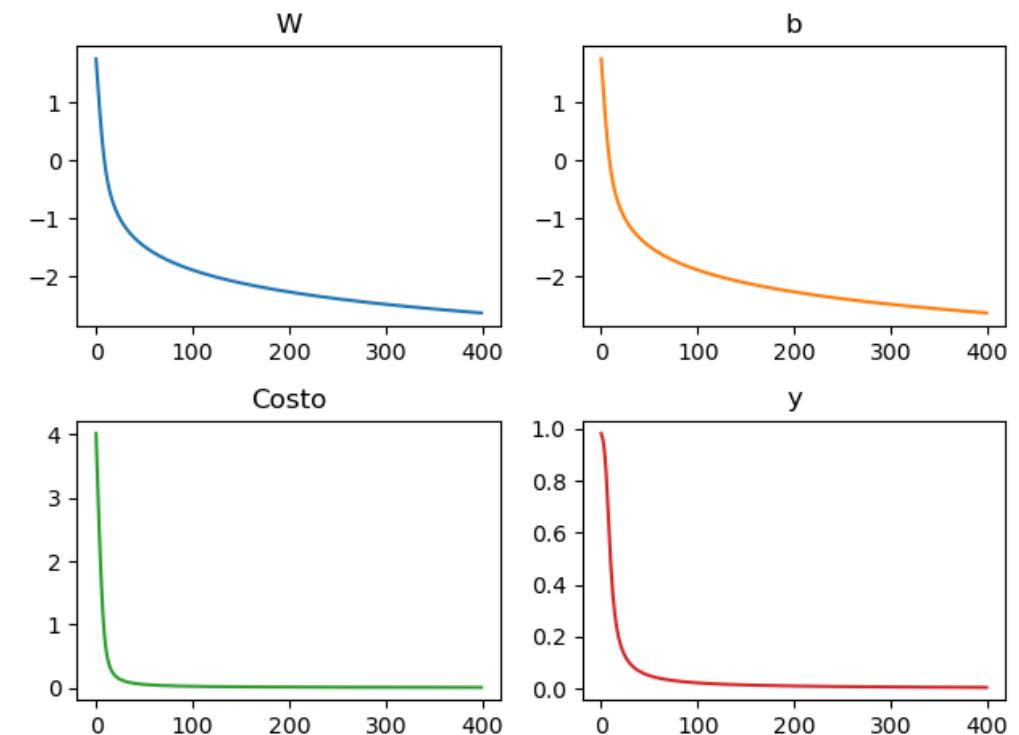
```
x = 1
T = 0
W = 0.6
b = 0.9
```

```
MAX_ITE, alfa = 2000, 0.25
ite = 0
C_ant, Costo = 0, 1
while (ite<MAX_ITE) and (np.abs(C_ant-Costo)>10e-05):
    C_ant = Costo
    neta = W * X + b
    y = 1.0/(1+np.exp(-neta))
    Error = T - y
    Costo = -(T*np.log(y)+(1-T)*np.log(1-y))
    gradiente_W = - Error * X
    gradiente_b = - Error
    W = W - alfa * gradiente_W
    b = b - alfa * gradiente_b
    ite = ite + 1
```



```
x = 1
T = 0
W = 2
b = 2
```

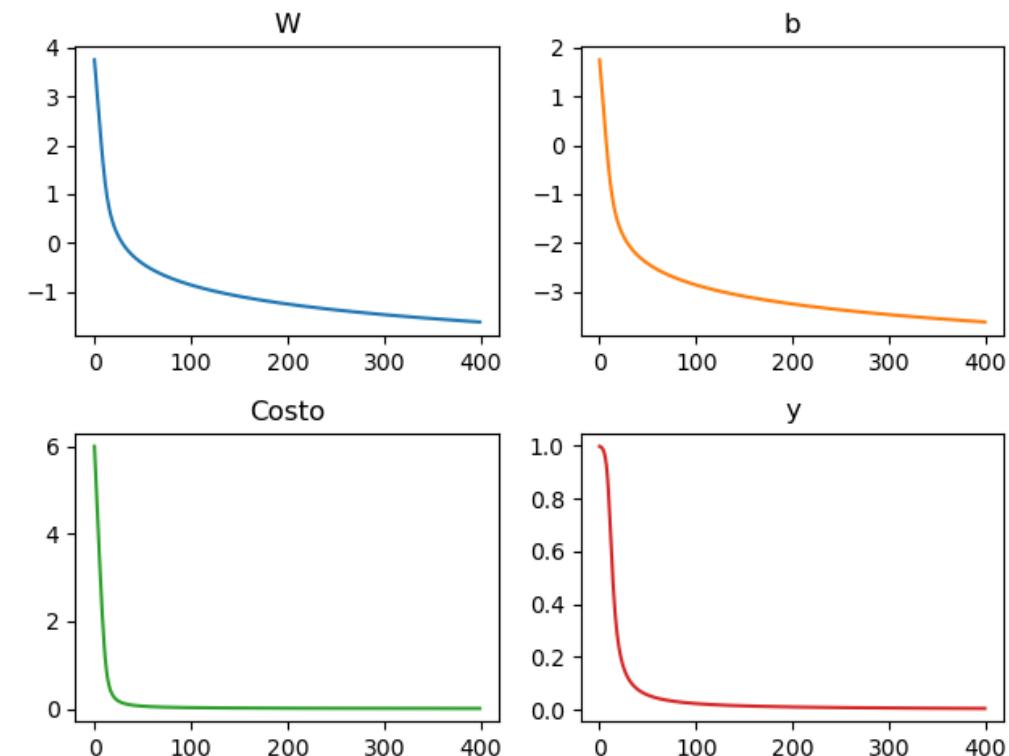
```
MAX_ITE, alfa = 2000, 0.25
ite = 0
C_ant, Costo = 0, 1
while (ite<MAX_ITE) and (np.abs(C_ant-Costo)>10e-05):
    C_ant = Costo
    neta = W * X + b
    y = 1.0/(1+np.exp(-neta))
    Error = T - y
    Costo = -(T*np.log(y)+(1-T)*np.log(1-y))
    gradiente_W = - Error * X
    gradiente_b = - Error
    W = W - alfa * gradiente_W
    b = b - alfa * gradiente_b
    ite = ite + 1
```



```
x = 1
T = 0
```

```
W = 4
b = 2
```

```
MAX_ITE, alfa = 2000, 0.25
ite = 0
C_ant, Costo = 0, 1
while (ite<MAX_ITE) and (np.abs(C_ant-Costo)>10e-05):
    C_ant = Costo
    neta = W * X + b
    y = 1.0/(1+np.exp(-neta))
    Error = T - y
    Costo = -(T*np.log(y)+(1-T)*np.log(1-y))
    gradiente_W = - Error * X
    gradiente_b = - Error
    W = W - alfa * gradiente_W
    b = b - alfa * gradiente_b
    ite = ite + 1
```



Funciones de costo

Entropía Cruzada Binaria

- Mejor ajuste para problemas de clasificación binaria.
- Produce **gradientes más grandes** cuando las predicciones están muy alejadas de las etiquetas verdaderas, acelerando el entrenamiento.

La entropía cruzada es más adecuada para clasificación binaria, ya que aprovecha la naturaleza probabilística de la sigmoide.

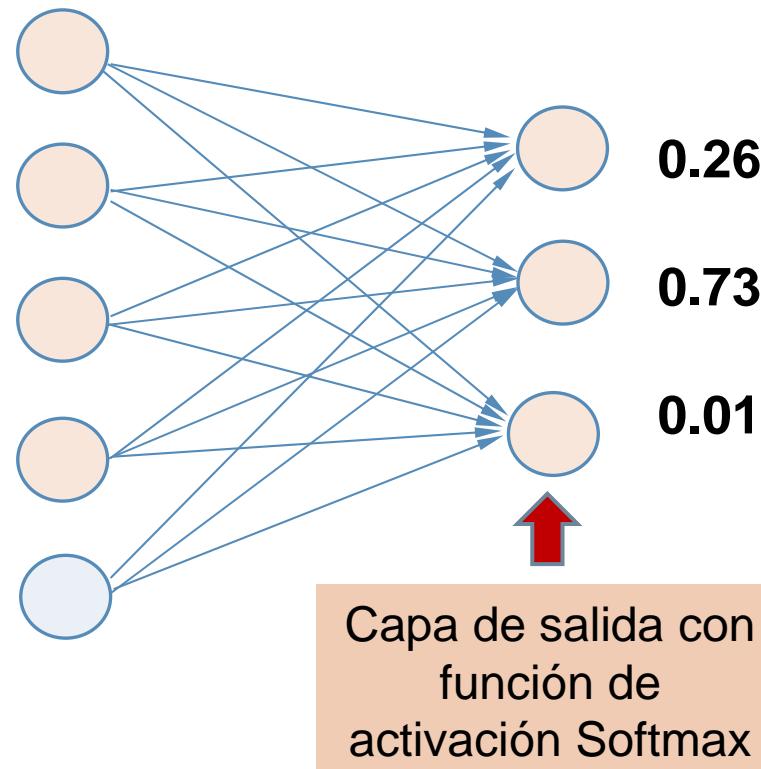
Error Cuadrático Medio

- Usualmente empleado en problemas de **regresión**.
- Gradientes más pequeños cuando la salida está cerca de la sigmoide, lo que puede provocar **aprendizaje más lento**.

El ECM, aunque puede usarse en clasificación, no genera gradientes tan eficientes, especialmente cuando las predicciones son extremas (cercanas a 0 o 1).

Función Softmax

- Se utiliza como función de activación en la última capa para normalizar la salida de la red a una distribución de probabilidad.



Capa softmax

$$netaj = \sum_i w_{ji} x_i + b_j$$

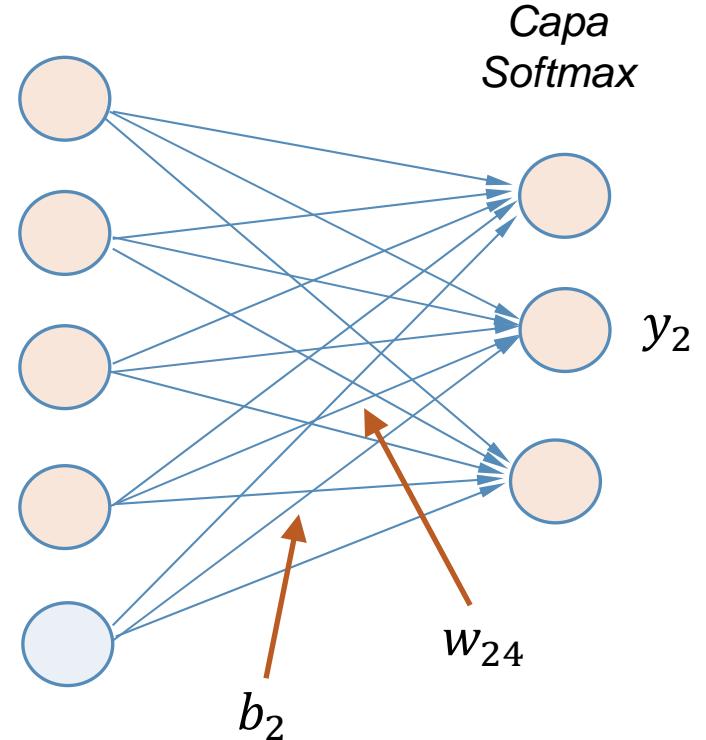
$$e^{netaj}$$

$$y_j = \frac{e^{netaj}}{\sum_k e^{netak}}$$

- La salida de la capa es una distribución de probabilidad

- $y_j > 0 \quad j = 1..k$

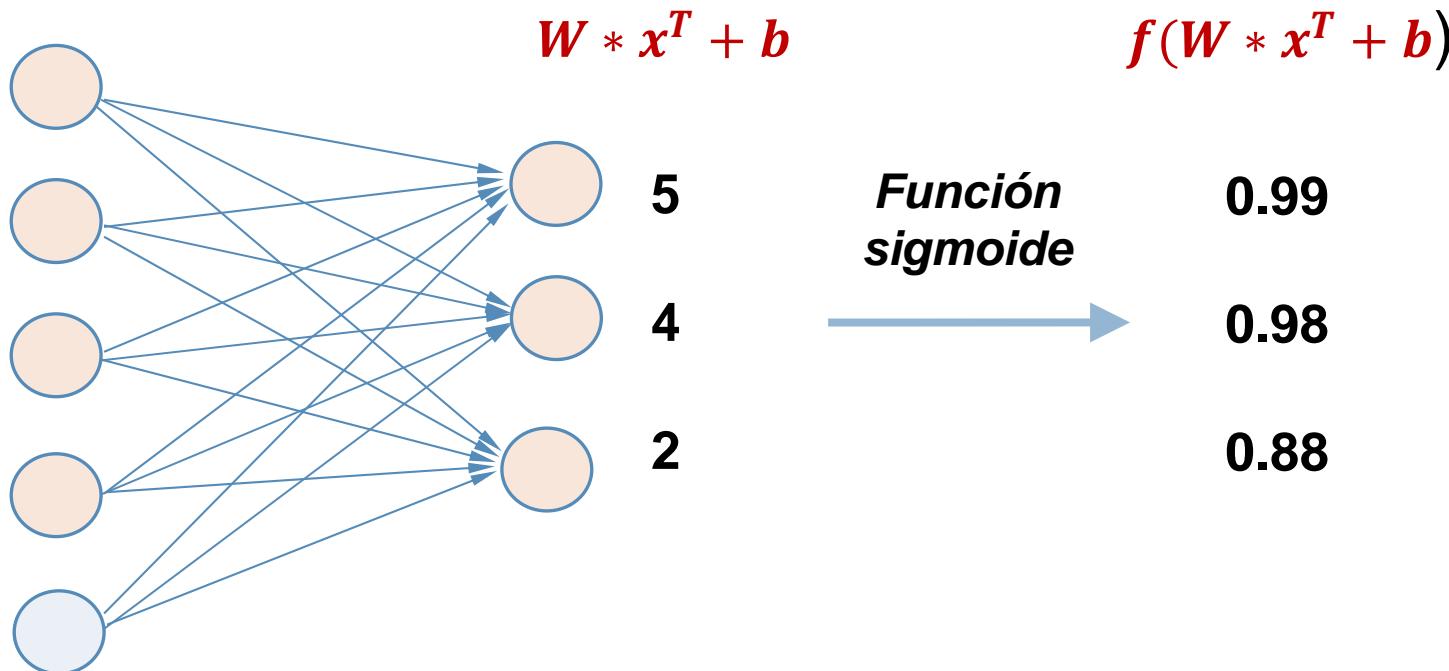
- $\sum_j y_j = 1$



Ver que el incremento en algún y_j producirá disminuciones en el resto

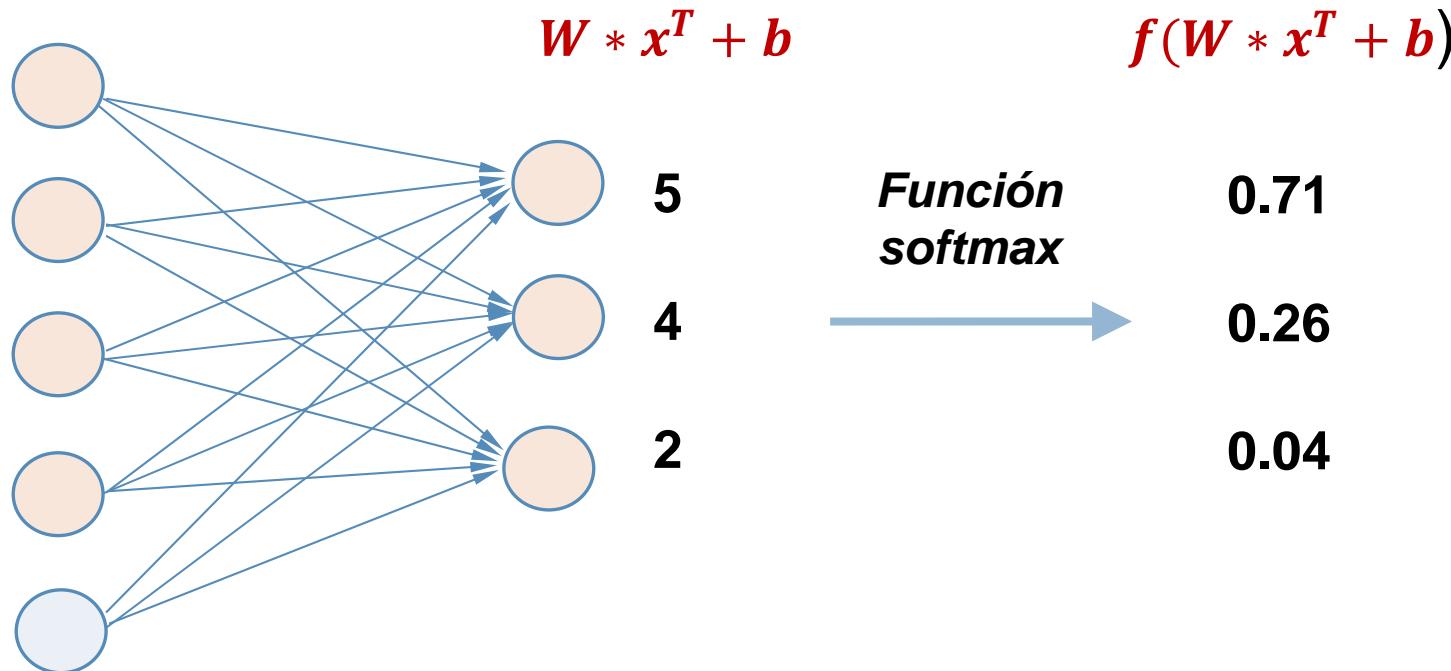
Función Softmax

□ Ejemplo



Función Softmax

□ Ejemplo



Capa Softmax

$$y_j = \frac{e^{netaj}}{\sum_k e^{netak}}$$

□ Función de costo: Entropía cruzada categórica

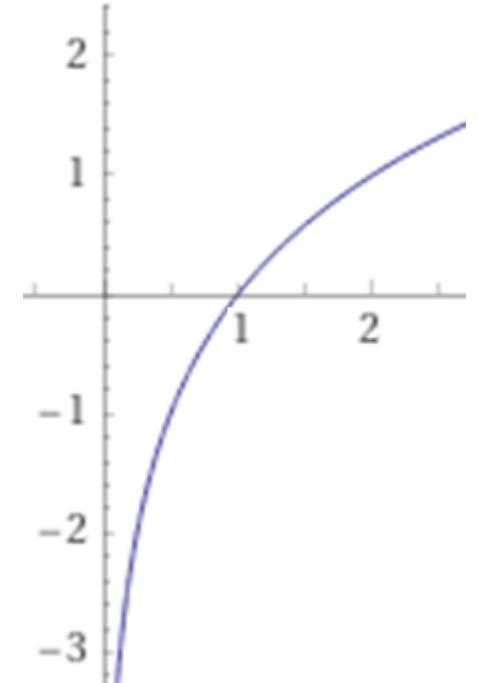
$$C = - \sum_k t_k \ln y_k$$

donde t es un vector binario que vale 1 sólo en la posición correspondiente al valor de clase esperado.

Luego

$$C = - \ln y_s$$

s es la neurona correspondiente al valor de clase esperado



Capa Softmax

$$y_j = \frac{e^{netaj}}{\sum_k e^{netak}}$$

□ Función de costo: Entropía cruzada categórica

$$C = -\ln y_s$$

S es la neurona correspondiente al valor de clase esperado

□ Derivada de la función de costo

$$\frac{\partial C}{\partial w_{jk}} = -(t_j - y_j) x_k$$

$$\frac{\partial C}{\partial b_j} = -(t_j - y_j)$$

Coincide con la derivada de la entropía cruzada binaria

Capa softmax

□ Función de costo: Entropía cruzada categórica

Entrada	Salida Softmax			Loss, L(a)
	cat	dog	horse	NLL
	0.71	0.26	0.04	
	0.02	0.00	0.98	
	0.49	0.49	0.02	
<i>La clase correcta está pintada de rojo</i>			$-\log(y_s)$ en la clase correcta	
			→	
				0.34
				0.02
				0.71
				Total = 1.07

- Sólo se evalúa en la neurona correspondiente a la salida esperada.
- Cuando más cerca está de 1 menor será el error.
- A menor valor de la neurona softmax correspondiente a la clase correcta, mayor error.

ClassRNMulticlae.py

```
nn = RNMulticlae (alpha=0.01, n_iter=50, cotaE=10E-07, FUN='sigmoid',
                   COSTO='ECM', random_state=None)
```

□ Parámetros de entrada

- **alpha**: valor en el intervalo (0, 1] que representa la velocidad de aprendizaje.
- **n_iter**: máxima cantidad de iteraciones a realizar.
- **cotaE**: termina si la diferencia entre dos errores consecutivos es menor que este valor.
- **FUN**: función de activación – ‘sigmoid’, ‘tanh’, ‘softmax’.
- **COSTO**: función de costo – ‘ECM’, ‘EC_binaria’, ‘EC’
- **random_state**: None si los pesos se inicializan en forma aleatoria, un valor entero para fijar la semilla

ClassRNMulticlasse.py

```
nn = RNMulticlasse(alpha=0.01, n_iter=50, cotaE=10E-07, FUN='sigmoid',
                     COSTO='ECM', random_state=None)
```

```
nn.fit(X, T)
```

□ Parámetros de entrada

- **X** : arreglo de NxM donde N es la cantidad de ejemplos y M la cantidad de atributos.
- **T** : arreglo de NxK donde N es la cantidad de ejemplos y K es la cantidad de neuronas de salida

□ Retorna

- **w_** : arreglo de KxM siendo K el tamaño de la capa de salida y M la cantidad de atributos de entrada.
- **b_** : arreglo de K elementos formado por los bias de cada una de las K neuronas de la capa de salida.
- **errors_**: errores cometidos en cada iteración.

ClassRNMulticlae.py

Y = nn.predict_nOut(X)

- Parámetros de entrada
 - X : arreglo de NxM donde N es la cantidad de ejemplos y M la cantidad de atributos.
- Retorna: un arreglo con el resultado de aplicar la neurona general entrenada previamente con fit() a la matriz de ejemplos X.
 - Y : arreglo de NxK donde N es la cantidad de ejemplos y K es la cantidad de neuronas de salida con valores continuos.

ClassRNMulticlae.py

Y = nn.predict(X)

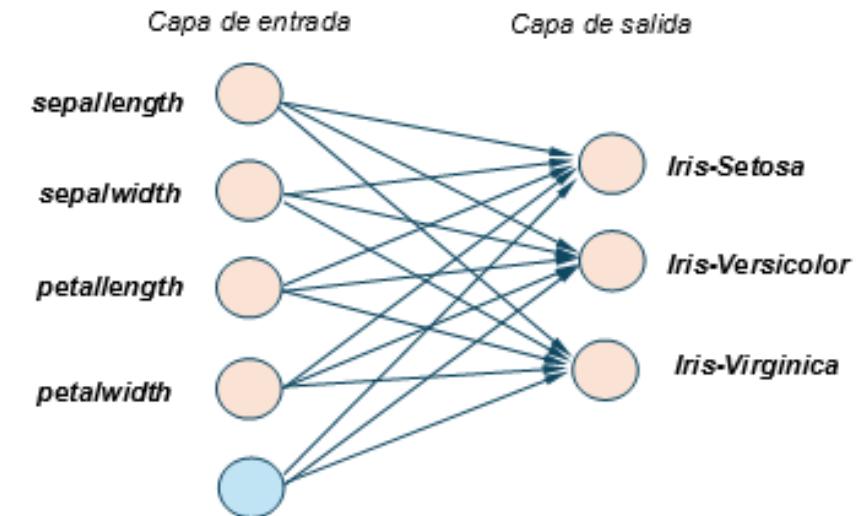
- Parámetros de entrada
 - X : arreglo de NxM donde N es la cantidad de ejemplos y M la cantidad de atributos.
- Retorna: un arreglo con el resultado de aplicar la neurona general entrenada previamente con fit() a la matriz de ejemplos X.
 - Y : arreglo de N elementos indicando para cada ejemplo el número de la clase predicha.

RNMulticlae_IRIS_RN.ipynb

Sigmoide vs Softmax

- Si la función de activación es **sigmoide**
 - Las neuronas de salida aprenden en forma independiente.
 - La salida no tiene que sumar 1.
 - Permite interpretar cada neurona como una probabilidad independiente.
 - Puede ser útil si las clases no son mutuamente excluyentes.
 - Es conveniente utilizar como función de costo la Entropía cruzada binaria (gradientes más grandes).

- Si la función de activación es **Softmax**
 - Las neuronas de salida aprenden en forma conjunta.
 - La suma de las salidas siempre es 1, lo que permite interpretar las salidas como una distribución de probabilidad.
 - Muy útil cuando las clases son mutuamente excluyentes.
 - Utiliza como función de costo la Entropía Cruzada Categórica.



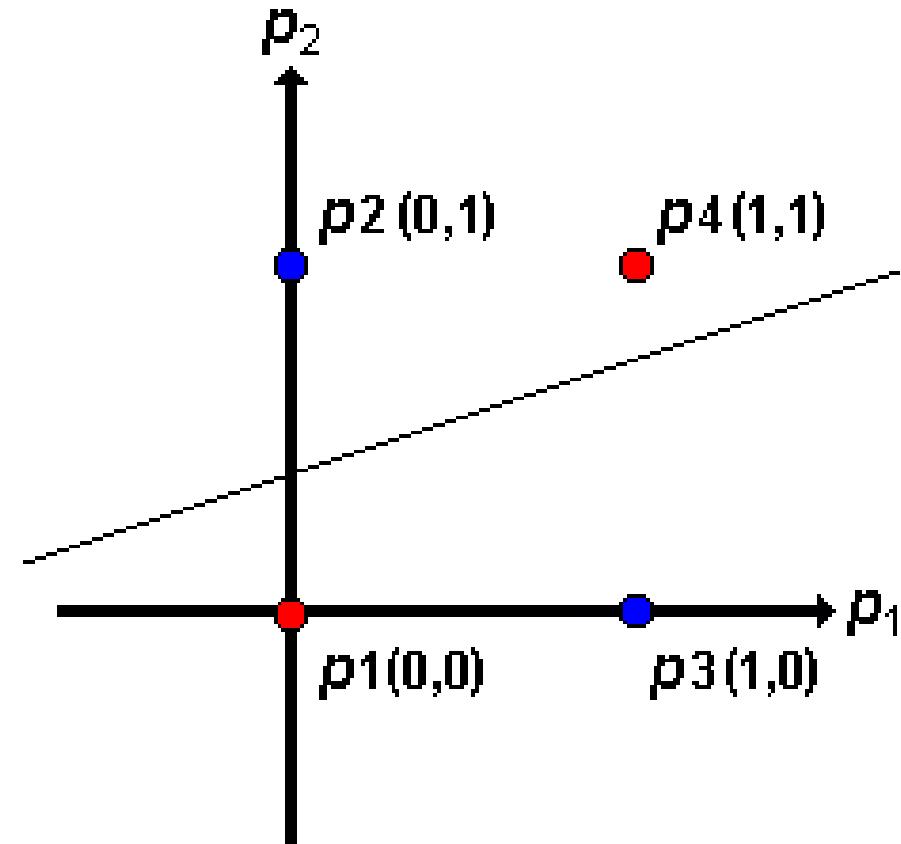
MULTIPERCEPTRÓN Y EL ALGORITMO BACKPROPAGATION



Redes multicapa

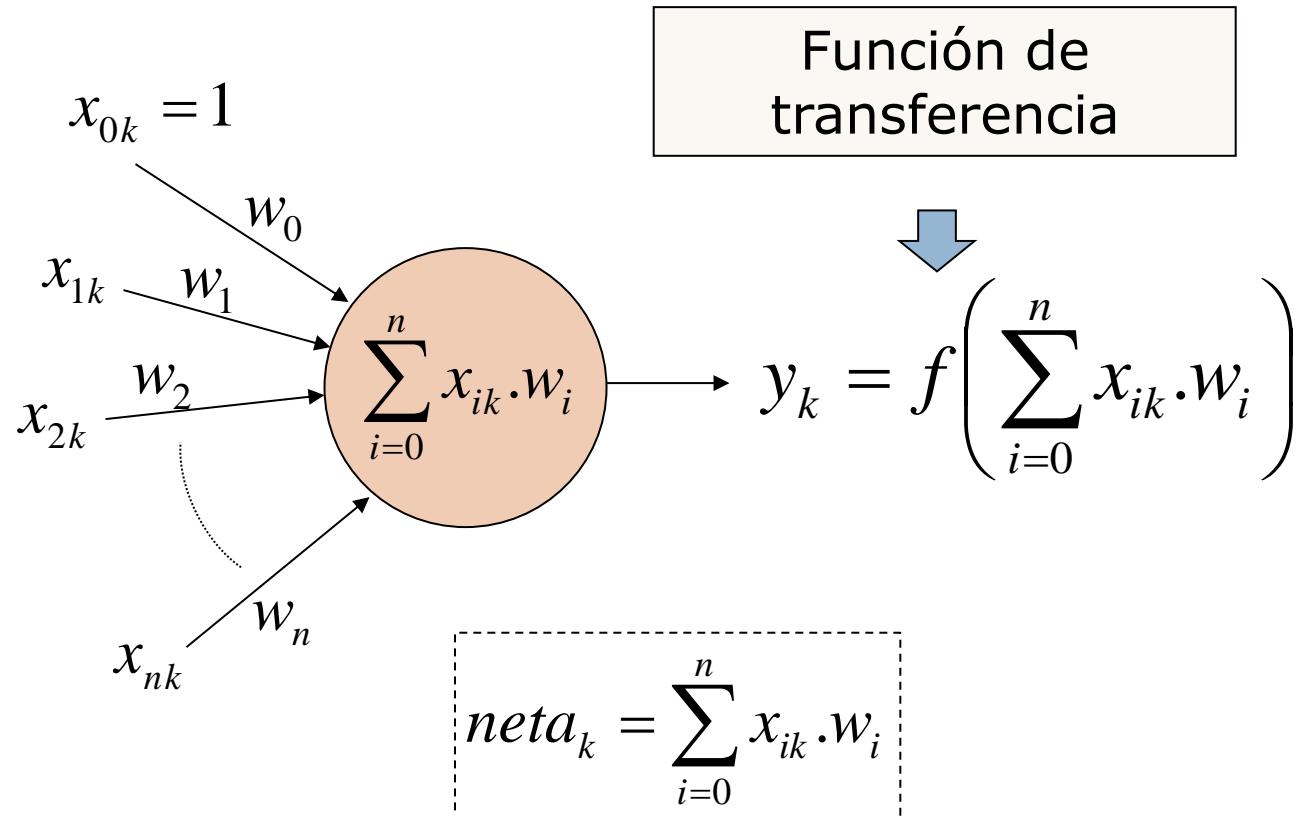
3

- Con una sola neurona no se puede resolver el problema del XOR porque no es linealmente separable.



