REDES NEURONALES ARTIFICIALES (ANN)

Sergi Ramírez

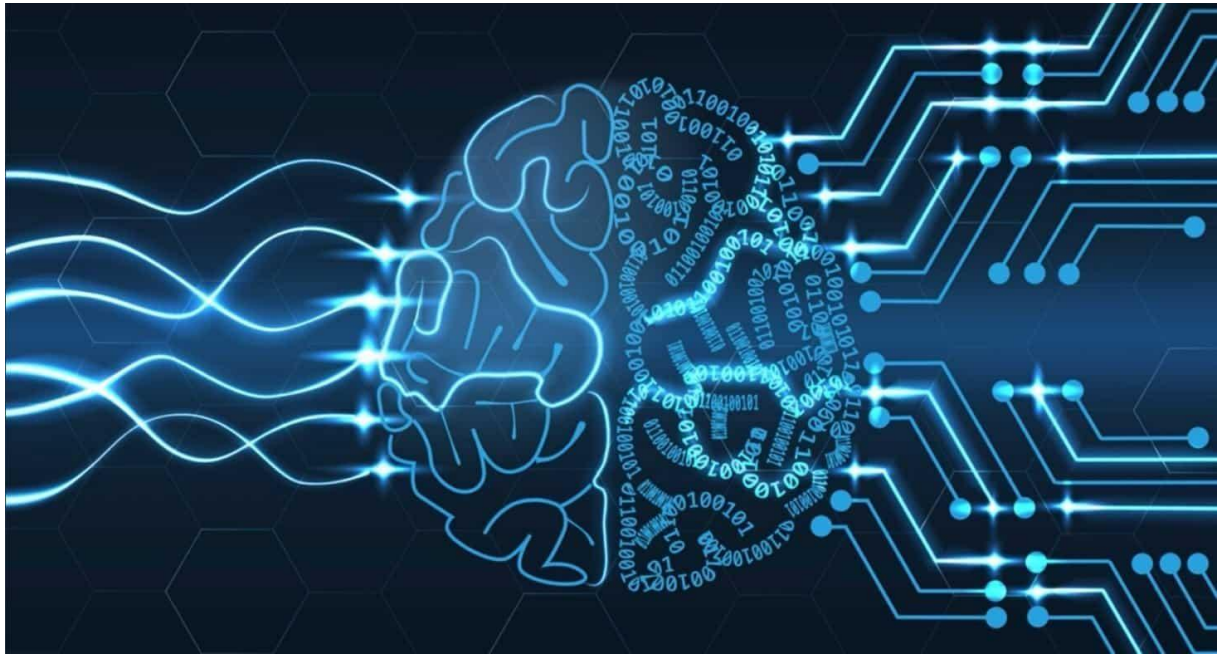
11 de Diciembre de 2025

Universidad de Barcelona - Universidad Politécnica de Cataluña

UNIVERSITAT DE BARCELONA

UNIVERSITAT POLITÈCNICA DE CATALUNYA BARCELONATECH

Índice



Deep Learning

Inteligencia Artificial

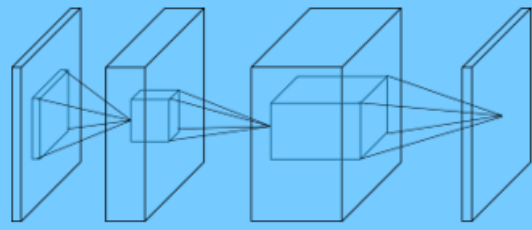
Técnicas que permiten que el computador imite el comportamiento humano

Machine Learning

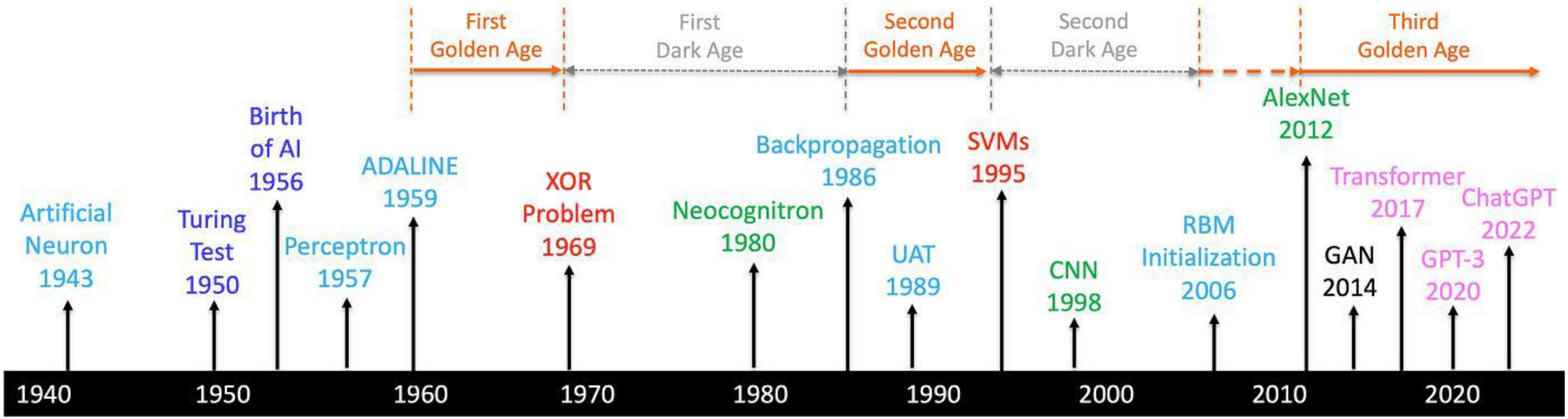
Tiene la habilidad de aprender sin haber sido explícitamente programado para la tarea

Deep Learning

Simula el aprendizaje humano, aprendiendo las mejores características



Un poco de historia



McCulloch-Pitts

Rosenblatt

Widrow-Hoff

Minsky-Papert

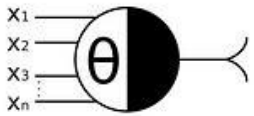
Rumelhart, Hinton et al.

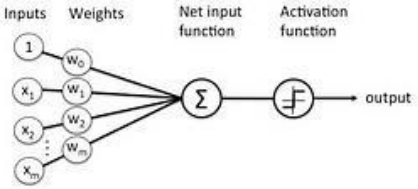
LeCun

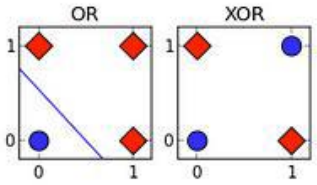
Hinton-Ruslan

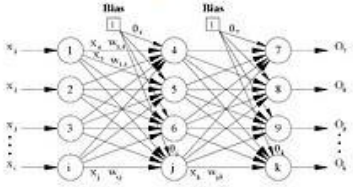
Krizhevsky et al.

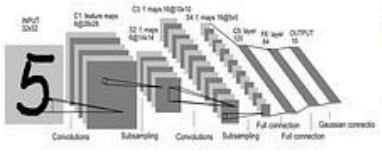
Vaswani

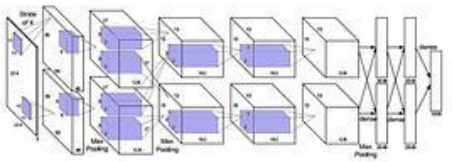












Deep Learning

Big Data

- Conjunto de datos grandes
- Recopilación y almacenamiento más sencillos

Hardware

- Unidades de procesamiento gráfico (GPU's)
- Paralelización masiva

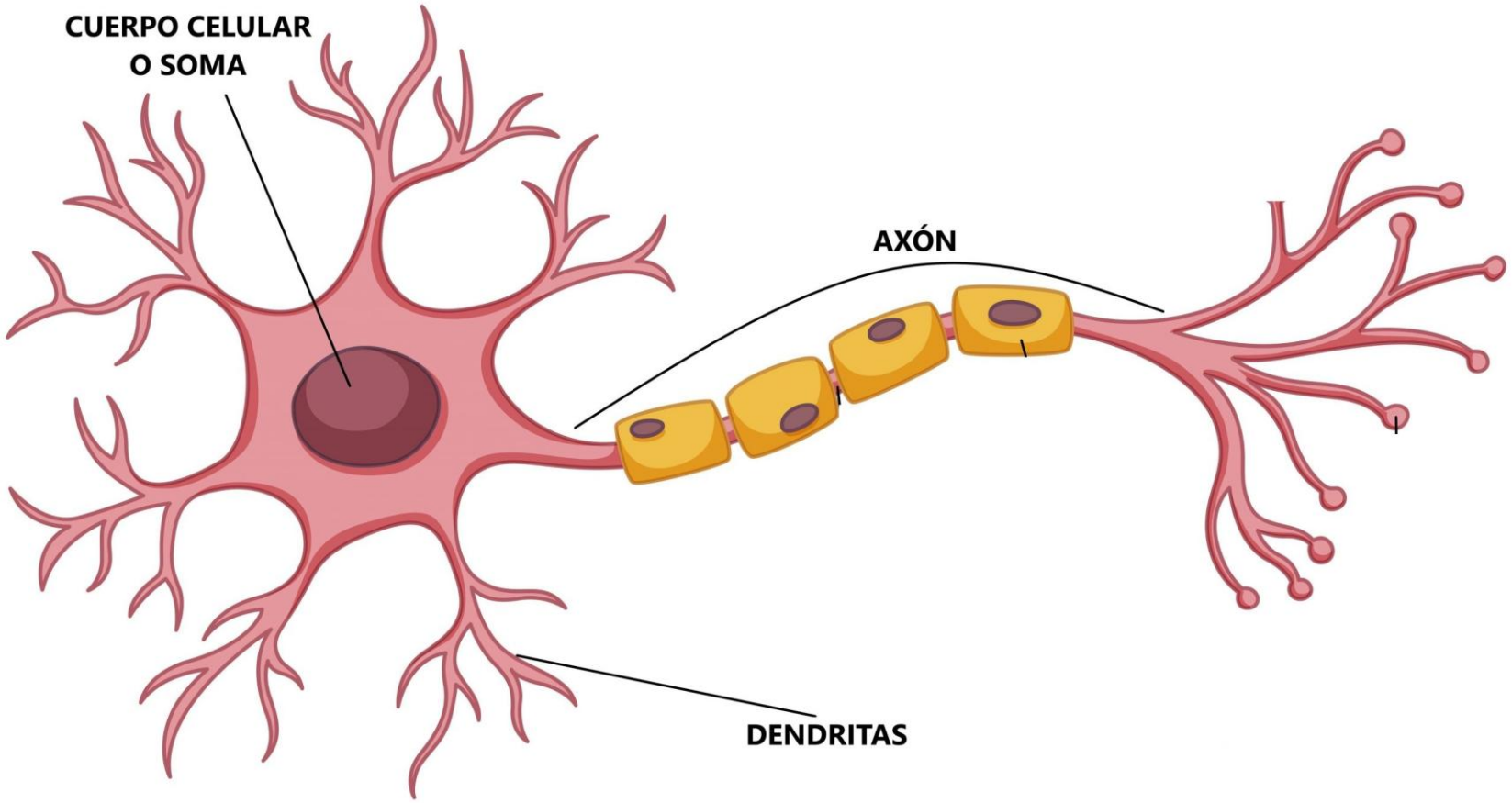
Software

- Técnicas mejoradas
- Nuevos modelos y cajas de herramientas

El modelo biológico

El cerebro humano contiene 10 billones de neuronas aproximadamente. Cada una está conectada con otras 10.000 neuronas.

El modelo biológico



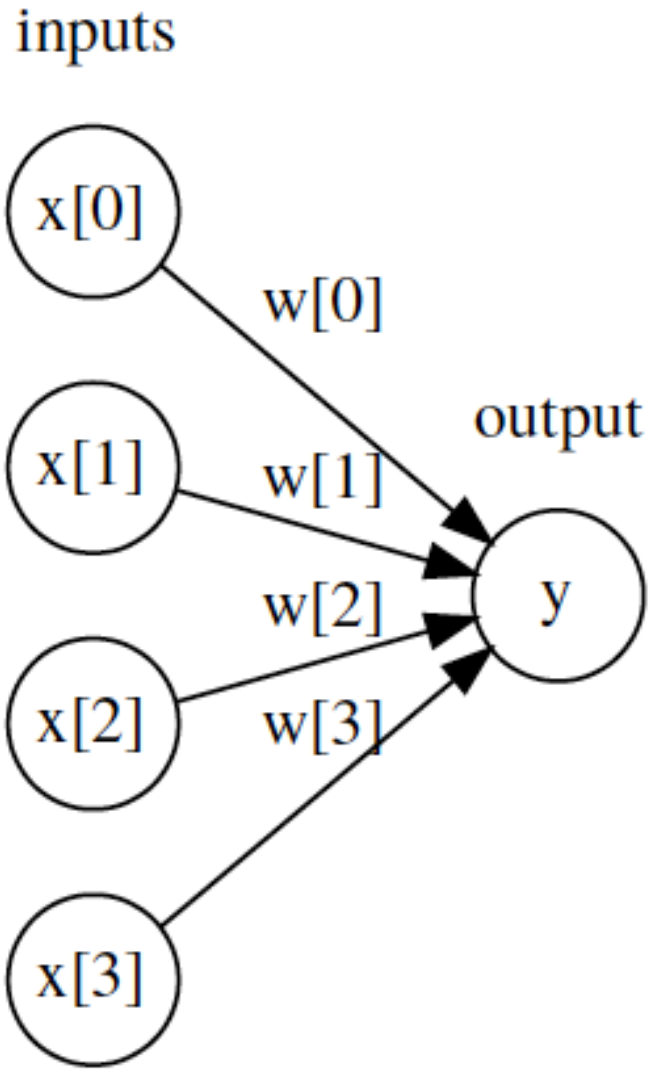
La señal se transmite a través de las neuronas

Artificial Neural Networks (ANN)