Neurona artificial

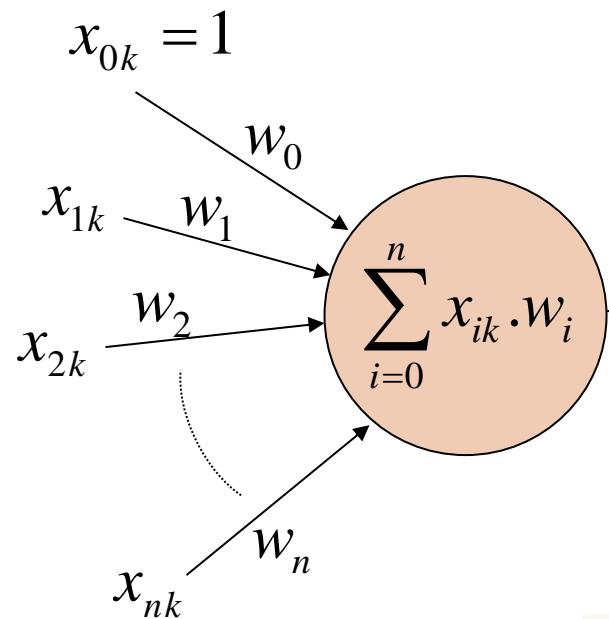
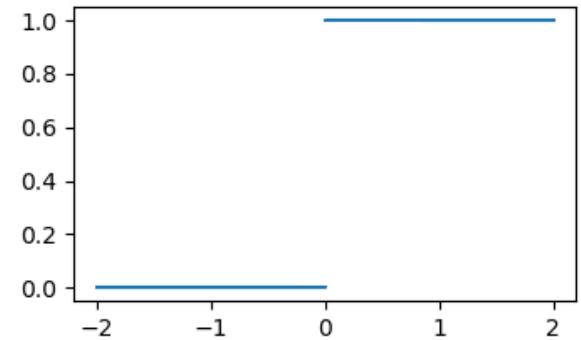
6



Perceptrón

7

$$f(x) = \begin{cases} 1 & \text{si } x \geq 0 \\ 0 & \text{si } x < 0 \end{cases}$$



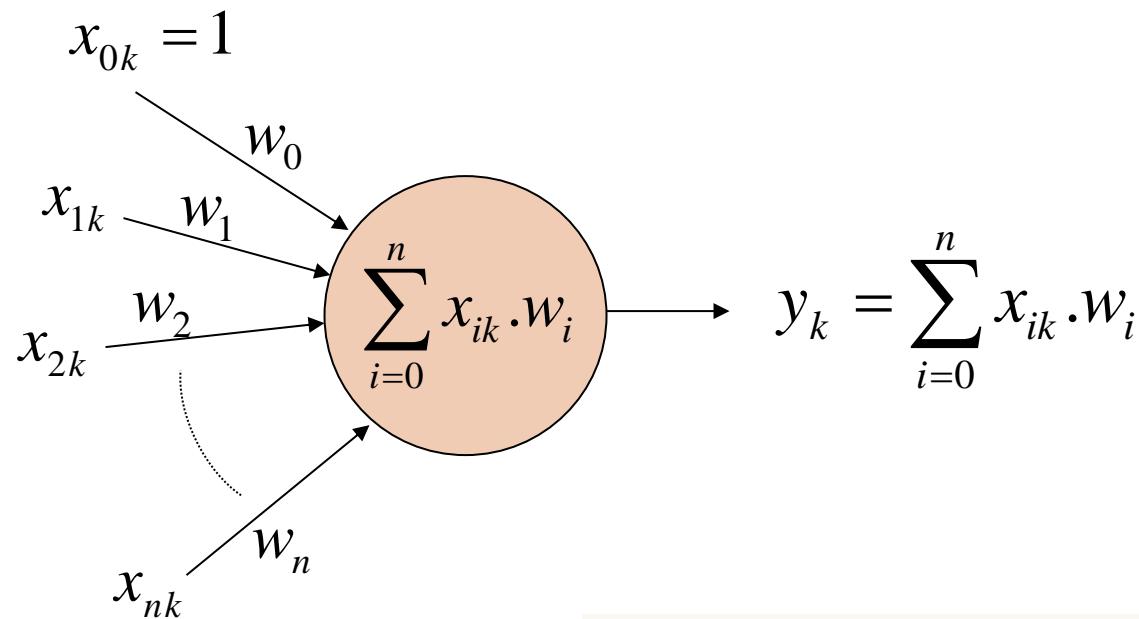
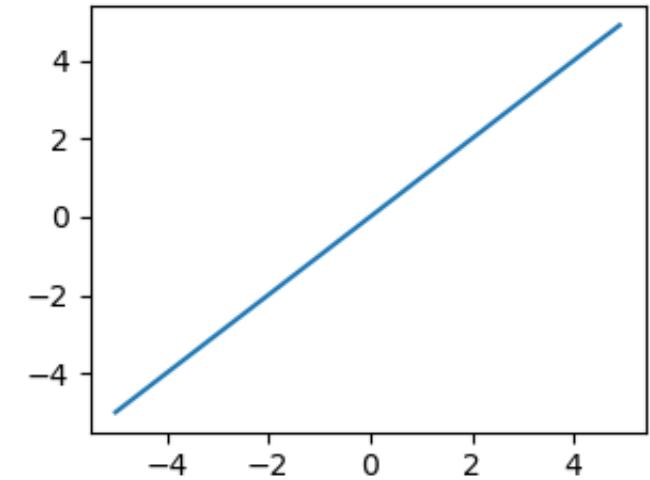
$$y_k = \begin{cases} 1 & \text{si } \sum_{i=0}^n x_{ik} \cdot w_i \geq 0 \\ 0 & \text{si } \sum_{i=0}^n x_{ik} \cdot w_i < 0 \end{cases}$$

- Actualiza el vector W modificando el ángulo que forma con cada ejemplo X_k

Combinador lineal

8

$$f(x) = x$$

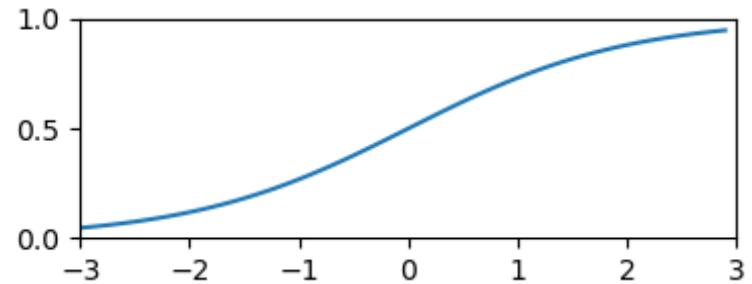
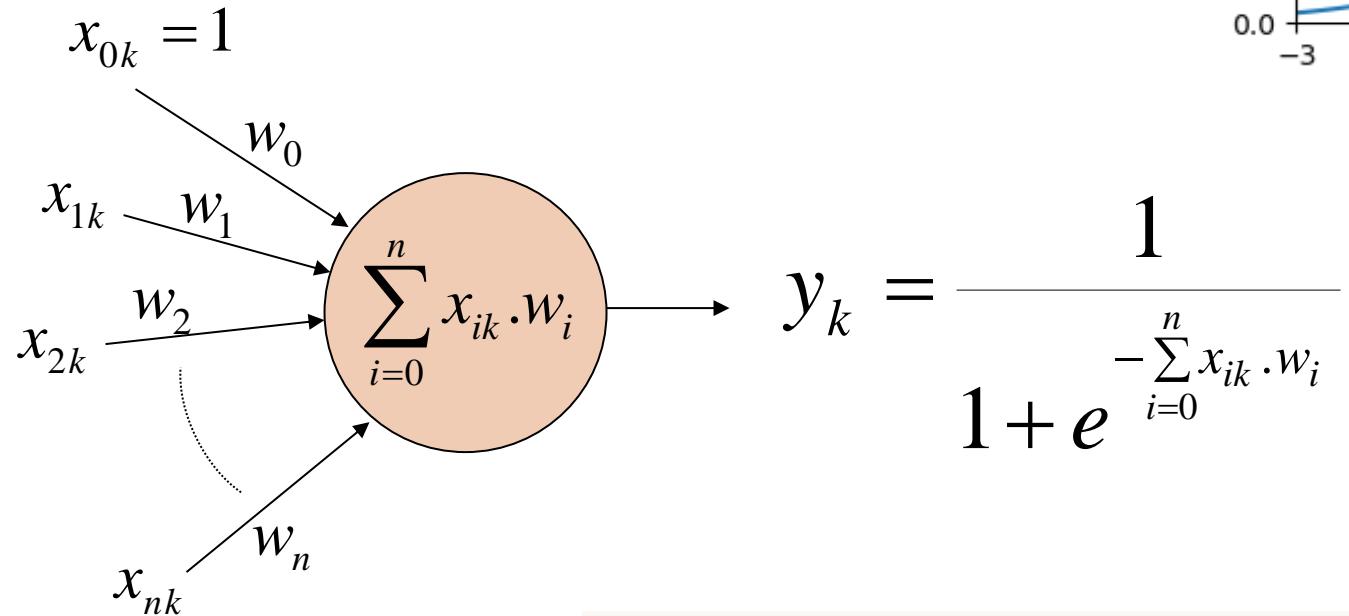


- Utiliza descenso por gradiente para actualizar el vector de pesos.

Neurona no lineal (logsig)

9

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

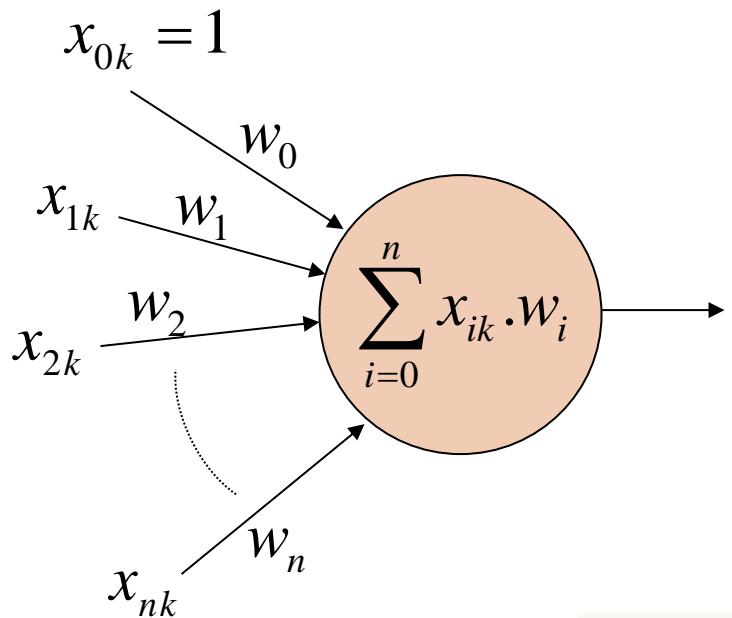


- Utiliza descenso por gradiente para actualizar el vector de pesos.

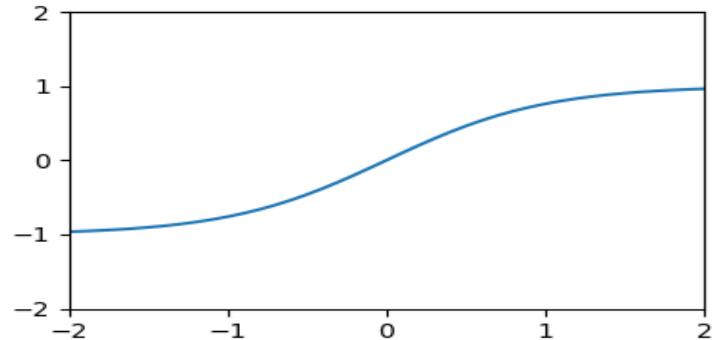
Neurona no lineal (tansig)

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

10



$$y_k = \frac{2}{1 + e^{-2 \sum_{i=0}^n x_{ik} \cdot w_i}} - 1$$

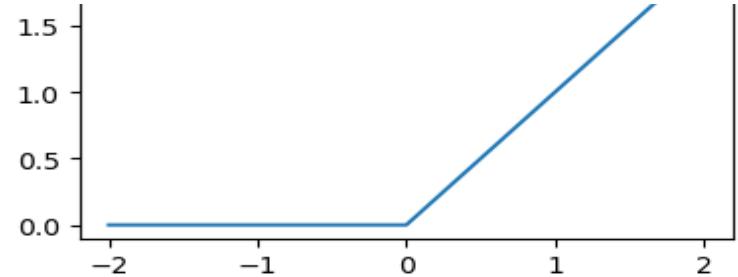
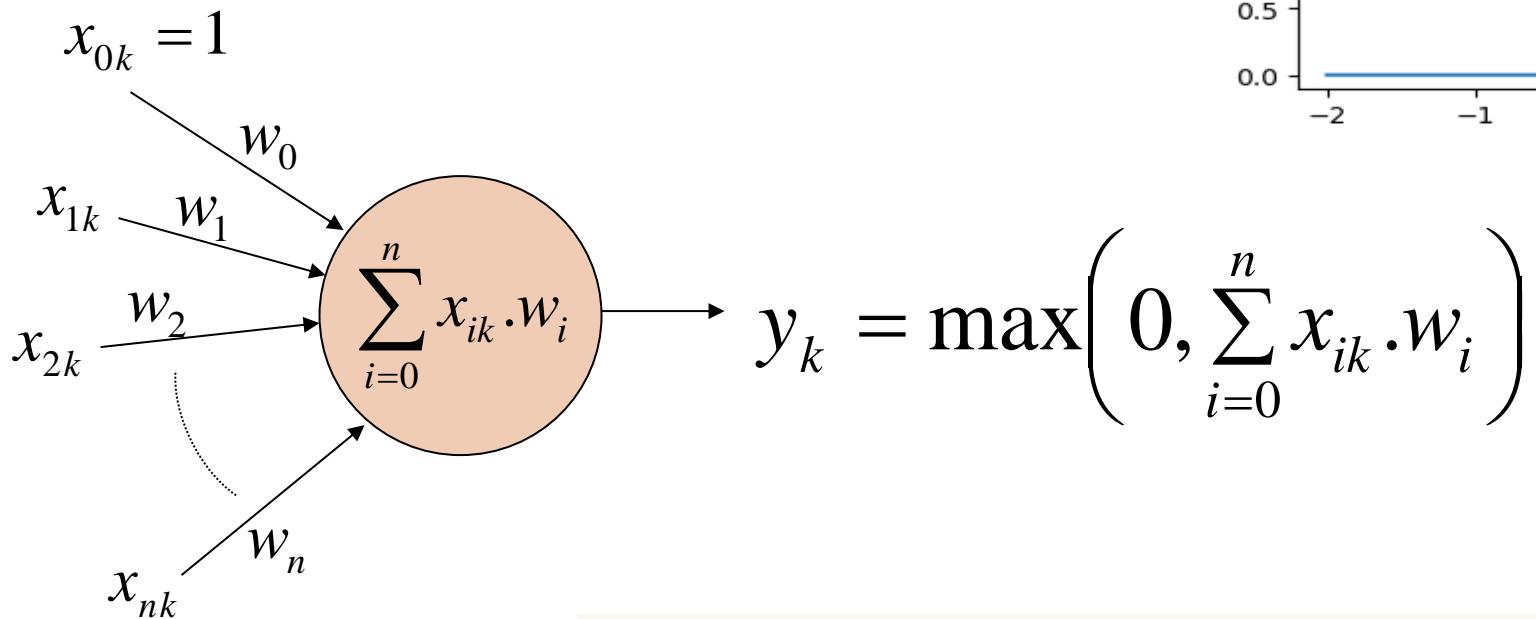


- Utiliza descenso por gradiente para actualizar el vector de pesos.

Neurona no lineal (Relu)

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

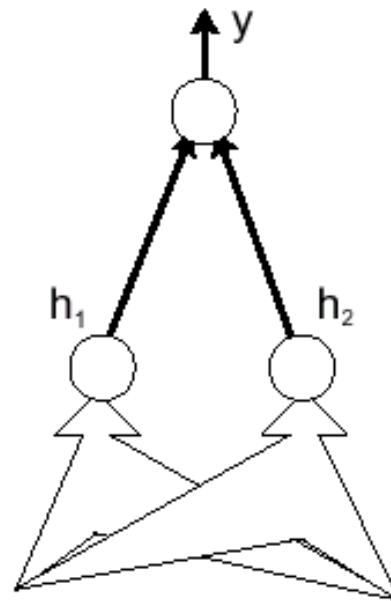
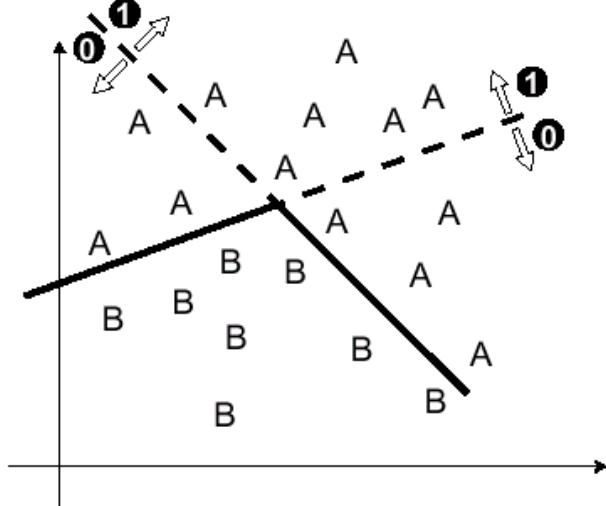
11



- Utiliza descenso por gradiente para actualizar el vector de pesos.

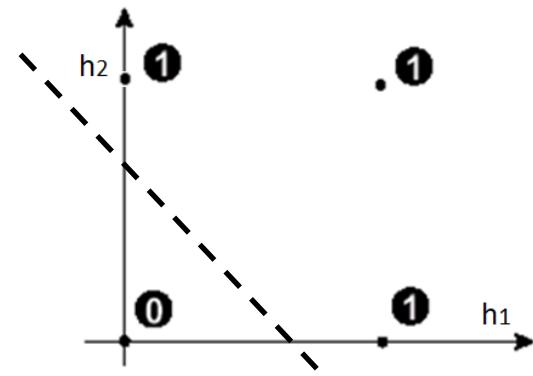
Problema no separable linealmente

12



h_1	h_2	y
0	0	0
0	1	1
1	0	1
1	1	1

$\left. \right\} A \leftrightarrow 1$



- Se busca obtener un algoritmo más general que permita integrar el aprendizaje entre las dos capas.

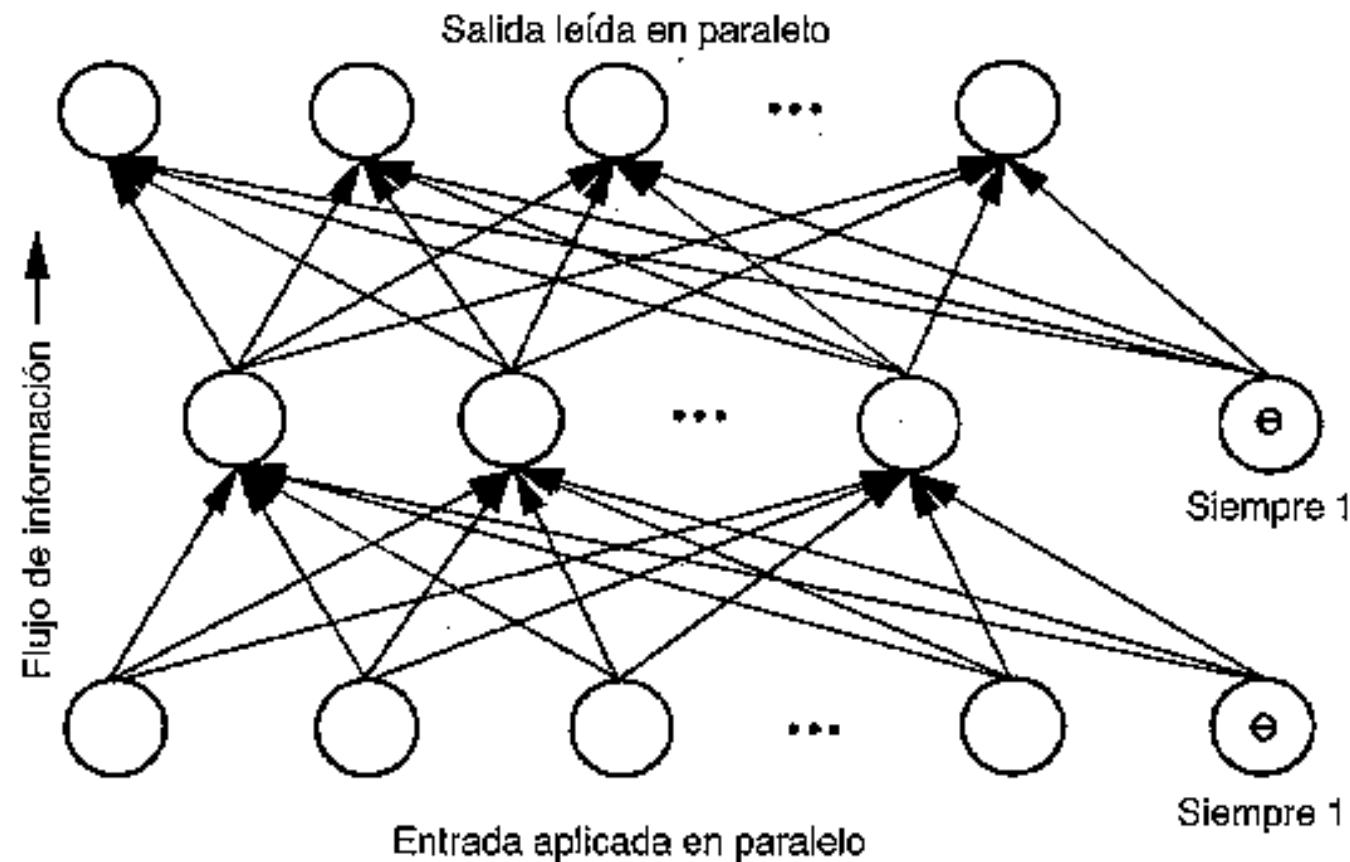
Problema no separable linealmente

15

- La idea es aplicar un descenso en la dirección del gradiente sobre la superficie de error expresada como una función de los pesos.
- Deberán tenerse en cuenta los pesos de los arcos que unen AMBAS capas.
- Dado que el aprendizaje es supervisado, para los nodos de salida se conoce la respuesta esperada a cada entrada. Por lo tanto, puede aplicarse la **regla delta** vista para el Combinador Lineal y la Neurona No Lineal.

Multiperceptrón - Arquitectura

16



Algoritmo backpropagation

17

Dado el siguiente conjunto de vectores

$$\{(x_1, y_1), \dots, (x_p, y_p)\}$$

que son ejemplos de correspondencia funcional

$$y = \phi(x) \quad x \in R^N, y \in R^M$$

se busca entrenar la red para que aprenda una aproximación

$$y' = \phi'(x)$$

Backpropagation. Capa Oculta

18

- Ejemplo de entrada

$$x_p = (x_{p1}, x_{p2}, \dots, x_{pN})^t$$

- Entrada neta de la j -ésima neurona de la capa oculta

$$neta_{pj}^h = \sum_{i=1}^n w_{ji}^h x_{pi} + \theta_j^h$$

- Salida de la j -ésima neurona de la capa oculta

$$i_{pj} = f_j^h(neta_{pj}^h)$$

Backpropagation. Capa de Salida

19

- Entrada neta de la k-ésima neurona de la capa de salida

$$neta_{pk}^o = \sum_{j=1}^L w_{kj}^o i_{pj} + \theta_k^o$$

- Salida de la k-ésima neurona de la capa de salida

$$o_{pk} = f_k^o(neta_{pk}^o)$$

Actualización de pesos

20

- Error en una sola unidad de la capa de salida

$$\delta_{pk} = (y_{pk} - o_{pk})$$

donde

- y es la salida deseada
- o es la salida real.
- p se refiere al p -ésimo vector de entrada
- k se refiere a la k -ésima unidad de salida

Actualización de pesos

21

- Se busca minimizar

$$E_p = \frac{1}{2} \sum_{k=1}^M \delta_{pk}^2$$

$$E_p = \frac{1}{2} \sum_{k=1}^M (y_{pk} - o_{pk})^2$$

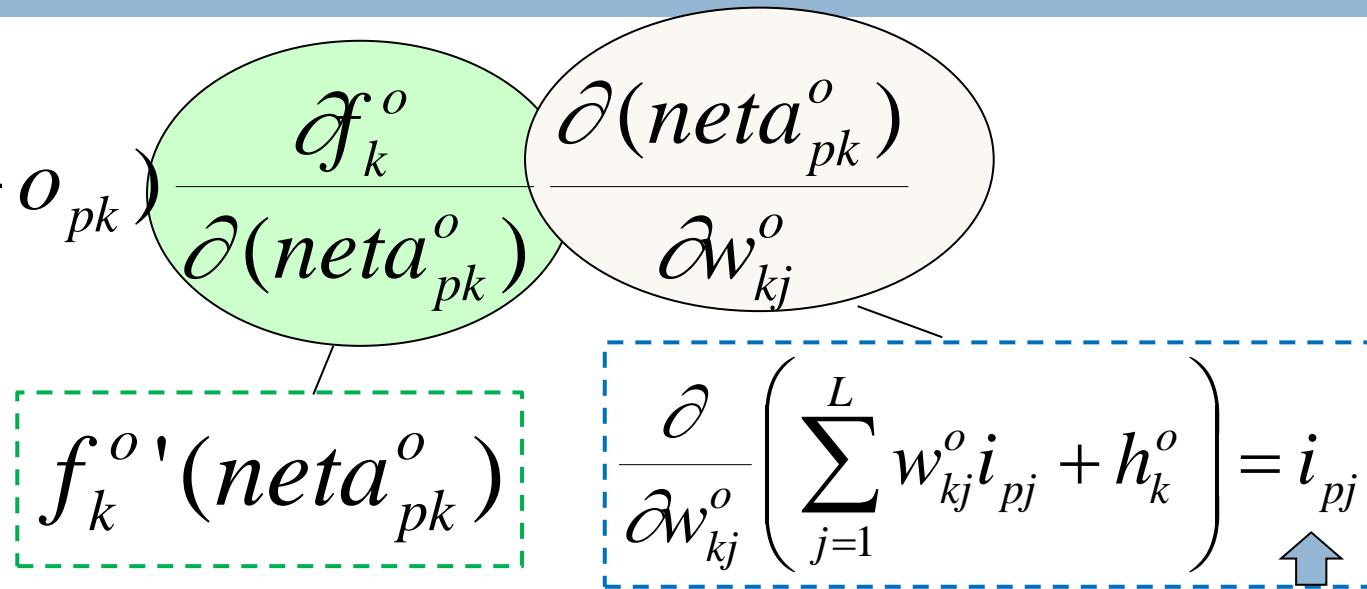
se tomará el valor negativo del gradiente

Actualización de pesos

$$\frac{\partial E_p}{\partial w_{kj}^o}$$

$$\frac{\partial E_p}{\partial w_{kj}^o}$$

Peso del arco que une la neurona j de la capa oculta y la neurona k de la capa de salida



Salida de la
neurona oculta j

Actualización de pesos

- Por lo tanto, para la capa de salida se tiene

$$\delta_{pk}^o = (y_{pk} - o_{pk}) f_k^{o'}(neto_{pk}^o)$$

$$w_{kj}^o(t+1) = w_{kj}^o(t) + \alpha \delta_{pk}^o i_{pj}$$

Actualización de pesos

- Corrección para los pesos de los arcos entre la capa de entrada y la oculta

$$\Delta_p w_{ji}^h(t) = \alpha \delta_{pj}^h x_{pi}$$

serán de la forma:

$$\delta_{pj}^h = f_j^{h'}(neto_{pj}^h) \sum_k \delta_{pk}^o w_{kj}^o$$

Backpropagation. Entrenamiento

25

- Aplicar un vector de entrada y calcular su salida.
- Calcular el error.
- Determinar en qué dirección (+ o -) debe cambiarse los pesos para reducir el error.
- Determinar la cantidad en que es preciso cambiar cada peso.
- Corregir los pesos de las conexiones.
- Repetir los pasos anteriores para todos los ejemplos hasta reducir el error a un valor aceptable.

Backpropagation. Resumen

34

- Aplicar el vector de entrada

$$x_p = (x_{p1}, x_{p2}, \dots, x_{pN})^t$$

- Calcular los valores netos de las unidades de la capa oculta

$$neto_{pj}^h = \sum_{i=1}^n w_{ji}^h x_{pi} + \theta_j^h$$

- Calcular las salidas de la capa oculta

$$i_{pj} = f_j^h(neto_{pj}^h)$$

Backpropagation. Resumen

35

- Calcular los valores netos de las unidades de la capa de salida

$$neto_{pk}^o = \sum_{j=1}^L w_{kj}^o i_{pj} + \theta_k^o$$

- Calcular las salidas

$$o_{pk} = f_k^o(neto_{pk}^o)$$

Backpropagation. Resumen

36

- Calcular los términos de error para las unidades de salida

$$\delta_{pk}^o = (y_{pk} - o_{pk}) f_k^{o'}(neto_{pk}^o)$$

- Calcular los términos de error para las unidades ocultas

$$\delta_{pj}^h = f_j^h'(neto_{pj}^h) \sum_k \delta_{pk}^o w_{kj}^o$$

Backpropagation. Resumen

37

- Se actualizan los pesos de la capa de salida

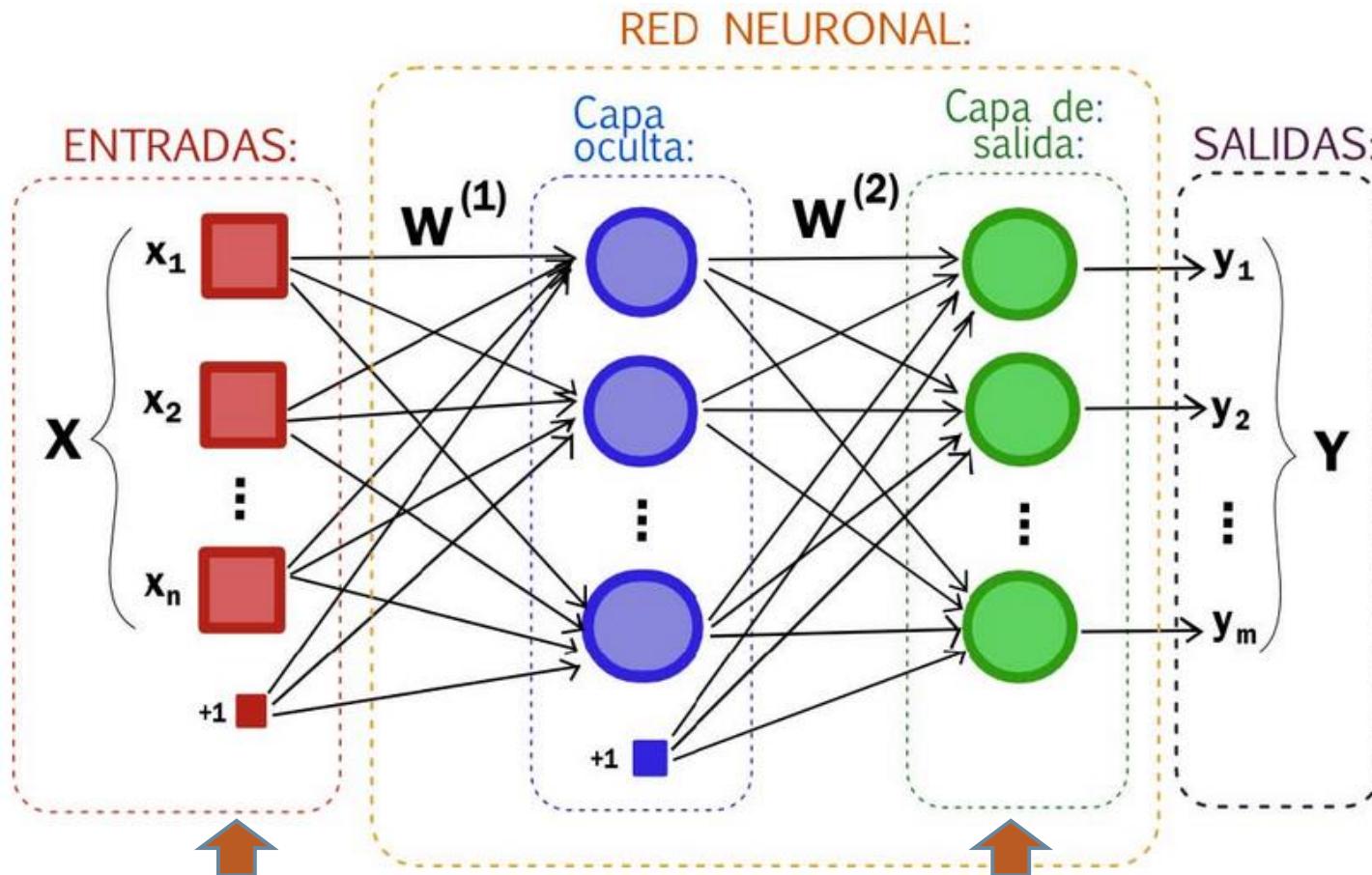
$$w_{kj}^o(t+1) = w_{kj}^o(t) + \alpha \delta_{pk}^o i_{pj}$$

- Se actualizan los pesos de la capa oculta

$$w_{ji}^h(t+1) = w_{ji}^h(t) + \alpha \delta_{pj}^h x_i$$

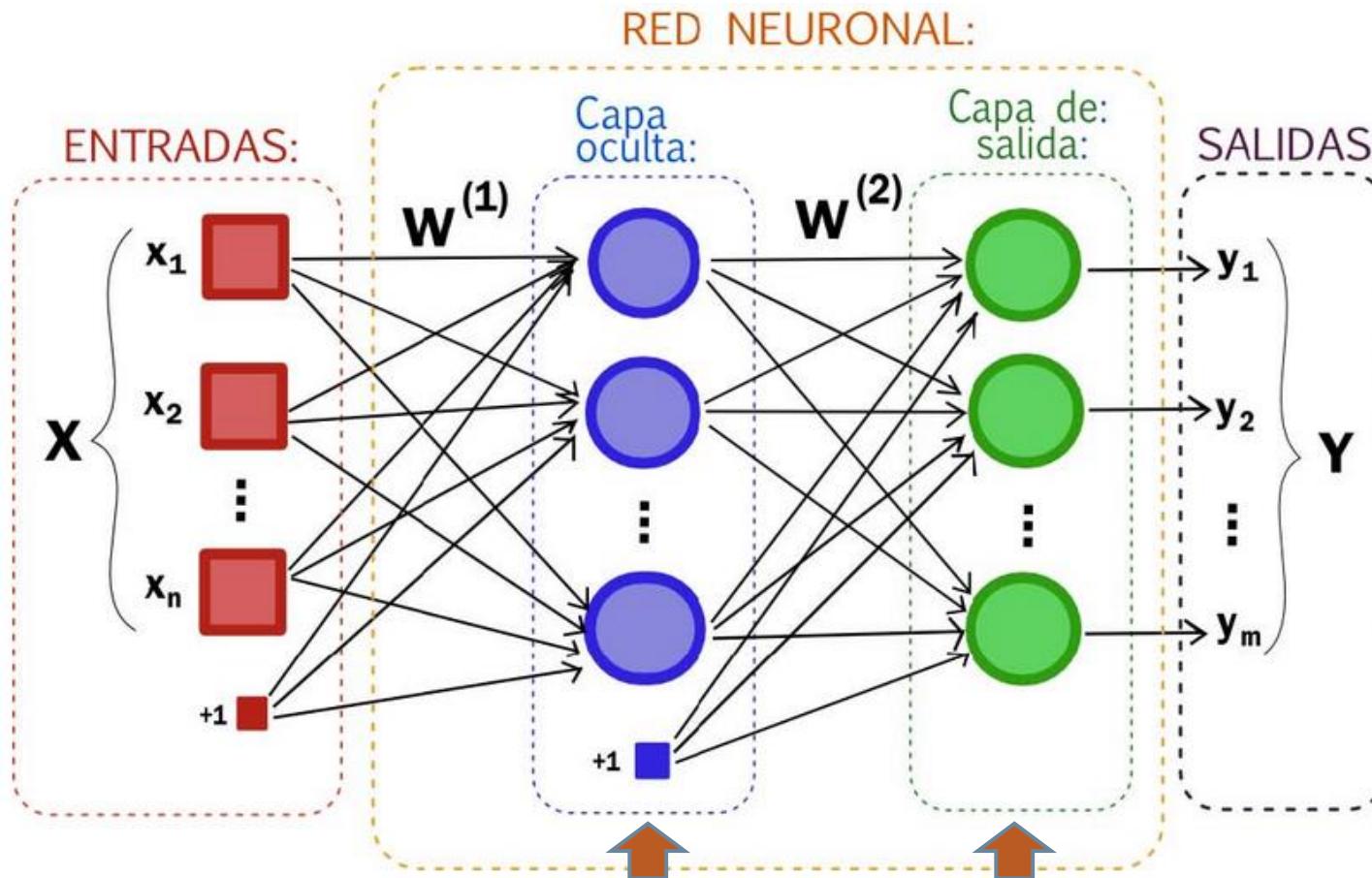
- Repetir hasta que el error resulte aceptable

Arquitectura de la red



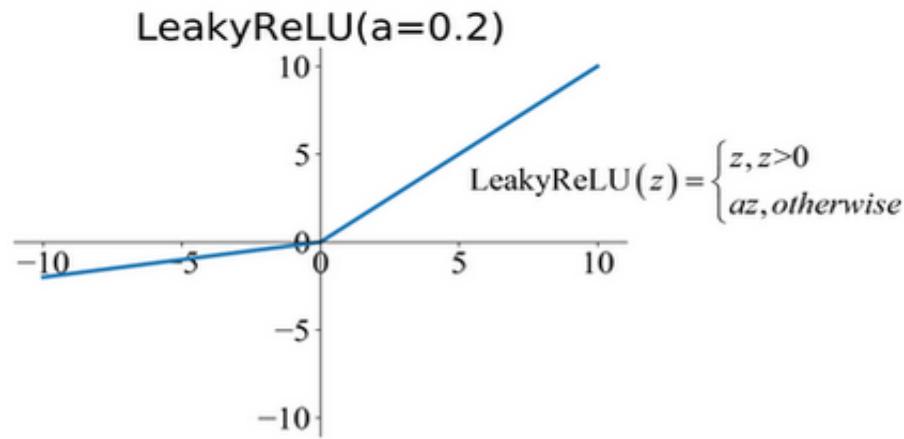
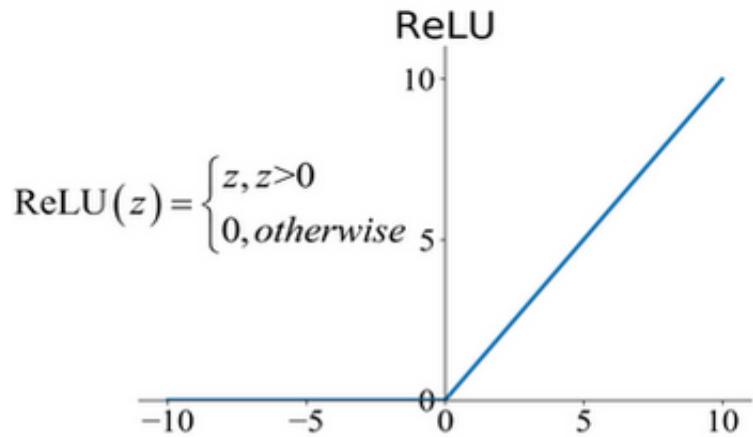
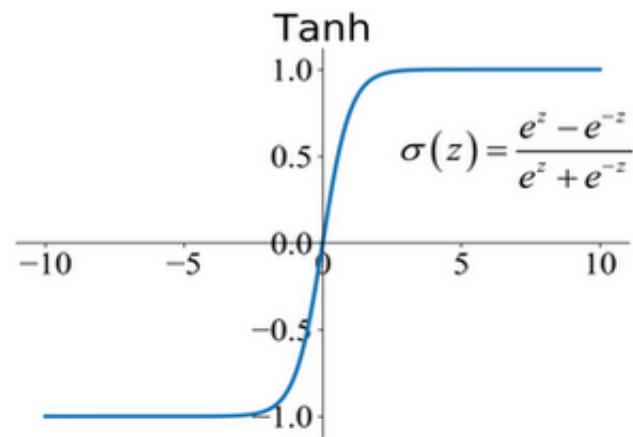
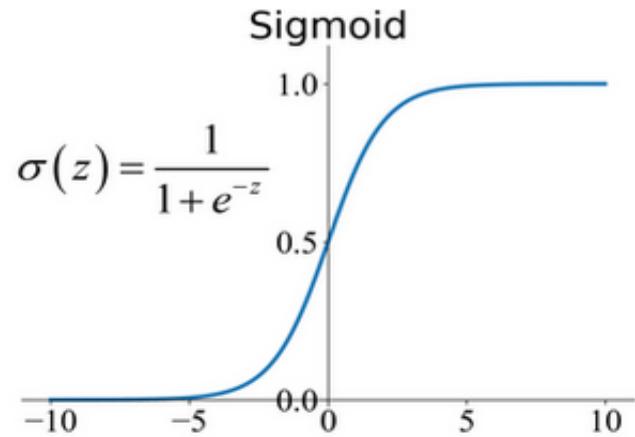
Las dimensiones de las capas de entrada y salida las define el problema

Arquitectura de la red

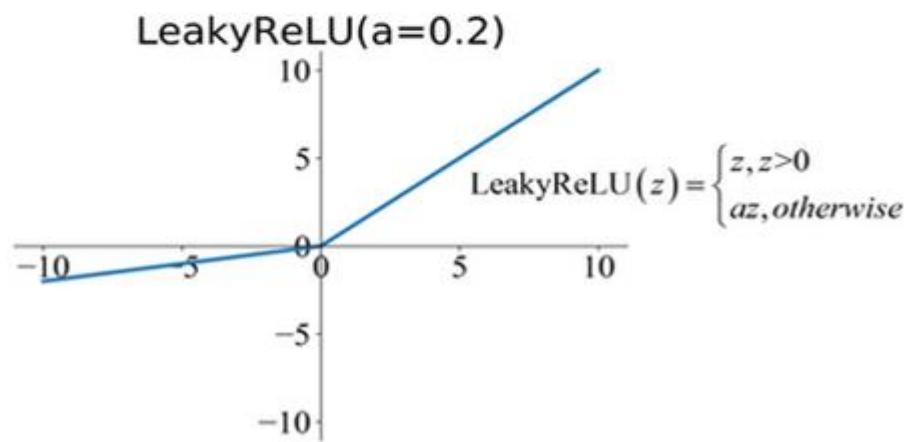
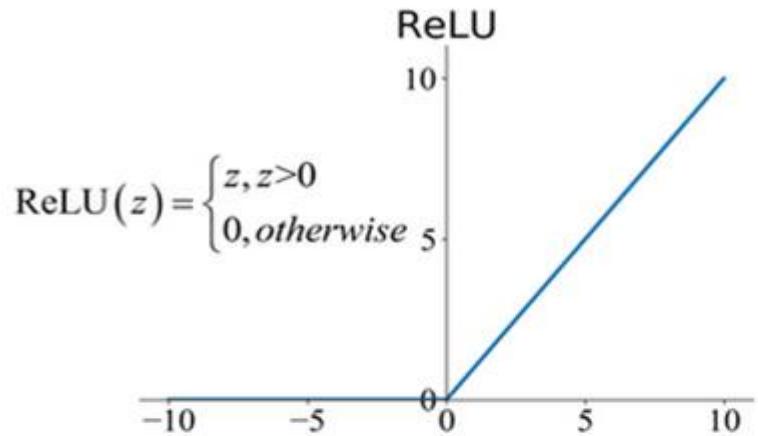


Su respuesta depende de la **Función de activación** elegida

Funciones de activación



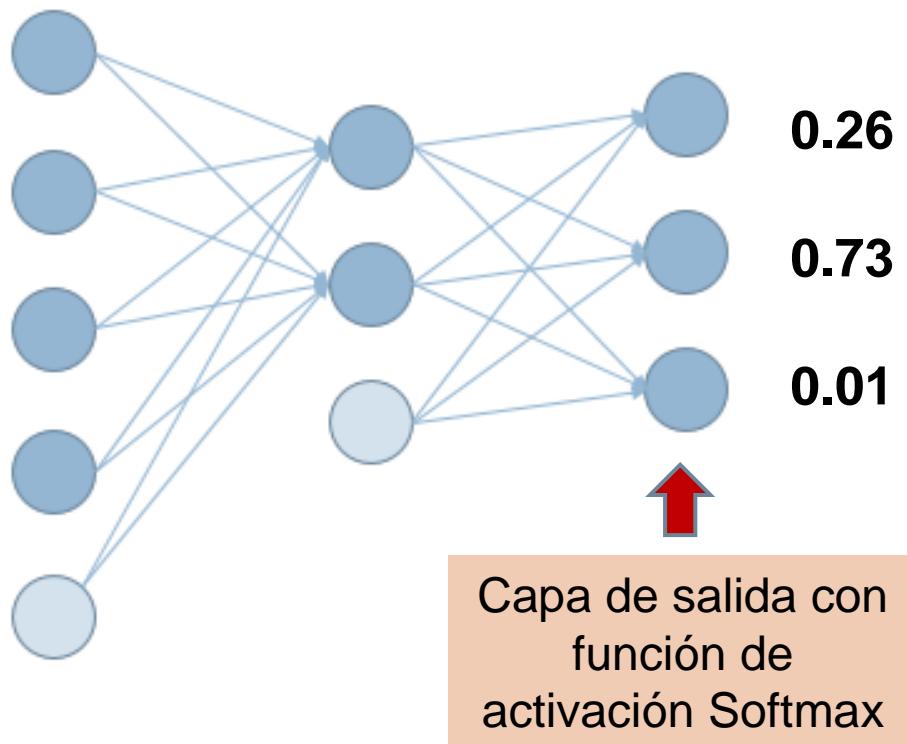
ReLU (Unidad Lineal Rectificada)



- Velocidad de aprendizaje (derivada)
- Velocidad de cómputo (fácil de calcular)
- Activa sólo algunas neuronas

Función Softmax

- Se utiliza como función de activación en la última capa para normalizar la salida de la red de manera que los valores sumen 1.



$$neta_j = \sum_i w_{ji} x_i + b_j$$

$$\hat{y}_j = \frac{e^{neta_j}}{\sum_k e^{neta_k}}$$

Keras

- Keras es una biblioteca de código abierto escrita en Python que facilita la creación de modelos complejos de aprendizaje profundo.
- Características
 - Prototipado rápido del modelo.
 - De alto nivel (programación a nivel de capa)
 - Usa las librerías de los frameworks vinculados
 - TensorFlow
 - Theano
 - Microsoft Cognitive Toolkit (CNTK)



Técnicas de optimización

- Descenso de gradiente estocástico (SGD) y el uso de mini-lotes 
- Capacidad de generalización de la red - Sobreajuste
- Mejoras introducidas
 - Momento: utiliza información de los gradientes anteriores
 - RMSProp: considera distintas magnitudes de cambio para reducir oscilaciones
 - Adam: combina los dos anteriores. Es el más usado.

Descenso de gradiente en mini-lotes

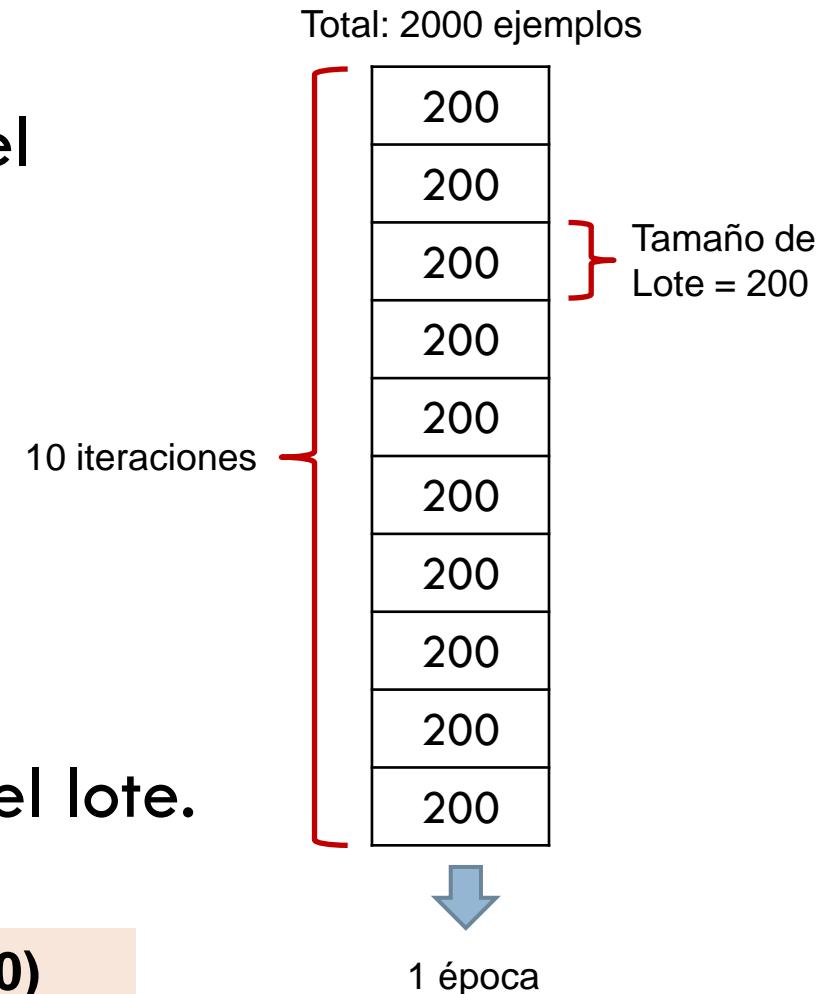
- En lugar de ingresar los ejemplos de a uno, ingresamos N a la red y buscamos minimizar el error cuadrático promedio del lote.

- La función de costo será

$$C = \frac{1}{N} \sum_{i=1}^N (d_i - f(neta_i))^2$$

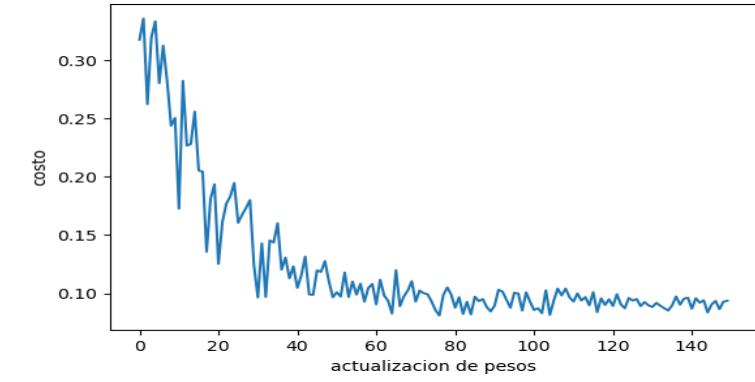
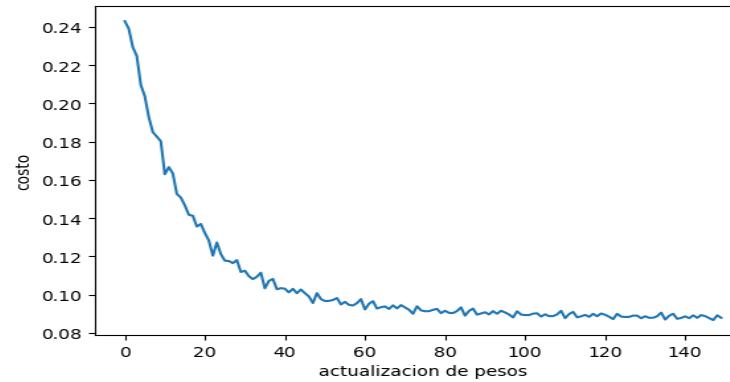
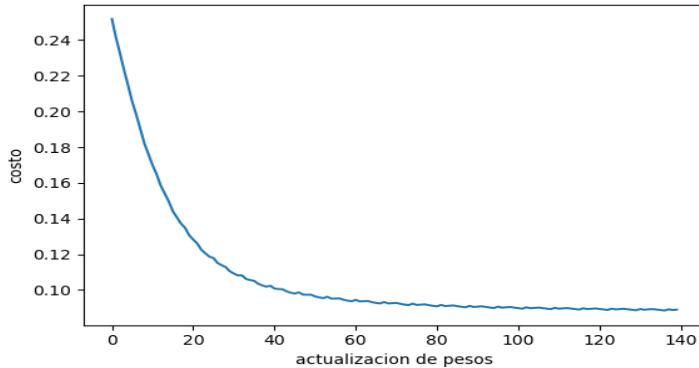
- N es la cantidad de ejemplos que conforman el lote.

```
model.fit(X, Y, epochs=2000, batch_size=200)
```



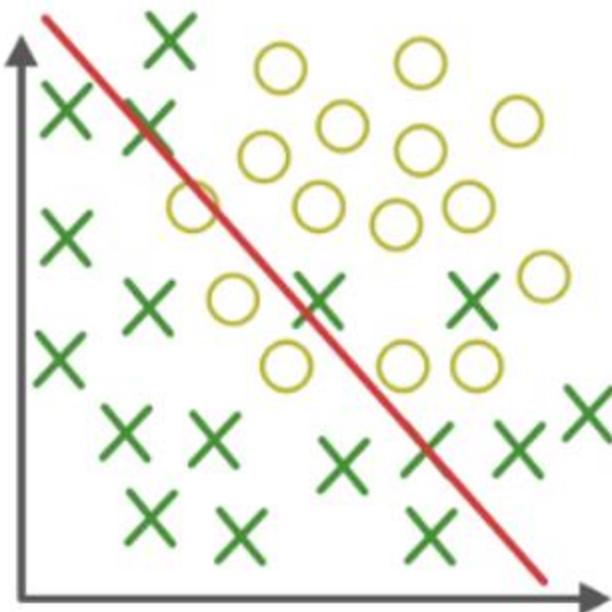
Descenso de gradiente

Batch	Mini-batch	Stochastic
Ingresa TODOS los ejemplos y luego actualiza los pesos.	Ingresa un LOTE de N ejemplos y luego actualiza los pesos	Ingresa UN ejemplo y luego actualiza los pesos
$C = \frac{1}{M} \sum_{i=1}^M (d_i - f(neta_i))^2$	$C = \frac{1}{N} \sum_{i=1}^N (d_i - f(neta_i))^2 \quad N \ll M$	$C = (d - f(neta))^2$

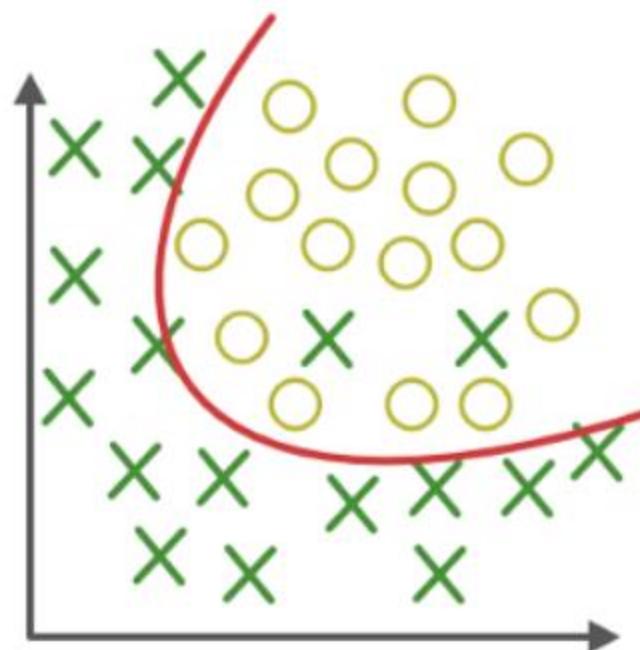


Ver [MLP_MNIST_8x8.ipynb](#) y [MLP_MNIST_8x8_miniLote.ipynb](#)

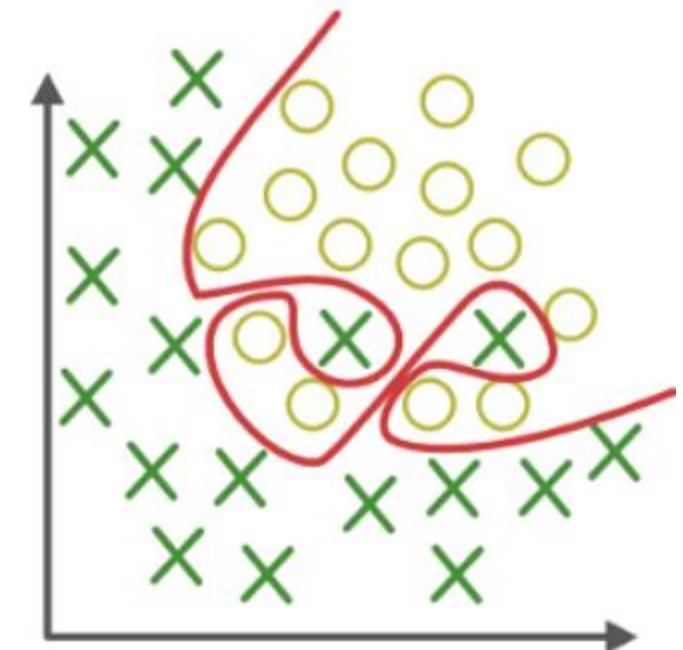
Capacidad de generalización de la red



Underfitting
(demasiado simple)



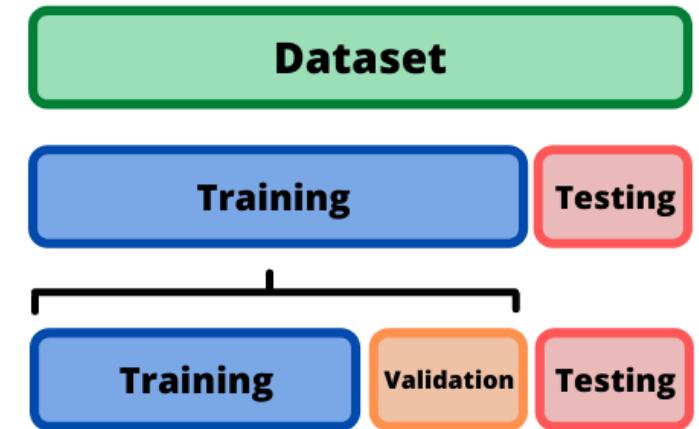
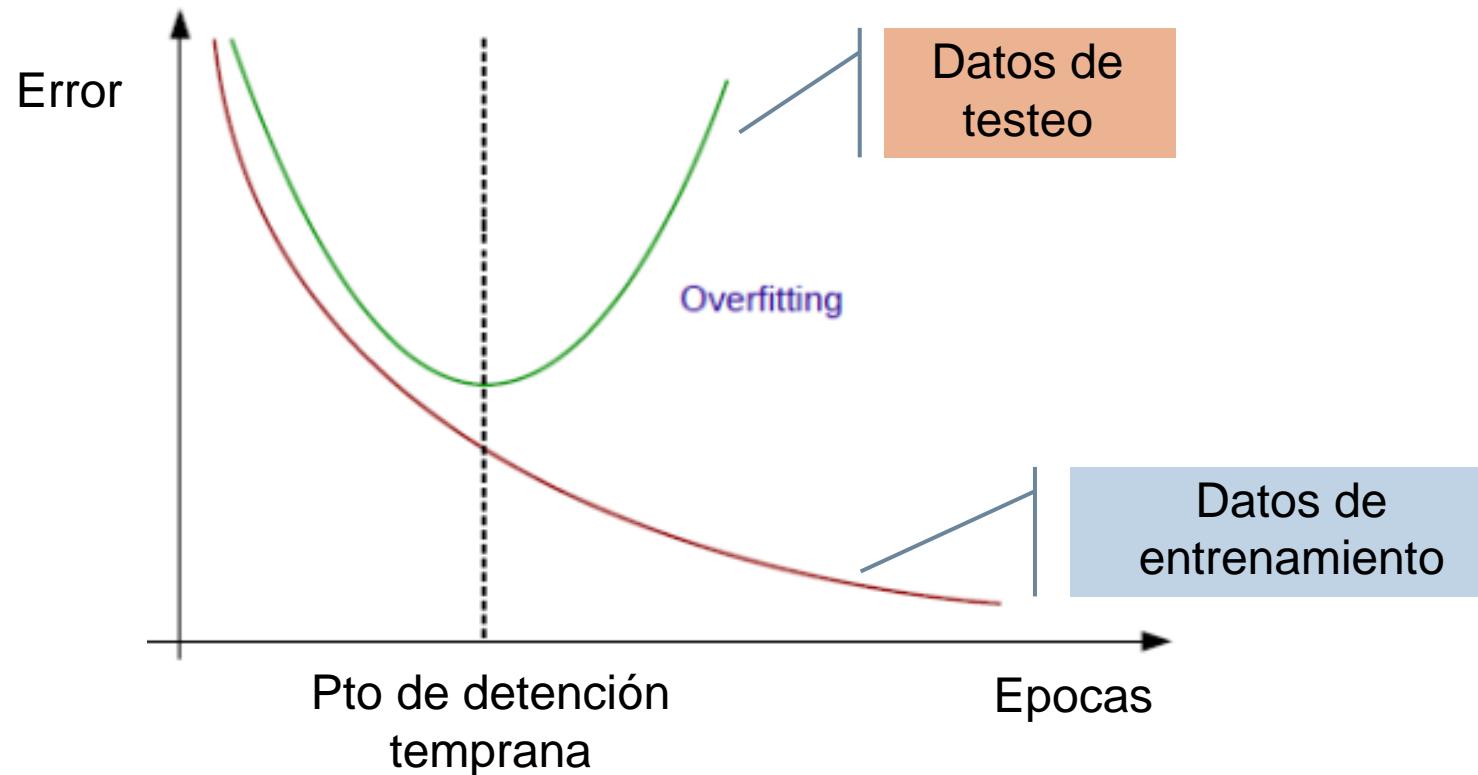
Generalización
correcta



Overfitting
(demasiados
parámetros)

Sobreajuste

□ Parada temprana (early-stopping)



Parada temprana

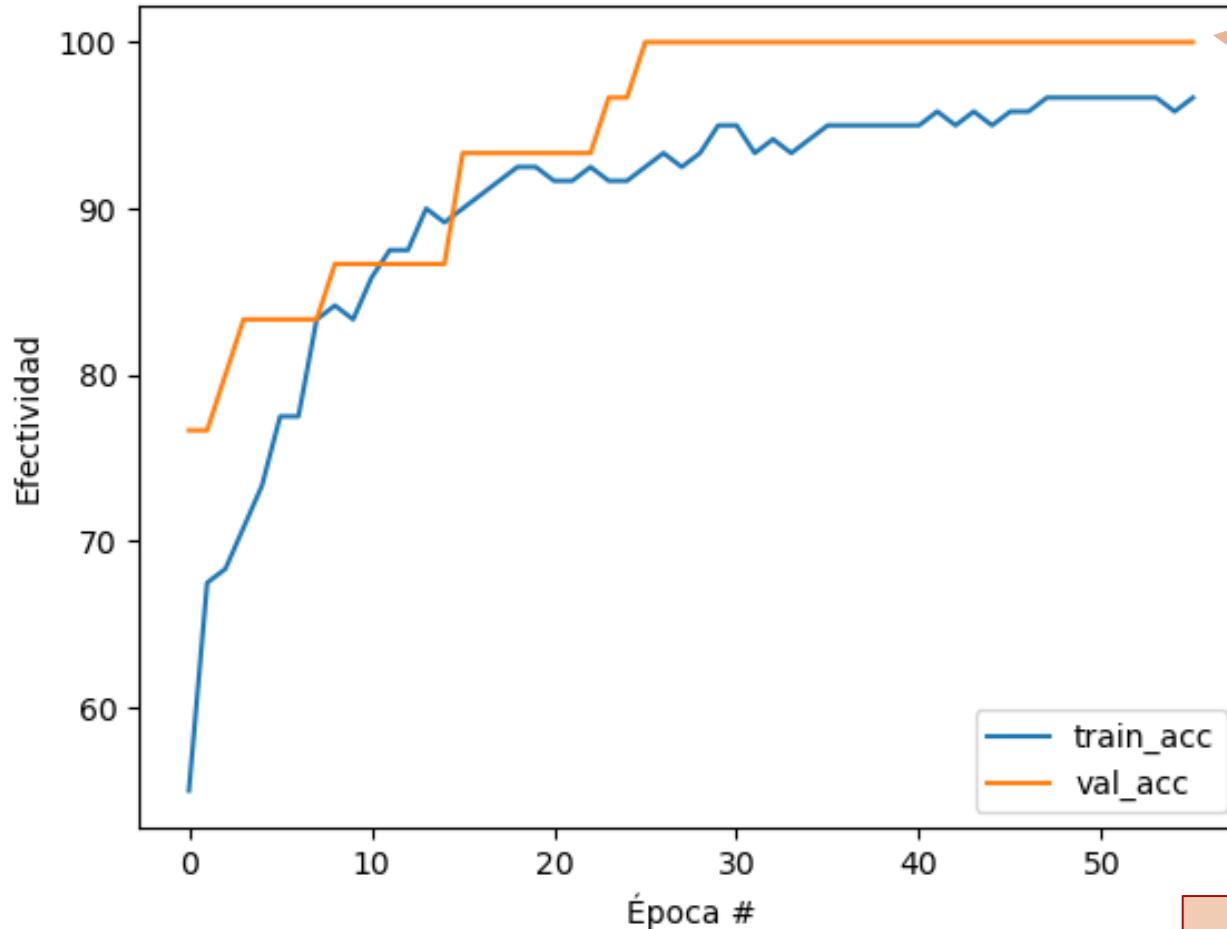
```
from keras.callbacks import EarlyStopping  
model = ...  
model.compile( ... )  
  
es = EarlyStopping(monitor='val_accuracy', patience=30, min_delta=0.0001)  
H = model.fit(x = X_train, y = Y_train, epochs=4000, batch_size = 20,  
               validation_data = (X_test, Y_test), callbacks=[es])  
  
print("Epocas = %d" % es.stopped_epoch)
```

EarlyStopping

- Detiene el entrenamiento cuando una métrica ha dejado de mejorar.
- Parámetros principales
 - **monitor**: valor a monitorear
 - **min_delta**: un cambio absoluto en el valor monitoreado inferior a **min_delta**, se considerará como que no hubo mejora.
 - **patience**: Número de épocas sin mejora tras las cuales se detendrá el entrenamiento.
 - **modo**: Uno de {"auto", "min", "max"}. En el modo "min", el entrenamiento se detendrá cuando el valor monitoreado haya dejado de disminuir; en el modo "max" se detendrá cuando el valor monitoreado haya dejado de aumentar; en el modo "auto", la dirección se infiere automáticamente del nombre del valor monitoreado.
 - **restore_best_weights**: Si se restauran los pesos del modelo de la época con el mejor resultado del valor monitoreado.

https://keras.io/api/callbacks/early_stopping/

Evolución del entrenamiento



*monitor='val_accuracy'
patience=30
min_delta=0.0001*

Keras_IRIS_Softmax_earlyStop.ipynb

Reducción del sobreajuste

- Si lo que se busca es reducir el sobreajuste puede probar
 - Incrementar la cantidad de ejemplos de entrenamiento.
 - Reducir la complejidad del modelo, es decir usar menos pesos (menos capas o menos neuronas por capa).
 - Aplicar una técnica de regularización
 - Regularización L2
 - Regularización L1
 - Dropout

Tienen por objetivo que los pesos de la red se mantengan pequeños

Sobreajuste - Regularización L2

- También conocida como técnica de decaimiento de pesos

$$C = C_o + \frac{\lambda}{2} \sum_k w_k^2$$

donde C_o es la función de costo original sin regularizar

- La derivada de la función de costo regularizada será

$$\frac{\partial C}{\partial w_k} = \frac{\partial C_o}{\partial w_k} + \lambda w_k$$

Sobreajuste - Regularización L2

Función de costo regularizada

$$C = C_0 + \frac{\lambda}{2} \sum_k w_k^2$$

Derivada

$$\frac{\partial C}{\partial w_k} = \frac{\partial C_0}{\partial w_k} + \lambda w_k$$

□ Actualización de los pesos

$$w_k = w_k - \alpha \frac{\partial C_0}{\partial w_k} - \lambda w_k$$

$$w_k = (1 - \lambda) w_k - \alpha \frac{\partial C_0}{\partial w_k}$$

Sobreajuste - Regularización L1

Función de costo regularizada

$$C = C_0 + \lambda \sum_k |w_k|$$

Derivada

$$\frac{\partial C}{\partial w_k} = \frac{\partial C_0}{\partial w_k} + \lambda sign(w_k)$$

□ Actualización de los pesos

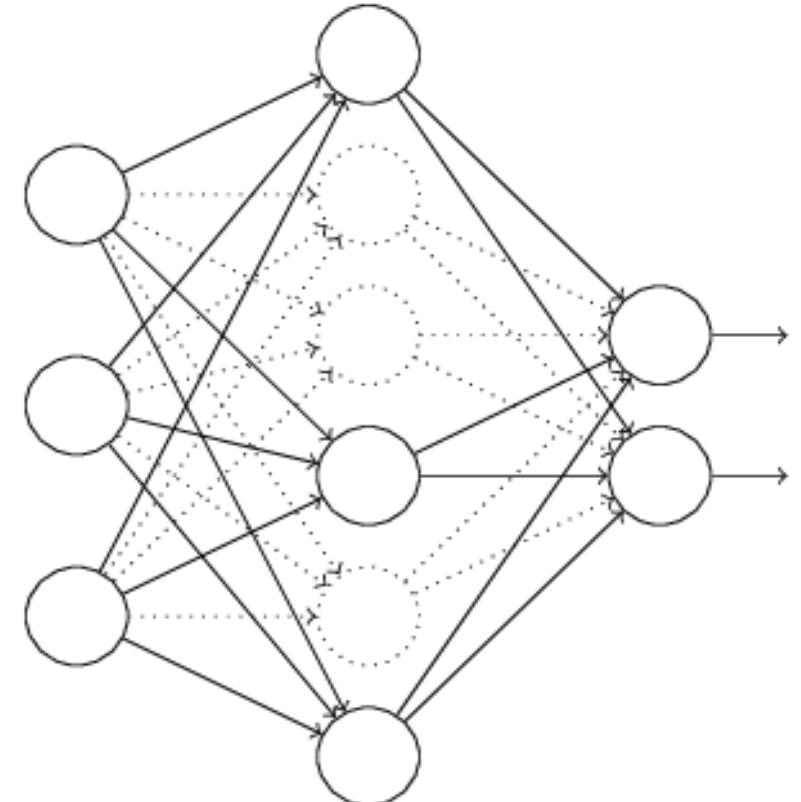
$$w_k = w_k - \alpha \frac{\partial C_0}{\partial w_k} - \lambda sign(w_k)$$

L1 vs L2

Regularización L1	Regularización L2
<ul style="list-style-type: none">• Lleva los pesos a 0: útil para que el modelo ignore características irrelevantes.• Selección automática de características: ideal para datos con muchas variables donde solo unas pocas son relevantes.• Aplicaciones: Modelos de alta dimensionalidad (por ejemplo, compresión de modelos, selección de características).	<ul style="list-style-type: none">• Mantiene todos los pesos pequeños: pero no los hace exactamente 0.• Mejor generalización: útil cuando se espera que todas las características sean relevantes.• Aplicaciones: Modelos profundos, evitar sobreajuste en redes neuronales complejas.