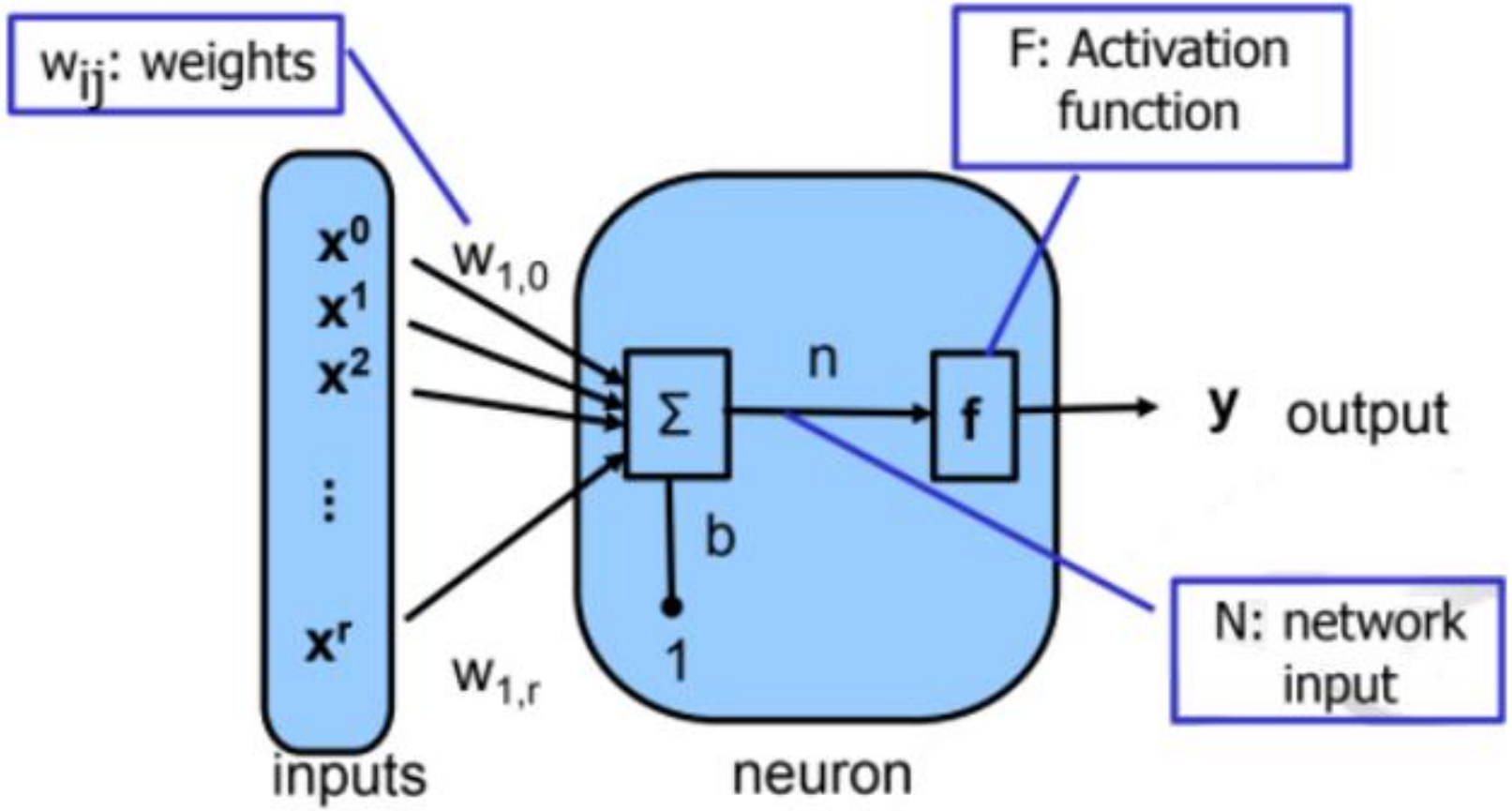
Recta de regresión lineal

$$y = w_i x_i + \cdots + w_d x_d + b$$

Cada neurona de la capa intermedia tiene un valor de sesgo.



Artificial Neural Networks (ANN)



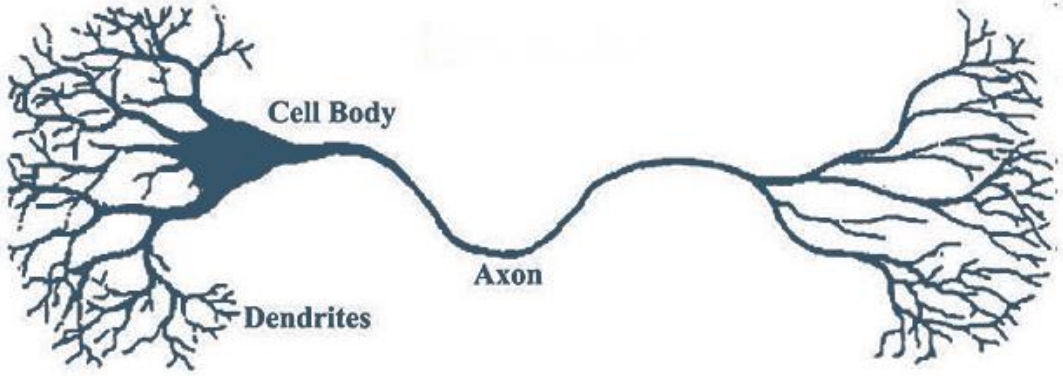
$$y_i = f(w_o + \sum_{j=1}^{P} W_j x_{ij} + \varepsilon_i)$$

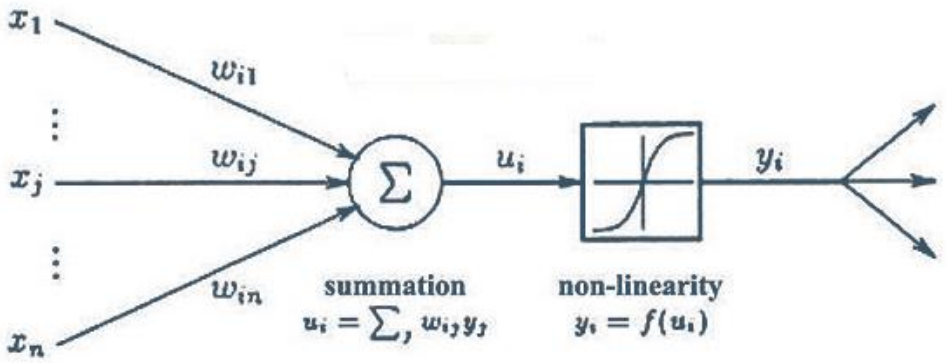
$$net_i = \sum_{i=0}^{P} W_j x_{ij}$$

Artificial Neural Networks (ANN)

Propiedades del cerebro emuladas por las ANN

1. Computación paralela y distribuida
2. Conexión densa de unidades básicas
3. Las conexiones se pueden modificar a través de la experiencia
4. Aprendizaje constante
5. Tolerancia a los errores



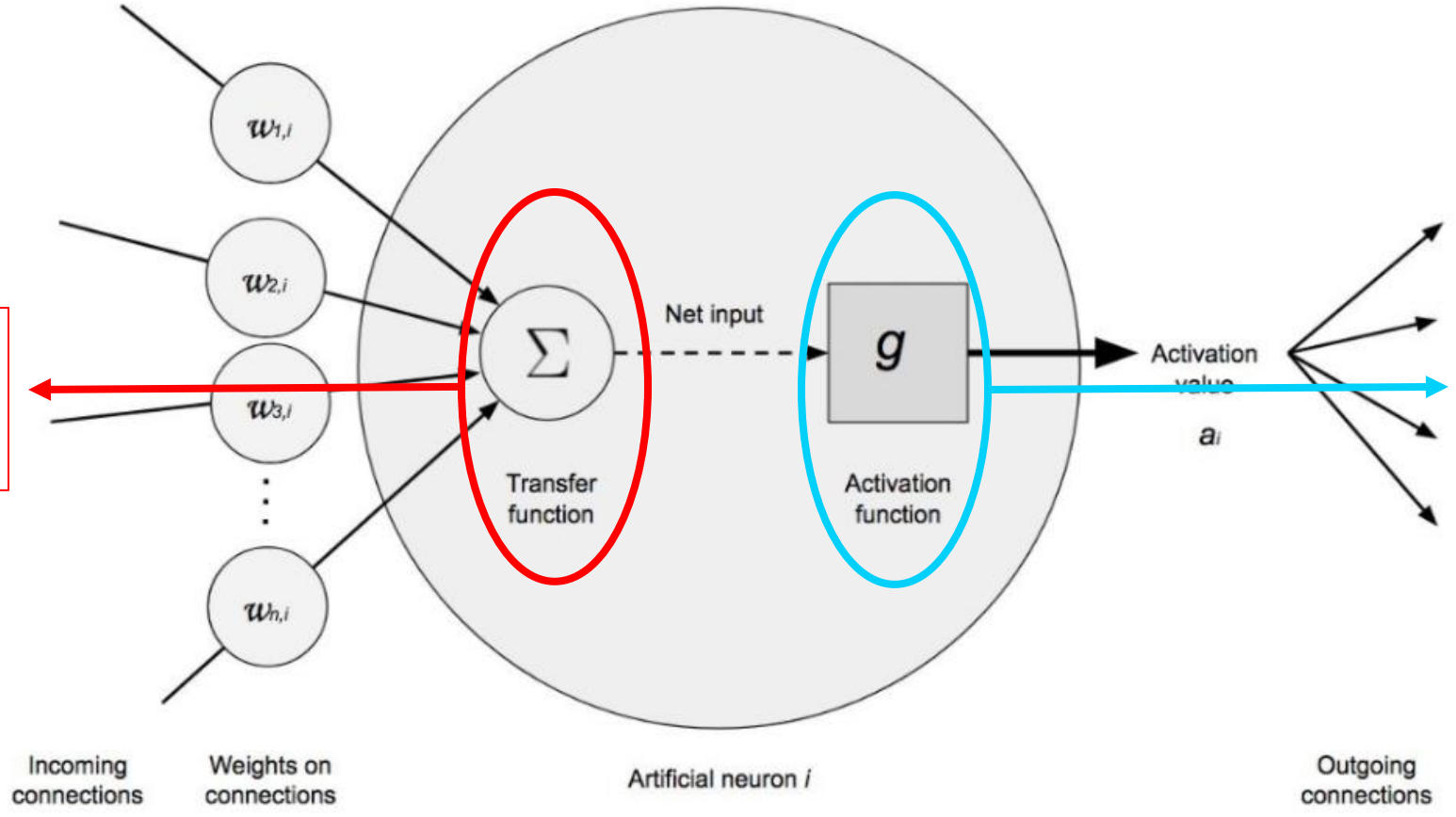


La neurona (Perceptrón)

La neurona es la unidad funcional de los modelos de redes.

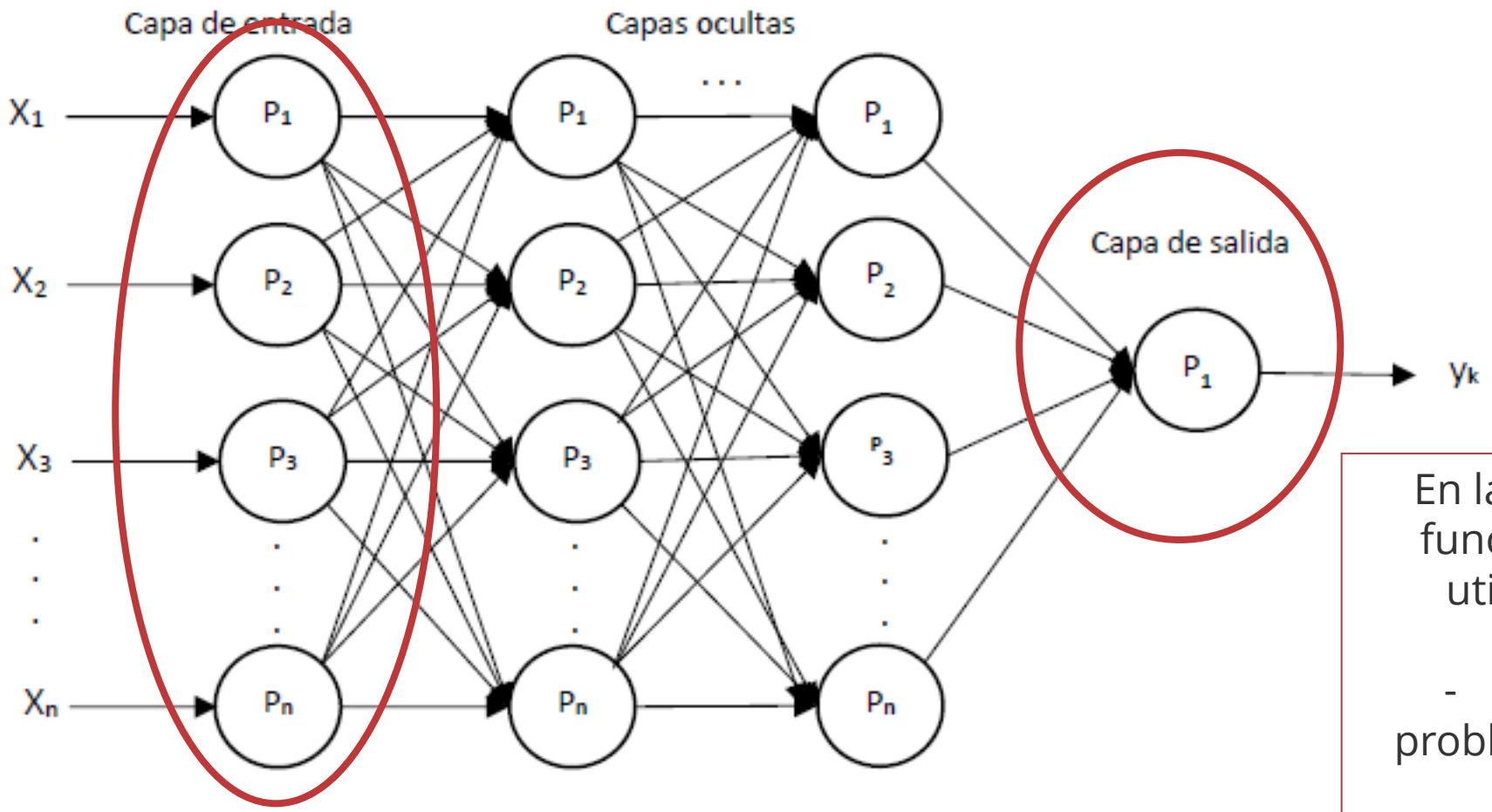
Dentro de cada neurona ocurren dos operaciones:

la suma ponderada de sus entradas



la aplicación de una función de activación.