Sobreajuste - Dropout

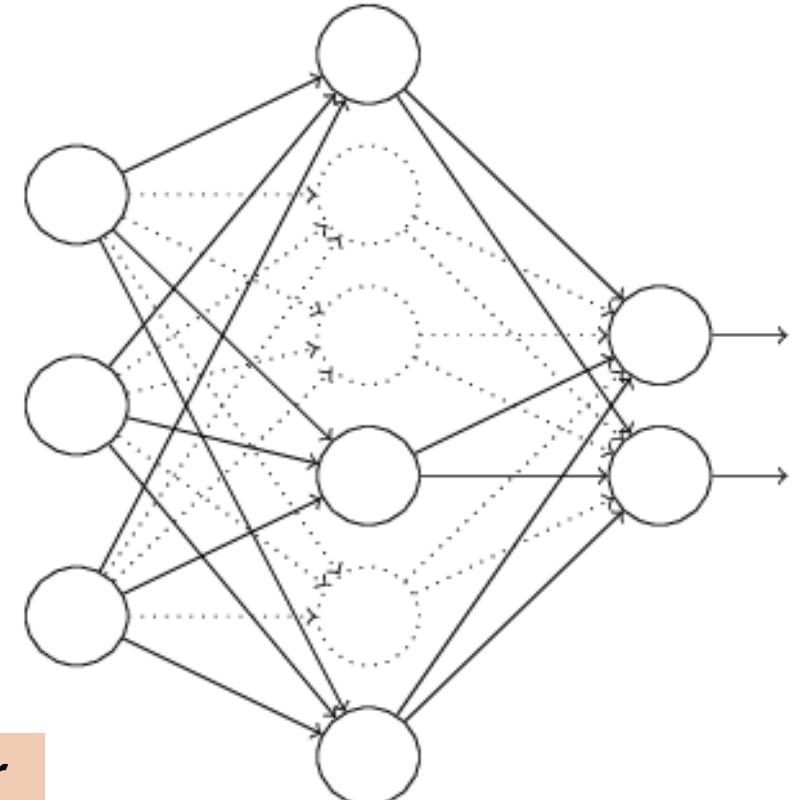
- No modifica la función de costo sino la arquitectura de la red.
- Proceso
 - Selecciona aleatoriamente las neuronas que no participarán en la próxima iteración y las “borra” temporalmente.
 - Actualiza los pesos (del mini lote si corresponde).
 - Restaura las neuronas “borradas”.
 - Repite hasta que se estabilice.



Keras dropout

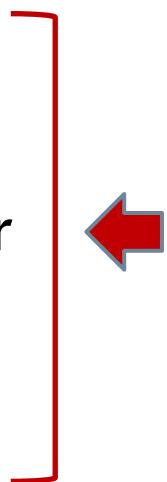
```
from keras.layers import Dense  
from keras.layers import Dropout  
...  
model.add(Dense(6, input_shape=[3]))  
model.add(Dropout(0.5))  
model.add(Dense(2))
```

*Probabilidad de anular cada entrada de la capa anterior
En este caso el 50% de las entradas serán anuladas*



Técnicas de optimización

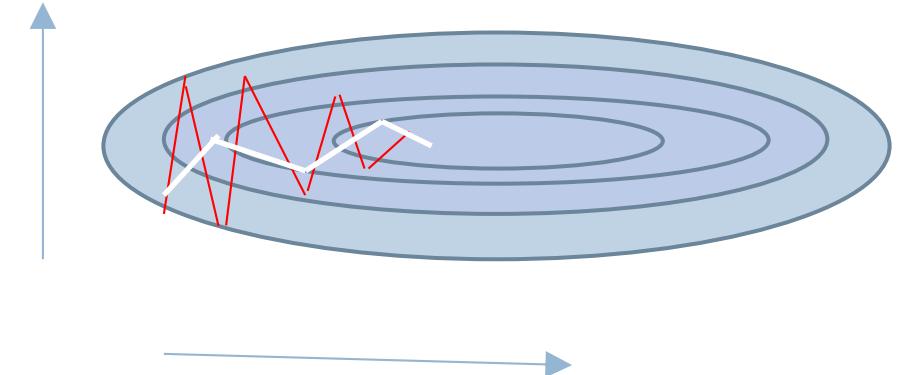
- Descenso de gradiente estocástico (SGD)
- Capacidad de generalización de la red - Sobreajuste
- Mejoras introducidas
 - Momento: utiliza información de los gradientes anteriores
 - RMSProp: considera distintas magnitudes de cambio para reducir oscilaciones
 - Adam: combina los dos anteriores. Es el más usado.



SGD con momento

$$v_t = \beta v_{t-1} + (1 - \beta)(\nabla C)_t$$

$$w_t = w_t - \alpha v_t$$



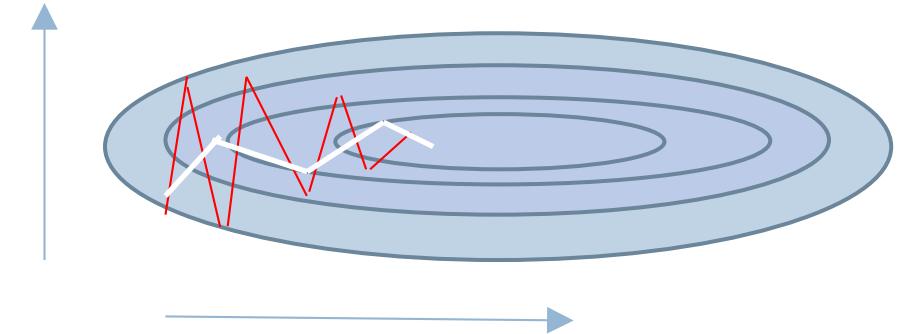
- Las modificaciones sobre W tienen en cuenta el promedio de los gradientes anteriores.
- La cantidad de gradientes anteriores a considerar son aprox. $\frac{1}{1-\beta}$
- Esto reduce las oscilaciones.

SGD con momento

$$v_t = \beta v_{t-1} + (1 - \beta)(\nabla C)_t$$

□ Usemos $\beta = 0.9$ en la iteración $t = 10$

$$v_{10} = 0.9 * v_9 + (1 - 0.9)(\nabla C)_{10}$$

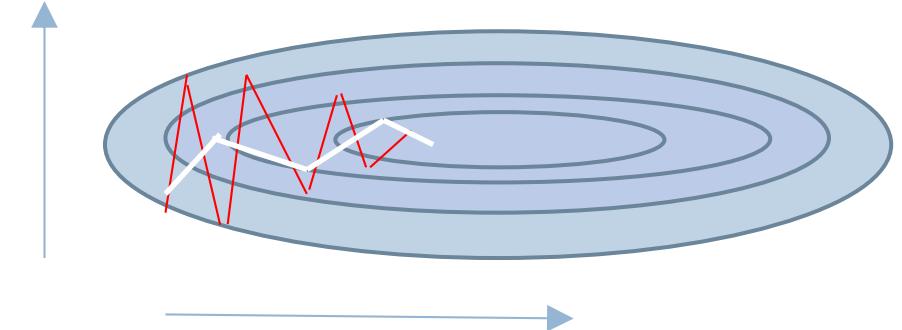


SGD con momento

$$v_t = \beta v_{t-1} + (1 - \beta)(\nabla C)_t$$

□ Usemos $\beta = 0.9$ en la iteración $t = 10$

$$v_{10} = 0.9 * v_9 + (1 - 0.9)(\nabla C)_{10} = 0.1 \nabla C_{10} + 0.9 v_9$$



SGD con momento

$$v_t = \beta v_{t-1} + (1 - \beta)(\nabla C)_t$$

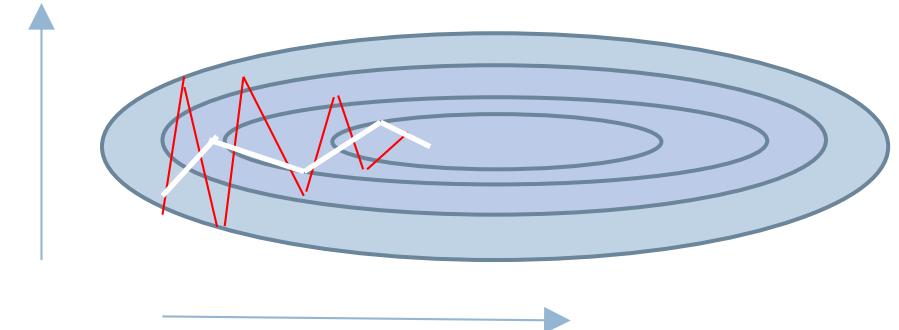
□ Usemos $\beta = 0.9$ en la iteración $t = 10$

$$v_{10} = 0.9 * v_9 + (1 - 0.9)(\nabla C)_{10} = 0.1 \nabla C_{10} + 0.9 v_9$$

$$v_{10} = 0.1 \nabla C_{10} + 0.9 (0.1 \nabla C_9 + 0.9 v_8)$$

$$v_{10} = 0.1 \nabla C_{10} + 0.1 * 0.9 \nabla C_9 + 0.9^2 v_8$$

$$v_{10} = 0.1 \nabla C_{10} + 0.1 * 0.9 \nabla C_9 + 0.9^2 (0.9 v_7 + 0.1 \nabla C_8)$$



SGD con momento

$$v_t = \beta v_{t-1} + (1 - \beta)(\nabla C)_t$$

□ Usemos $\beta = 0.9$ en la iteración $t = 10$

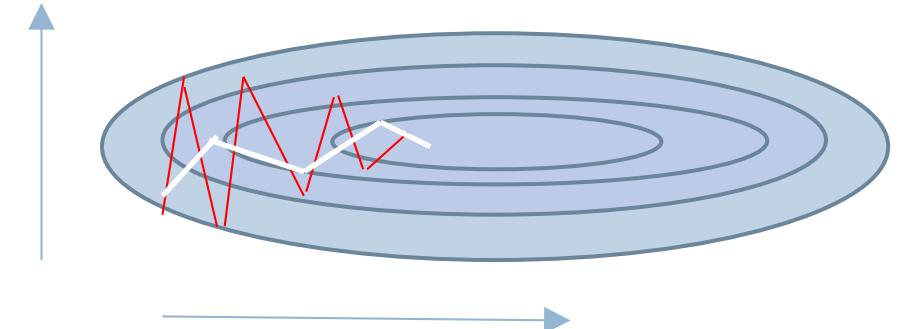
$$v_{10} = 0.9 * v_9 + (1 - 0.9)(\nabla C)_{10} = 0.1 \nabla C_{10} + 0.9 v_9$$

$$v_{10} = 0.1 \nabla C_{10} + 0.9 (0.1 \nabla C_9 + 0.9 v_8)$$

$$v_{10} = 0.1 \nabla C_{10} + 0.1 * 0.9 \nabla C_9 + 0.9^2 v_8$$

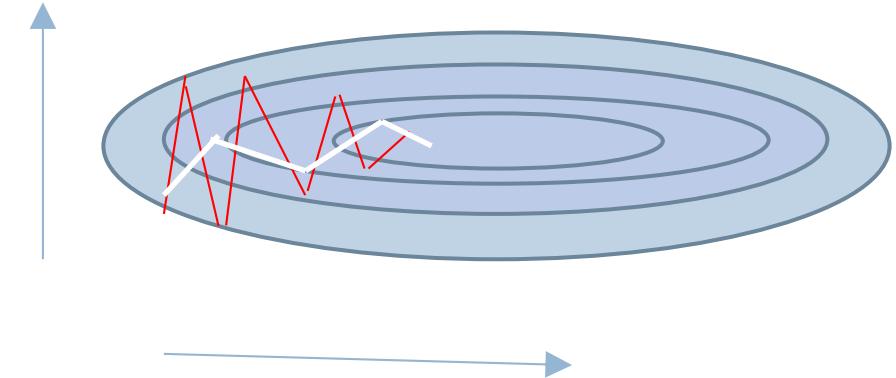
$$v_{10} = 0.1 \nabla C_{10} + 0.1 * 0.9 \nabla C_9 + 0.1 * 0.9^2 \nabla C_8 + 0.9^3 v_7 + \dots$$

La cantidad de gradientes anteriores a considerar son aprox. $\frac{1}{1-\beta}$ ∴ si $\beta=0.9$ serán aprox. 10



SGD con momento

```
vw = 0  
vb = 0  
for t in range(iteraciones):  
    Calcular gradientes  $\nabla_w$  y  $\nabla_b$   
    vw = beta * vw + (1-beta) *  $\nabla_w$   
    vb = beta * vb + (1-beta) *  $\nabla_b$   
    w = w - alfa * vw  
    b = b - alfa * vb
```

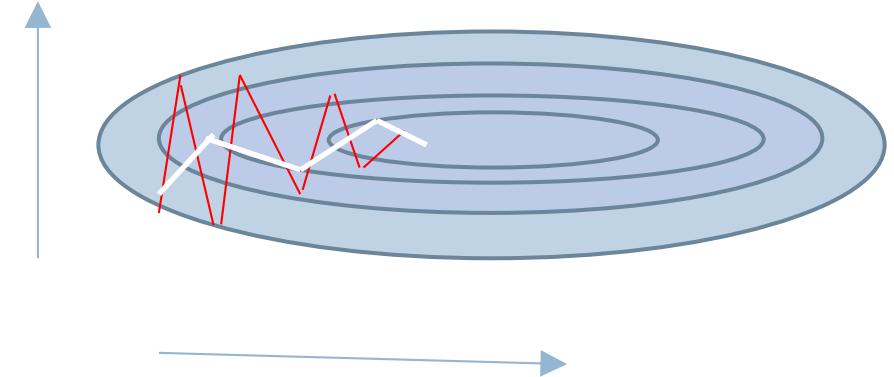


```
keras.optimizers.SGD(learning_rate=0.01, momentum=0.9)
```

RMSprop

$$s = \beta s + (1 - \beta) (\nabla C)^2$$

$$w = w - \alpha \frac{\nabla C}{\sqrt{s + \epsilon}}$$



- Las modificaciones sobre W tienen en cuenta el promedio de los gradientes anteriores.
- Las modificaciones más grandes serán divididas por coeficientes más grandes; por lo tanto se reducen.
- Las modificaciones más chicas se incrementan.

Es más eficiente que SGD+Momento

RMSprop

```
from keras.optimizers import RMSprop  
  
X,Y = cargar_datos()  
  
model = Sequential()  
model.add(...)  
  
model.compile(  
    loss='categorical_crossentropy',  
    optimizer = RMSprop(learning_rate=0.001),  
    metrics=['accuracy'])  
  
model.fit(X,Y, epochs=10, batch_size=32)
```

ADAM

<https://keras.io/api/optimizers/adam/>

- Combina momento y RMSprop

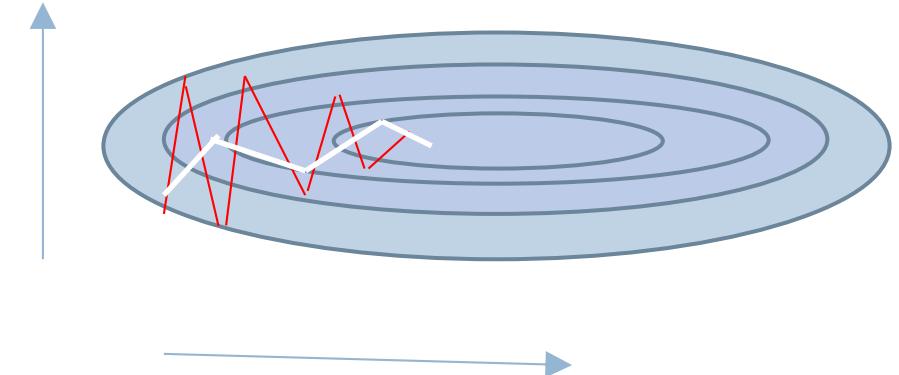
$$v = \beta_1 v + (1 - \beta_1) \nabla C$$

$$s = \beta_2 s + (1 - \beta_2) (\nabla C)^2$$

$$w = w - \alpha \frac{v}{\sqrt{s + \epsilon}}$$

- Los valores recomendados son $\beta_1 = 0.9$ y $\beta_2 = 0.999$

```
model.compile(optimizer='adam', loss='mse')
```



Evaluación del modelo

- Matriz de confusión
- Métricas
 - Accuracy
 - Precisión
 - Recall
 - F1-score
 - AUC, Curva ROC 

Clasificación binaria

- Los resultados se etiquetan como positivos (P) o negativos (N)
- Luego, la matriz de confusión tendrá la siguiente forma:

	Predice P	Predice N	
Clase P	VP	FN	$P = VP + FN$
Clase N	FP	VN	$N = FP + VN$

- Tasa de verdaderos positivos → $TVP = VP / P$ (Sensibilidad)
- Tasa de Falsos Positivos → $TFP = FP / N$ (Falsas alarmas)
- Tasa de verdaderos negativos → $TVN = VN / N$ (Especificidad)

A

VP=6	FN=6
FP=0	VN=8
6	14

$$TVP = 6/12 = 0.5$$

$$TFP = 0/8 = 0$$

B

VP=9	FN=3
FP=6	VN=2
15	5

$$TVP = 9/12 = 0.75$$

$$TFP = 6/8 = 0.75$$

C

VP=12	FN=0
FP=2	VN=6
14	6

$$TVP = 12/12 = 1$$

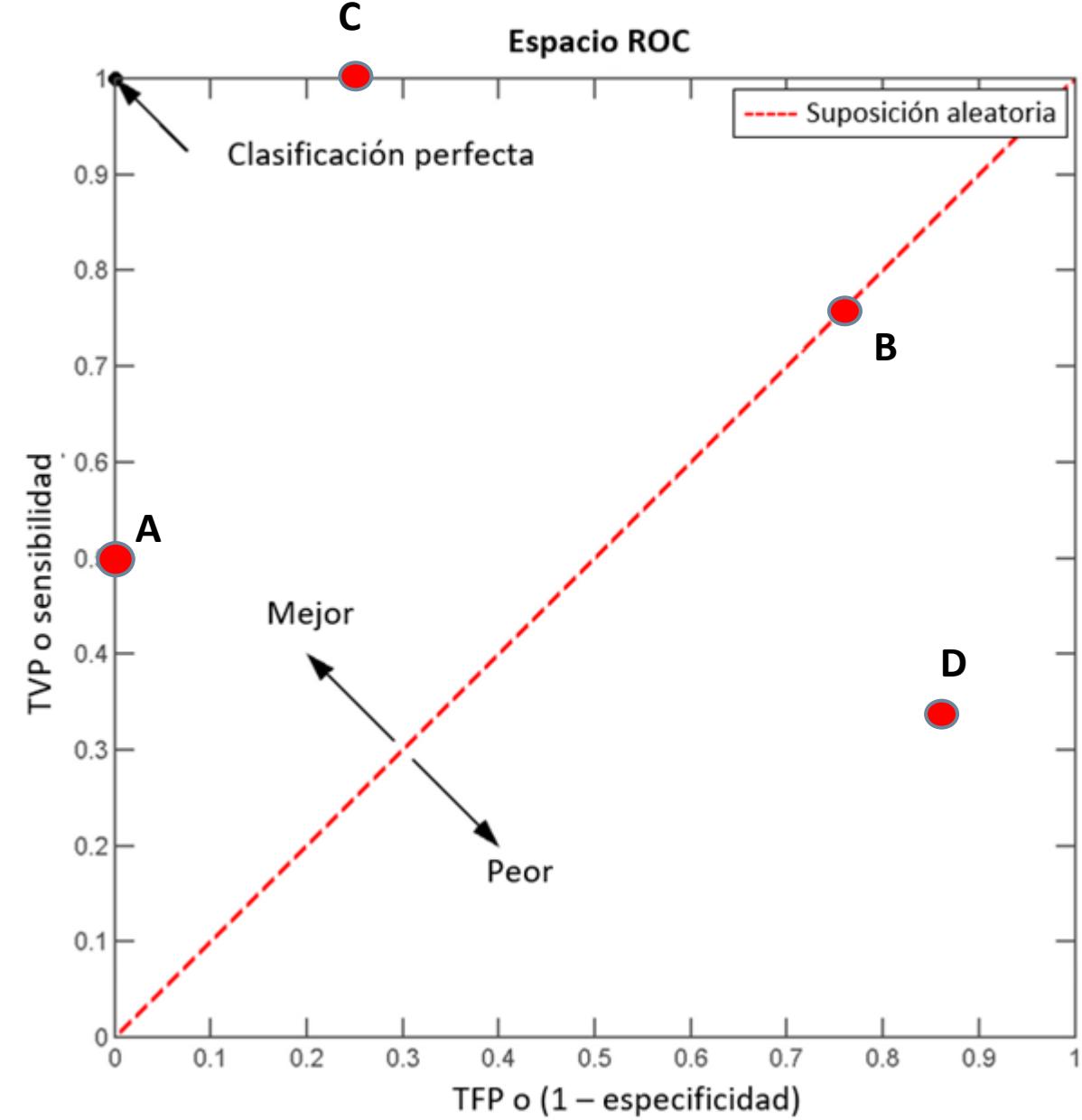
$$TFP = 2/8 = 0.25$$

D

VP=4	FN=8
FP=7	VN=1
11	9

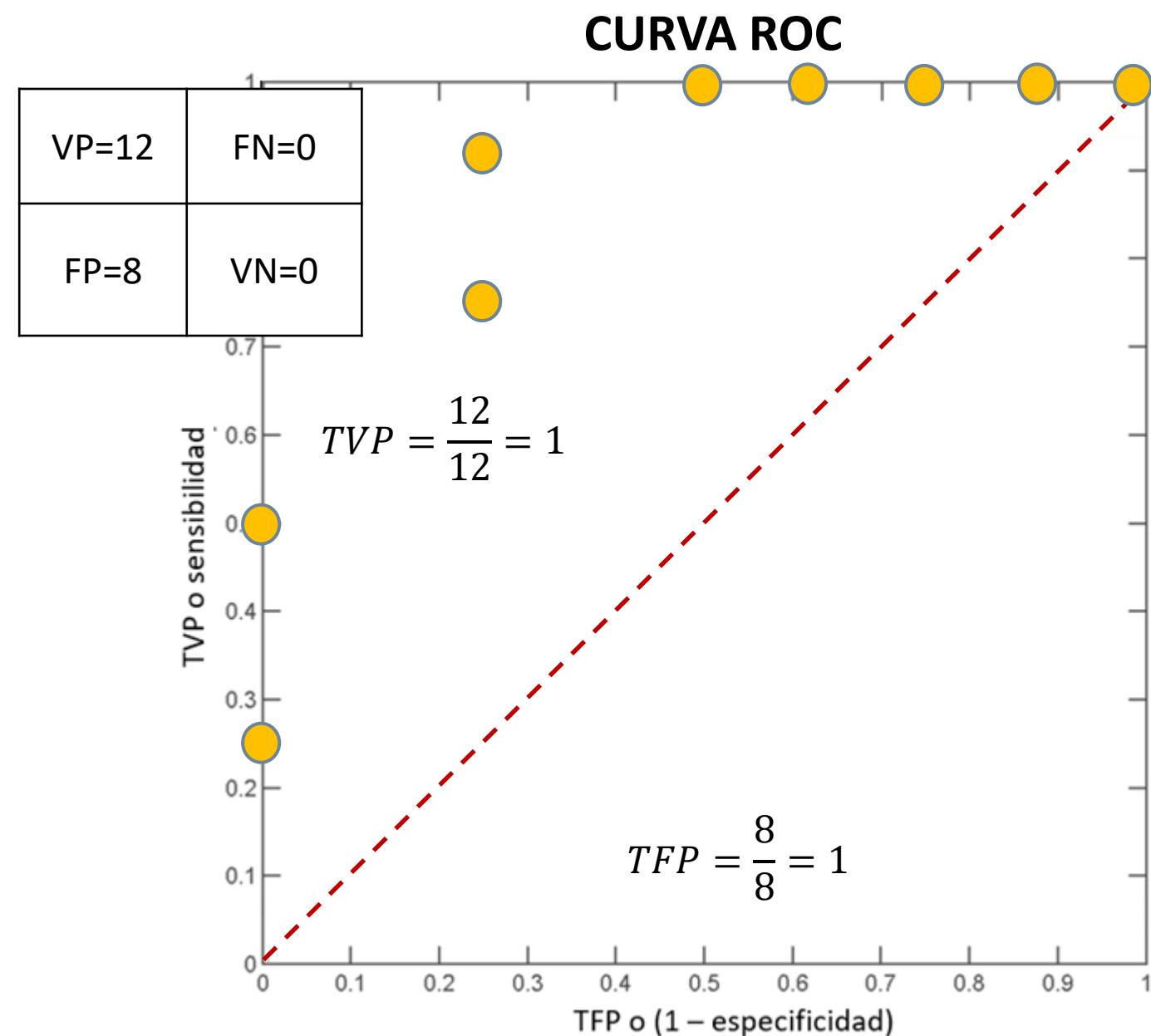
$$TVP = 4/12 = 0.33$$

$$TFP = 7/8 = 0.875$$

C

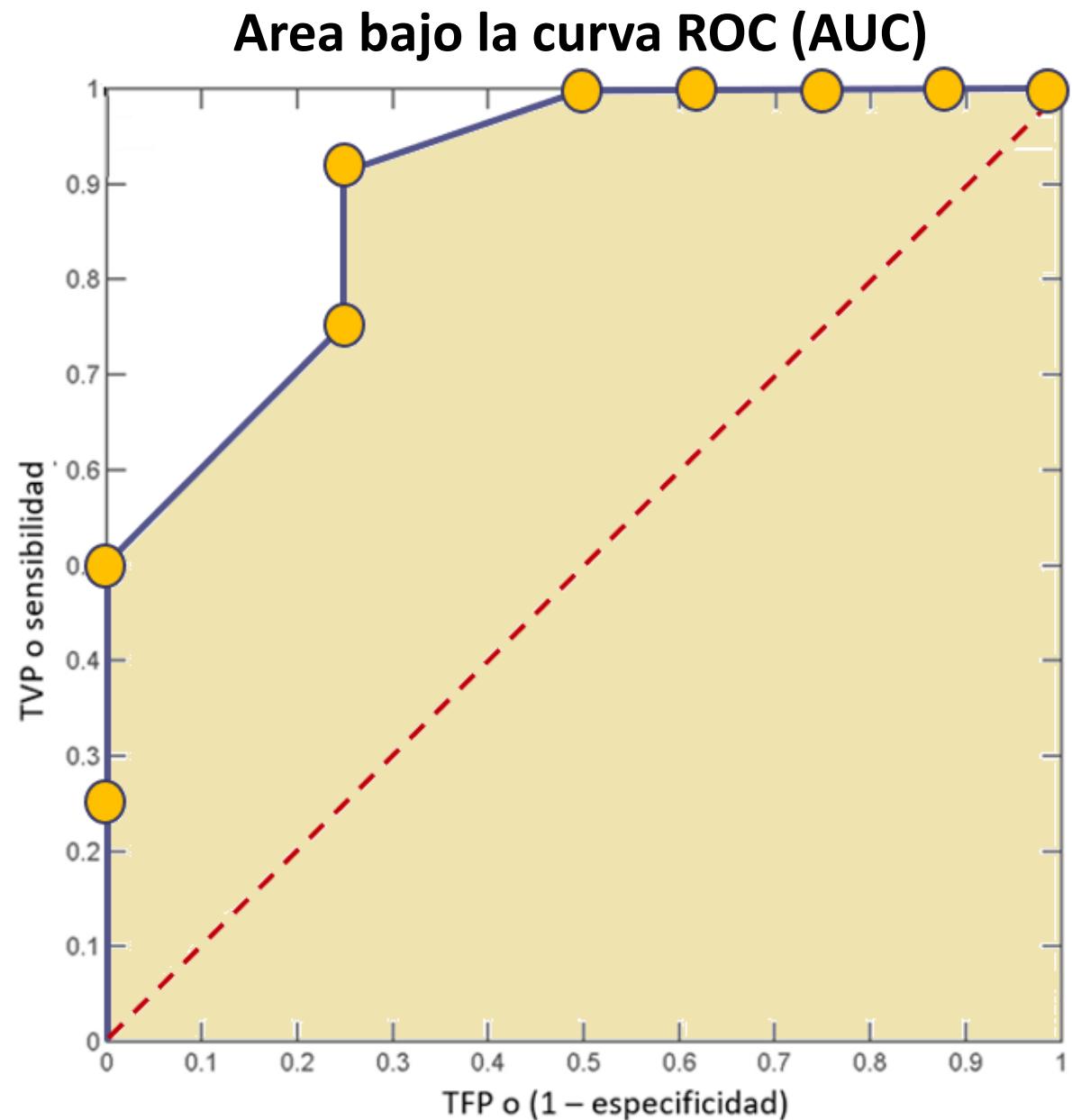
ID	Clase	Confianza	Predice
5	Mina	0.99	Mina
7	Mina	0.99	Mina
9	Mina	0.99	Mina
1	Mina	0.9	Mina
10	Mina	0.9	Mina
20	Mina	0.9	Mina
8	Roca	0.8	Mina
14	Mina	0.8	Mina
15	Mina	0.8	Mina
18	Roca	0.8	Mina
19	Mina	0.8	Mina
3	Mina	0.7	Mina
6	Mina	0.7	Mina
12	Mina	0.65	Mina
4	Roca	0.6	Mina
16	Roca	0.6	Mina
11	Roca	0.5	Mina
2	Roca	0.4	Mina
13	Roca	0.3	Mina
17	Roca	0.1	Mina

12 minas y 8 rocas



ID	Clase	Confianza	Predice
5	Mina	0.99	
7	Mina	0.99	
9	Mina	0.99	
1	Mina	0.9	
10	Mina	0.9	
20	Mina	0.9	
8	Roca	0.8	
14	Mina	0.8	
15	Mina	0.8	
18	Roca	0.8	
19	Mina	0.8	
3	Mina	0.7	
6	Mina	0.7	
12	Mina	0.65	
4	Roca	0.6	
16	Roca	0.6	
11	Roca	0.5	
2	Roca	0.4	
13	Roca	0.3	
17	Roca	0.1	

12 minas y 8 rocas

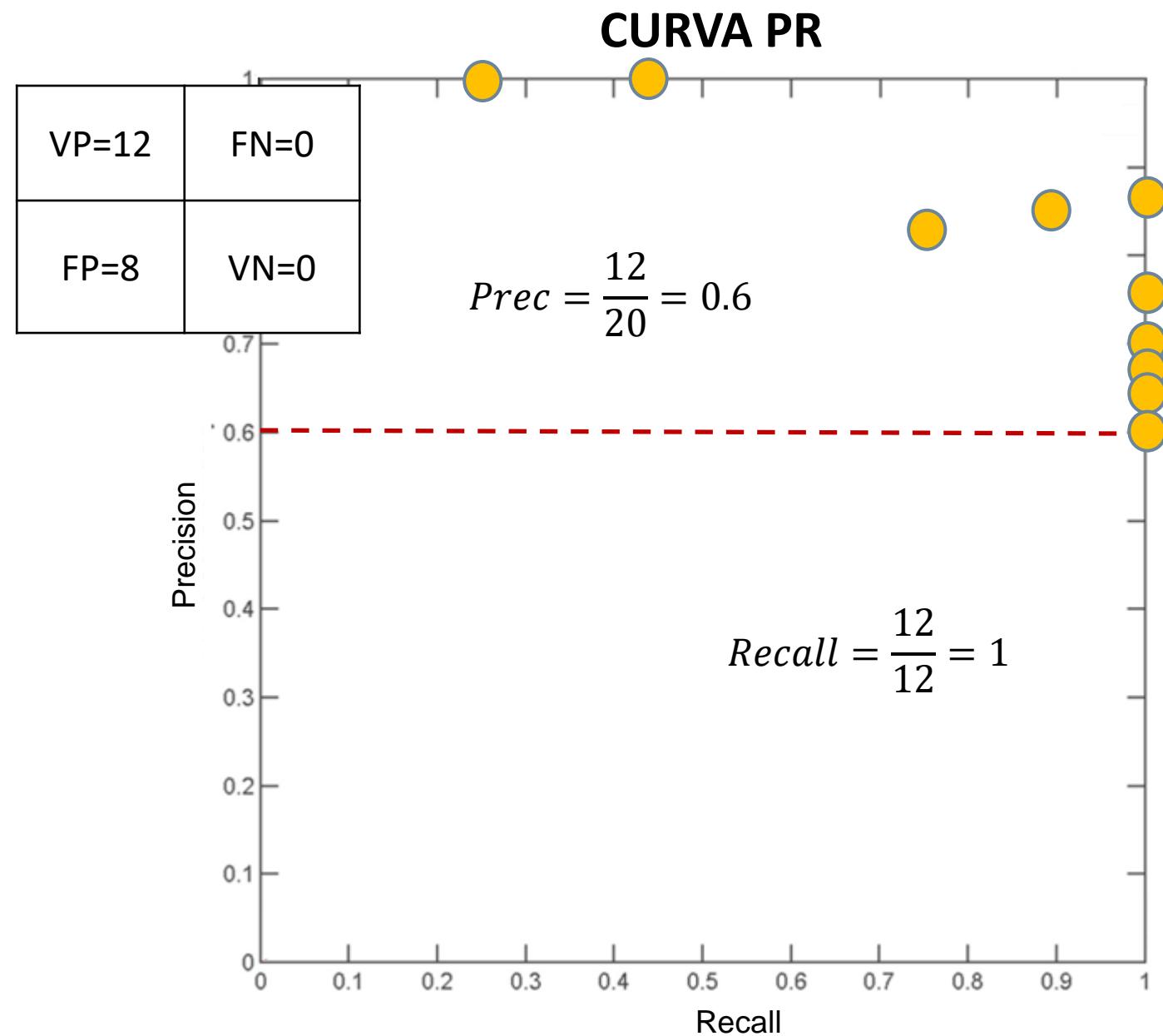


AUC ROC

- **AUC (Area Under the Curve)** ROC es el área bajo la curva ROC.
 - Representa la capacidad de un modelo para diferenciar entre clases.
 - Un valor de AUC ROC cercano a 1 indica un buen rendimiento, mientras que un valor de 0.5 indica un rendimiento similar al de una clasificación aleatoria.
- **¿Cuándo se usa AUC ROC?**
 - Se utiliza cuando las clases están **más o menos balanceadas** o cuando se quiere medir qué tan bien un modelo puede **separar entre las clases positivas y negativas**.
 - Es útil para ver el **rendimiento general del modelo** a través de diferentes umbrales, especialmente en casos donde se puede ajustar el umbral de clasificación.

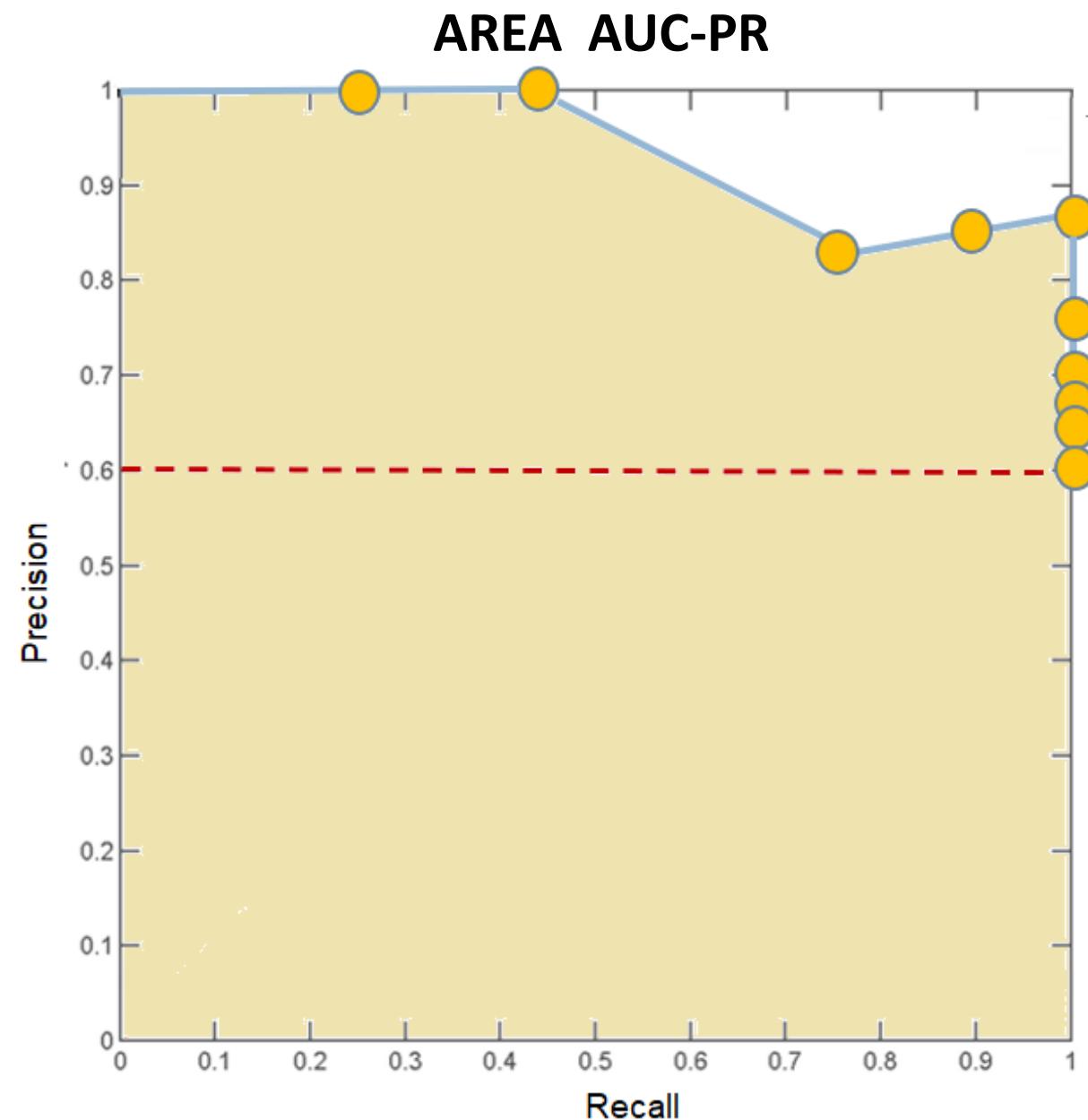
ID	Clase	Confianza	Predice
5	Mina	0.99	Mina
7	Mina	0.99	Mina
9	Mina	0.99	Mina
1	Mina	0.9	Mina
10	Mina	0.9	Mina
20	Mina	0.9	Mina
8	Roca	0.8	Mina
14	Mina	0.8	Mina
15	Mina	0.8	Mina
18	Roca	0.8	Mina
19	Mina	0.8	Mina
3	Mina	0.7	Mina
6	Mina	0.7	Mina
12	Mina	0.65	Mina
4	Roca	0.6	Mina
16	Roca	0.6	Mina
11	Roca	0.5	Mina
2	Roca	0.4	Mina
13	Roca	0.3	Mina
17	Roca	0.1	Mina

12 minas y 8 rocas



ID	Clase	Confianza	Predice
5	Mina	0.99	Mina
7	Mina	0.99	Mina
9	Mina	0.99	Mina
1	Mina	0.9	Mina
10	Mina	0.9	Mina
20	Mina	0.9	Mina
8	Roca	0.8	Mina
14	Mina	0.8	Mina
15	Mina	0.8	Mina
18	Roca	0.8	Mina
19	Mina	0.8	Mina
3	Mina	0.7	Mina
6	Mina	0.7	Mina
12	Mina	0.65	Mina
4	Roca	0.6	Mina
16	Roca	0.6	Mina
11	Roca	0.5	Mina
2	Roca	0.4	Mina
13	Roca	0.3	Mina
17	Roca	0.1	Mina

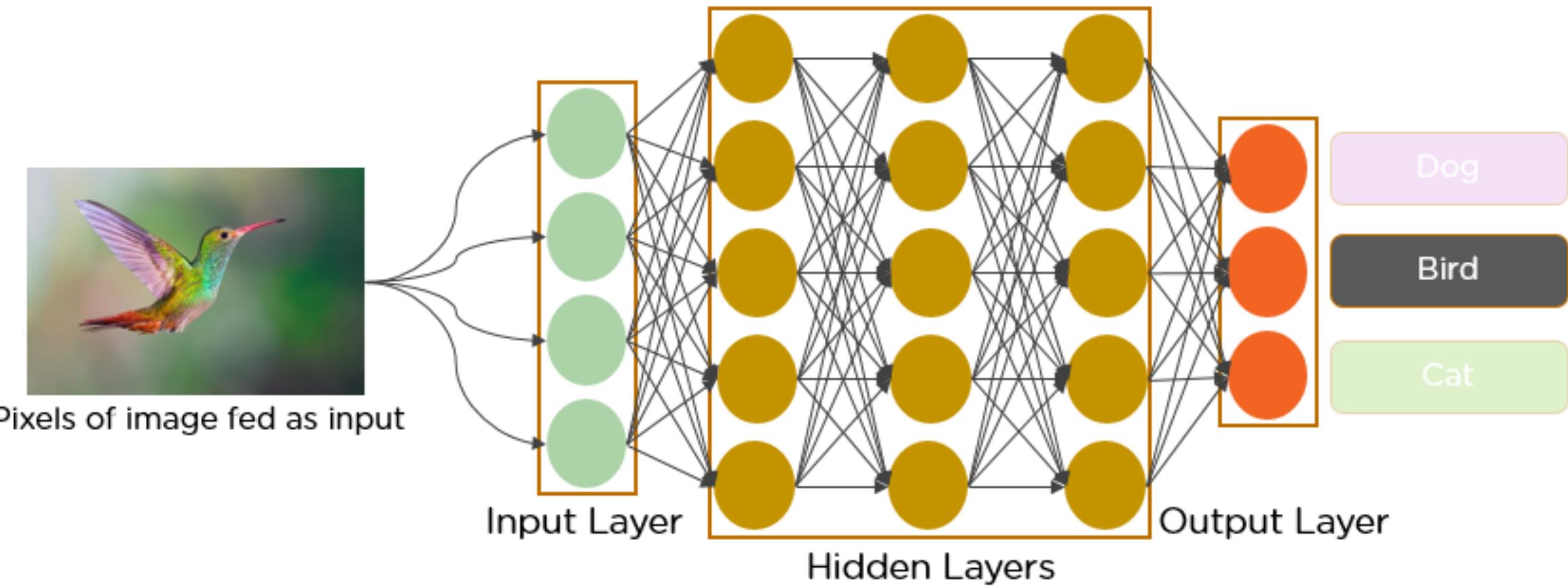
12 minas y 8 rocas



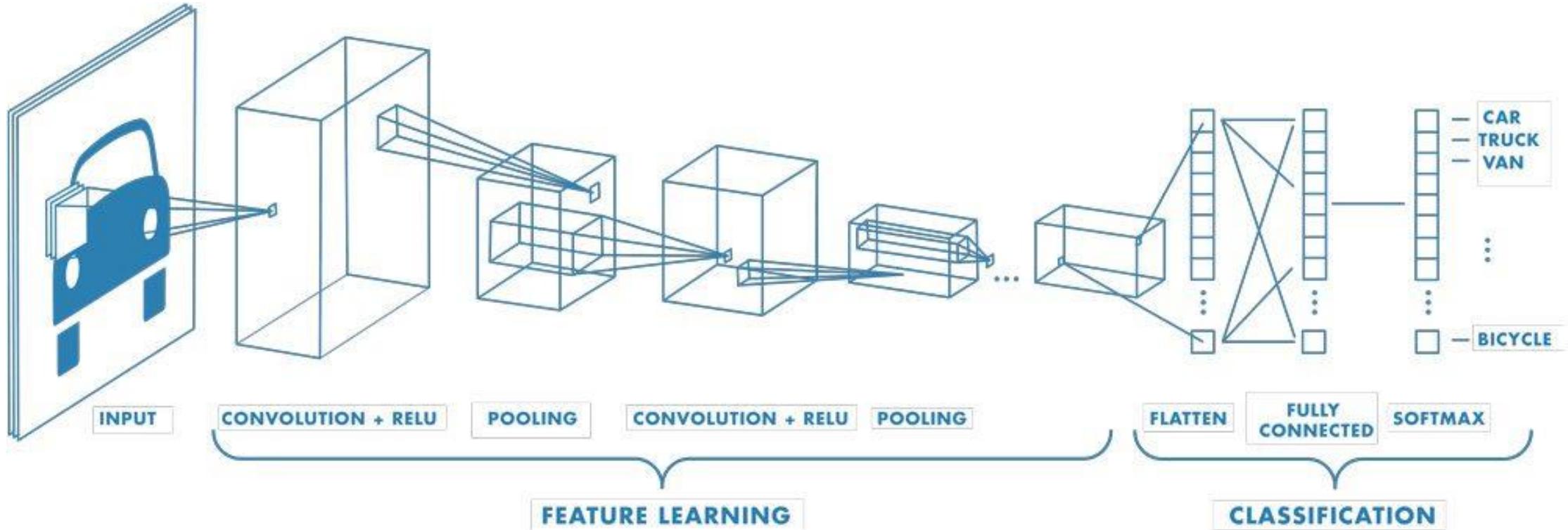
AUC PR

- **AUC PR** es el área bajo la curva Precision-Recall.
 - Esta métrica es útil cuando se mide el **balance entre la precisión y el recall**.
 - En problemas donde hay un gran desbalance de clases, AUC PR proporciona una visión más ajustada de cómo está funcionando el modelo en la clase minoritaria.
- **¿Cuándo se usa AUC PR?**
 - Se utiliza cuando las clases están **desbalanceadas**.
 - Si las clases están **desbalanceadas** (ej: se tienen muchos más negativos que positivos), el **FPR** puede ser muy bajo simplemente porque hay pocos falsos positivos en comparación con la gran cantidad de verdaderos negativos. Esto puede hacer que el AUC ROC parezca alto, incluso si el modelo no está funcionando bien en la clase minoritaria (positiva).
 - En este tipo de escenarios, el AUC ROC puede ser **engañoso**, porque un buen rendimiento en la clase mayoritaria puede ocultar el mal desempeño en la clase minoritaria.

Clasificación de imágenes



Red Neuronal Convolucional



Filtros en el dominio espacial

- En este proceso se relaciona un conjunto de píxeles próximos al píxel objetivo con la finalidad de obtener una información útil



*Usaremos máscaras o kernels de **convolución***

Convolución 2D

- La operación de convolución de una imagen con un filtro o kernel permite destacar ciertas características de dicha imagen.



-1	-1	-1
-1	8	-1
-1	-1	-1



Convolución 2D

- Filtro de detección de bordes horizontal



1	1	1
0	0	0
-1	-1	-1



Convolución 2D

- Filtro de detección de bordes diagonal



-1	0
0	1



Convolución 2D

- Filtro de detección de bordes diagonal



0	-1
1	0

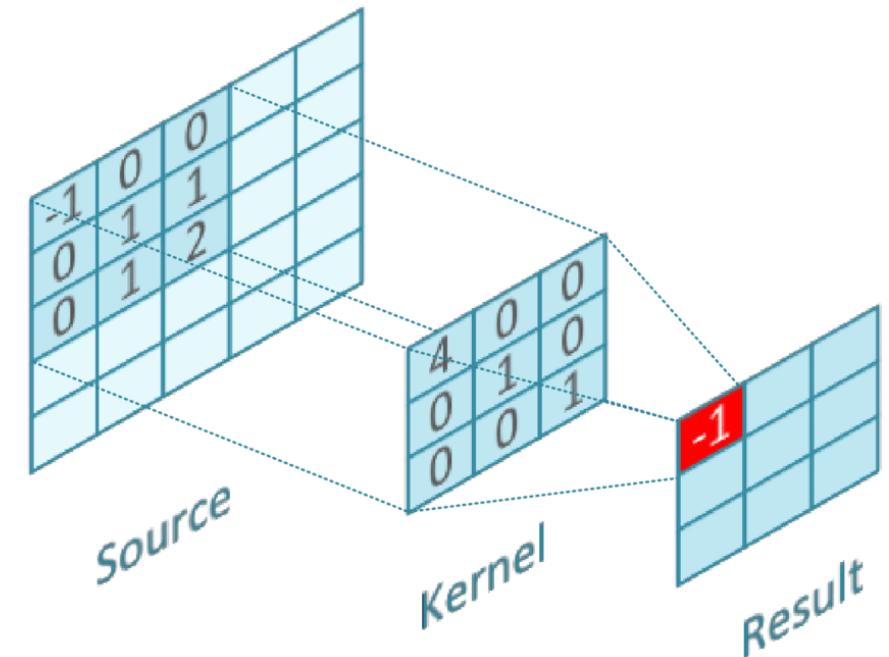


Convolución 2D

- La convolución discreta de dos funciones f y g se define como

$$(f * g)[x, y] = \sum_{n_1=-\infty}^{\infty} \sum_{n_2=-\infty}^{\infty} f[n_1, n_2] \cdot g[x - n_1, y - n_2]$$

- La función g se desplaza antes de multiplicar.



Convolución 2D

Entrada

60	113	56	139	85
73	121	54	84	128
131	99	70	129	127
80	57	115	69	134
104	126	123	95	130

Kernel

0	-1	0
-1	5	-1
0	-1	0

Salida

	266	-61	-30	
	116	-47	295	
	-135	256	-128	

□ Parámetros

- **Kernel_size:** tamaño del filtro o kernel. En este caso =3
- **Stride:** desplazamiento del filtro cada vez que se aplica. En este caso = 1

Convolución 2D - Padding

- Usando padding conservamos el tamaño de la imagen original

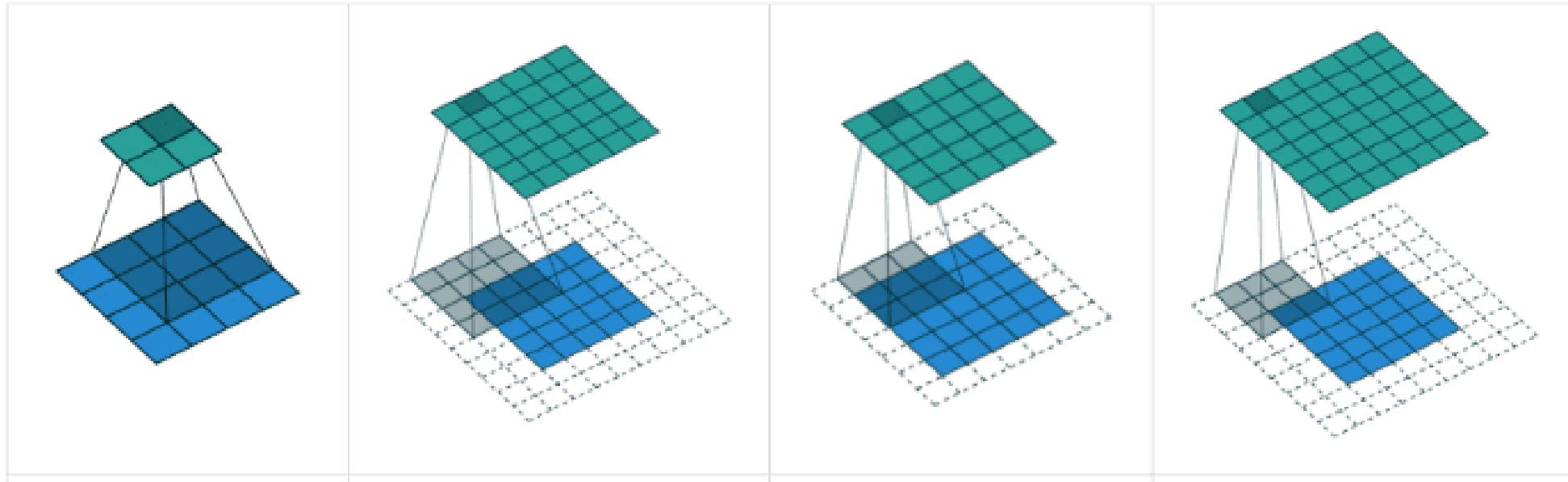
0	0	0	0	0	0	0
0	60	113	56	139	85	0
0	73	121	54	84	128	0
0	131	99	70	129	127	0
0	80	57	115	69	134	0
0	104	126	123	95	130	0
0	0	0	0	0	0	0

Kernel

0	-1	0
-1	5	-1
0	-1	0

114				

Convolución 2D - Padding



Resultado 2x2
(4-3+1)

"valid"

Resultado 6x6
(5+2*2-4+1)

Resultado 5x5
(5+2*1-3+1)

"same"

Resultado 7x7
(5+2*2-3+1)

Filtros y Extracción de Características

2	0	2	0	2
0	3	0	3	0
2	0	5	0	2
0	3	0	3	0
2	0	2	0	2

2	0	0	0	0
0	3	0	0	0
0	0	1	0	0
0	0	0	3	0
0	0	0	0	2

0	0	0	0	2
0	0	0	3	0
0	0	1	0	0
0	3	0	0	0
2	0	0	0	0

- ¿Qué representa cada nuevo valor?

Cada nuevo valor representa el grado de coincidencia entre el filtro y la sección correspondiente de la imagen original

Filtros y Extracción de Características

2	0	2	0	2
0	3	0	3	0
2	0	5	0	2
0	3	0	3	0
2	0	2	0	2

2	0	0	0	0
0	3	0	0	0
0	0	1	0	0
0	0	0	3	0
0	0	0	0	2

0	0	0	0	2
0	0	0	3	0
0	0	1	0	0
0	3	0	0	0
2	0	0	0	0

- ¿El resultado del filtro es una imagen?

Si, pero el nuevo valor es una intensidad relacionada con la coincidencia del filtro

Filtros y Extracción de Características

2	0	2	0	2
0	3	0	3	0
2	0	5	0	2
0	3	0	3	0
2	0	2	0	2

2	0	0	0	0
0	3	0	0	0
0	0	1	0	0
0	0	0	3	0
0	0	0	0	2

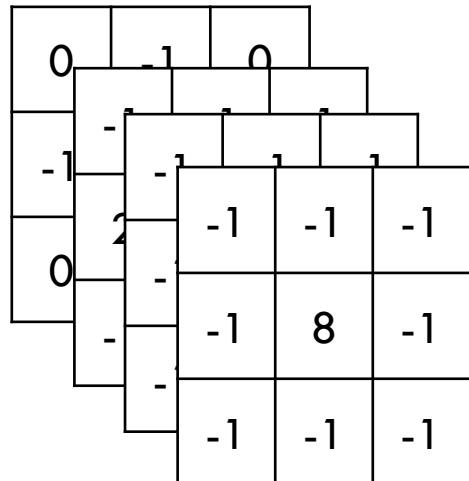
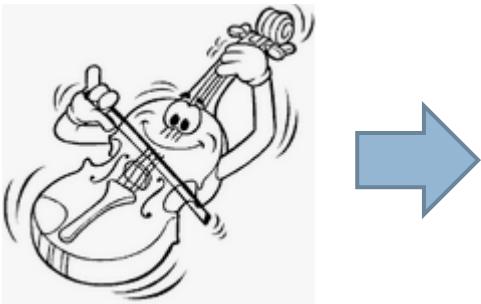
0	0	0	0	2
0	0	0	3	0
0	0	1	0	0
0	3	0	0	0
2	0	0	0	0

- Si aplicamos un nuevo filtro al resultado, ¿qué sucede?

Un nuevo filtro relaciona las características de filtros anteriores, agregando un nivel de abstracción en la interpretación de la imagen

Capa convolucional

- Está formada por uno o varios filtros.
- Su función es hacer la convolución de cada filtro sobre cada ejemplo de entrada.



```
Conv2D(cant_filtros,  
       kernel_size=k,  
       strides=(n,m)  
       activation='relu',  
       padding='same')
```

Input Volume (+pad 1) (7x7x3)

 $x[:, :, 0]$

0	0	0	0	0	0	0
0	0	0	1	0	2	0
0	1	0	2	0	1	0
0	1	0	2	2	0	0
0	2	0	0	2	0	0
0	2	1	2	2	0	0
0	0	0	0	0	0	0

 $x[:, :, 1]$

0	0	0	0	0	0	0
0	2	1	2	1	1	0
0	2	1	2	0	1	0
0	0	2	1	0	1	0
0	1	2	2	2	2	0
0	0	1	2	0	1	0
0	0	0	0	0	0	0

 $x[:, :, 2]$

0	0	0	0	0	0	0
0	2	1	1	2	0	0
0	1	0	0	1	0	0
0	0	1	0	0	0	0
0	1	0	2	1	0	0
0	2	2	1	1	1	0
0	0	0	0	0	0	0

Filter W0 (3x3x3)

 $w0[:, :, 0]$

-1	0	1
0	0	1
1	-1	1

 $w0[:, :, 1]$

-1	0	1
1	-1	0
0	1	0

Bias $b0 (1 \times 1 \times 1)$
 $b0[:, :, 0]$

1

Filter W1 (3x3x3)

 $w1[:, :, 0]$

0	1	-1
0	-1	0
0	-1	1

 $w1[:, :, 1]$

-1	0	0
1	-1	0
1	-1	0

 $w1[:, :, 2]$

-1	1	-1
0	-1	-1
1	0	0

Output Volume (3x3x2)

 $o[:, :, 0]$

2	3	3
3	7	3
8	10	-3
-8	-8	-3
-3	1	0
-3	-8	-5

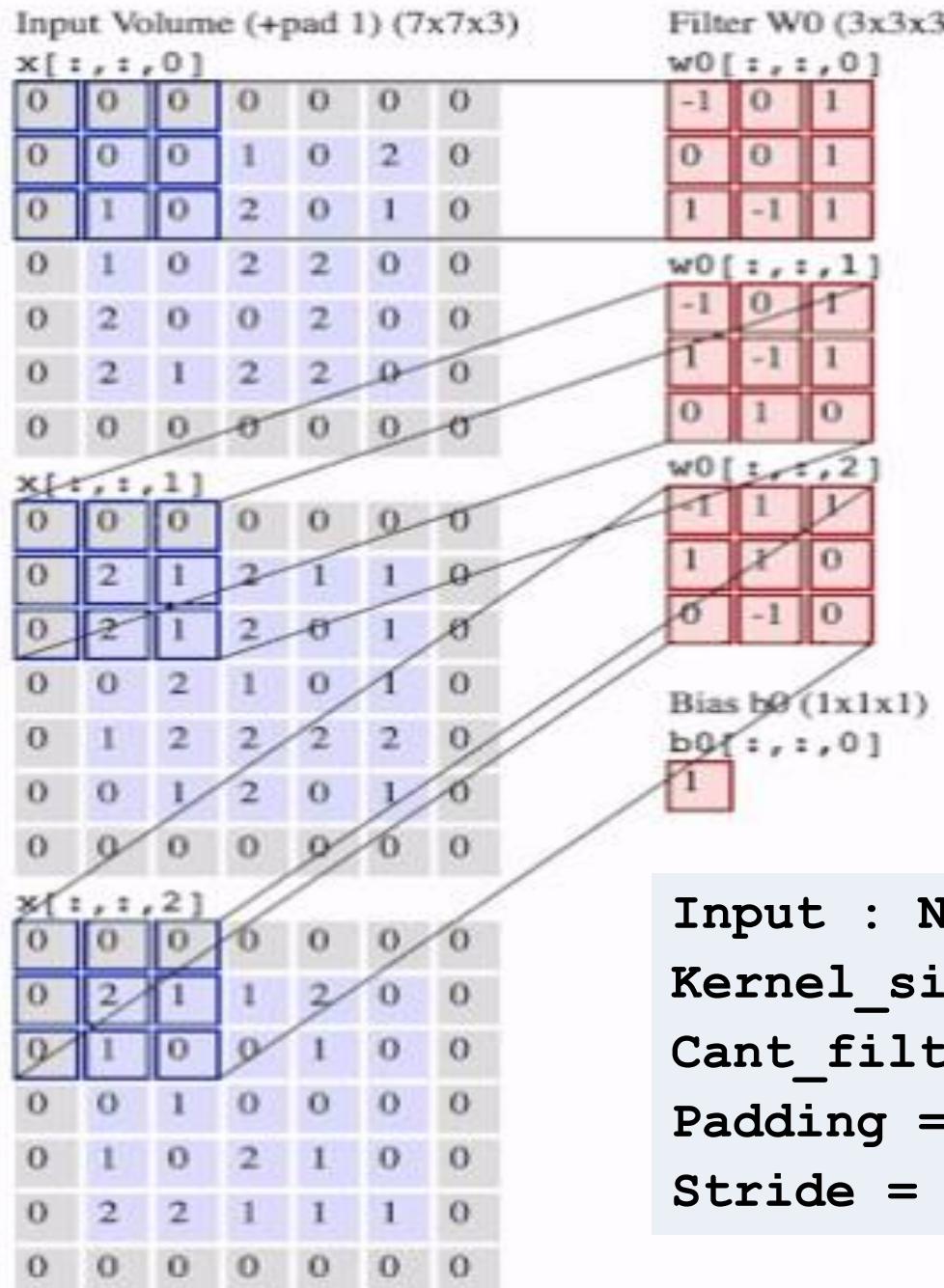
 $b1 (1 \times 1 \times 1)$

0

Input : $N \times M \times N_c = 5 \times 5 \times 3$ Kernel_size= $K \times K \times N_c = 3 \times 3 \times 3$ Cant_filtros = $N_f = 2$

Padding = P = 1

Stride = S = 2



Filter W1 (3x3x3)

w1[::, ::, 1]							
-1	0	0					
1	-1	0					
1	-1	0					

Output Volume (3x3x2)

o[::, ::, 1]							
-8	-8	-3					
-3	1	0					
-3	-8	-5					

Output : $T \times U \times N_f = 3 \times 3 \times 2$

$$T = (N + 2*P - K) / S + 1$$

$$= (5 + 2*1 - 3) / 2 + 1 = 3$$

$$U = (M + 2*P - K) / S + 1 = 3$$

Input : $N \times M \times N_c = 5 \times 5 \times 3$

Kernel_size= $K \times K \times N_c = 3 \times 3 \times 3$

Cant_filtros = $N_f = 2$

Padding = $P = 1$

Stride = $S = 2$

MNIST

```
model = Sequential()
model.add(Conv2D(64, kernel_size=3, activation="relu",
                 input_shape=input_shape))
model.add(Flatten())
model.add(Dense(10, activation='softmax'))
model.summary()
```

Layer (type)	Output Shape	Param #
conv2d_4 (Conv2D)	(None, 26, 26, 64)	640
flatten_10 (Flatten)	(None, 43264)	0
dense_10 (Dense)	(None, 10)	432650
Total params: 433,290		

¿Por qué la salida es de 26x26 si las imágenes son de 28x28?

$$(N + 2*P - K)/S + 1$$

$$(28 + 0 - 3)/1 + 1 = 26$$

MNIST

```
model = Sequential()  
model.add(Conv2D(64, kernel_size=3, activation="relu",  
                input_shape=input_shape))  
model.add(Flatten())  
model.add(Dense(10, activation='softmax'))  
model.summary()
```

Layer (type)	Output Shape	Param #
conv2d_4 (Conv2D)	(None, 26, 26, 64)	640
flatten_10 (Flatten)	(None, 43264)	0
dense_10 (Dense)	(None, 10)	432650
Total params: 433,290		

¿Por qué la capa convolucional tiene 640 parámetros?

$$N_f * K * K * N_c + N_f$$
$$64 * \underbrace{3 * 3}_{\text{Tamaño del filtro}} + 64$$

↑ Cantidad de filtros ↑ Tamaño del filtro ↑ Bias

MNIST

```
model = Sequential()  
model.add(Conv2D(64, kernel_size=3, activation="relu",  
                input_shape=input_shape))  
model.add(Flatten())  
model.add(Dense(10, activation='softmax'))  
model.summary()
```

Layer (type)	Output Shape	Param #
conv2d_4 (Conv2D)	(None, 26, 26, 64)	640
flatten_10 (Flatten)	(None, 43264)	0
dense_10 (Dense)	(None, 10)	432650
Total params: 433,290		

Cantidad de parámetros de la capa Flatten

$$\begin{array}{c} N_f \quad * \quad T \quad * \quad U \\ 64 \quad * \quad \underbrace{26 \quad * \quad 26}_{\text{Tamaño de la imagen filtrada}} \\ \uparrow \qquad \qquad \qquad \end{array}$$

Cantidad de filtros

MNIST

```
model = Sequential()
model.add(Conv2D(64, kernel_size=3, activation="relu", input_shape=input_shape))
model.add(Flatten())
model.add(Dense(15, activation='tanh'))
model.add(Dense(10, activation='softmax'))
model.summary()
```

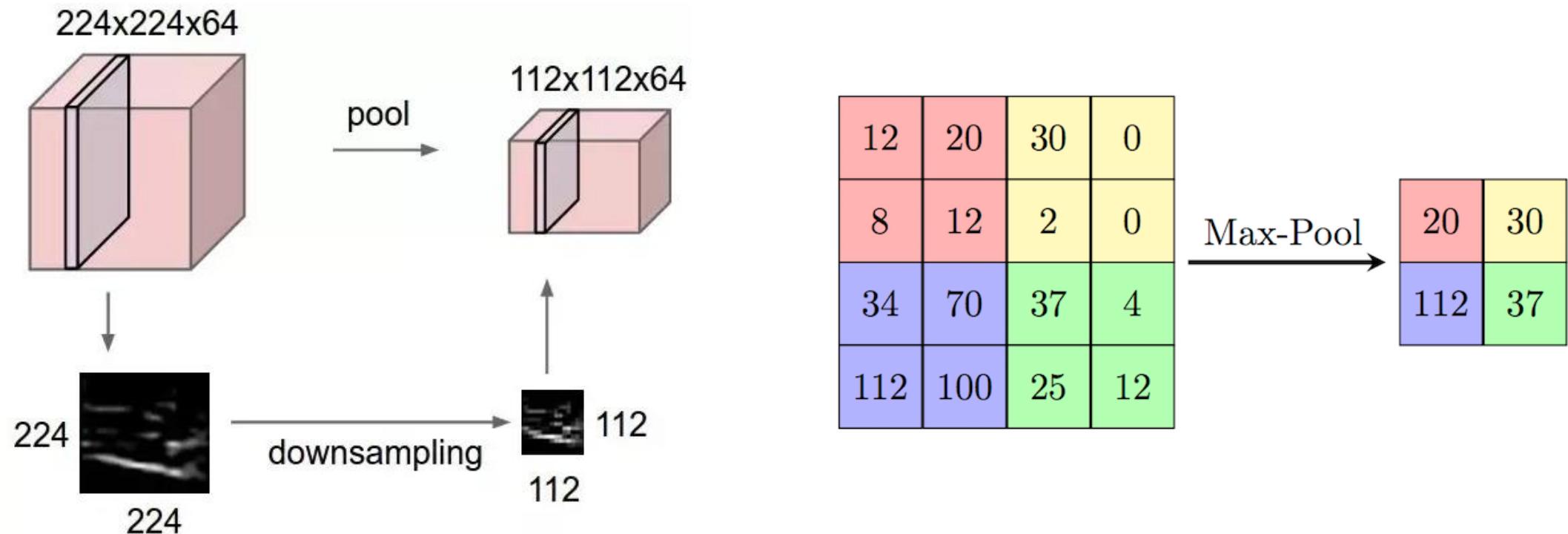
Layer (type)	Output Shape	Param #
conv2d (Conv2D)	(None, 26, 26, 64)	640
flatten (Flatten)	(None, 43264)	0
dense (Dense)	(None, 15)	648975
dense_1 (Dense)	(None, 10)	160
Total params: 649,775		

Cantidad de parámetros o pesos de la capa oculta del multiperceptrón

$$43264 * 15 + 15$$

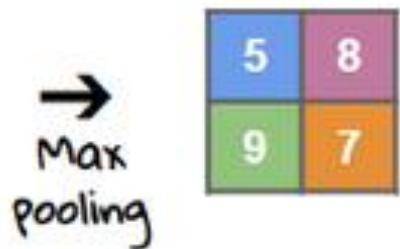
Capa de pooling

- La capa de pooling reduce el tamaño de la salida de la capa convolucional. Se trata de una convolución con un stride igual al tamaño del kernel que calcula la función sobre todos los pixels.

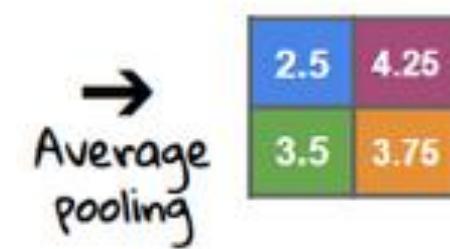


Pooling

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



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



- La reducción de tamaño permite eliminar parte del ruido y extraer datos más significativos.
- Reduce el exceso de ajuste y acelera el cálculo

Capa MaxPooling2D

```
model = Sequential()
model.add(Conv2D(64, kernel_size=3, activation="relu", input_shape=input_shape))
model.add(MaxPooling2D(pool_size=(2,2)))
model.add(Flatten())
model.add(Dense(15, activation='tanh'))
model.add(Dense(10, activation='softmax'))
model.summary()
```

Layer (type)	Output Shape	Param #
=====		
conv2d_1 (Conv2D)	(None, 26, 26, 64)	640
=====		
max_pooling2d (MaxPooling2D)	(None, 13, 13, 64)	0
=====		
flatten_1 (Flatten)	(None, 10816)	0
=====		
dense_2 (Dense)	(None, 15)	162255
=====		
dense_3 (Dense)	(None, 10)	160
=====		
Total params: 163,055		,

Luego de aplanar la cantidad de entradas se redujo un 75%

Antes eran 43264 y ahora quedaron 10816

MNIST

```
model = Sequential()
model.add(Input(shape=(28, 28, 1)))
model.add(Conv2D(32, kernel_size=3, strides=(1,1), activation='relu'))
model.add(MaxPooling2D(pool_size=(2,2), name='max_pool_1'))
model.add(Conv2D(16, kernel_size=3, strides=(1,1), activation='relu'))
model.add(MaxPooling2D(pool_size=(2,2), name='max_pool_2'))
model.add(Flatten())
model.add(Dense(10, activation='softmax'))
model.summary()
```

¿Por qué la salida es de
26x26 si las imágenes son
de 28x28?