La neurona (Perceptrón)



Para la capa de entrada la función de activación es la unidad.

En la capa de salida la función de activación utilizada suele ser:

- **Identidad** para problemas de regresión
- **Softmax** para clasificación

La neurona (Perceptrón)

El **teorema de la convergencia** del perceptrón establece que, en los problemas en los que haya dos clases linealmente separables, es siempre posible encontrar unos pesos que realicen la separación en un número finito de iteraciones (Novikoff, 1962)

la mayoría de los problemas que se estudian no son linealmente separables

No es posible obtener un conjunto de variables que separen perfectamente, de forma lineal, las observaciones de cada clase..

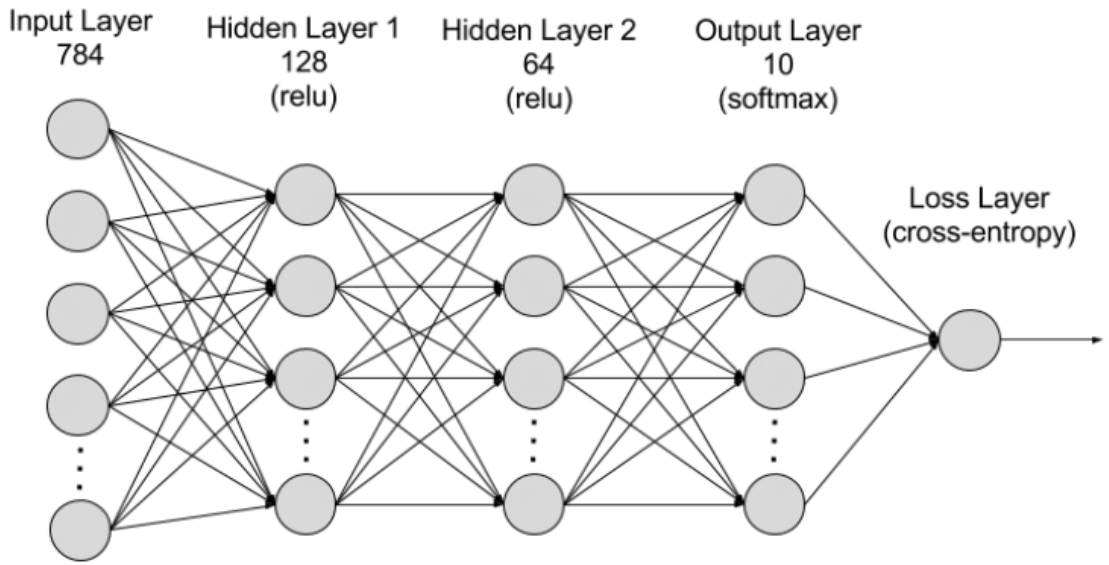
Algoritmo Pocket
Guarda la mejor solución obtenida hasta el final del entrenamiento.

Algoritmo Maxover
Halla el margen de separación máximo permitiendo clasificaciones incorrectas.

Algoritmo de Voto
Utiliza múltiples perceptrones combinando sus salidas

Función de activación

Las funciones de activación **controlan qué información** se propaga desde una capa a la siguiente (***forward propagation***).



Convierten el valor de entrada (combinación de los input, pesos y sesgo) en un nuevo valor dentro del rango (0, 1) o (-1, 1).

Combinar funciones de activación no lineales con múltiples capas, son capaces de aprender relaciones no lineales.

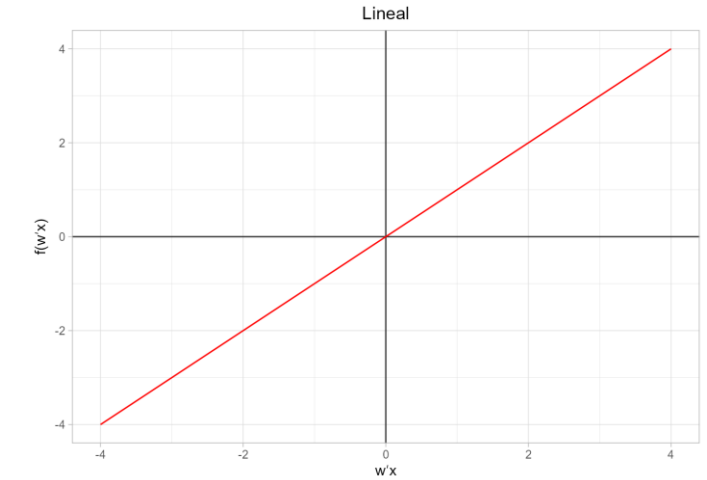
Cuando el valor de activación de una neurona es cero, se dice que la neurona está **inactiva**

Función de activación

Función Lineal

Se trata de una función identidad donde la salida tiene el mismo valor que la entrada. Normalmente se aplica en problemas de regresión lineal.

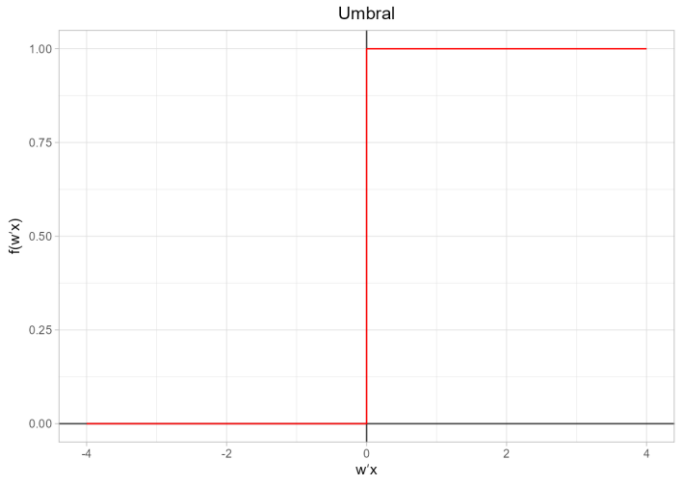
$$\boldsymbol{Lineal(x) = x}$$



Función Umbral (escalón)

Si el valor de entrada es menor que el umbral, u, la salida es 0. En caso contrario, la salida es 1. Si el umbral es 0, la función se reduce a observar el signo del valor de la salida.

$$\mathbf{Umbral}(x) = \begin{cases} \mathbf{0} & \boldsymbol{si\ x < umbral} \\ \mathbf{1} & \boldsymbol{en\ otro\ caso} \end{cases}$$

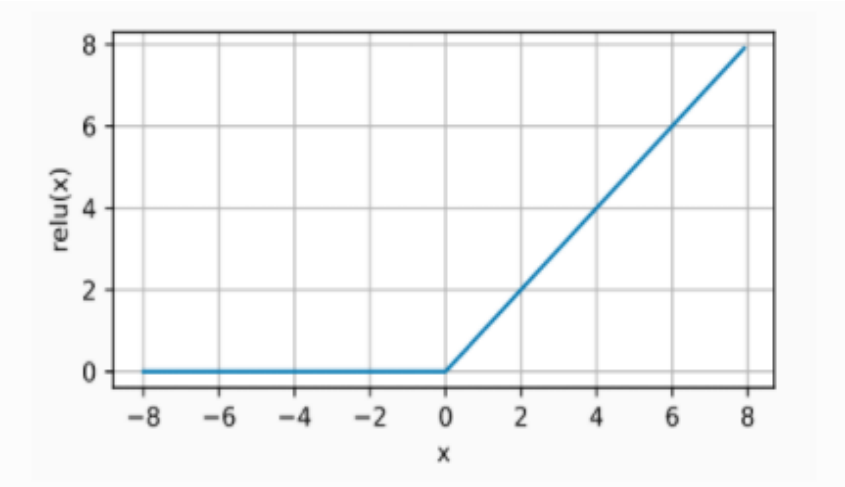


Función de activación

Rectified linear unit (ReLU)

Aplica una transformación que activa la neurona solo si el *input* está por encima de cero. Mientras el valor de entrada está por debajo de cero, el valor de salida es cero, pero cuando es superior, el valor de salida aumenta de forma lineal con el de entrada.

$$\boldsymbol{ReLU(x) = \max(0, x)}$$

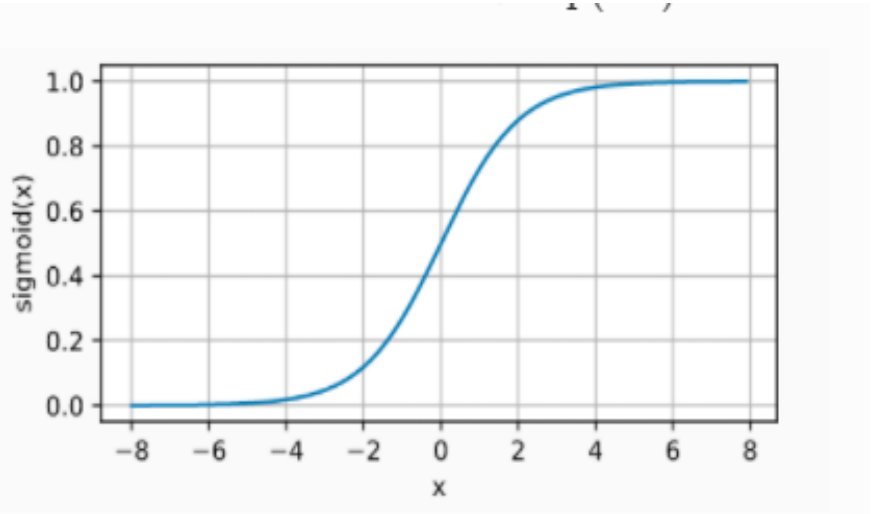


Sigmoide

Transforma valores en el rango de (-inf, +inf) a valores en el rango (0, 1).

Es la función por defecto es en las neuronas de la capa de salida de los modelos de **clasificación binaria**, ya que su salida puede interpretarse como probabilidades.

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

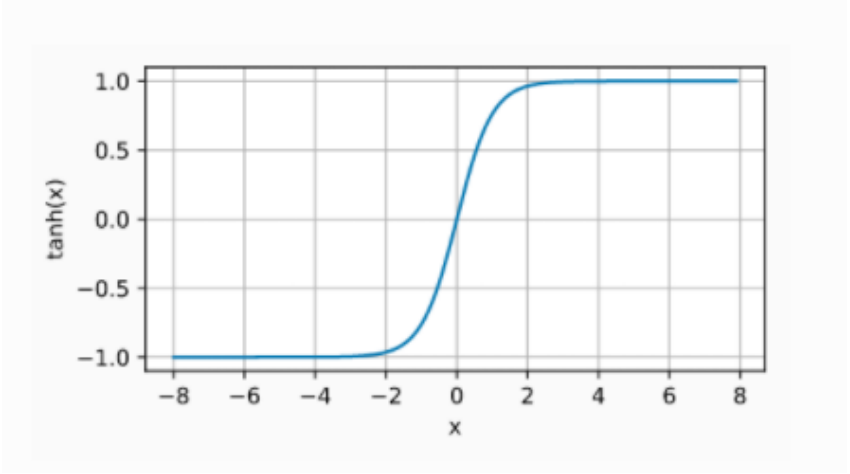


Función de activación

Tangente hiperbólica (Tanh)

La función de activación su salida está acotada en el rango (-1, 1).

$$\boldsymbol{tanh(x) = \frac{1 - exp(-2x)}{1 + exp(-2x)}}$$



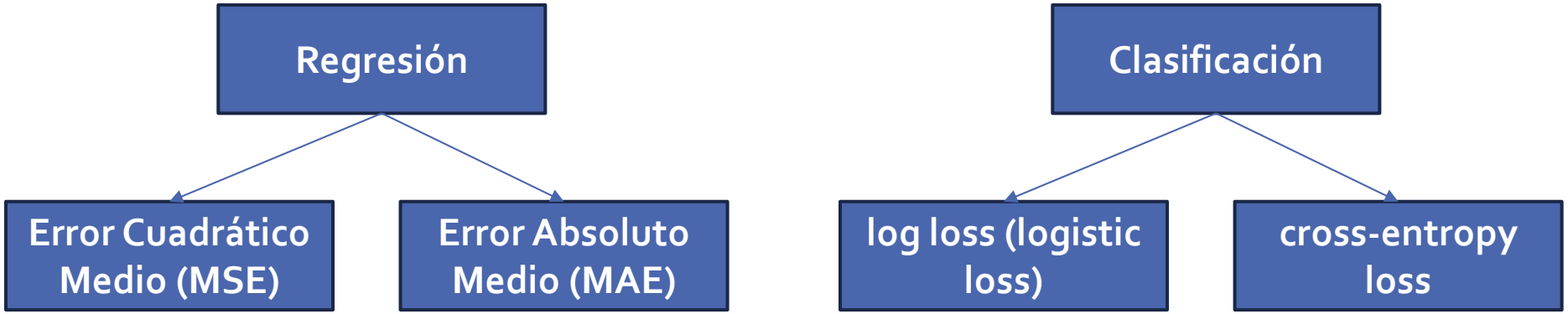
Sin las funciones de activación, las redes neuronales solo pueden aprender relaciones lineales.

Función de coste (loss function)

La **función de coste** (o de pérdida) se encarga de **cuantificar la distancia entre el valor real y el valor predicho** por la red (mide cuánto se equivoca la red al realizar predicciones)

Cuanto más próximo a cero es el valor de coste, mejor son las predicciones de la red (menor error), siendo cero cuando las predicciones se corresponden exactamente con el valor real.

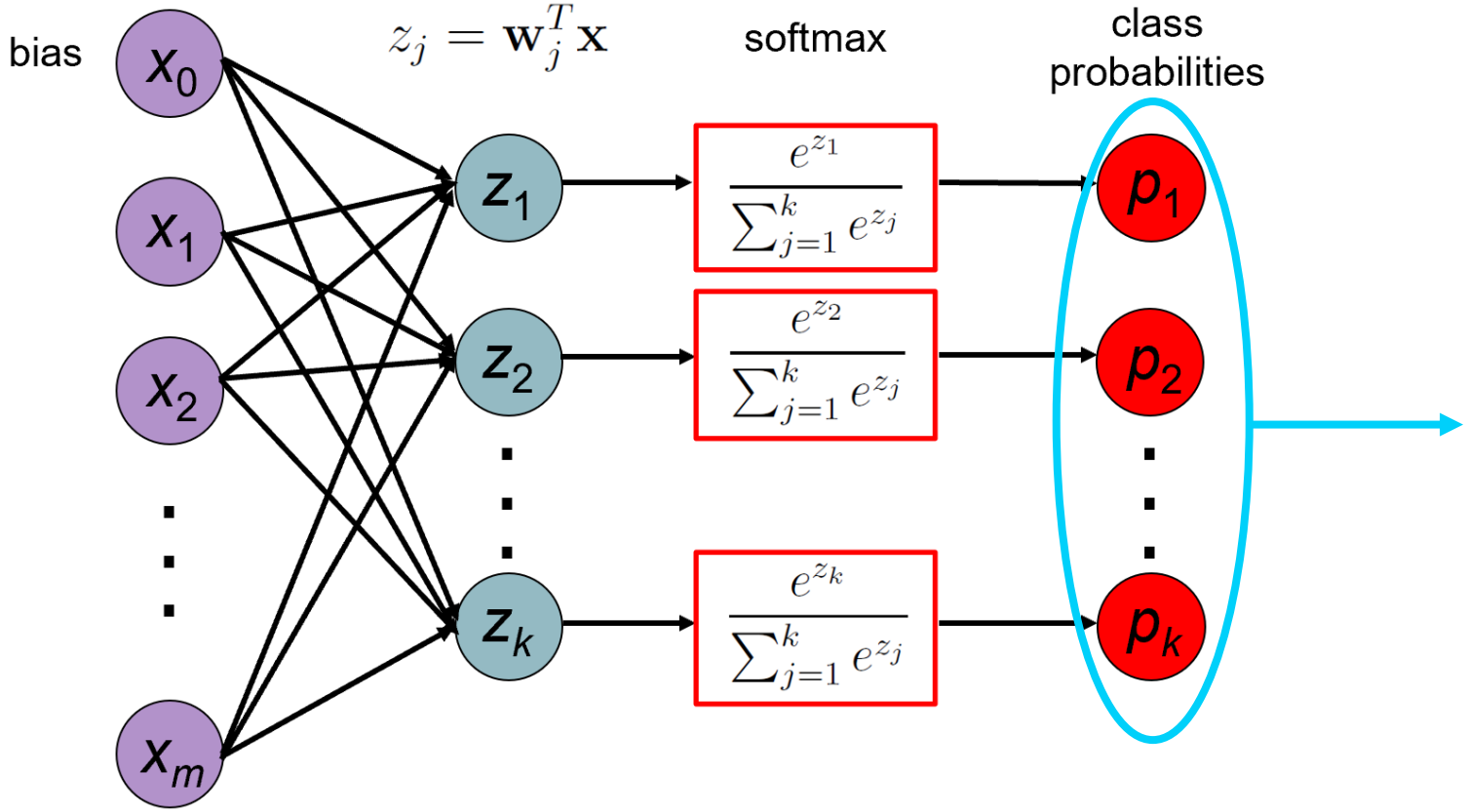
Dependiendo del tipo de problema es necesario utilizar una función de coste u otra.



Función de coste (loss function)

Log loss, logistic loss o cross-entropy loss

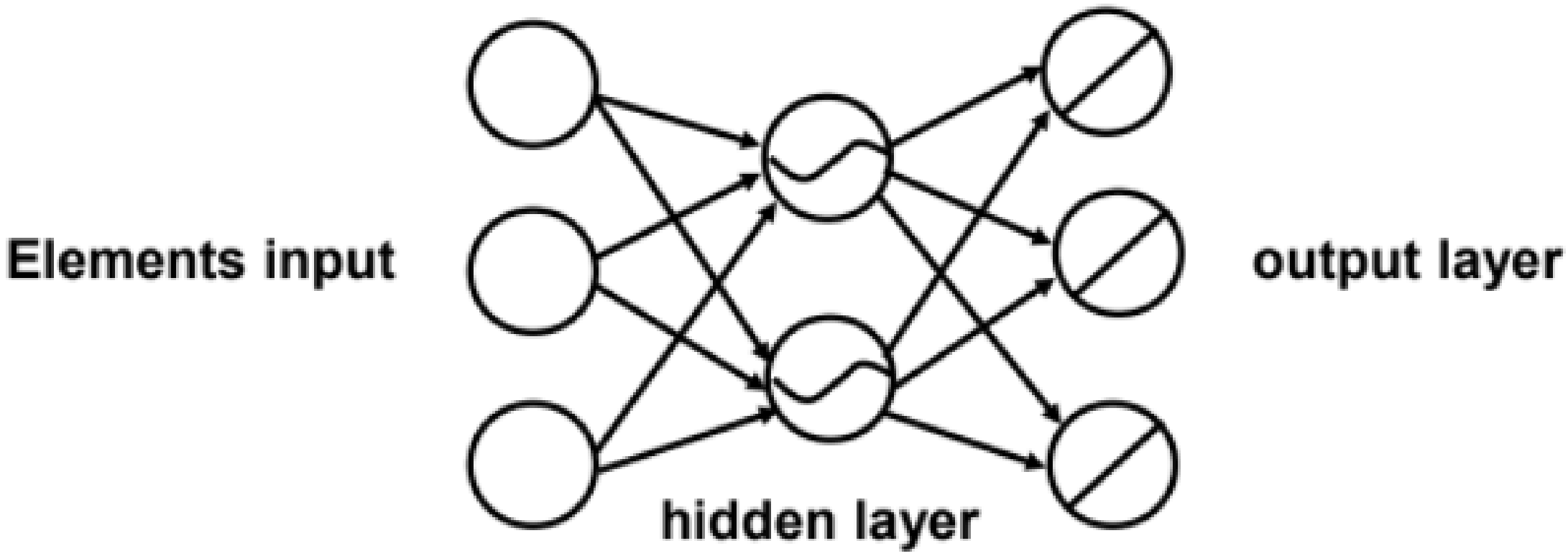
En problemas de clasificación, la capa de salida utiliza como función de activación la función ***softmax***.



La red devuelve la **probabilidad** de que la observación predicha pertenezca a cada una de las posibles clases.

Artificial Neural Networks (ANN)

Estructura de una red multi-capas

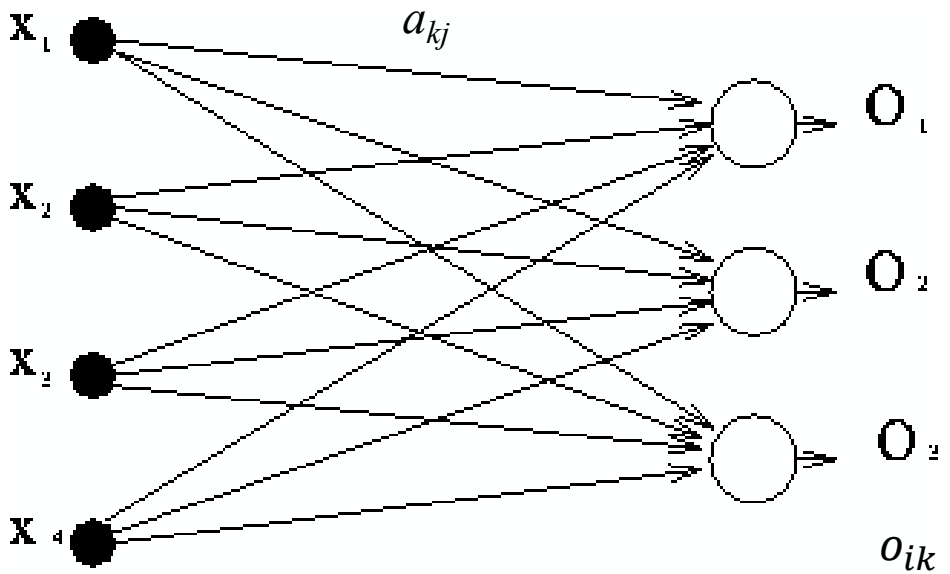


El perceptrón (Rosemblatt, 1957)

Red neuronal con una capa

El perceptrón (red neuronal con una capa) sólo puede discriminar regiones que sean linealmente separables.

Se trata de un sistema de ecuaciones de regresión simultáneas con función de activación no lineal.



$$o_{ik} = g(y_{ik})$$

$$y_{ik} = \sum_{j=1}^{P} a_{kj} x_{ij}$$

Función de coste

$$E_i = \frac{1}{2} \sum_{k=1}^{N} (g(a'x) - t_{ik})^2$$

Estimación de pesos

$$a_{kj}^{t+1} = a_{kj}^{t} - \eta \frac{\partial E_i}{\partial a_{kj}}$$

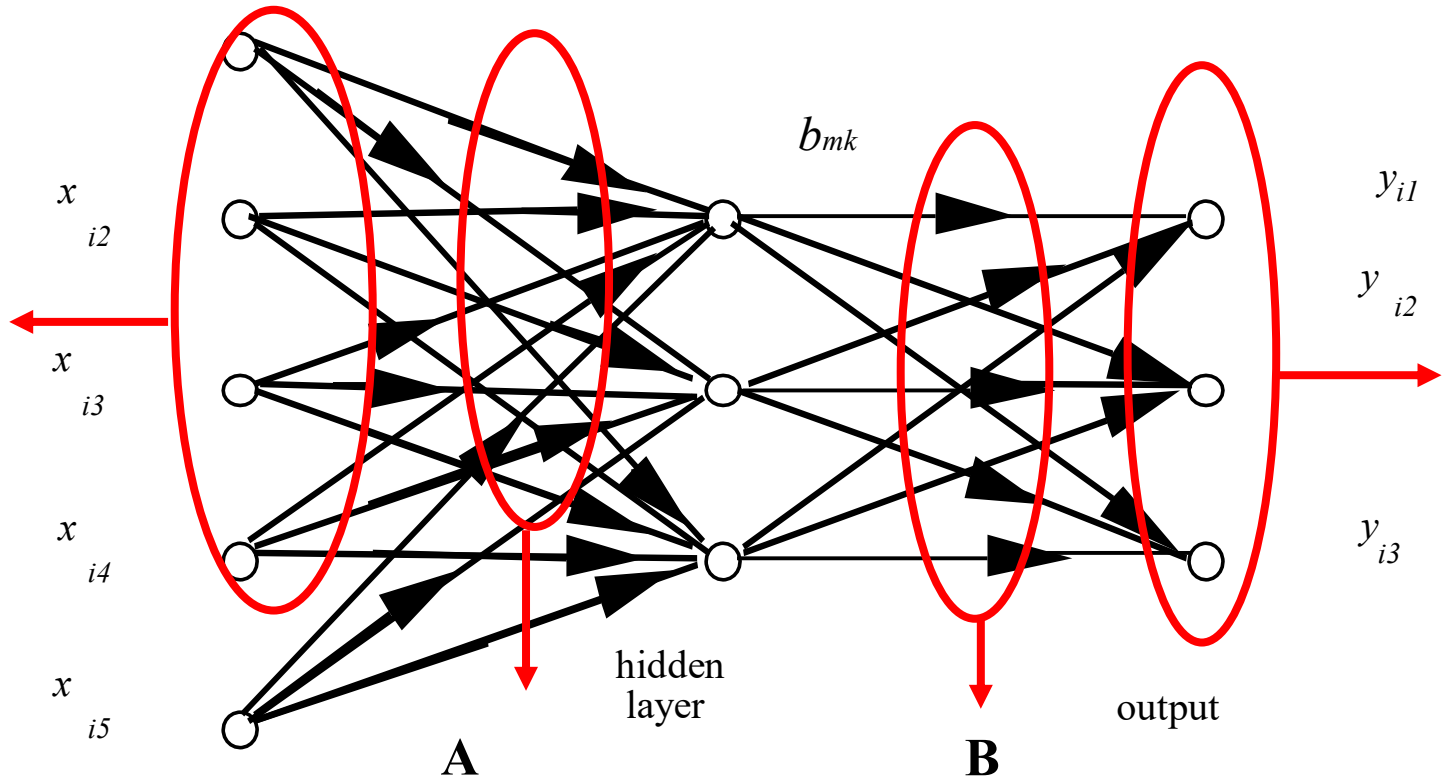
$$\frac{\partial E_i}{\partial a_{kj}} = \frac{(a_{ijk} - t_{ik}) g'(y_{ik})}{x_{ik}}$$

Perceptrón multicapa supervisado

Peceptrón con una capa oculta (red neuronal de 2 capas)

$\boldsymbol{X}(n,p)$

p variables de entrada (normalizada)



$\boldsymbol{Y}(n,q)$

q variables de salida, binarias o continuas

matriz (p, c) de pesos entre las neuronas de entrada y las de la capa oculta,

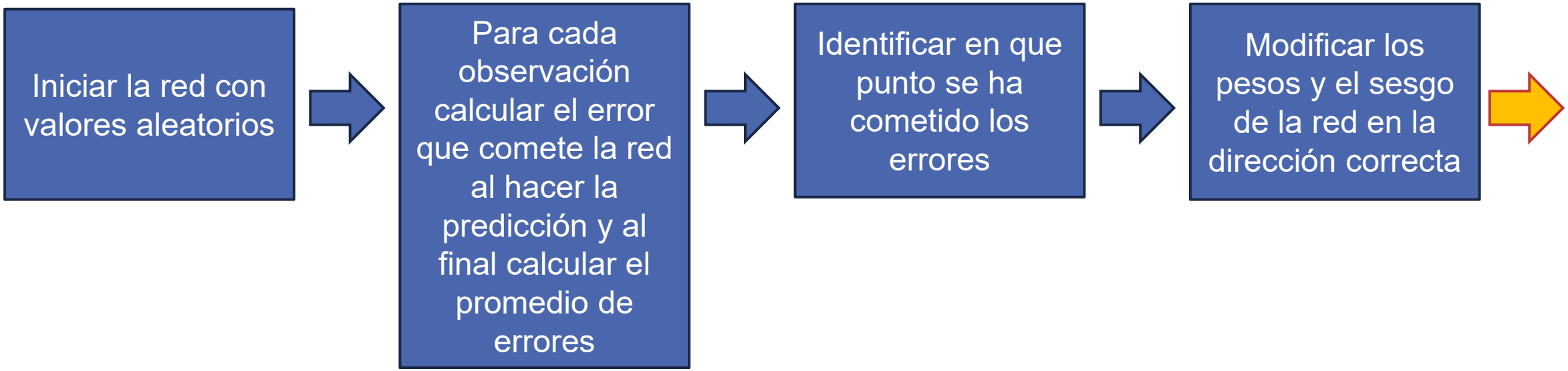
la matriz (c, q) de pesos entre las neuronas de la capa oculta y las de salida.

oculta,

Entrenamiento

El proceso de entrenamiento consiste en **ajustar el valor de los pesos y sesgo** de tal forma que, las predicciones que se generen tengan el **menor error posible**.