Layer (type)	Output Shape	Param #
conv2d_2 (Conv2D)	(None, 26, 26, 32)	320
max_pool_1 (MaxPooling2D)	(None, 13, 13, 32)	0
conv2d_3 (Conv2D)	(None, 11, 11, 16)	4624
max_pool_2 (MaxPooling2D)	(None, 5, 5, 16)	0
flatten_1 (Flatten)	(None, 400)	0
dense_1 (Dense)	(None, 10)	4010
<hr/>		
Total params: 8,954		

$$(N + 2*P - K)/S + 1$$

$$(28 + 0 - 3)/1 + 1 = 26$$

MNIST

```
model = Sequential()
model.add(Input(shape=(28, 28, 1)))
model.add(Conv2D(32, kernel_size=3, strides=(1,1), activation='relu'))
model.add(MaxPooling2D(pool_size=(2,2), name='max_pool_1'))
model.add(Conv2D(16, kernel_size=3, strides=(1,1), activation='relu'))
model.add(MaxPooling2D(pool_size=(2,2), name='max_pool_2'))
model.add(Flatten())
model.add(Dense(10, activation='softmax'))
model.summary()
```

¿Por qué la capa convolucional tiene 320 parámetros?

Layer (type)	Output Shape	Param #
conv2d_2 (Conv2D)	(None, 26, 26, 32)	320
max_pool_1 (MaxPooling2D)	(None, 13, 13, 32)	0
conv2d_3 (Conv2D)	(None, 11, 11, 16)	4624
max_pool_2 (MaxPooling2D)	(None, 5, 5, 16)	0
flatten_1 (Flatten)	(None, 400)	0
dense_1 (Dense)	(None, 10)	4010

Total params: 8,954

$$N_f * K * K * N_c + N_f$$

32 * 3 * 3 * 1 + 32

↑ Cantidad de filtros ↓ Tamaño del filtro ↗ Bias

MNIST

```
model = Sequential()  
model.add(Input(shape=(28, 28, 1)))  
model.add(Conv2D(32, kernel_size=3, strides=(1,1), activation='relu'))  
model.add(MaxPooling2D(pool_size=(2,2), name='max_pool_1'))  
model.add(Conv2D(16, kernel_size=3, strides=(1,1), activation='relu'))  
model.add(MaxPooling2D(pool_size=(2,2), name='max_pool_2'))  
model.add(Flatten())  
model.add(Dense(10, activation='softmax'))  
model.summary()
```

¿Por qué la 2da.capa convolucional tiene 4624 parámetros?

Layer (type)	Output Shape	Param #
conv2d_2 (Conv2D)	(None, 26, 26, 32)	320
max_pool_1 (MaxPooling2D)	(None, 13, 13, 32)	0
conv2d_3 (Conv2D)	(None, 11, 11, 16)	4624
max_pool_2 (MaxPooling2D)	(None, 5, 5, 16)	0
flatten_1 (Flatten)	(None, 400)	0
dense_1 (Dense)	(None, 10)	4010

Total params: 8,954

$$\text{N}_f * \text{K} * \text{K} * \text{N}_c + \text{N}_f$$
$$16 * 3 * 3 * 32 + 16$$

Cantidad de Tamaño del Bias
filtros filtro

MNIST

```
model = Sequential()
model.add(Input(shape=(28, 28, 1)))
model.add(Conv2D(32, kernel_size=3, strides=(1,1), activation='relu'))
model.add(MaxPooling2D(pool_size=(2,2), name='max_pool_1'))
model.add(Conv2D(16, kernel_size=3, strides=(1,1), activation='relu'))
model.add(MaxPooling2D(pool_size=(2,2), name='max_pool_2'))
model.add(Flatten())
model.add(Dense(10, activation='softmax'))
model.summary()
```

Layer (type)	Output Shape	Param #
conv2d_2 (Conv2D)	(None, 26, 26, 32)	320
max_pool_1 (MaxPooling2D)	(None, 13, 13, 32)	0
conv2d_3 (Conv2D)	(None, 11, 11, 16)	4624
max_pool_2 (MaxPooling2D)	(None, 5, 5, 16)	0
flatten_1 (Flatten)	(None, 400)	0
dense_1 (Dense)	(None, 10)	4010

Total params: 8,954

**Luego del 2do.
MaxPooling se
aplanan las 16
salidas de 5x5**

MNIST

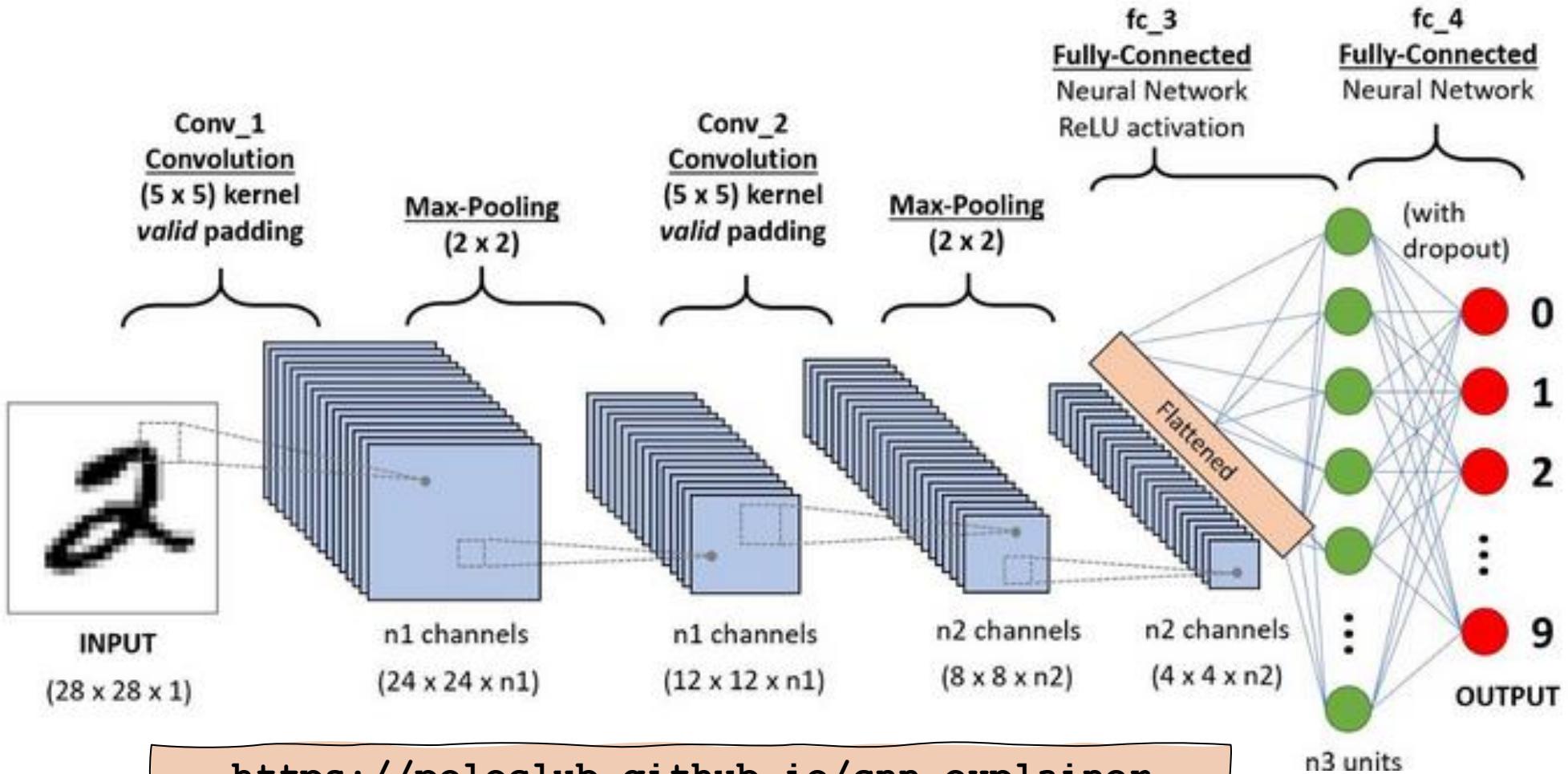
```
model = Sequential()
model.add(Input(shape=(28, 28, 1)))
model.add(Conv2D(32, kernel_size=3, strides=(1,1), activation='relu'))
model.add(MaxPooling2D(pool_size=(2,2), name='max_pool_1'))
model.add(Conv2D(16, kernel_size=3, strides=(1,1), activation='relu'))
model.add(MaxPooling2D(pool_size=(2,2), name='max_pool_2'))
model.add(Flatten())
model.add(Dense(10, activation='softmax'))
model.summary()
```

Layer (type)	Output Shape	Param #
conv2d_2 (Conv2D)	(None, 26, 26, 32)	320
max_pool_1 (MaxPooling2D)	(None, 13, 13, 32)	0
conv2d_3 (Conv2D)	(None, 11, 11, 16)	4624
max_pool_2 (MaxPooling2D)	(None, 5, 5, 16)	0
flatten_1 (Flatten)	(None, 400)	0
dense_1 (Dense)	(None, 10)	4010

Total params: 8,954

¿Por qué la capa de salida tiene 4010 parámetros?

Reconocimiento de dígitos MNIST

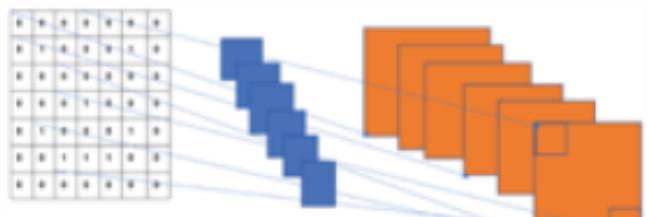


Resumen

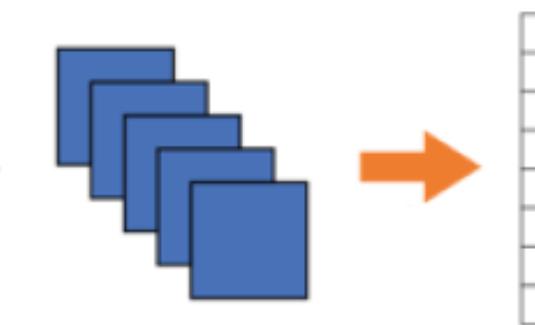
- Las capas convolucionales 2D contienen los filtros que, se entranan junto con los demás parámetros de la red y que permiten detectar características.
- El resultado de aplicar un filtro o máscara es un mapa de características de $T \times U \times 1$ dependiendo del padding, stride y tamaño de kernel usados.
- La salida de la capa convolucional es una nueva “imagen” de $T \times U \times F$ siendo F el número de filtros.
- Las capas de pooling reducen la dimensionalidad haciendo más rápido y eficaz el entrenamiento.
- Se suelen intercalar capas de convolución con capas de pooling hasta llegar a la parte feedforward donde para ingresar “aplanamos” las “imágenes”.

Modelos pre-entrenados

BASE CONVOLUCIONAL (extracción de características)

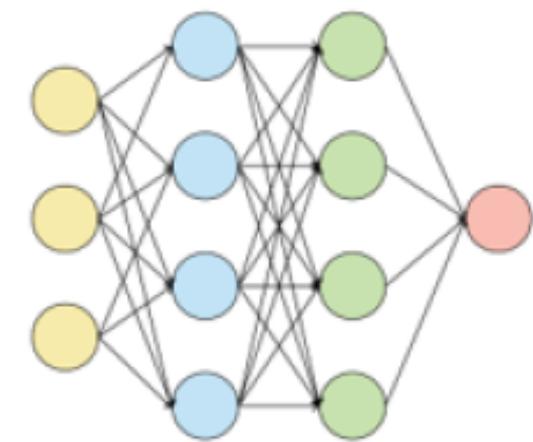


Convolutional Layer



Pooling and Flattening

CLASIFICADOR



Artificial Neural Network

Redes pre-entrenadas en Keras

- Las siguientes redes pre-entrenadas pueden ser consideradas como las capas convolucionales base.
- Se utilizan estas redes y se ajusta un clasificador (ANN):
 - VGG16 
 - Inception V3
 - Xception
 - ResNet50
 - MobileNet

Red neuronal convolucional con 16 capas propuesto por K. Simonyan y A. Zisserman de la Universidad de Oxford

El modelo se presentó al Desafío de Reconocimiento Visual a Gran Escala de ImageNet (ILSVRC) en 2014.

Redes pre-entrenadas en Keras

- Las siguientes redes pre-entrenadas pueden ser consideradas como las capas convolucionales base.
- Se utilizan estas redes y se ajusta un clasificador (ANN):
 - VGG16
 - Inception V3
 - Xception
 - ResNet50 
 - MobileNet

Red neuronal convolucional entrenada con el conjunto de datos de ImageNet que tiene 50 capas y que ganó el primer puesto en el ILSVRC en 2015.

Ajuste fino (fine-tuning) de modelos pre-entrenados

- Se busca reusar la base convolucional y mejorar la respuesta del clasificador en una tarea específica.
- Pasos para efectuar el **ajuste fino**
 - Añadir un clasificador (RNA) sobre un sistema preentrenado.
 - Fijar la base convolucional y entrenar la red.
 - Entrenar conjuntamente el clasificador añadido y la base convolucional.

Consultar Capítulo 8 del libro “The Deep Learning with Keras Workshop. An Interactive Approach to Understanding Deep Learning with Keras (2020)”

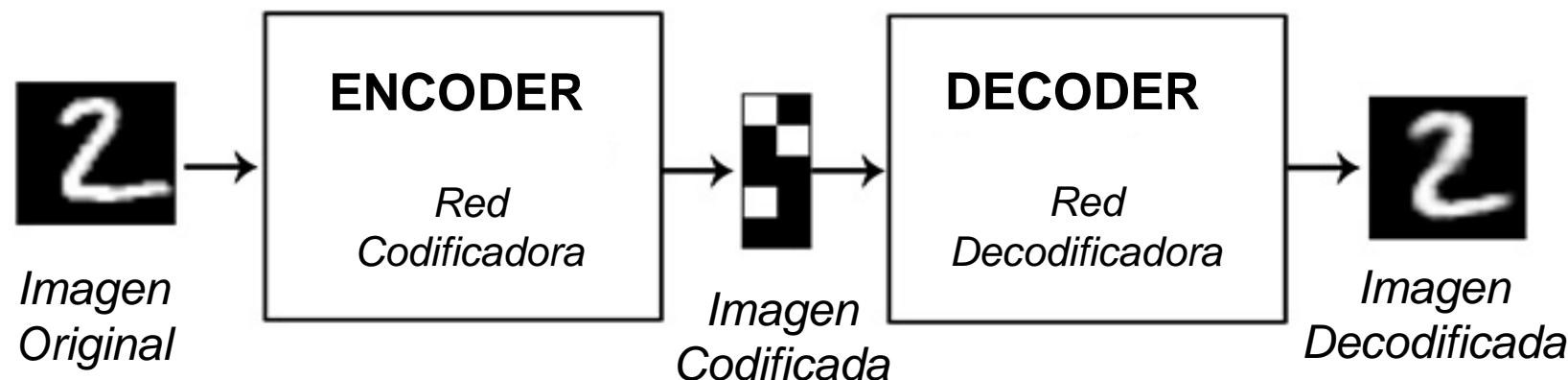
VGG16_cars_flowers.ipynb

AUTOENCODERS (AUTO CODIFICADORES)



Autoencoding (Auto codificación)

- Es un algoritmo de compresión de datos en el que las funciones de compresión y descompresión son
 - Específicas de los datos
 - Con pérdida
 - Aprendidas automáticamente



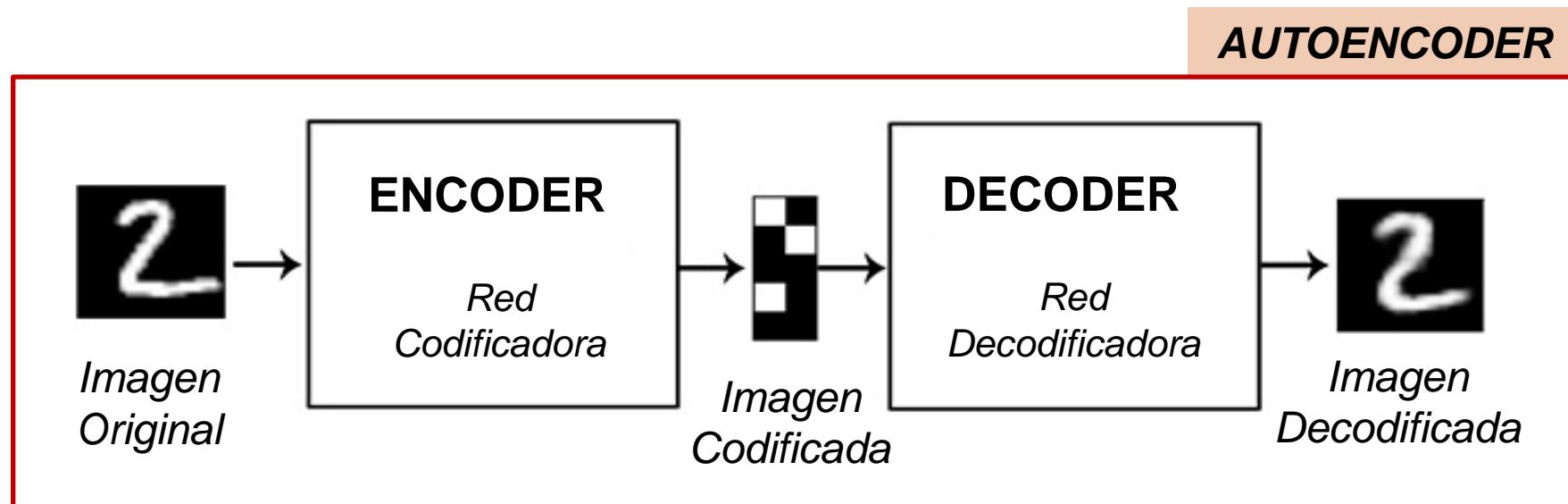
Autoencoder

□ Entrenamiento

□ La entrada y la salida son iguales.

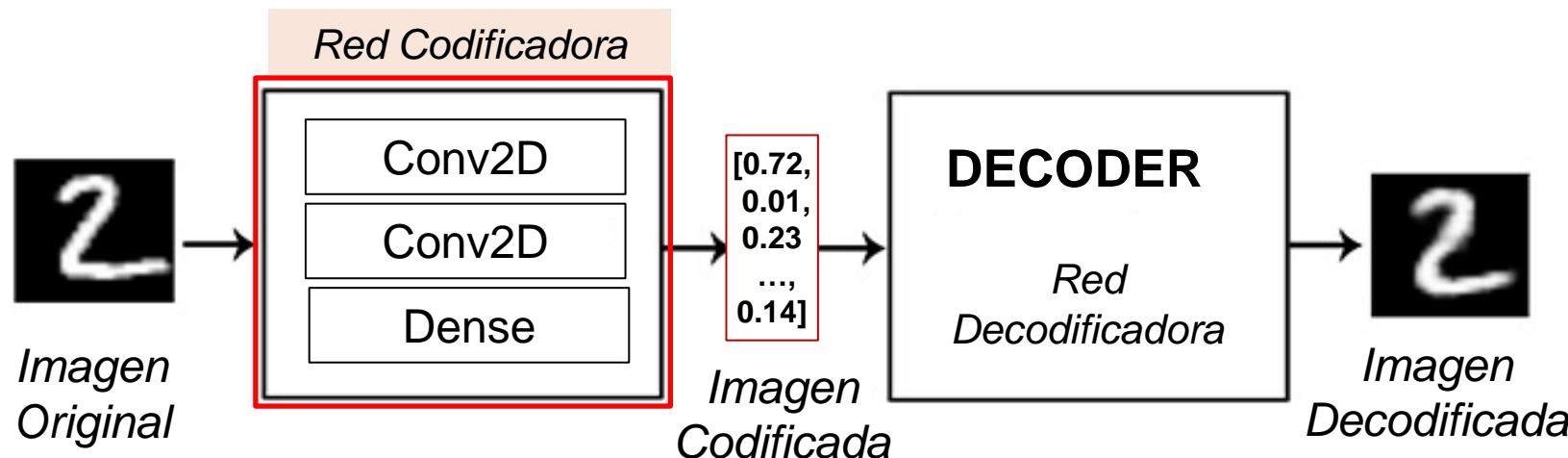
□ Entrenamiento NO supervisado aunque usa mecanismos supervisados.

□ Tres modelos: encoder, decoder, autoencoder=decoder(encoder(x))



Red Codificadora

- Genera una representación vectorial de tamaño K
 - $K \ll$ tamaño original de la imagen
- La nueva representación comprime la imagen
- Es una red neuronal feedforward o convolucional



Red Decodificadora

- Recibe la imagen codificada y genera una imagen de igual tamaño que la imagen de entrada.
- Descomprime la imagen.
- Es una red neuronal feedforward o convolucional

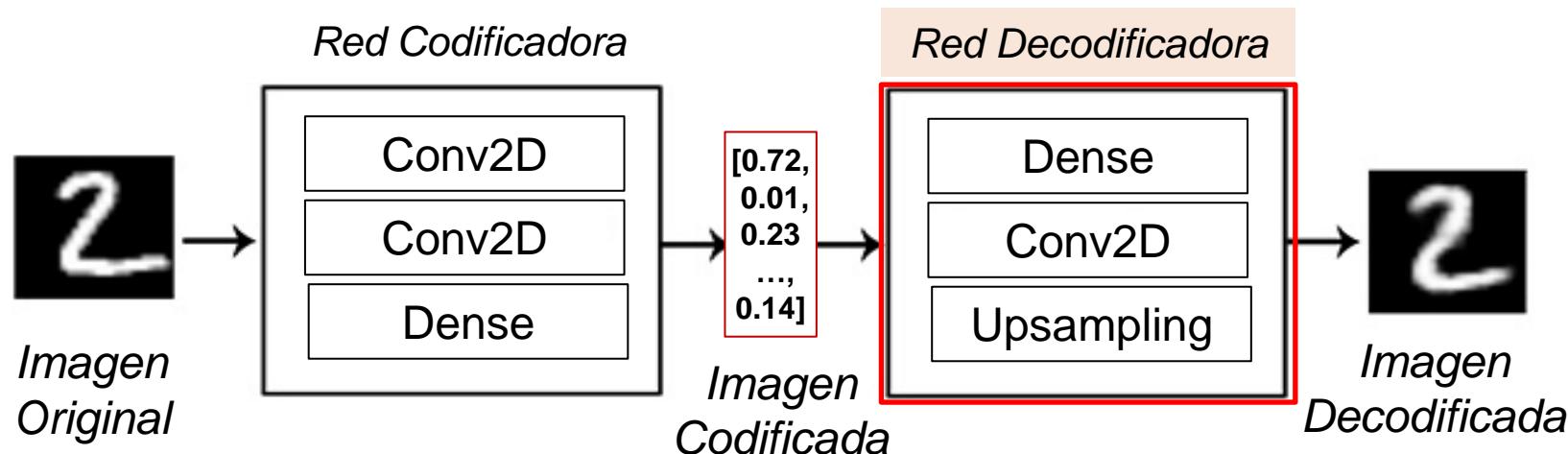
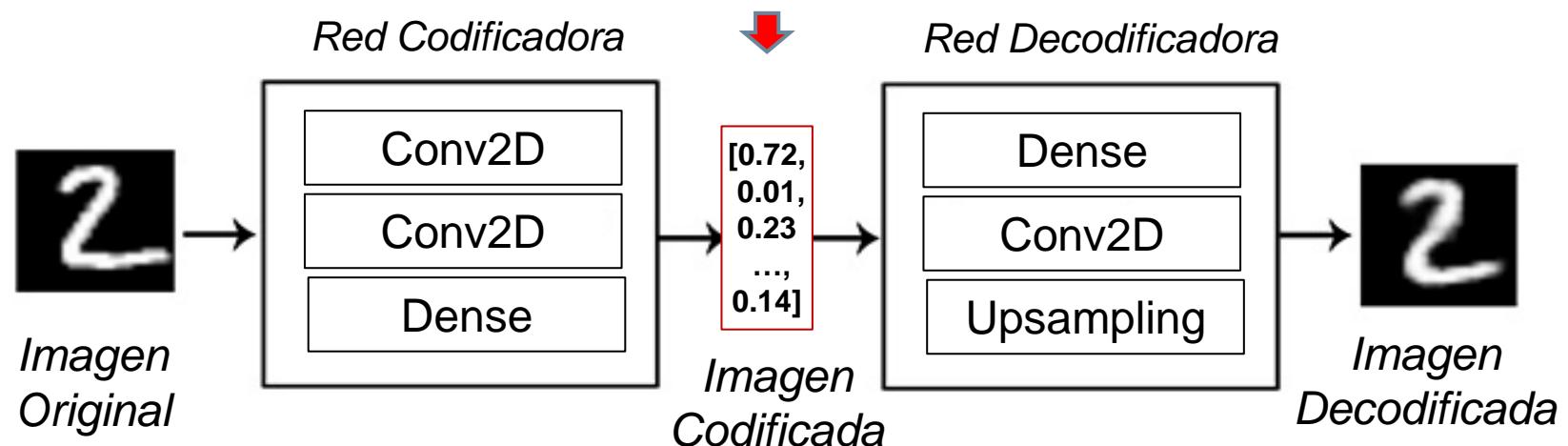


Imagen codificada

- Es un vector numérico de tamaño K (vector latente)
 - El valor de K es arbitrario
 - A mayor valor de K mejora la representación y se reduce la compresión.
- Su valor se infiere

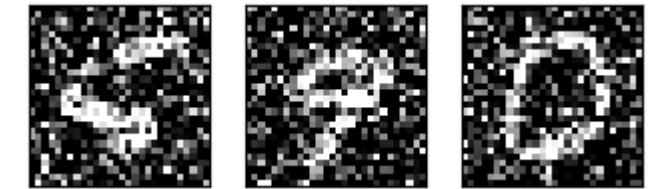


Autoencoder convolucional

Layer (type)	Output Shape	Param #
input_5 (InputLayer)	[(None, 28, 28, 1)]	0
conv2d_17 (Conv2D)	(None, 28, 28, 16)	160
max_pooling2d_7 (MaxPooling2D)	(None, 14, 14, 16)	0
conv2d_18 (Conv2D)	(None, 14, 14, 8)	1160
max_pooling2d_8 (MaxPooling2D)	(None, 7, 7, 8)	0
conv2d_19 (Conv2D)	(None, 7, 7, 8)	584
max_pooling2d_9 (MaxPooling2D)	(None, 4, 4, 8)	0
conv2d_20 (Conv2D)	(None, 4, 4, 8)	584
up_sampling2d_7 (UpSampling2D)	(None, 8, 8, 8)	0
conv2d_21 (Conv2D)	(None, 8, 8, 8)	584
up_sampling2d_8 (UpSampling2D)	(None, 16, 16, 8)	0
conv2d_22 (Conv2D)	(None, 14, 14, 16)	1168
up_sampling2d_9 (UpSampling2D)	(None, 28, 28, 16)	0
conv2d_23 (Conv2D)	(None, 28, 28, 1)	145
<hr/>		
Total params: 4,385		

Imágenes con ruido

```
(x_train, _), (x_test, _) = mnist.load_data()  
  
x_train = x_train.astype('float32') / 255.  
x_test = x_test.astype('float32') / 255.  
  
x_train = np.reshape(x_train, (len(x_train), 28, 28, 1))  
x_test = np.reshape(x_test, (len(x_test), 28, 28, 1))  
  
noise_factor = 0.5  
  
x_train_noisy = x_train + noise_factor * np.random.normal(loc=0.0, scale=1.0,  
                                                               size=x_train.shape)  
  
x_test_noisy = x_test + noise_factor * np.random.normal(loc=0.0, scale=1.0,  
                                                               size=x_test.shape)  
  
x_train_noisy = np.clip(x_train_noisy, 0., 1.)  
x_test_noisy = np.clip(x_test_noisy, 0., 1.)
```

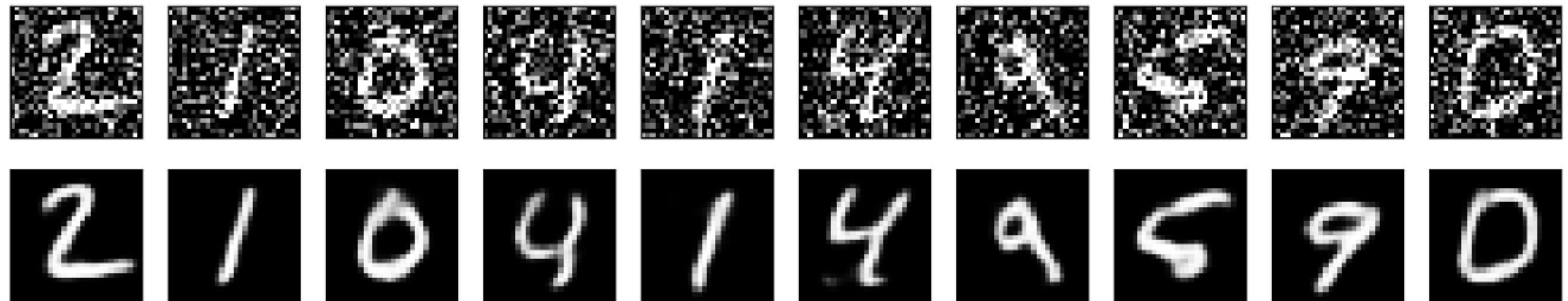


Resultado de la decodificación

```
autoencoder.fit(x_train_noisy, x_train, epochs=100, batch_size=128,  
                 shuffle=True, validation_data=(x_test_noisy, x_test))
```

Predicción del Autoencoder para las imágenes de testeo

```
decoded_imgs = autoencoder.predict(x_test_noisy)
```

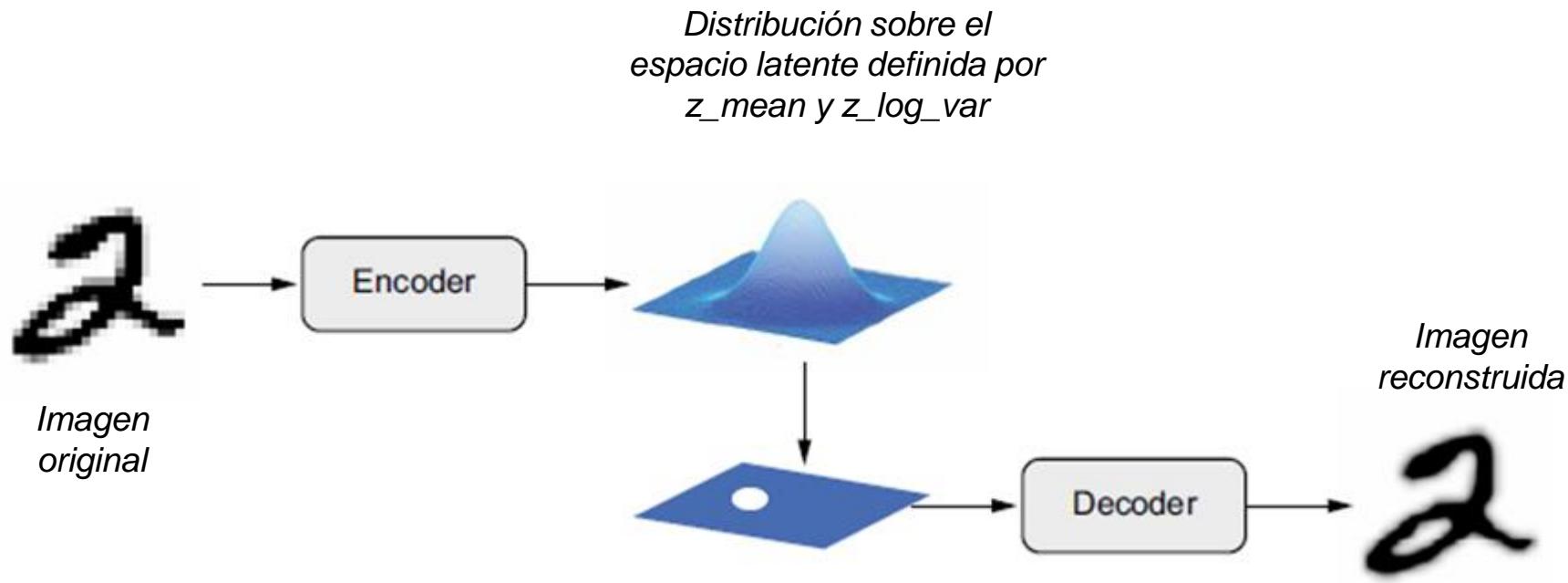


Detección de fraudes

- El conjunto de datos contiene transacciones realizadas por tarjetas de crédito en septiembre de 2013 por titulares de tarjetas europeas.
- Este conjunto de datos presenta las transacciones que se produjeron en dos días, donde tenemos 492 fraudes de 284.807 transacciones.
- El conjunto de datos es muy desequilibrado, la clase positiva (fraudes) representa el 0,172% de todas las transacciones
 - [Link al dataset](#)

Autoencoder_detecta_fraude.ipynb

Variational autoencoders (VAE)