El entrenamiento de la red sigue los siguientes pasos:



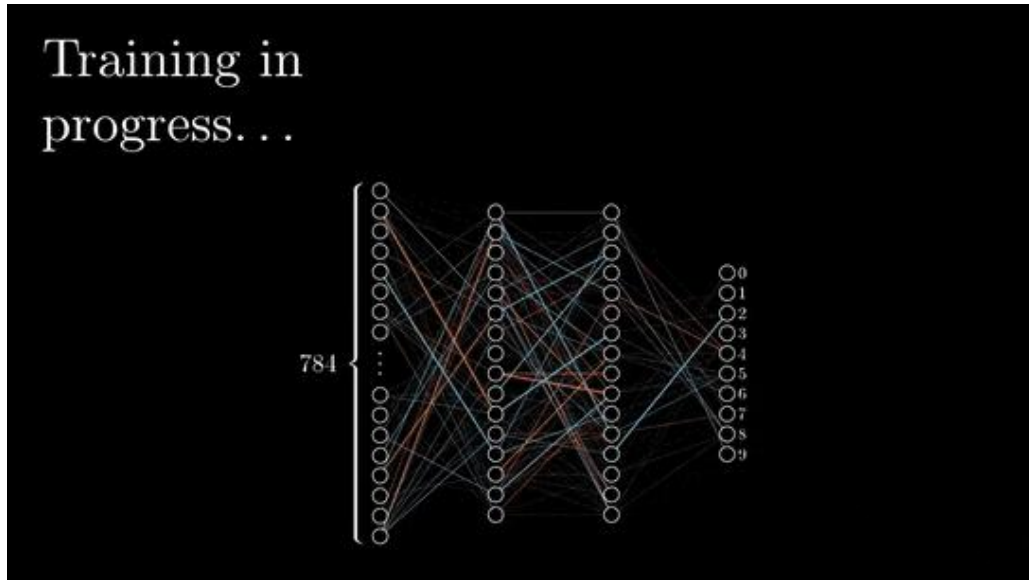
Para realizar este proceso recurrimos a la **retropropagación** (backpropagation) y la **optimización por descenso del gradiente** (gradient descent).

Entrenamiento

Backpropagation

Algoritmo que permite **cuantificar la influencia** que tiene cada peso y sesgo en las predicciones de la red.

Hace uso de la regla de la cadena para calcular el gradiente (vector formado por las derivadas parciales de una función).

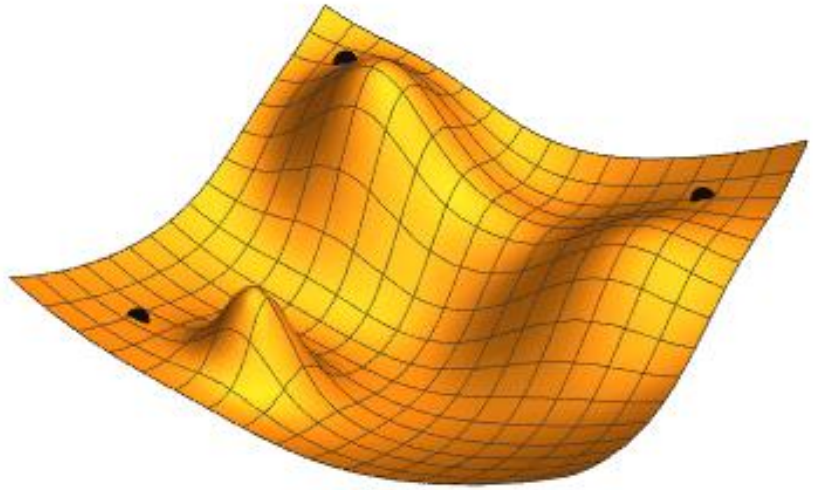


Mide cuánta “responsabilidad” ha tenido ese parámetro en el error cometido. Gracias a esto, se puede identificar qué pesos de la red hay que modificar para mejorarla.

Entrenamiento

Descenso del gradiente

Algoritmo de optimización que permite **minimizar una función** haciendo actualizaciones de sus parámetros en la dirección del valor negativo de su gradiente.



Suele utilizarse el gradiente estocástico que consiste en dividir el conjunto de entrenamiento en lotes (*minibatch* o *batch*) y actualizar los parámetros de la red con cada uno.

Preprocesado (transformaciones)

One hot encoding (Binarización)

Consiste en crear nuevas variables dummy con cada uno de los niveles de las variables cualitativas.

One Hot Encoding

color	color_red	color_blue	color_green
red	1	0	0
green	0	0	1
blue	0	1	0
red	1	0	0

Modificación del peso de la varianza en el modelo

Para predictores numéricos y con magnitud de varianzas muy diferentes es necesario igualar los predictores ya que aquellos que tengan más varianza dominarán el modelo, aunque no sean los que más relación tienen con la variable respuesta.
Existen principalmente 2 estrategias para evitarlo:

- **Centralizar**: Consiste en restar a cada valor la media del predictor
- **Normalizar**

Preprocesado (transformaciones)

Normalización o Estandarización

Consiste en transformar los datos de forma que todos los predictores estén en la misma escala

Normalización

Dividir cada predictor entre su desviación tipo después de centrarlo

Estandarización

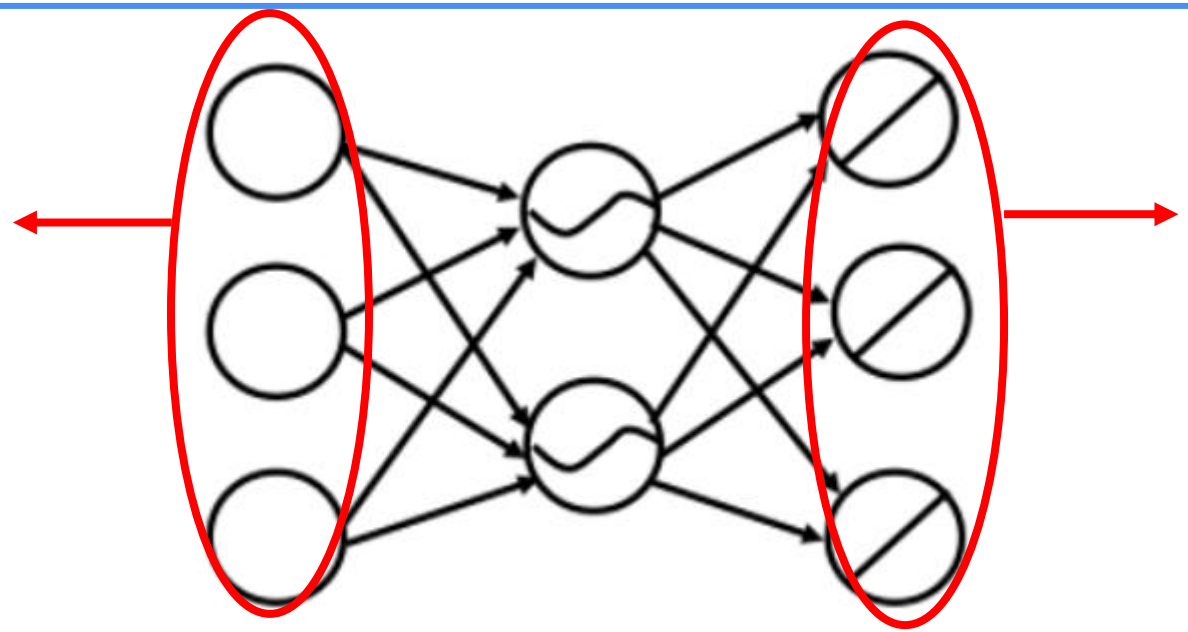
Transforma los datos para incorporarlo dentro del rango [0, 1]

Hiperparámetros

Número y tamaño de capas

El número de capas y el número de neuronas determinan la complejidad del modelo y su potencial capacidad de aprendizaje.

La **capa de entrada** tiene tantas neuronas como predictores



La capa de salida tiene:

- **Regresión**: una neurona
- **Clasificación**: tantas como clases menos una.

Cuantas más neuronas y capas, mayor la complejidad de las relaciones que puede aprender el modelo.

OJOOO!! Cómo cada neurona está conectada por pesos al resto de neuronas de las capas adyacentes, el número de parámetros a aprender aumenta y con ello el tiempo de entrenamiento.

Hiperparámetros

Learning rate

Cómo de rápido pueden cambiar los parámetros de un modelo a medida que se optimiza

Características:

- depende de los datos e interacciona con el resto de hiperparámetros
- *learning rate* muy grande, el modelo no será capaz de aprender
- *learning rate* muy pequeño, el proceso tardará demasiado y no llegará a completarse

Recomendaciones:

- Utilizar un *learning rate* lo más pequeño posible siempre
- Utilizar valores mayores al inicio y pequeños al final

Hiperparámetros

Algoritmo de optimización

Recomendaciones:

- Conjunto de **datos pequeños**: *l-bfgs*
- Conjunto de **datos grandes**: *adam* o *rmsprop*

La elección del algoritmo de optimización puede tener un impacto notable en el aprendizaje de los modelos, sobre todo en *deep learning.*

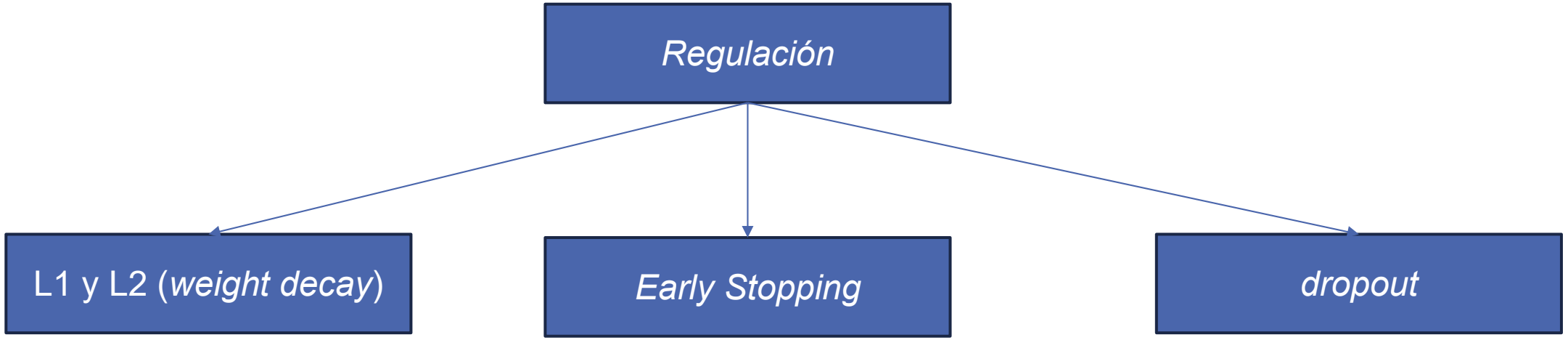
Hiperparámetros

Regulación

Tienen el objetivo de reducir el sobreajuste (*overfitting*) de los modelos

Un modelo con sobreajuste memoriza los datos de entrenamiento, pero es incapaz de predecir correctamente nuevas observaciones.

Los modelos de redes neuronales son modelos sobre parametrizados, por lo tanto, las regularizaciones son fundamentales

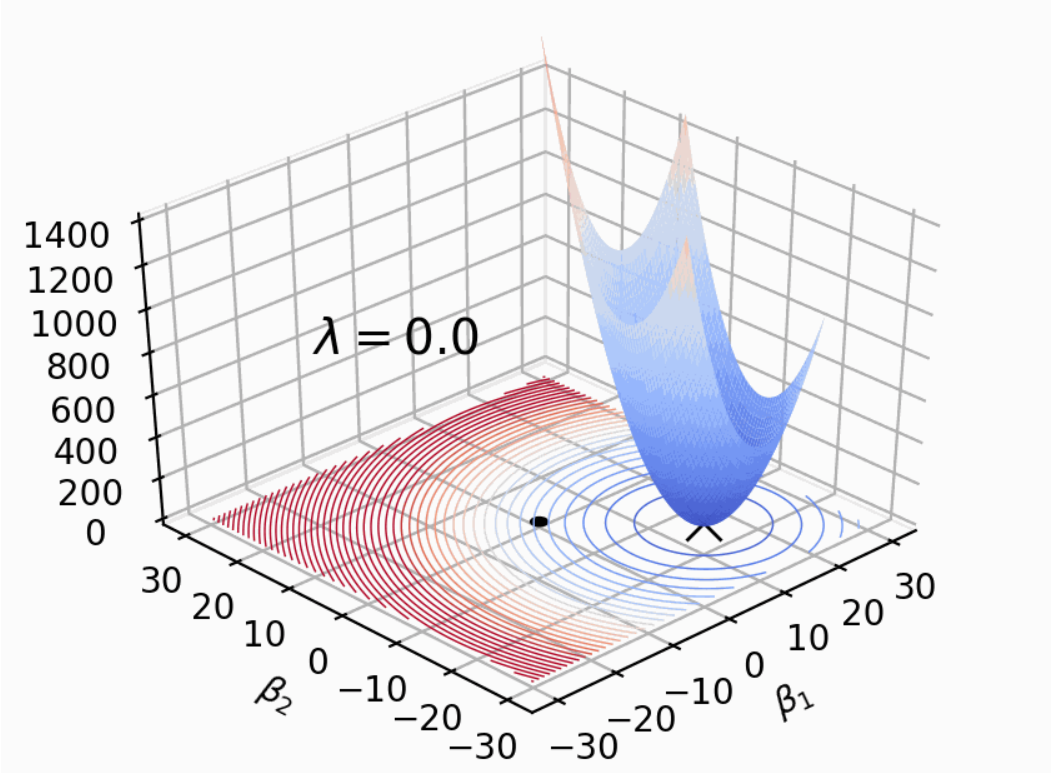


Hiperparámetros

Regulación L1 y L2 (*weight decay*)

El objetivo es evitar que los pesos tomen valores excesivamente elevados.

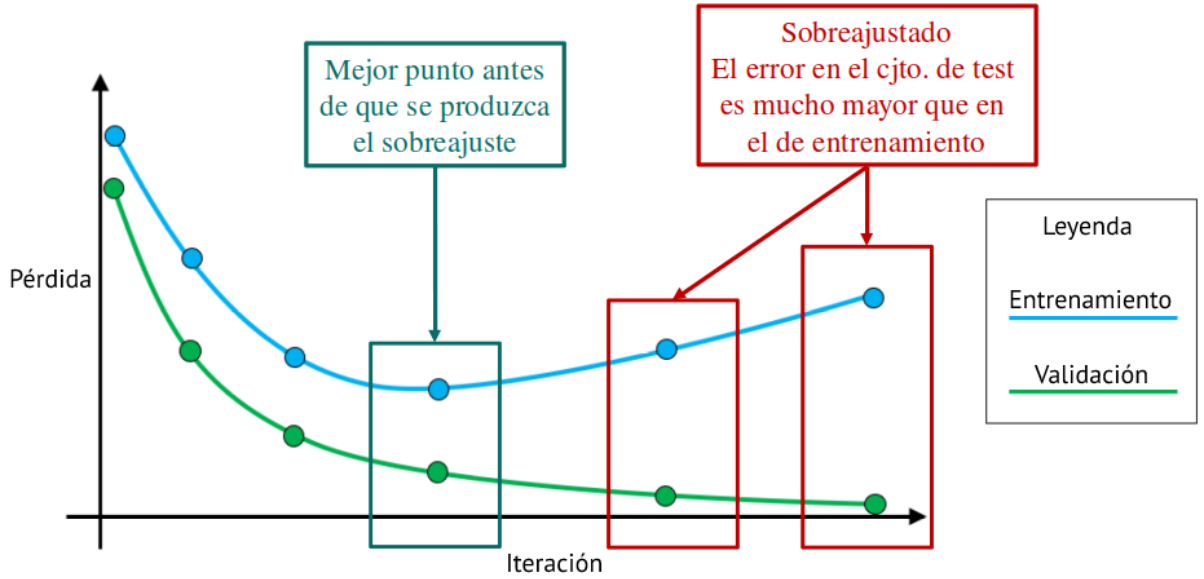
Se evita que unas pocas neuronas dominen el comportamiento de la red y se fuerza a que las características poco informativas (ruido) tengan pesos próximos o iguales a cero.



Hiperparámetros

Early Stopping

Se trata de parar el entrenamiento antes de que se produzca el sobreajuste y seleccionar ese modelo como final. Para ello se utilizan dos conjuntos: uno de entrenamiento y otro de validación. Cuando las curvas de pérdida de ambos conjuntos comienzan a divergir, se para el entrenamiento y se selecciona el modelo resultante del momento anterior al comienzo de la divergencia

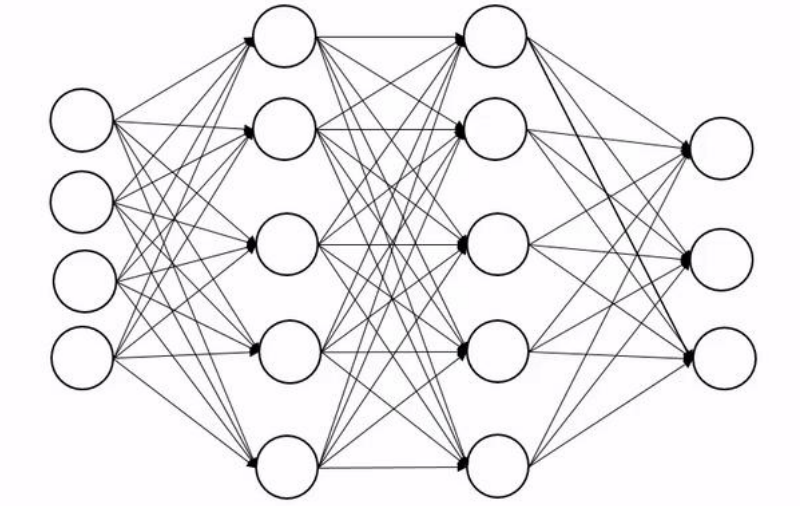


Hiperparámetros

Dropout

Consiste en desactivar aleatoriamente una serie de neuronas durante el proceso de entrenamiento.

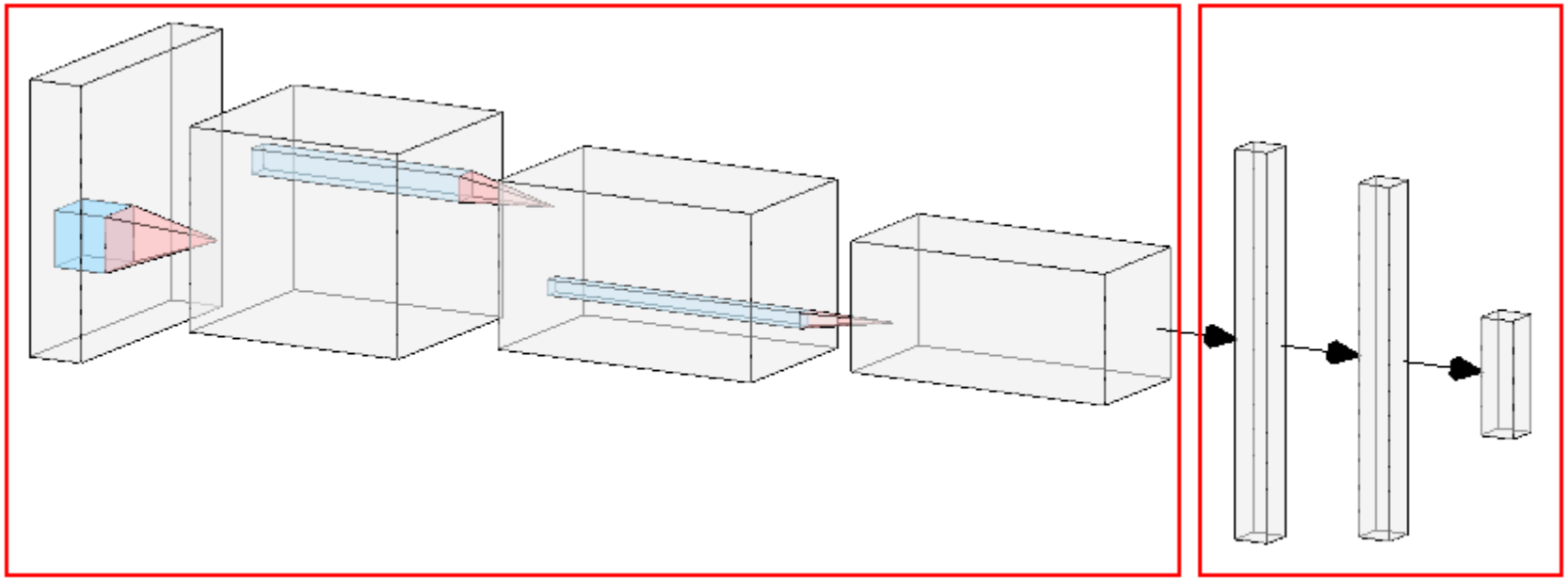
Durante cada iteración del entrenamiento, se ponen a cero los pesos de una fracción aleatoria de neuronas por capa.



El porcentaje de neuronas que suele desactivarse por capa (***dropout rate***) suele ser un valor entre 0.2 y 0.5.

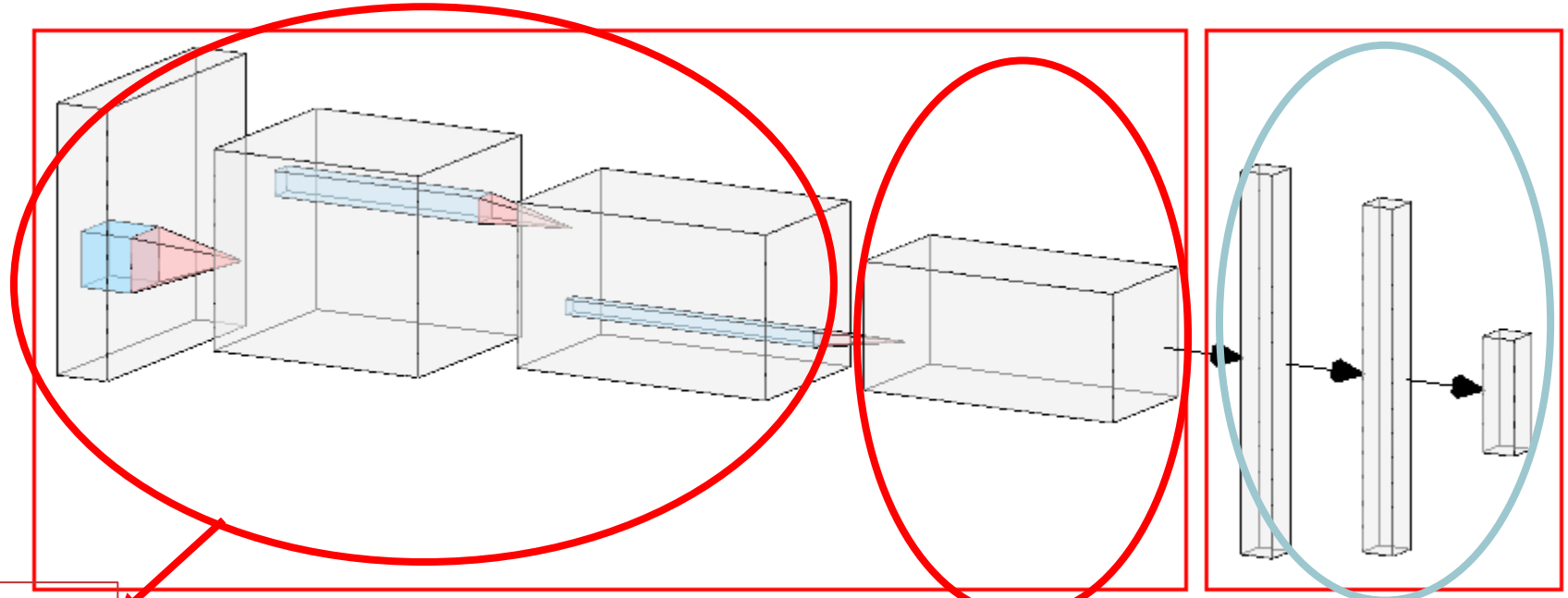
Convolutional Neuronal Networks (CNN)

Las **redes neuronales convolucionales (CNN)** son una extensión de las redes ANN en las que se incluyen capas convolucionales para aprender a extraer, de forma automática, las características de los datos de entrenamiento al inicio de la arquitectura



(LeCun & Bengio, 1995)

Convolutional Neuronal Networks (CNN)



Aprenden a extraer características generales de los datos de entrada

Aprenden a extraer características mucho más específicas

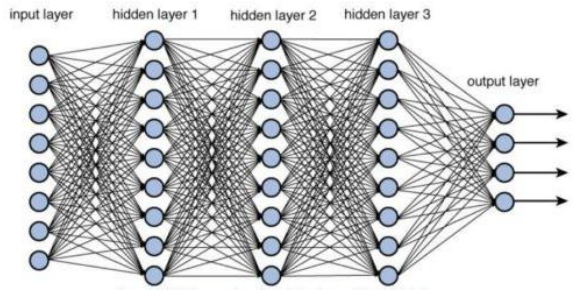
Las capas densas o totalmente conectadas se encargan de realizar la clasificación/regresión

Cuanto más **profunda** (larga) es la red, mayor cantidad de detalles podrá distinguir

Convolutional Neuronal Networks (CNN)

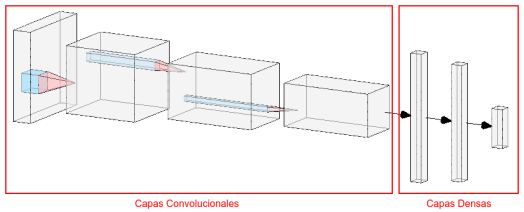
Convolución

ANN



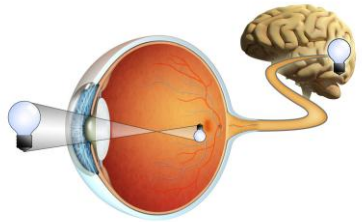
Pueden utilizarse con los valores de color de una imagen como variables para reconocer qué hay en ella, no permiten extraer información de con dependencia espacial.

CNN



A partir de las convoluciones se puede extraer características que contengan dependencia espacial.

Se inspiran en cómo el ser humano percibe y procesa las características de los objetos.