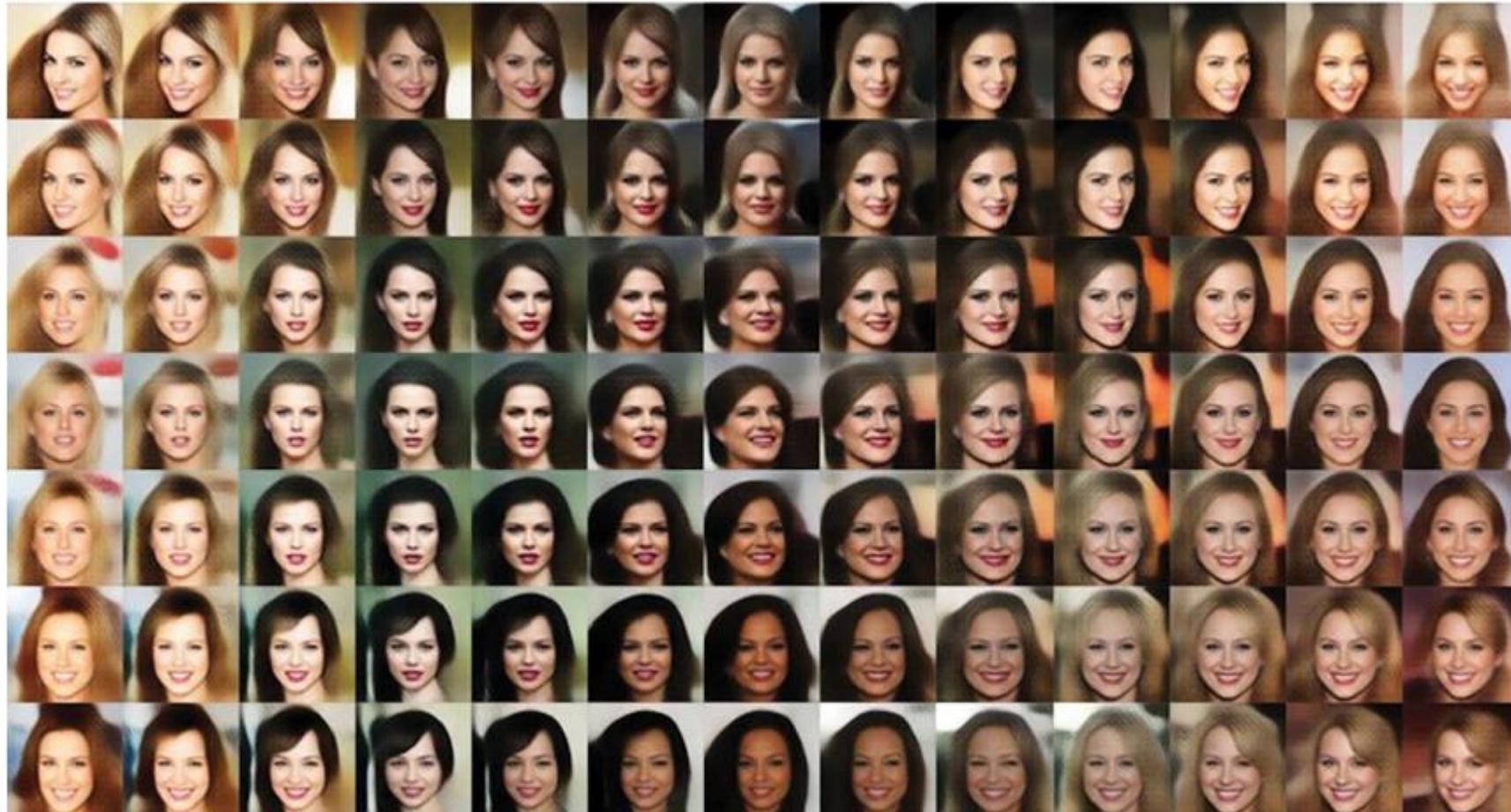
Variational-autoencoders.ipynb

Variational autoencoders (VAE)



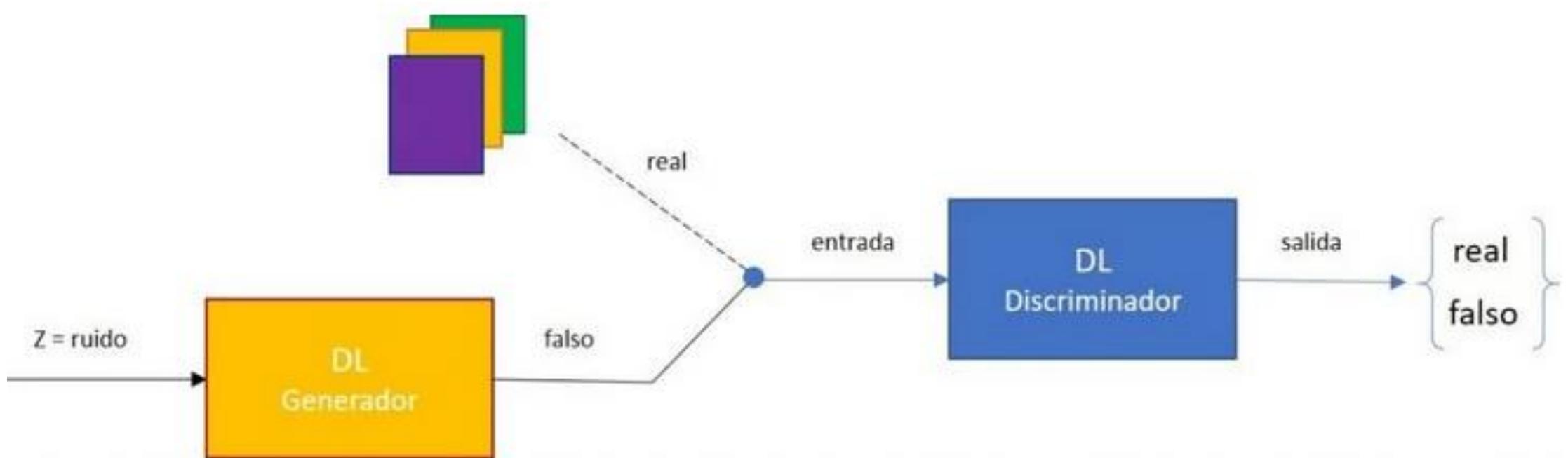
Variational-encoder.ipynb

Variational autoencoders (VAE)

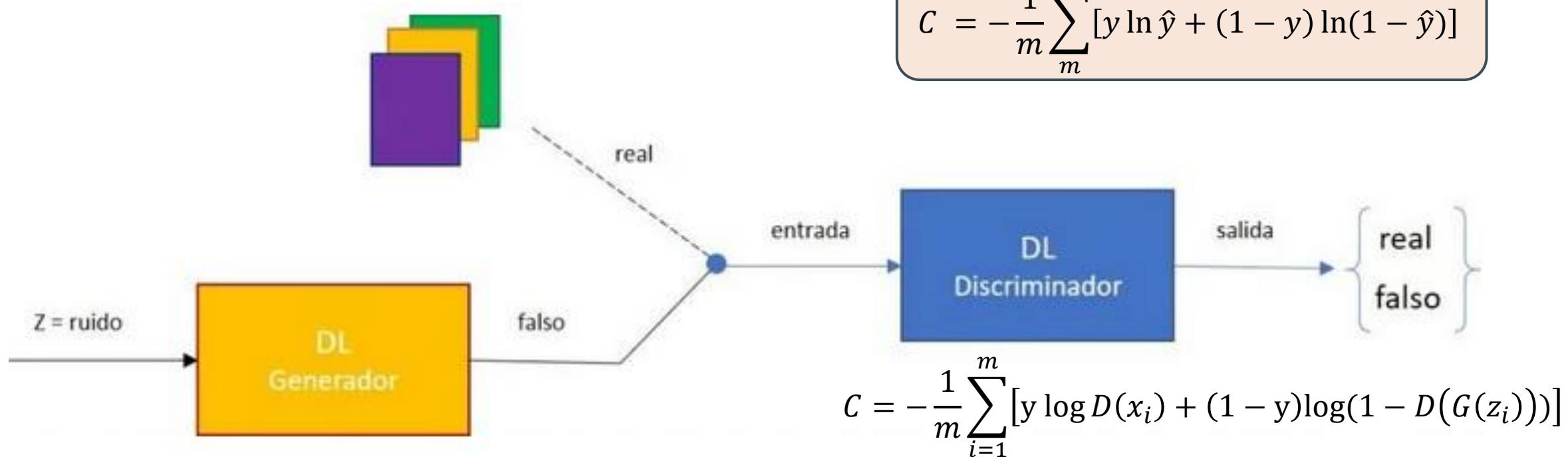


Un espacio continuo de caras generadas por Tom White usando VAEs
(Figura 12.14 del libro de Chollet)

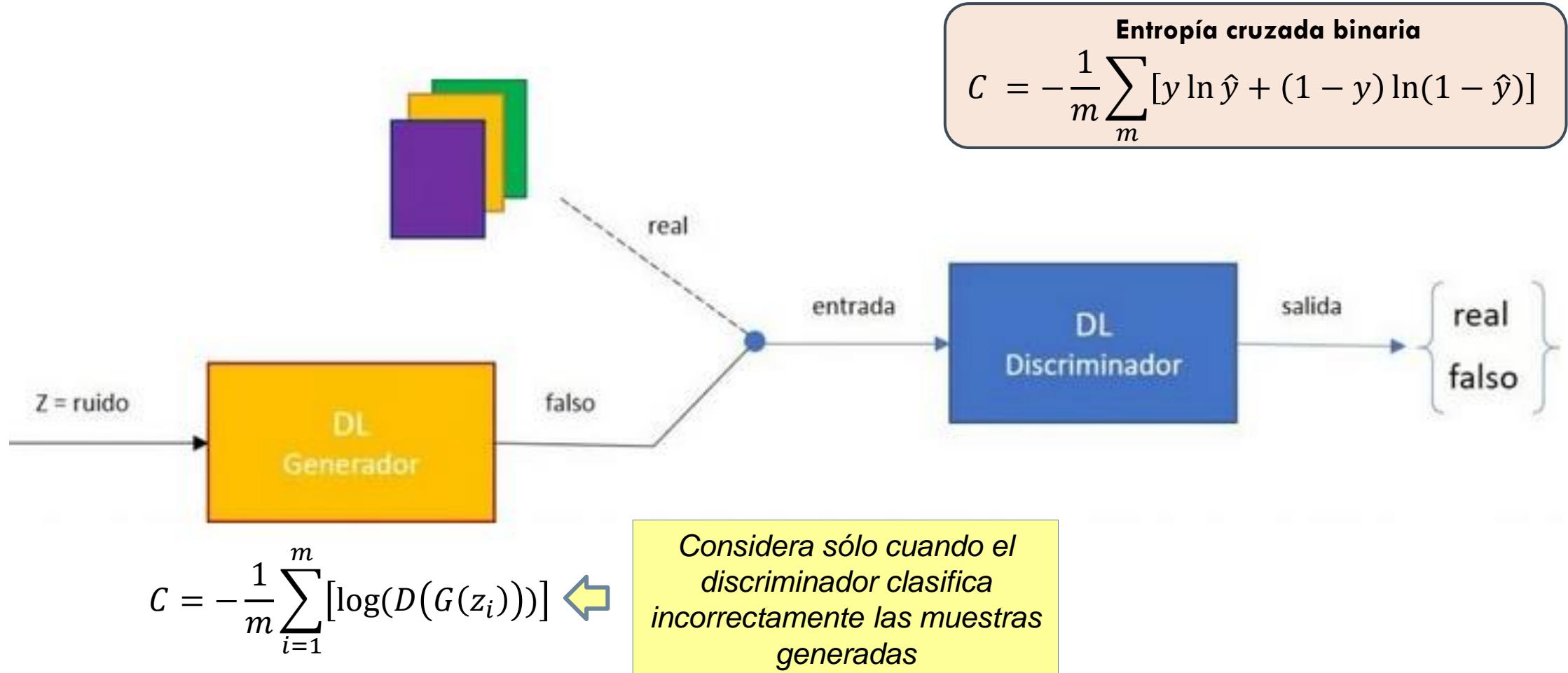
Redes generativas adversarias (GAN)



Redes generativas adversarias (GAN)



Redes generativas adversarias (GAN)



Discriminador

```
def build_discriminator(img_shape):
    model = Sequential()
    model.add(Flatten(input_shape=img_shape))
    model.add(Dense(512))
    model.add(LeakyReLU(alpha=0.2))
    model.add(Dense(256))
    model.add(LeakyReLU(alpha=0.2))
    model.add(Dense(1, activation='sigmoid'))
    return model
```

```
img_shape = (28, 28, 1)
```

Crear y compilar el discriminador

```
discriminator = build_discriminator(img_shape)
discriminator.compile(loss='binary_crossentropy', optimizer=Adam(0.0002, 0.5), metrics=['accuracy'])
```

Layer (type)	Output Shape	Param #
=====		
flatten (Flatten)	(None, 784)	0
dense (Dense)	(None, 512)	401920
leaky_re_lu (LeakyReLU)	(None, 512)	0
dense_1 (Dense)	(None, 256)	131328
leaky_re_lu_1 (LeakyReLU)	(None, 256)	0
dense_2 (Dense)	(None, 1)	257
=====		
Total params: 533,505		
Trainable params: 533,505		
Non-trainable params: 0		

Generador

```
def build_generator(latent_dim):
    model = Sequential()
    model.add(Dense(256, input_dim=latent_dim))
    model.add(LeakyReLU(alpha=0.2))
    model.add(BatchNormalization(momentum=0.8))
    model.add(Dense(512))
    model.add(LeakyReLU(alpha=0.2))
    model.add(BatchNormalization(momentum=0.8))
    model.add(Dense(1024))
    model.add(LeakyReLU(alpha=0.2))
    model.add(BatchNormalization(momentum=0.8))
    model.add(Dense(28 * 28 * 1, activation='tanh'))
    model.add(Reshape((28, 28, 1)))
    return model
```

```
latent_dim = 100
```

Crear el generador

```
generator = build_generator(latent_dim)
```

Layer (type)	Output Shape	Param #
=====		
dense_3 (Dense)	(None, 256)	25856
leaky_re_lu_2 (LeakyReLU)	(None, 256)	0
batch_normalization	(None, 256)	1024
dense_4 (Dense)	(None, 512)	131584
leaky_re_lu_3 (LeakyReLU)	(None, 512)	0
batch_normalization_1	(None, 512)	2048
dense_5 (Dense)	(None, 1024)	525312
leaky_re_lu_4 (LeakyReLU)	(None, 1024)	0
batch_normalization_2	(None, 1024)	4096
dense_6 (Dense)	(None, 784)	803600
reshape (Reshape)	(None, 28, 28, 1)	0
=====		
Total params: 1,493,520		
Trainable params: 1,489,936		
Non-trainable params: 3,584		

Modelo combinado. Generador+Discriminador

```
latent_dim = 100
```

El generador toma ruido como entrada y genera imágenes

```
z = Input(shape=(latent_dim,))  
img = generator(z)
```

En el modelo combinado solo se entrena el generador
discriminator.trainable = False

Rta del discriminador
valid = discriminator(img)

Modelo combinado (stacked generador y discriminador)

```
combined = Model(z, valid)  
combined.compile(loss='binary_crossentropy', optimizer=Adam(0.0002, 0.5))
```

Layer (type)	Output Shape	Param #
=====		
input_1 (InputLayer)	[(None, 100)]	0
sequential_1 (Sequential)	(None, 28, 28, 1)	1493520
sequential (Sequential)	(None, 1)	533505
=====		
Total params: 2,027,025		
Trainable params: 1,489,936		
Non-trainable params: 537,089		

Entrenamiento del modelo

for epoch in range(epochs):

```
    idx = np.random.randint(0, X_train.shape[0], batch_size)
    imgs = X_train[idx]

    noise = np.random.normal(0, 1, (batch_size, latent_dim))
    gen_imgs = generator.predict(noise)

    d_loss_real = discriminator.train_on_batch(imgs, valid)
    d_loss_fake = discriminator.train_on_batch(gen_imgs, fake)
    d_loss = 0.5 * np.add(d_loss_real, d_loss_fake)
```

Discriminador

Seleccionar un conjunto aleatorio de imágenes reales

```
    noise = np.random.normal(0, 1, (batch_size, latent_dim))
    g_loss = combined.train_on_batch(noise, valid)
```

Entrenamiento del modelo

```
for epoch in range(epochs):
```

```
    idx = np.random.randint(0, X_train.shape[0], batch_size)  
    imgs = X_train[idx]
```

```
    noise = np.random.normal(0, 1, (batch_size, latent_dim))  
    gen_imgs = generator.predict(noise)
```

Generar un conjunto de imágenes falsas

Discriminador

```
    d_loss_real = discriminator.train_on_batch(imgs, valid)  
    d_loss_fake = discriminator.train_on_batch(gen_imgs, fake)  
    d_loss = 0.5 * np.add(d_loss_real, d_loss_fake)
```

```
    noise = np.random.normal(0, 1, (batch_size, latent_dim))  
    g_loss = combined.train_on_batch(noise, valid)
```

Entrenamiento del modelo

```
for epoch in range(epochs):
```

```
    idx = np.random.randint(0, X_train.shape[0], batch_size)
    imgs = X_train[idx]

    noise = np.random.normal(0, 1, (batch_size, latent_dim))
    gen_imgs = generator.predict(noise)

    d_loss_real = discriminator.train_on_batch(imgs, valid)
    d_loss_fake = discriminator.train_on_batch(gen_imgs, fake)
    d_loss = 0.5 * np.add(d_loss_real, d_loss_fake)
```

Discriminador

```
    noise = np.random.normal(0, 1, (batch_size, latent_dim))
    g_loss = combined.train_on_batch(noise, valid)
```

Entrenar el discriminador

Entrenamiento del modelo

```
for epoch in range(epochs):
```

```
    idx = np.random.randint(0, X_train.shape[0], batch_size)
    imgs = X_train[idx]

    noise = np.random.normal(0, 1, (batch_size, latent_dim))
    gen_imgs = generator.predict(noise)

    d_loss_real = discriminator.train_on_batch(imgs, valid)
    d_loss_fake = discriminator.train_on_batch(gen_imgs, fake)
    d_loss = 0.5 * np.add(d_loss_real, d_loss_fake)

    noise = np.random.normal(0, 1, (batch_size, latent_dim))
    g_loss = combined.train_on_batch(noise, valid)
```

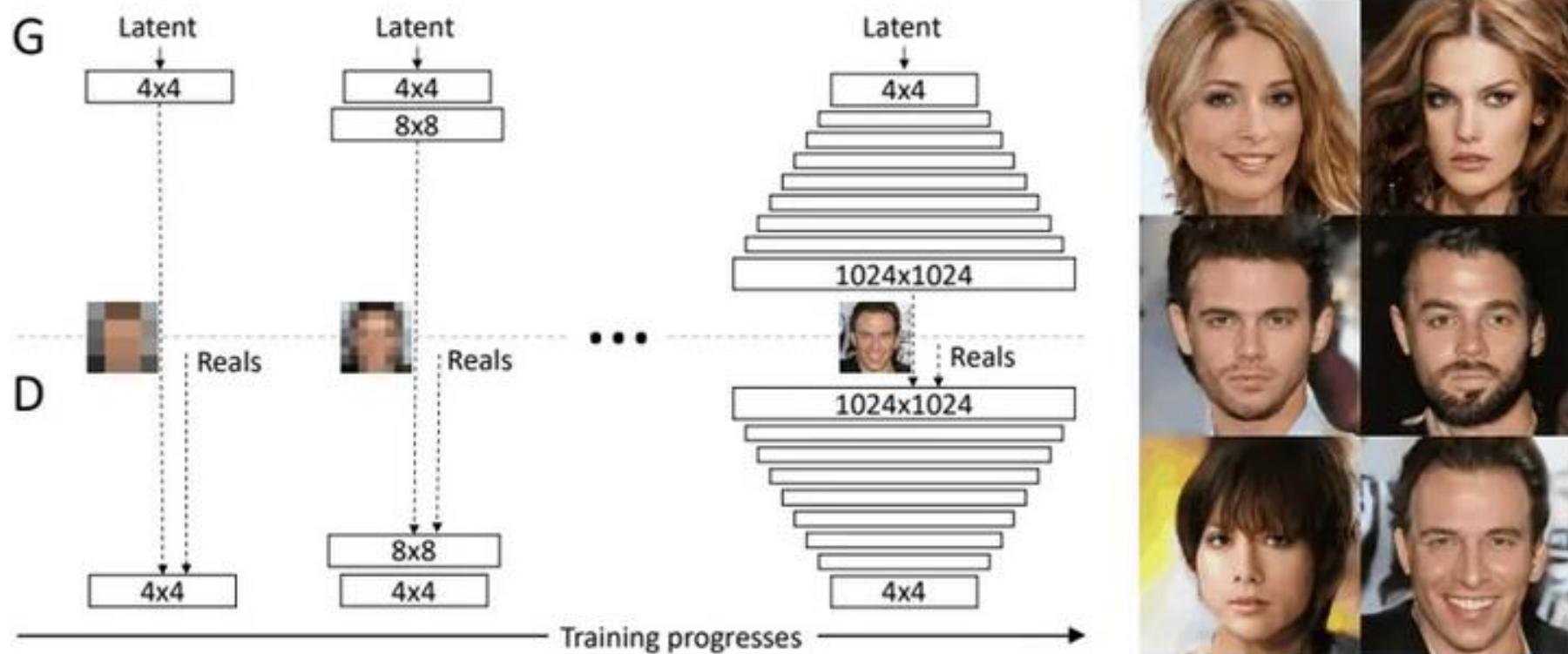
Discriminador

Entrena el generador para engañar al discriminador

GAN_MNIST.ipynb

Redes generativas progresivas

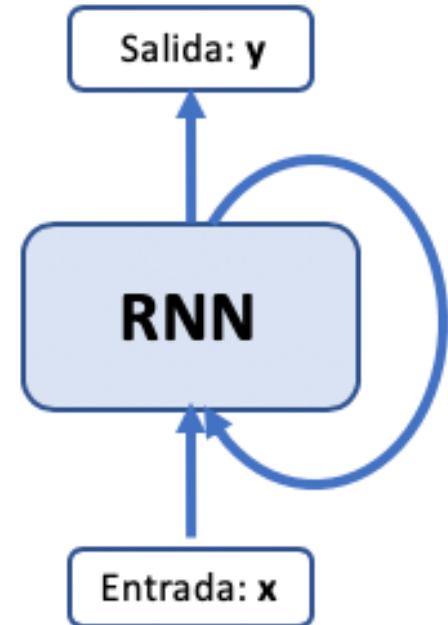
- La calidad generativa del modelo mejora incrementando el tamaño de las imágenes en forma progresiva



REDES NEURONALES RECURRENTES

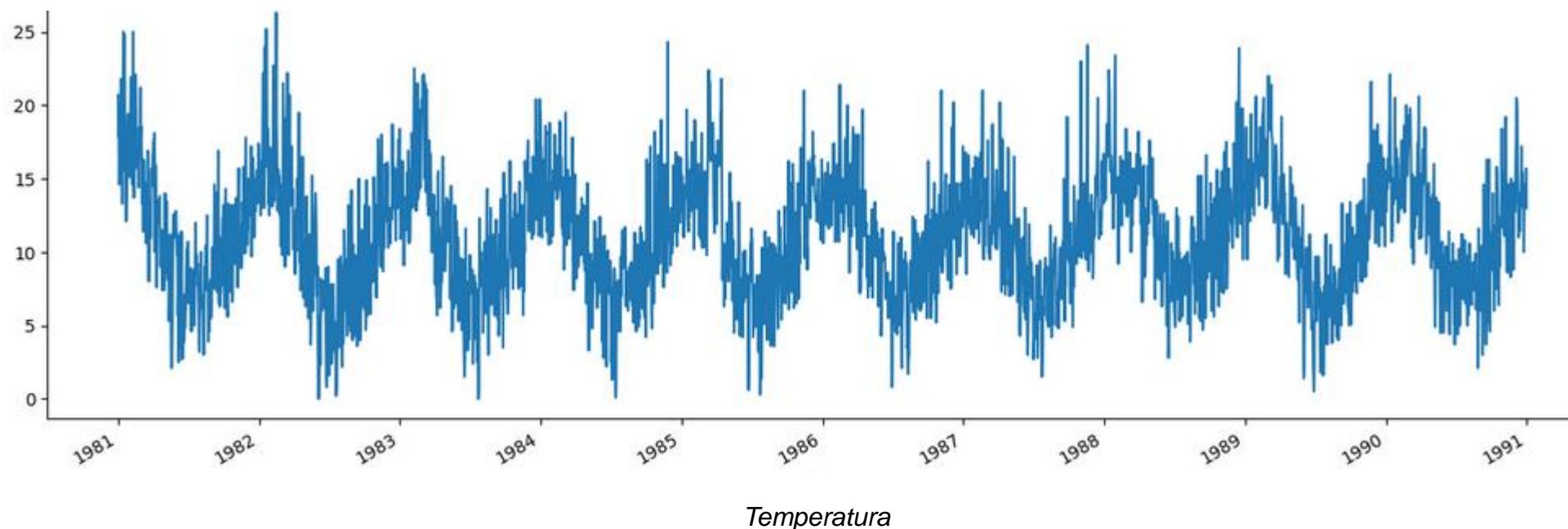
Red Neuronal Recurrente (RNN)

- Una red neuronal recurrente es un tipo de red diseñada para procesar secuencias de datos, donde la información tiene una estructura temporal o secuencial.
- Las RNN utilizan conexiones recurrentes que les permiten mantener una especie de "memoria" de información previa y utilizarla para procesar entradas futuras en la secuencia.



Serie Temporal

- Una serie temporal es un conjunto de datos ordenados en el tiempo.
- Los datos están igualmente espaciados en el tiempo, lo que significa que se registraron cada hora, minuto, mes o trimestre.



Preprocesamiento de los datos

- Antes de comenzar a trabajar es preciso analizar
 - Verificar que no haya **valores faltantes**. De ser necesario interpolar.
 - Verificar la periodicidad de las muestras
 - Eliminar duplicados.
 - Si hay diferencias en la periodicidad, reinterpolar el data set.
 - Verificar que la media de cada atributo no ha sufrido grandes modificaciones con todos estos cambios.

Red Neuronal Recurrente (RNN)

- Según la cantidad de series temporales que se tengan en cuenta como entrada
 - Univariada
 - Multivariada
- Según la cantidad de valores que se predigan en cada instante de tiempo
 - One-step
 - Multi-step

Red Neuronal Recurrente (RNN)

□ Enfoques

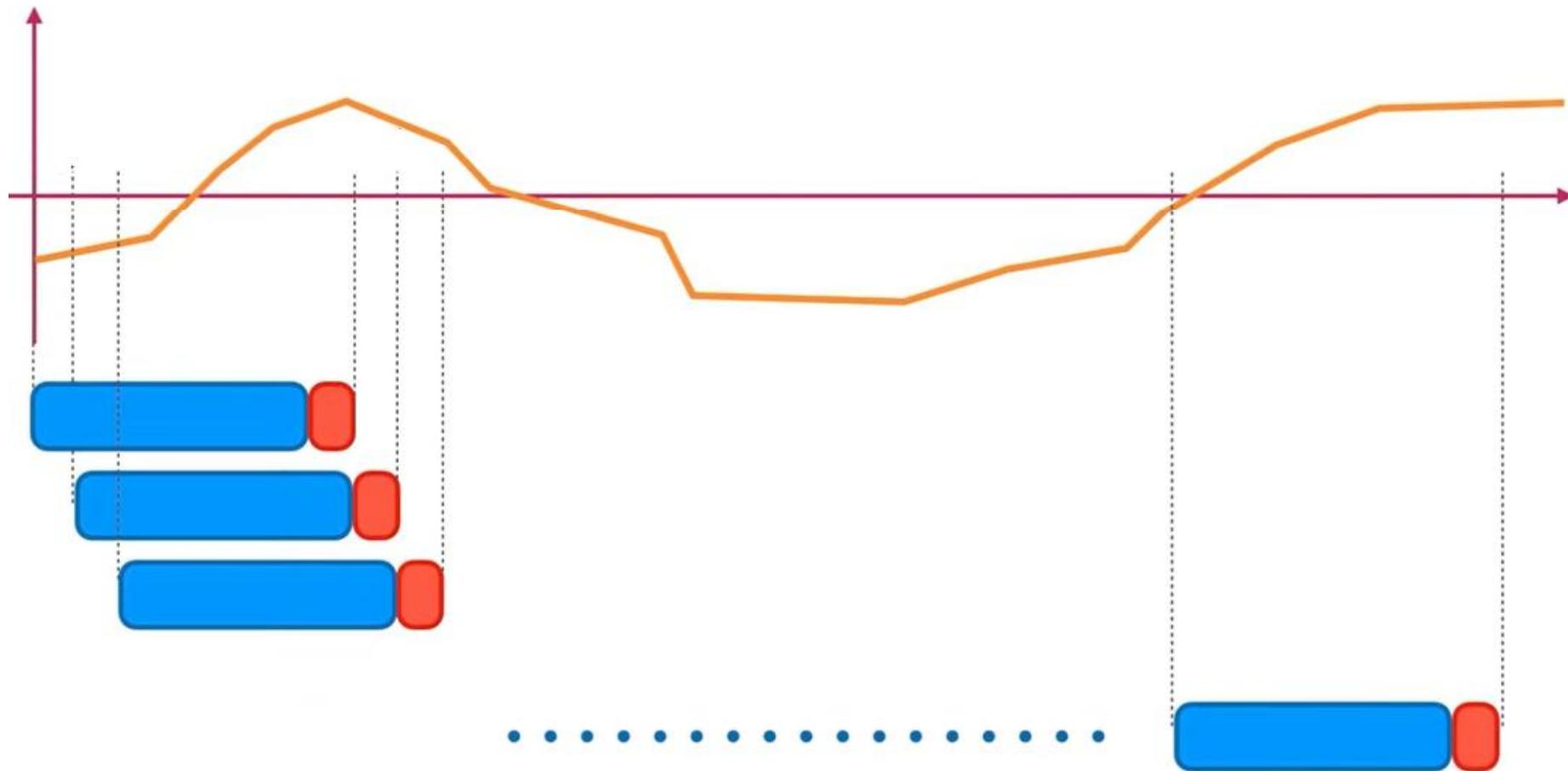
- Univariado + one-step
- Univariado + multi-step
- Multivariado + one-step
- Multivariado + multi-step



□ Tipos de RNN

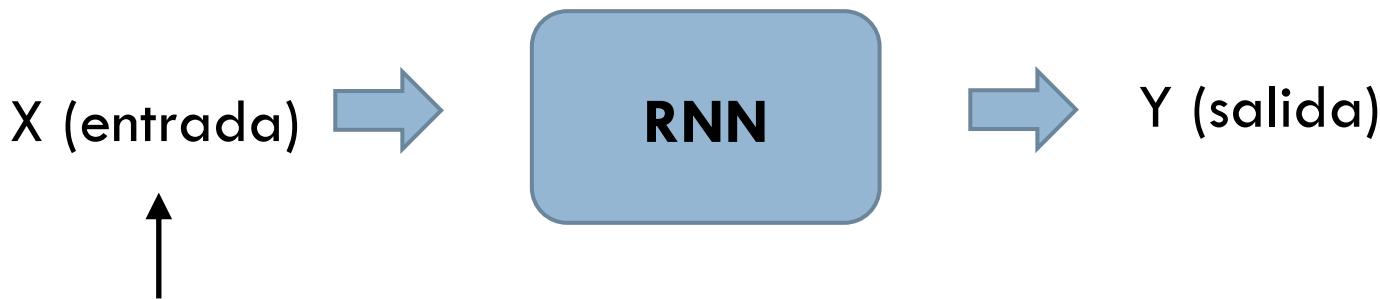
- SimpleRNN
- LSTM

Preparando los datos de entrada



Ingresando los datos a la RNN

- Según la documentación de Keras



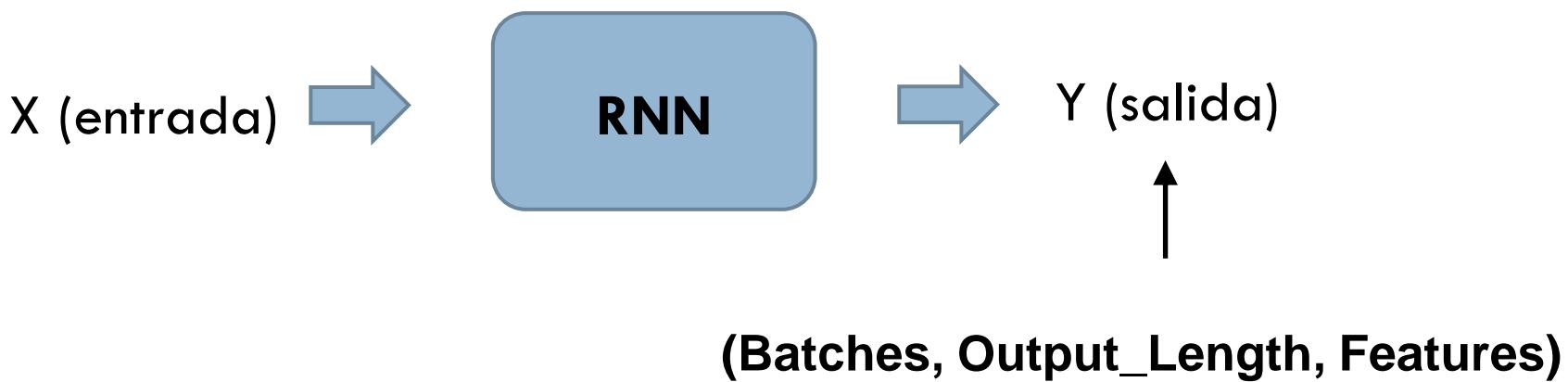
(Batches, Input_Length, Features)

donde

- **Batches** : cant.de datos (ventanas) de entrenamiento
- **Input_Length**: Longitud de la ventana de entrada
- **Features**: cantidad de series o variables a utilizar

Ingresando los datos a la RNN

- Según la documentación de Keras



donde

- **Batches** : cant.de predicciones (ídem a los de entrada)
- **Output_Length**: Longitud de la ventana de salida
- **Features**: cantidad de series o variables a predecir

Preparación de los datos

```
x, y = crear_dataset_supervisado(datos, 5, 1) ←
```

Genera secuencias de
entrada de longitud 5 y
de salida de longitud 1

```
Datos = [-8.05 -8.88 -8.81 -9.05 -9.63 -9.67 -9.17 -8.1 ... ]
```

```
x = [[[ -8.05]
       [-8.88]
       [-8.81]
       [-9.05]
       [-9.63]]
      [[ -8.88]
       [-8.81]
       [-9.05]
       [-9.63]
       [-9.67]]]
```

```
y = [[[ -9.67]
       [[ -9.17]]]
```

02a_UniVar_OneStep_SimpleRNN.ipynb

Preparación de los datos

- Si se ingresan secuencias de longitud 5 y se predice el valor siguiente

```
secuencia = [-8.05 -8.88 -8.81 -9.05 -9.63 -9.67 -9.17 -8.1 ... ]
```

```
entrada = [[[ -8.05]
             [-8.88]
             [-8.81]
             [-9.05]          (Batches, Input_Length, Features)
             [-9.63]]           # sec            5            1
             [[ -8.88]
              [-8.81]
              [-9.05]
              [-9.63]
              [-9.67]]
             ...]
```

```
salida = [[[ -9.67]]
             [[ -9.17]]          (Batches, Output_Length, Features)
             ...]]                # sec            1            1
```