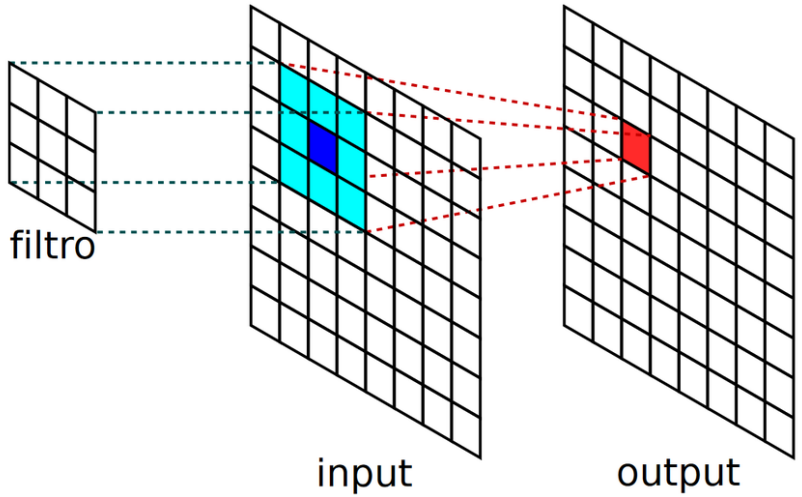


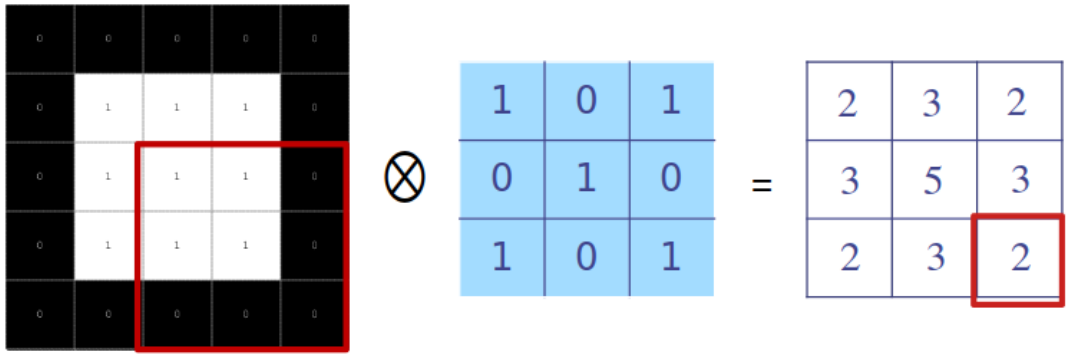
Convolutional Neuronal Networks (CNN)

Convolución

Una convolución aplica un filtro sobre la entrada siguiendo un proceso de ventana deslizante.

El **filtro** (o *kernel*) es una matriz con pesos que se centra en los valores de la entrada para calcular una media ponderada, siendo las ponderaciones los valores del filtro (Bueno et al, 2015)





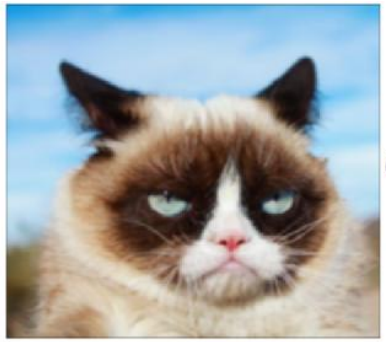
El tamaño de los filtros suele ser impar.

Convolutional Neuronal Networks (CNN)

Convolución

La elección de unos u otros valores del filtro dará lugar a matrices de salida que realcen o suavicen ciertas partes de la entrada.

Original



Gaussiano

$$\frac{1}{16}\cdot\begin{bmatrix}1 & 2 & 1\\ 2 & 4 & 2\\ 1 & 2 & 1\end{bmatrix}$$

Realce bordes

$$\begin{bmatrix}0 & -1 & 0\\ -1 & 5 & -1\\ 0 & -1 & 0\end{bmatrix}$$

Laplaciano

$$\begin{bmatrix}-1 & -1 & -1\\ -1 & 8 & -1\\ -1 & -1 & -1\end{bmatrix}$$

Los pesos de los filtros se ajustaban de forma manual según el problema a resolver. En la actualidad, se ajustan durante el proceso de entrenamiento de la CNN junto con el resto de pesos de la red, lo cual permite encontrar los valores del filtro que maximicen la precisión de la red.

Convolutional Neuronal Networks (CNN)

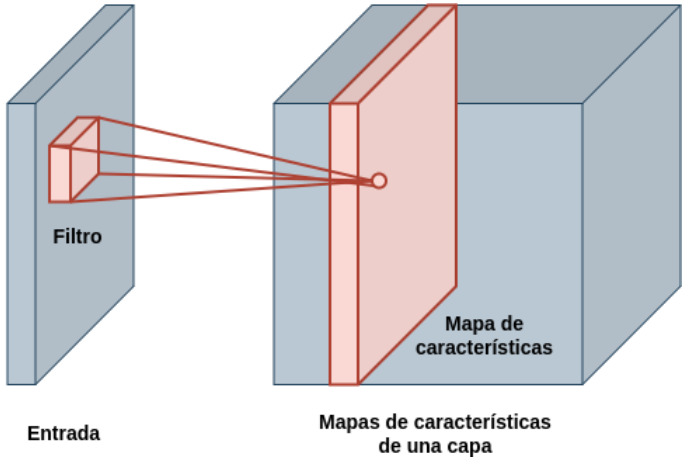
Neuronas Convolucionales

Las capas convolucionales no están compuestas por perceptrones, sino por neuronas convolucionales que aplican los filtros de convolución sobre la salida de la capa anterior.

Los pesos de estas neuronas se organizan en forma de matriz (cada matriz un filtro)

Cada neurona (matriz) pasarán por la función de activación para obtener finalmente lo que se conoce como **mapa de activaciones**.

La representación de este mapa de activaciones se realiza a través de una matriz 3D.



Convolutional Neuronal Networks (CNN)

Neuronas Convolucionales

La mayoría de las CNN utilizan la **ReLU** como función de activación, o alguna variante de esta. Esta función de activación funciona muy bien con el método del descenso del gradiente que se utiliza para actualizar los pesos.

En el caso de que la entrada no sea una matriz 2D sino una matriz 3D como, por ejemplo, una imagen con varios canales de color o un conjunto de mapas de activación, los filtros contarán con una tercera dimensión

Convolutional Neuronal Networks (CNN)

Padding

Se le denomina ***padding*** al incremento de la entrada con un relleno cuando al aplicar el filtro convolucional en la entrada, la matriz resultante es más pequeña. Esto sucede debido a que el filtro no se puede aplicar en los bordes de la matriz.

3	6	8
1	9	4
5	2	7

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

Valor Constante

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

Valores del borde

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

Espejo

Convolutional Neuronal Networks (CNN)

stride

Se denomina ***stride*** al desplazamiento que hace el filtro por la matriz. El desplazamiento básico es de 1. A veces se utiliza un desplazamiento superior a 1 cuando queremos reducir el volumen de datos a tratar.

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				

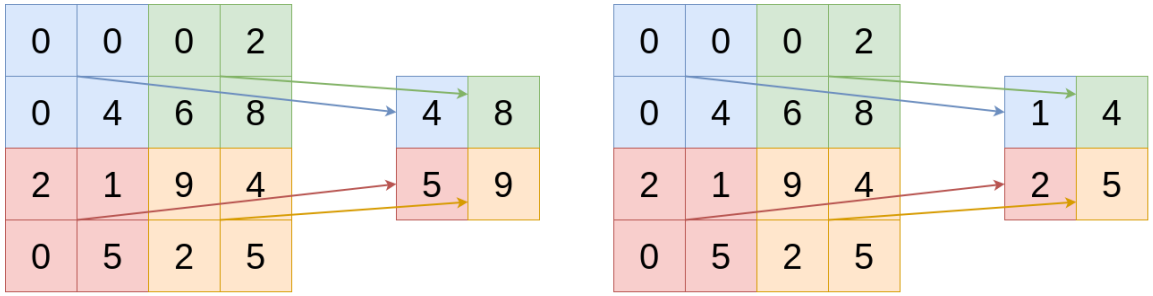
Convolutional Neuronal Networks (CNN)

pooling

La ejecución en secuencia de varias capas convolucionales es muy efectiva a la hora de decidir si ciertas características están o no presentes en la entrada.

Sin embargo, una de sus limitaciones es que mantiene la localización espacial de las características y pequeños movimientos del contenido de la imagen producirían mapas de características diferentes.

El ***pooling*** son capas que agrupan un número de valores adyacentes de los mapas de características obteniendo un nuevo conjunto de mapas más pequeños.

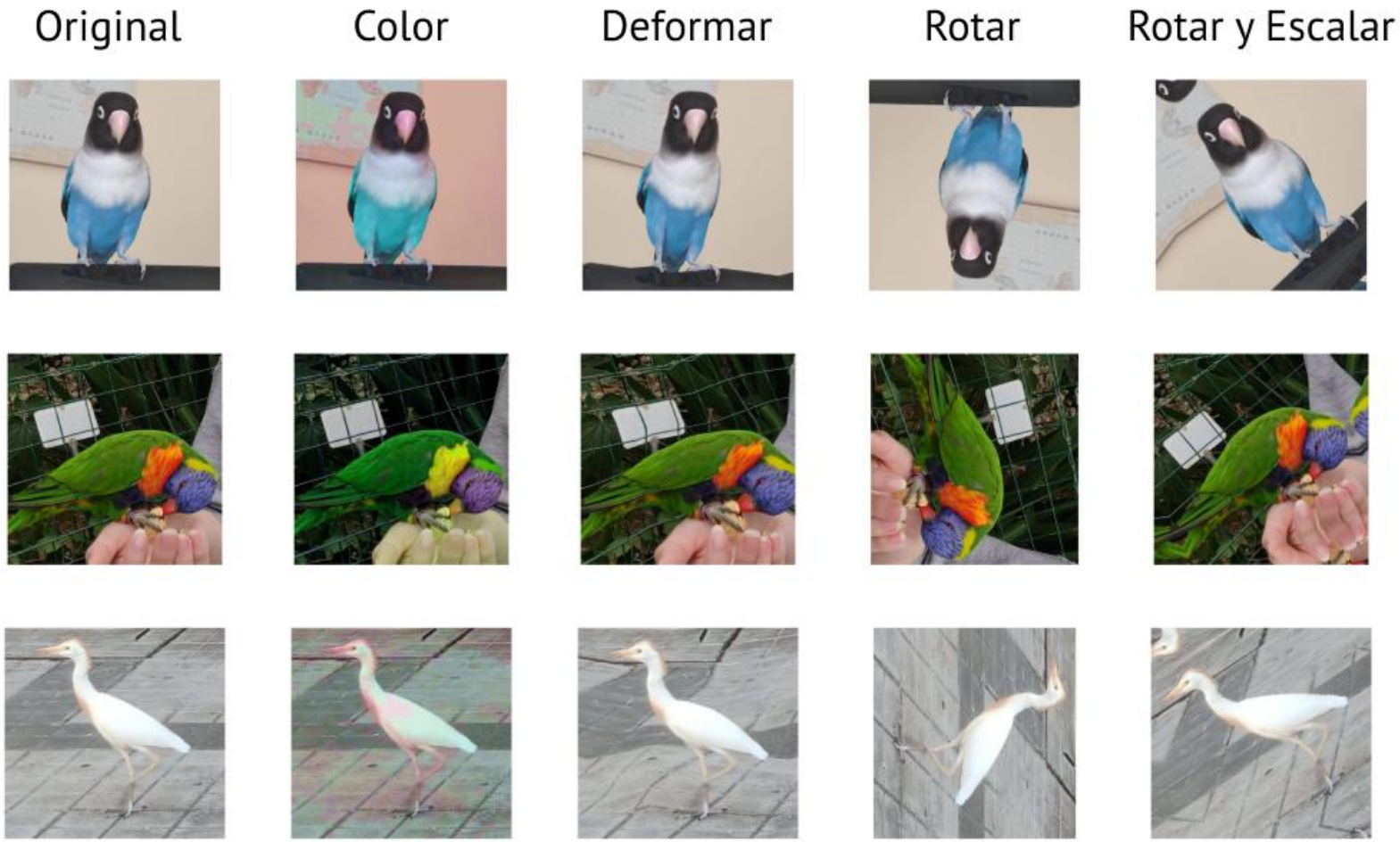


Filtros de 2x2 y *strides* de 2

El tamaño típico es 2x2

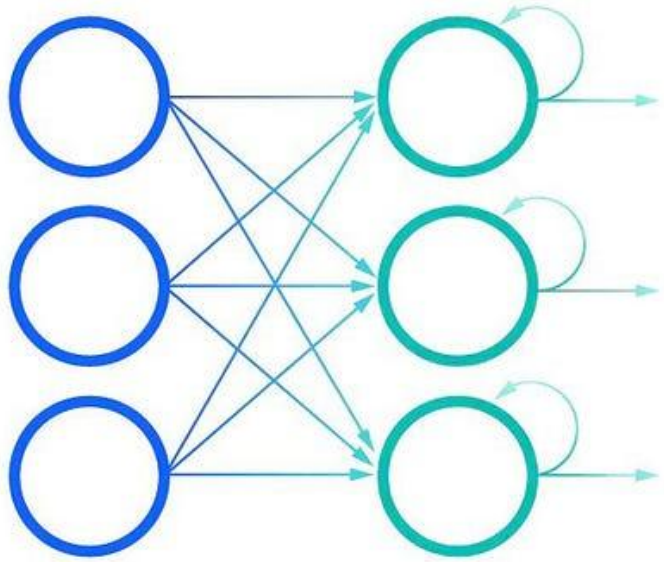
Convolutional Neuronal Networks (CNN)

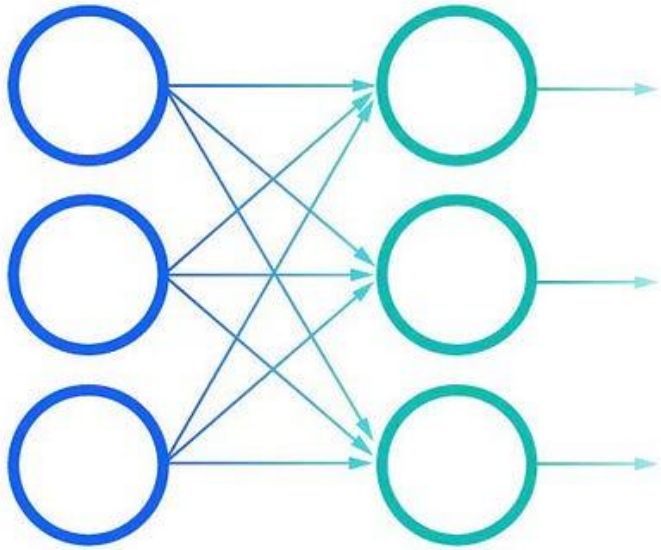
Data augmentation



Recurrent Neuronal Networks (RNN)

Las RNN están diseñadas para gestionar datos secuenciales (con longitudes de entrada y salida variables), tomando la salida de pasos anteriores como entrada para el paso actual. Son esenciales para tareas que involucran datos de series temporales o cualquier dato donde el orden sea importante.



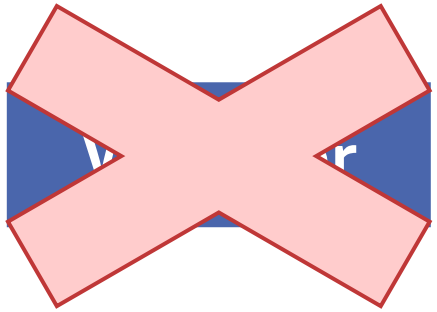


Recurrent Neuronal Networks (RNN)

Cuentan con una "memoria" que captura información sobre lo calculado hasta el momento y permite utilizar entradas previas para influir en la salida actual.

Ejemplo:

Feeling	Under	The	Weather



Recurrent Neuronal Networks (RNN)

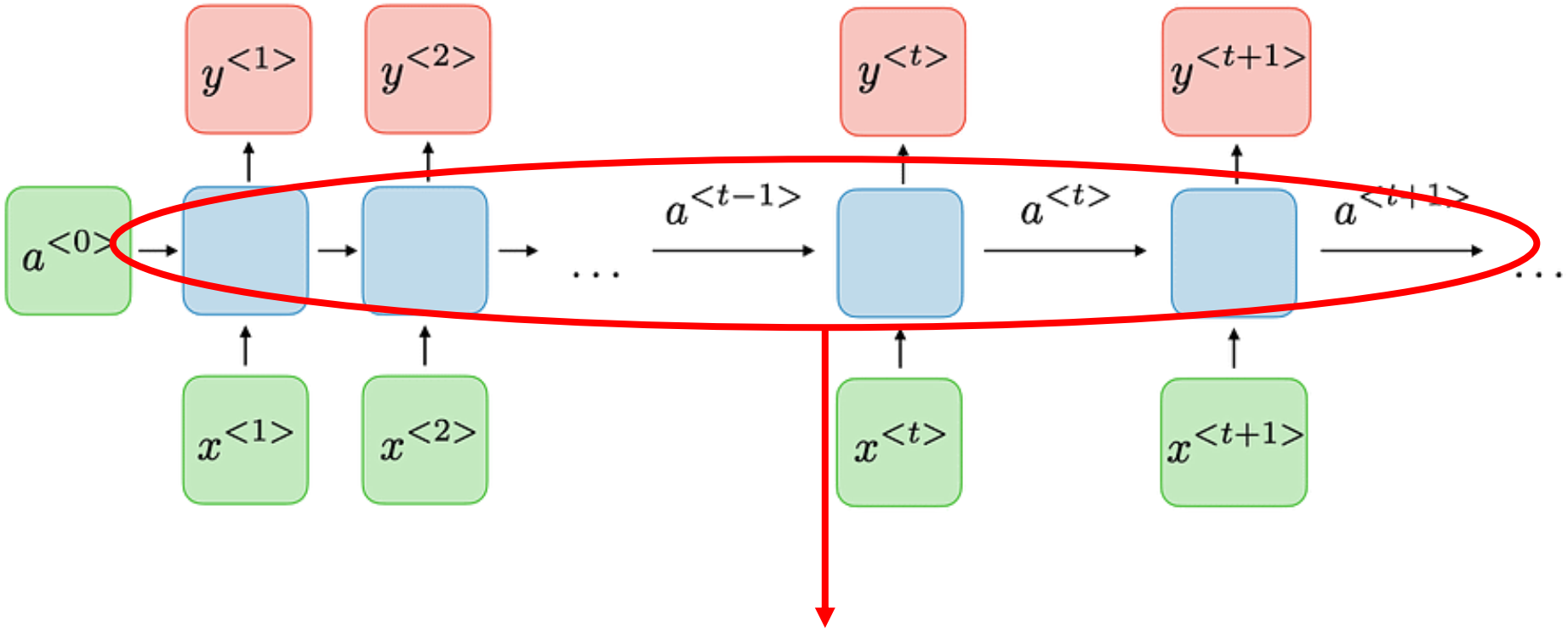
Si bien las RNN son potentes, presentan sus propios desafíos, principalmente los problemas **de gradiente explosivo y evanescente** . Estos problemas se definen por la magnitud del gradiente, que es la pendiente de la función de pérdida a lo largo de la curva de error.

- Si el gradiente es demasiado pequeño, continúa haciéndose más pequeño, actualizando los parámetros de peso hasta que se vuelven insignificantes, es decir, 0. Cuando eso ocurre, el algoritmo deja de aprender.

- Los gradientes explosivos se producen cuando el gradiente es demasiado grande, lo que crea un modelo inestable. En este caso, los pesos del modelo aumentarán demasiado y, finalmente, se representarán como NaN.

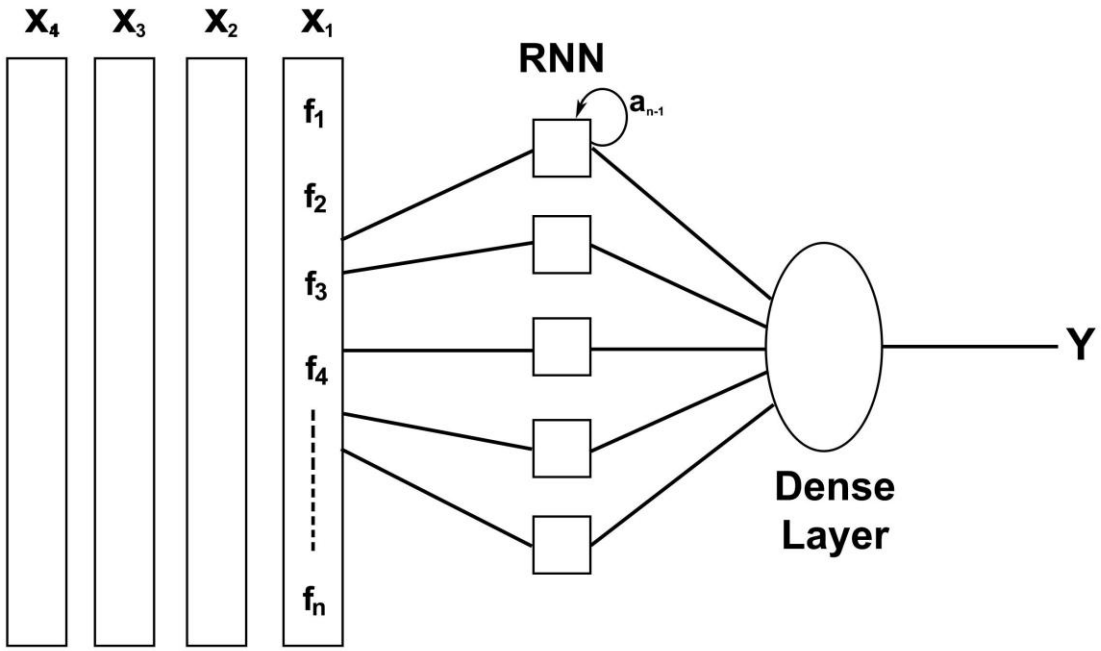
Una solución a estos problemas es reducir la cantidad de capas ocultas dentro de la red neuronal, eliminando parte de la complejidad del modelo RNN.

Recurrent Neuronal Networks (RNN)



Estado oculto que captura información sobre la secuencia

Recurrent Neuronal Networks (RNN)



- **One-to-One**: Red neuronal estándar (clasificación de imágenes)
- **One-to-Many**: Entrada única, salida secuencial (subtítulos de imágenes)
- **Many-to-One:** Entrada secuencial, salida única (análisis de sentimientos).
- **Many-to-Many (misma longitud):** Entrada y salida de secuencia de la misma longitud (reconocimiento de entidad nombrada)
- **Many-to-Many (diferentes longitudes):** Entrada y salida de secuencias de diferentes longitudes (traducción automática)

Recurrent Neuronal Networks (RNN)

Tipos de RNN

Redes neuronales recurrentes bidireccionales (BRNN)

Las BRNN procesan datos tanto en dirección directa como inversa, lo que permite que las redes tengan contexto pasado como futuro.

Útil cuando la salida en el momento *t* depende de entradas futuras

Redes de memoria a largo plazo a corto plazo (LSTM)

Los LSTM están diseñados para manejar el problema del gradiente de desaparición mediante la introducción de una celda de memoria que puede mantener la información durante largos períodos.

Útil cuando se requieren memoria a largo plazo, escribir ensayos o reconocimiento de voz

Unidades Recurrentes Cerradas (GRU)

Los GRU son una versión simplificada de los LSTM con solo dos puertas: puerta de reinicio y puerta de actualización.

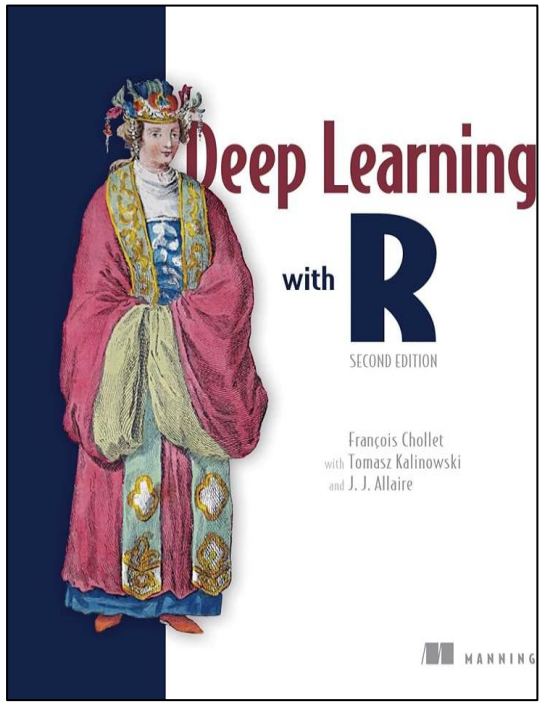
- Menos parámetros que LSTM, lo que los hace más rápidos de entrenar.
- Rendimiento comparable al de los LSTM en muchas tareas

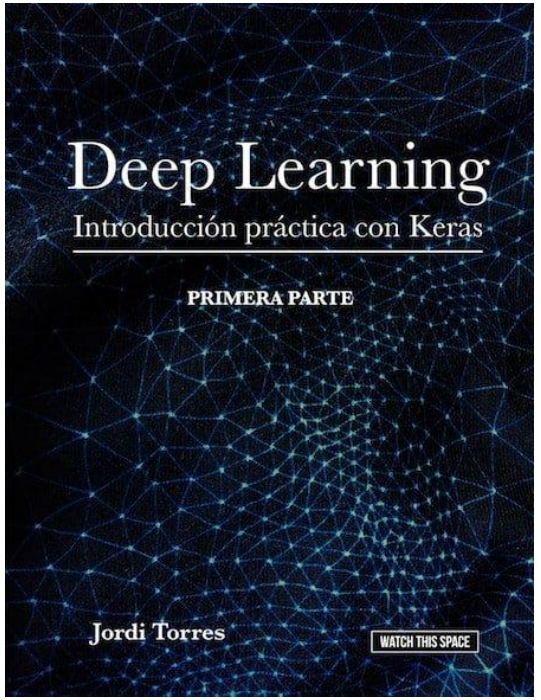
Recurrent Neuronal Networks (RNN)

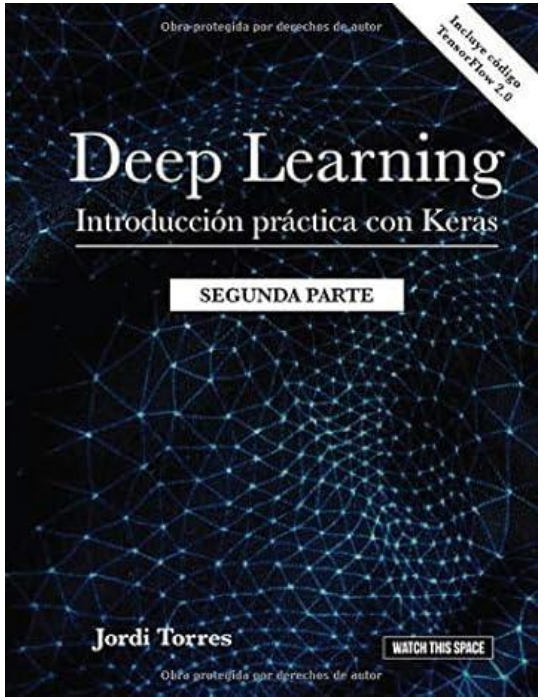
Aplicaciones de RNN

- Procesamiento de lenguaje natural (PLN)
- Reconocimiento de voz
- Traducción automática
- Predicción de series de tiempo
- Generación musical

Bibliografia







https://torres.ai/

Deep Learning An MIT Press book

UNIVERSITAT DE BARCELONA

UNIVERSITAT POLITÈCNICA DE CATALUNYA BARCELONATECH