SimpleRNN

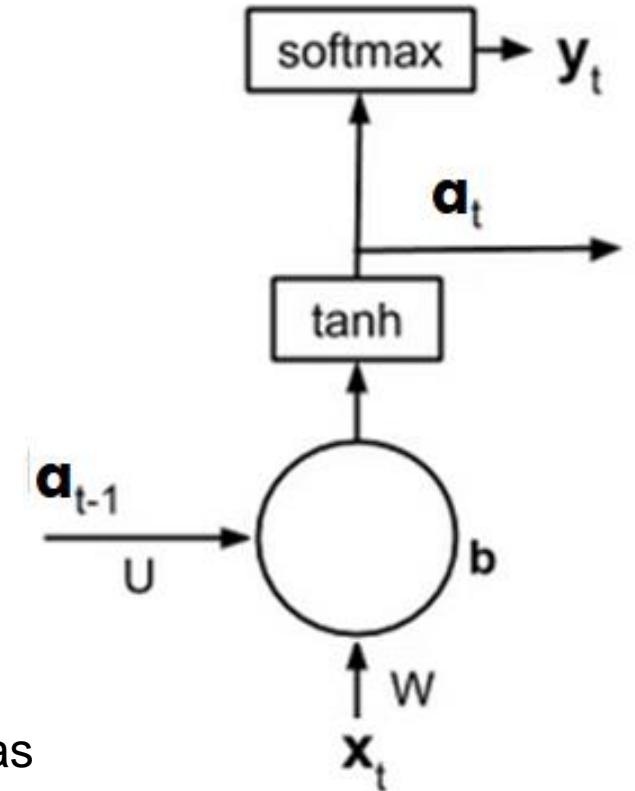
02a_UniVar_OneStep_SimpleRNN.ipynb

$$h_t = \tanh(Wx_t + Ua_{t-1} + b)$$

$$y_t = \text{softmax}(a_t)$$

$$x_t \in \mathbb{R}^d; W \in \mathbb{R}^{h \times d}; U \in \mathbb{R}^{h \times h}; b \in \mathbb{R}^h$$

La dimensión del vector a_t coincide con la cantidad h de neuronas ocultas



```

from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Input, SimpleRNN, Dense

modelo = Sequential(name='modelo')
modelo.add(SimpleRNN(4, input_shape=(24,1), name='Oculta'))
modelo.add(Dense(1, activation='linear', name='Salida'))

modelo.summary()

```

Model: "modelo"

Layer (type)	Output Shape	Param #
<hr/>		
Oculta (SimpleRNN)	(None, 4)	24
Salida (Dense)	(None, 1)	5
<hr/>		
Total params: 29 (116.00 Byte)		

$$x_t \in \mathbb{R}^d;$$

$$W \in \mathbb{R}^{h \times d}; b \in \mathbb{R}^h$$

$$U \in \mathbb{R}^{h \times h}$$

1 de c/u por capa

```

from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Input, SimpleRNN, Dense

modelo2 = Sequential(name='modelo2')
modelo2.add(Input(shape=(24,1), name='Entrada'))
modelo2.add(SimpleRNN(10, return_sequences=True, name='Oculta1'))
modelo2.add(SimpleRNN(10, name='Oculta2'))
modelo2.add(Dense(1, activation='linear', name='Salida'))

modelo2.summary()

```

Model: "modelo2"

Layer (type)	Output Shape	Param #
=====		
Oculta1 (SimpleRNN)	(None, 24, 10)	120
Oculta2 (SimpleRNN)	(None, 10)	210
Salida (Dense)	(None, 1)	11
=====		
Total params: 341 (1.33 KB)		

$$x_t \in \mathbb{R}^d;$$

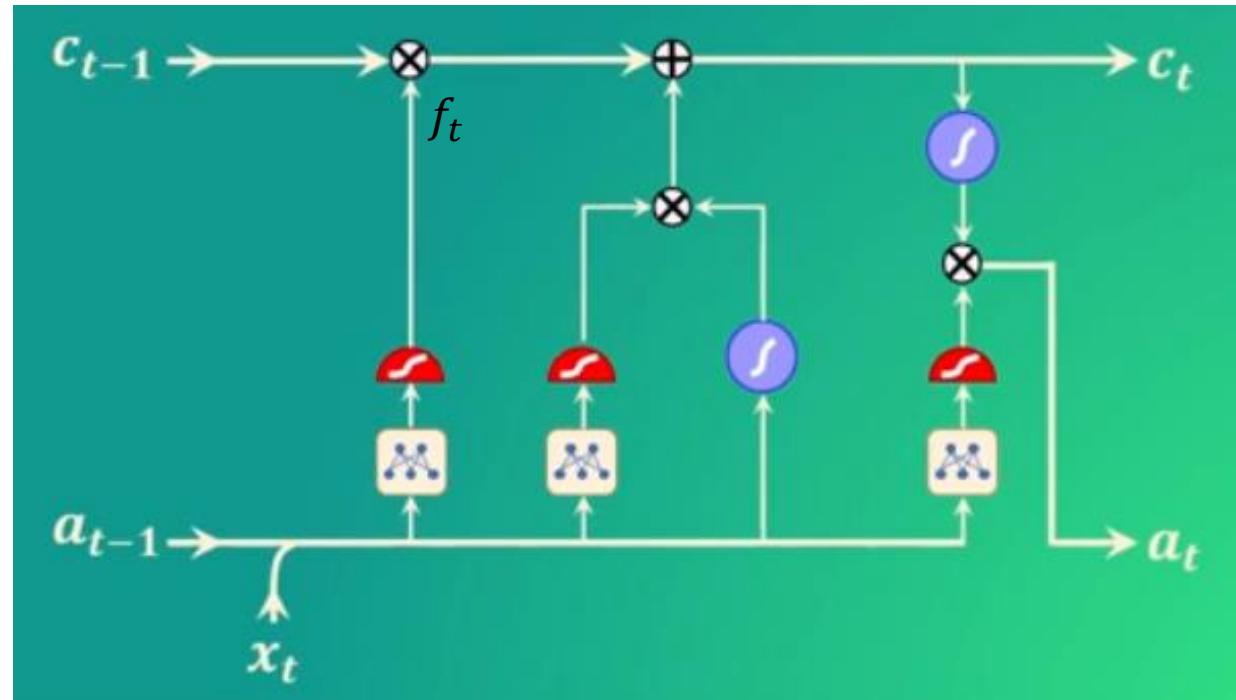
$$W \in \mathbb{R}^{hxd}; b \in \mathbb{R}^h$$

$$U \in \mathbb{R}^{hxh}$$

1 de c/u por capa

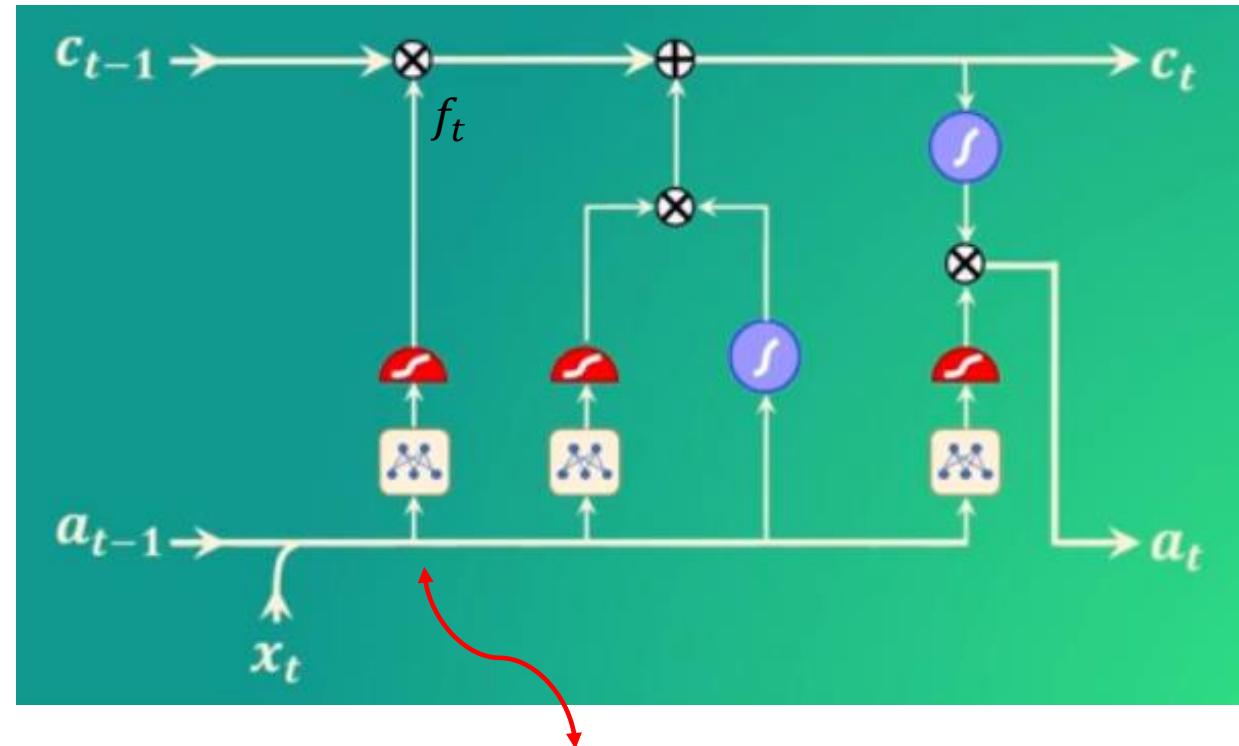
LSTM

- La red LSTM agrega una celda de estado o MEMORIA para resolver el problema de olvido de la red recurrente simple



LSTM

$$f_t = \text{sig}(W_f x_t + U_f a_{t-1} + b_f)$$

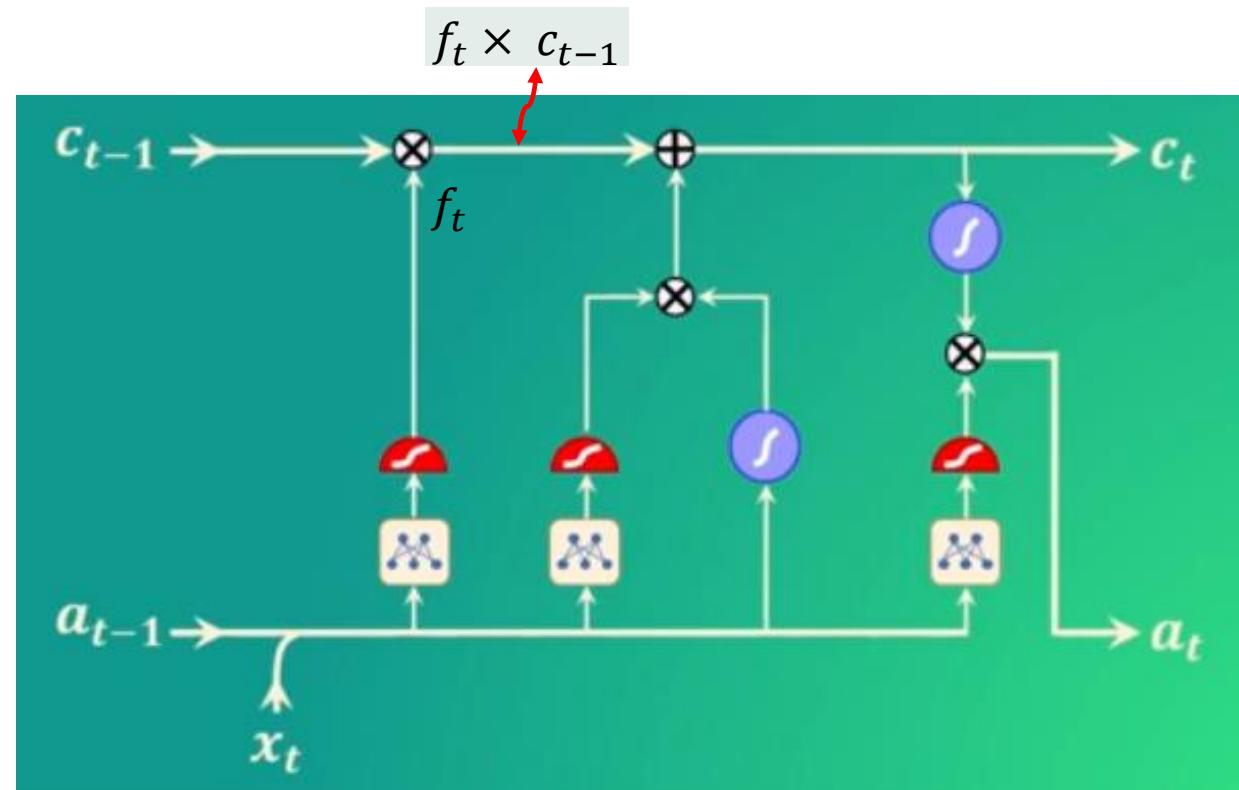


Forget Gate

Retorna un vector con valores entre 0 y 1.
Con valores cercanos a 0 “olvida” y con los cercanos a 1 “recuerda”

LSTM

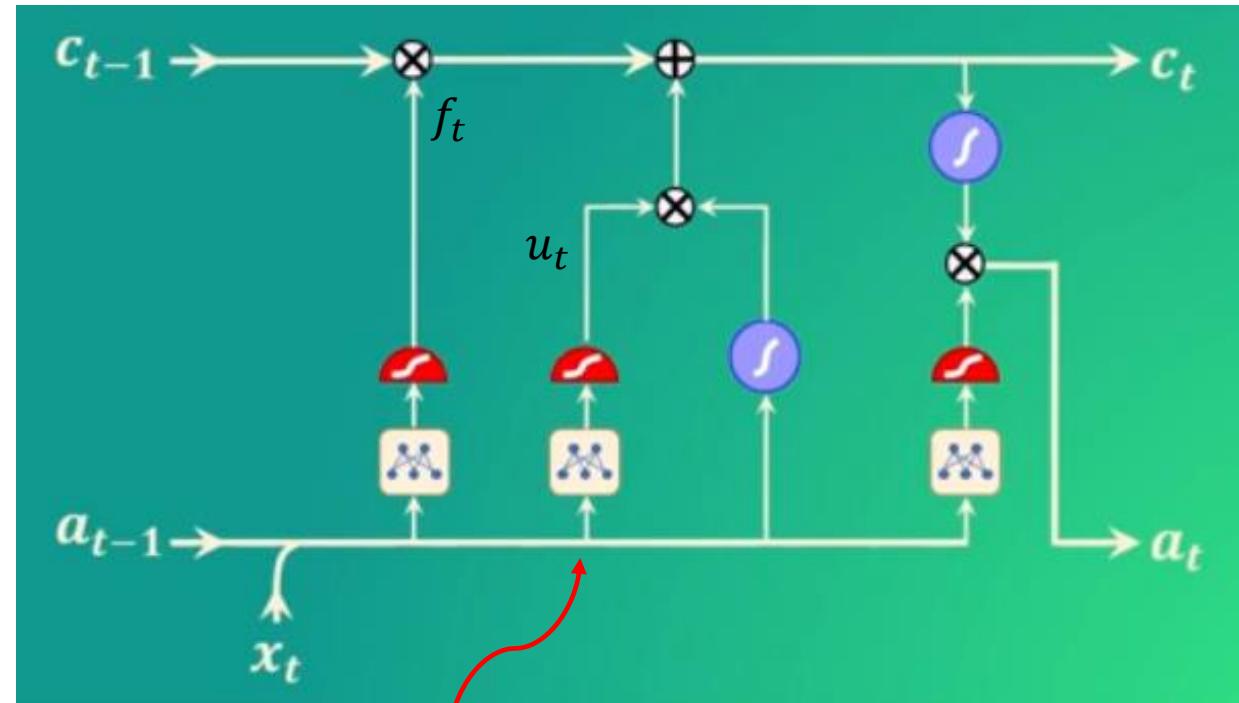
$$f_t = \text{sig}(W_f x_t + U_f a_{t-1} + b_f)$$



LSTM

$$f_t = \text{sig}(W_f x_t + U_f a_{t-1} + b_f)$$

$$u_t = \text{sig}(W_i x_t + U_i a_{t-1} + b_i)$$



Update Gate

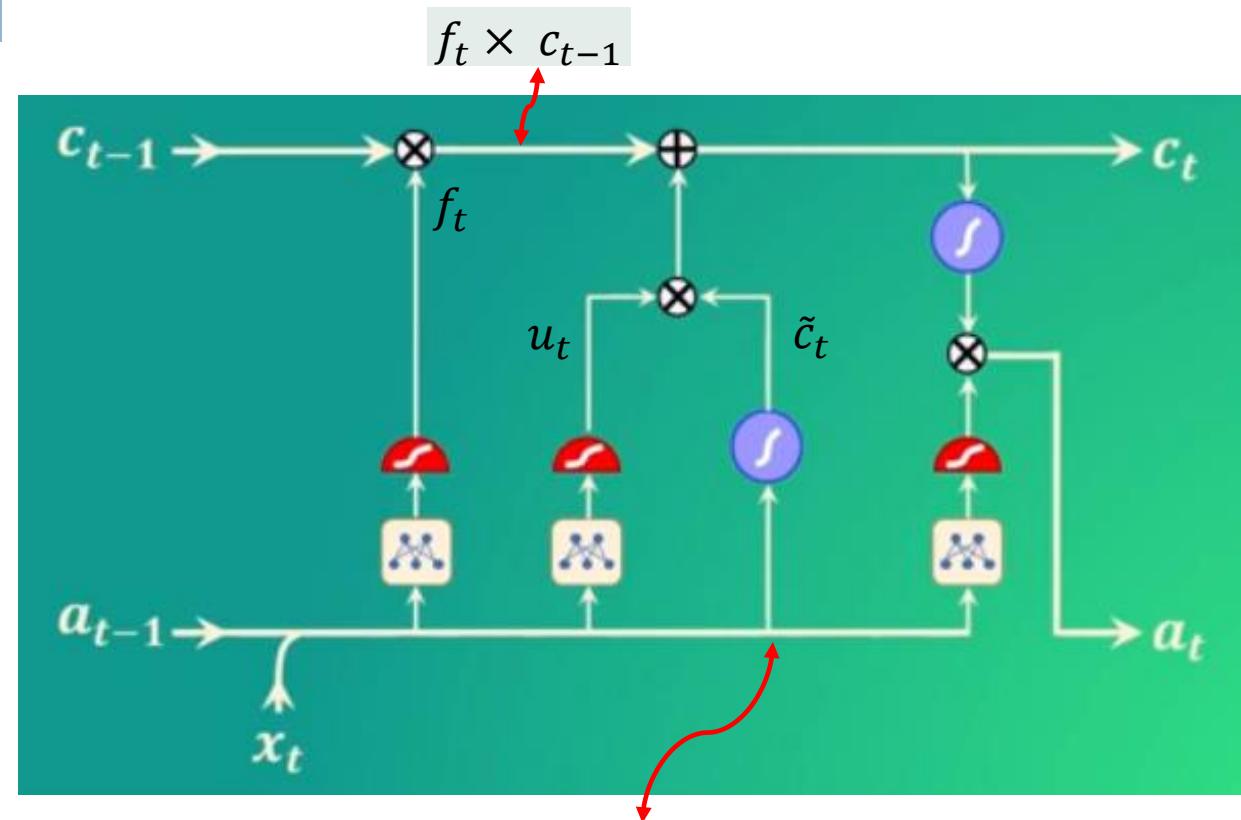
Retorna un vector con valores entre 0 y 1.
Con valores cercanos a 1 serán preservados

LSTM

$$f_t = \text{sig}(W_f x_t + U_f a_{t-1} + b_f)$$

$$u_t = \text{sig}(W_i x_t + U_i a_{t-1} + b_i)$$

$$\tilde{c}_t = \tanh(W_c x_t + U_c a_{t-1} + b_c)$$



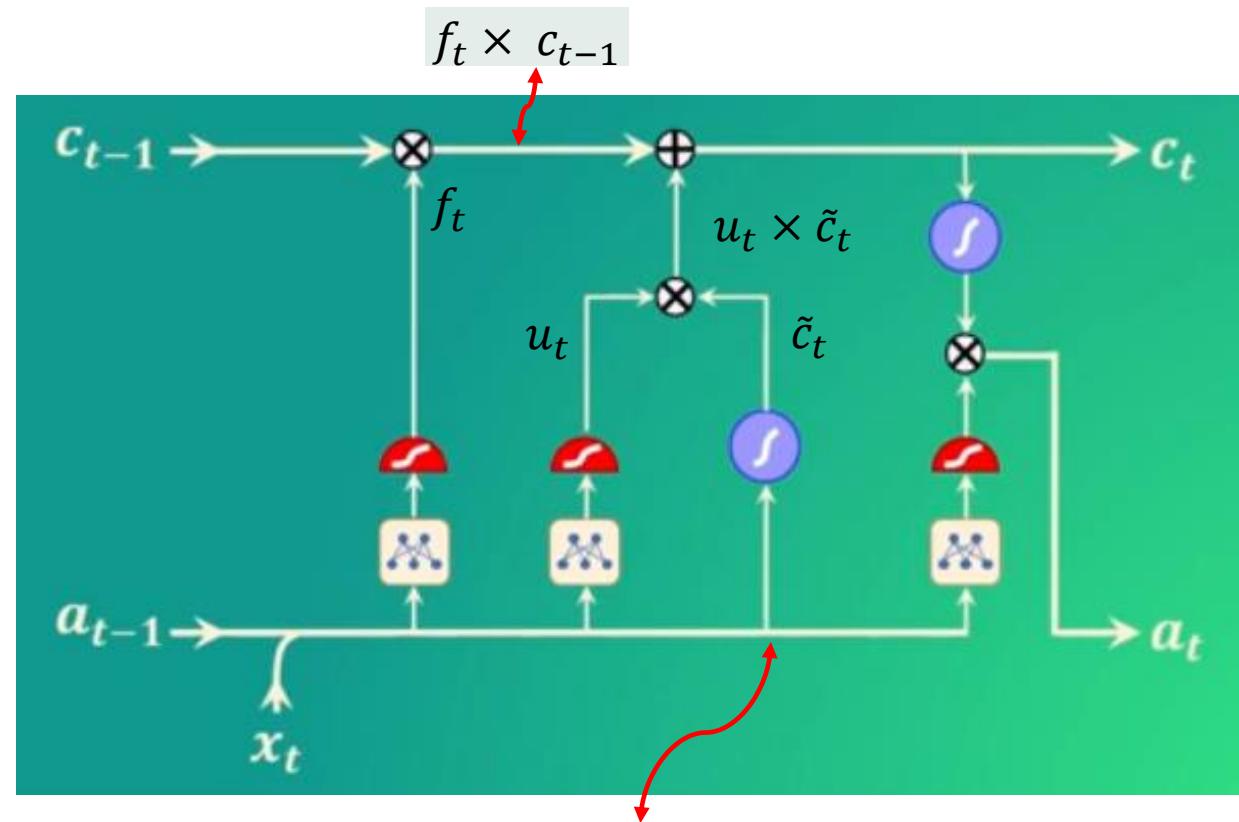
\tilde{c}_t es un vector de valores candidatos a formar parte de la nueva memoria

LSTM

$$f_t = \text{sig}(W_f x_t + U_f a_{t-1} + b_f)$$

$$u_t = \text{sig}(W_i x_t + U_i a_{t-1} + b_i)$$

$$\tilde{c}_t = \tanh(W_c x_t + U_c a_{t-1} + b_c)$$



\tilde{c}_t es un vector de valores candidatos a formar parte de la nueva memoria.

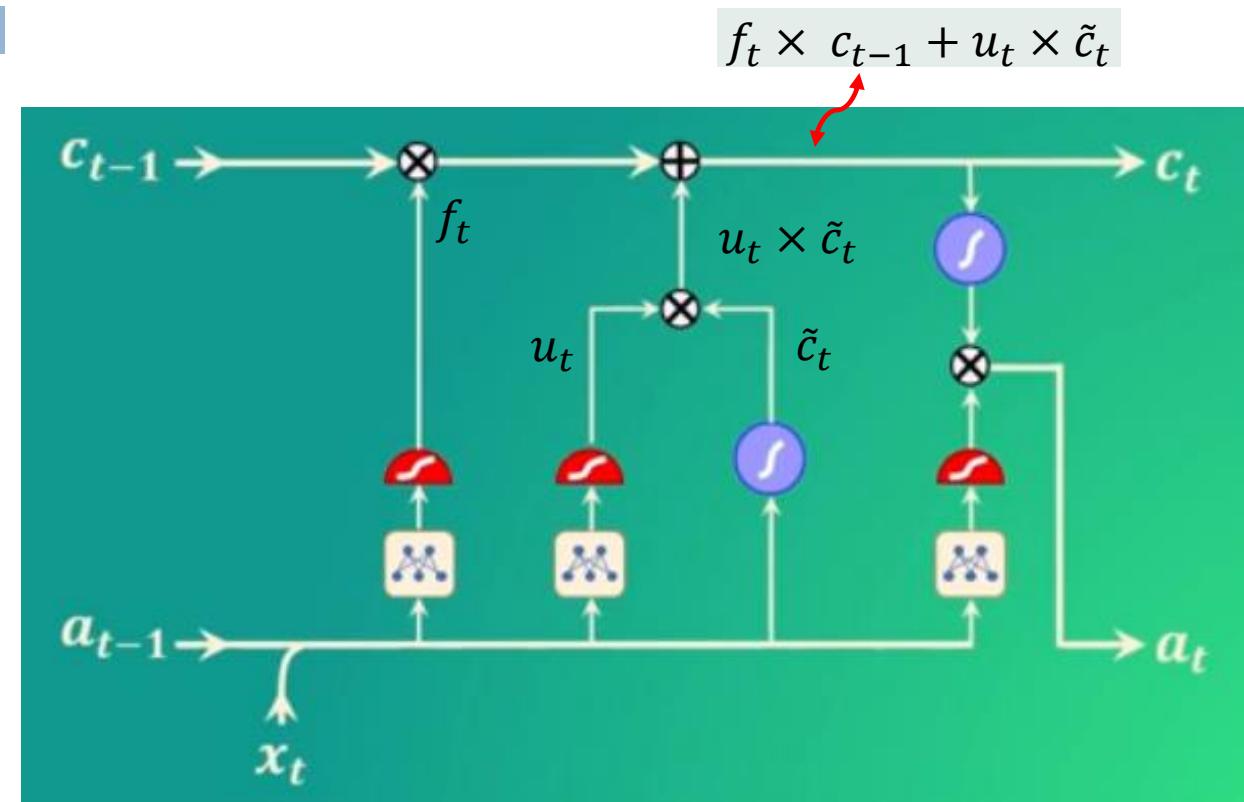
Estos valores se filtran utilizando u_t

LSTM

$$f_t = \text{sig}(W_f x_t + U_f a_{t-1} + b_f)$$

$$u_t = \text{sig}(W_i x_t + U_i a_{t-1} + b_i)$$

$$\tilde{c}_t = \tanh(W_c x_t + U_c a_{t-1} + b_c)$$



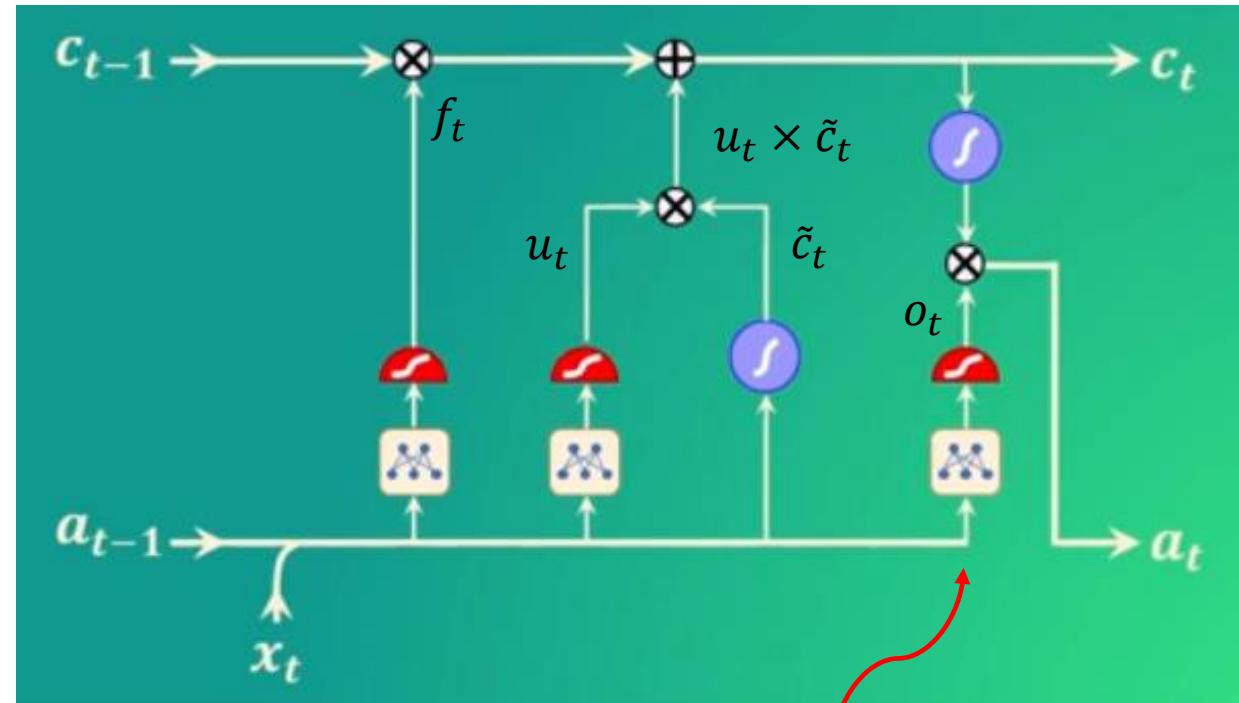
LSTM

$$f_t = \text{sig}(W_f x_t + U_f a_{t-1} + b_f)$$

$$u_t = \text{sig}(W_i x_t + U_i a_{t-1} + b_i)$$

$$\tilde{c}_t = \tanh(W_c x_t + U_c a_{t-1} + b_c)$$

$$o_t = \text{sig}(W_o x_t + U_o a_{t-1} + b_o)$$



Determina qué valores de la memoria formarán parte del nuevo estado oculto a_t

LSTM

02b_UniVar_OneStep_LSTM.ipynb

$$f_t = \text{sig}(W_f x_t + U_f a_{t-1} + b_f)$$

$$u_t = \text{sig}(W_i x_t + U_i a_{t-1} + b_i)$$

$$\tilde{c}_t = \tanh(W_c x_t + U_c a_{t-1} + b_c)$$

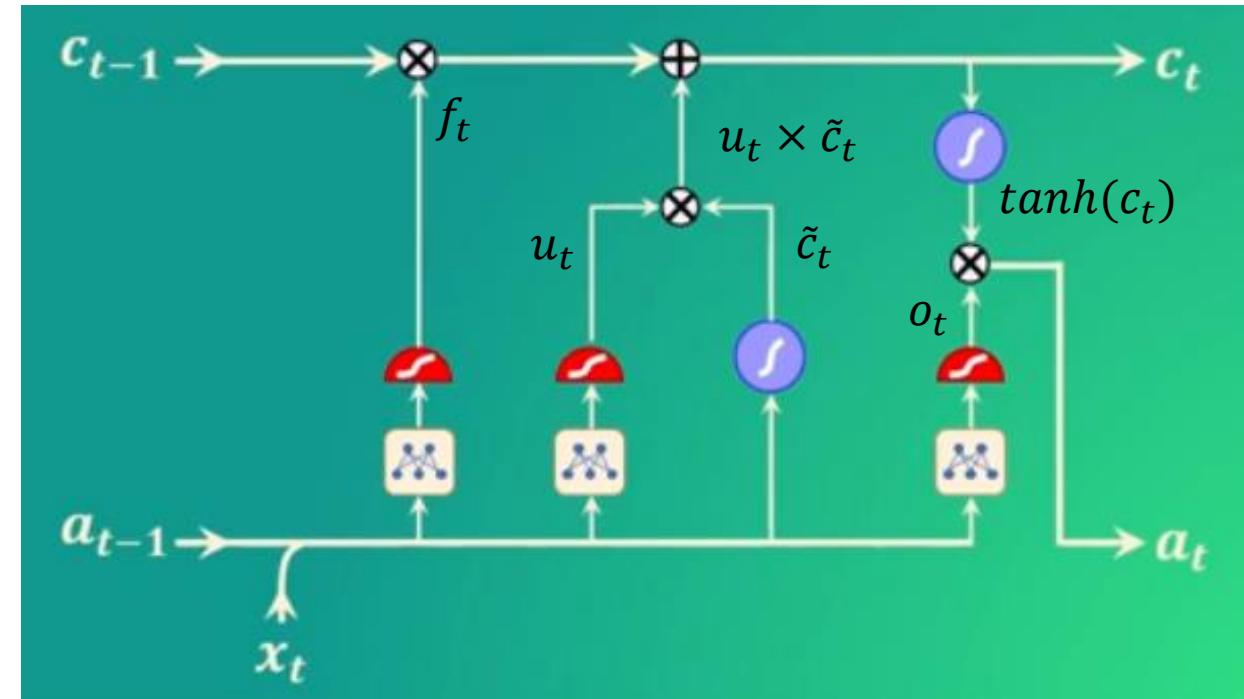
$$o_t = \text{sig}(W_o x_t + U_o a_{t-1} + b_o)$$

$$c_t = f_t \times c_{t-1} + u_t \times \tilde{c}_t$$

$$a_t = o_t \times \tanh(c_t)$$

La dimensión de los vectores f_t , u_t , \tilde{c}_t y o_t coincide con la cantidad h de neuronas ocultas

$$x_t \in \mathbb{R}^d; W \in \mathbb{R}^{h \times d}; U \in \mathbb{R}^{h \times h}; b \in \mathbb{R}^h$$



```

from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import LSTM, Dense

INPUT_SHAPE = (24,1)
OUTPUT_LENGTH = 1
N_UNITS = 1

modelo = Sequential()
modelo.add(LSTM(N_UNITS, input_shape=INPUT_SHAPE))
modelo.add(Dense(OUTPUT_LENGTH, activation='linear'))
modelo.summary()

```

$$x_t \in \mathbb{R}^d;$$

$$W \in \mathbb{R}^{h \times d}; b \in \mathbb{R}^h$$

$$U \in \mathbb{R}^{h \times h}$$

4 de c/u por capa

Model: "sequential"

Layer (type)	Output Shape	Param #
<hr/>		
lstm (LSTM)	(None, 1)	12
dense (Dense)	(None, 1)	2
<hr/>		
Total params: 14 (56.00 Byte)		

```

from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import LSTM, Dense

INPUT_SHAPE = (24,1)
OUTPUT_LENGTH = 1
N_UNITS = 3

modelo = Sequential()
modelo.add(LSTM(N_UNITS, input_shape=INPUT_SHAPE))
modelo.add(Dense(OUTPUT_LENGTH, activation='linear'))
modelo.summary()

```

$$x_t \in \mathbb{R}^d;$$

$$W \in \mathbb{R}^{h \times d}; b \in \mathbb{R}^h$$

$$U \in \mathbb{R}^{h \times h}$$

4 de c/u por capa

Model: "sequential"

Layer (type)	Output Shape	Param #
<hr/>		
lstm (LSTM)	(None, 3)	60
dense (Dense)	(None, 1)	4
<hr/>		
Total params: 64 (256.00 Byte)		