Módulo 4: Aprendizaje Automático

Temario de la Clase 5
6 de septiembre del 2024
Redes Neuronales Artificiales

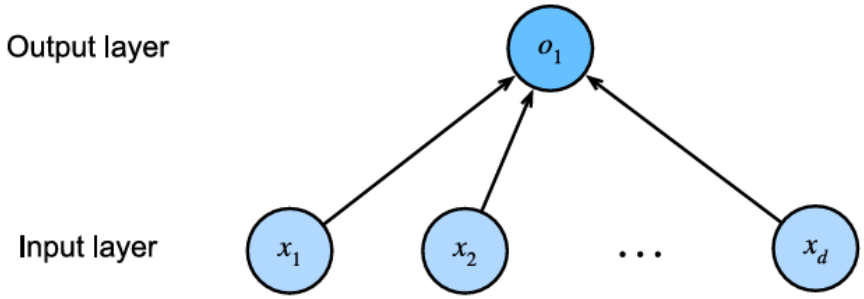
- Bias - Varianza
- Norma de un vector y una matriz
- Regularización
- Normalización de datasets
- Redes neuronales artificiales
- Origen biológico y evolución histórica
- Estructura de una RNA
- Funciones de activación

Repaso de modelos lineales

Un modelo lineal para regresión tenía un determinado número de entradas y una salida.

$$\hat{y} = w_1 x_1 + w_2 x_2 + \cdots + w_d x_d + b$$

$$\hat{y} = \sum_{i=1}^{d} w_i x_i + b = \boldsymbol{w}^T \boldsymbol{x} + b$$



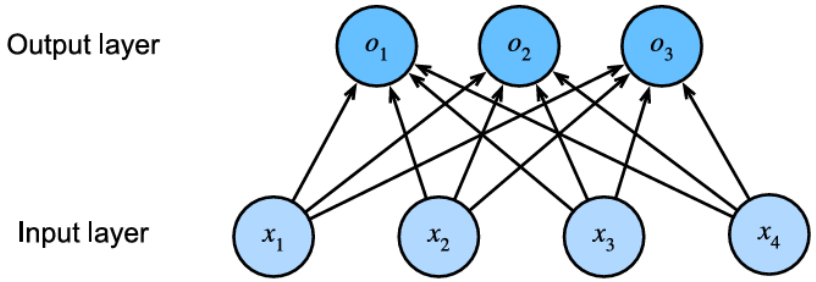
Repaso de modelos lineales

Un modelo lineal para clasificación tenía un mayor número de nodos de salida.

$$o_1 = x_1 w_{11} + x_2 w_{12} + x_3 w_{13} + x_4 w_{14} + b_1,$$

$$o_2 = x_1 w_{21} + x_2 w_{22} + x_3 w_{23} + x_4 w_{24} + b_2,$$

$$o_3 = x_1 w_{31} + x_2 w_{32} + x_3 w_{33} + x_4 w_{34} + b_3.$$



Repaso de modelos lineales

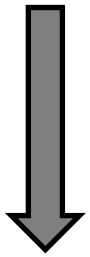
	Regresión lineal	Clasificación binaria	Clasificación multi-clase
Función de error	Error cuadrático medio	Entropía cruzada binaria	Entropía cruzada categórica
Función de activación	--	Sigmoide	Softmax
Número de salidas del modelo	1	1	K (número de clases)
Modelo	Lineal	Lineal + función de activación	K factores lineales + función de activación

Bias - Varianza

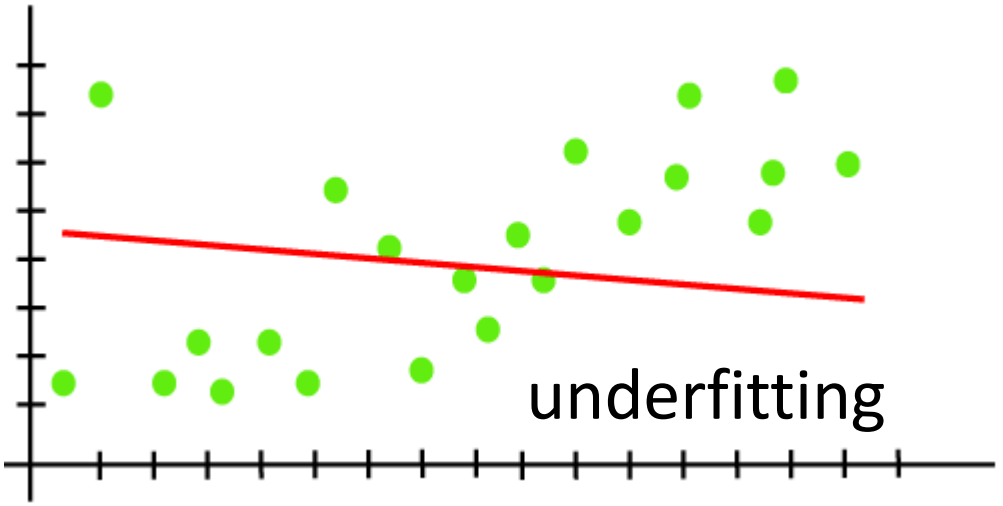
Bias

Es la cantidad de supuestos que tiene el modelo.

Más supuestos



Mayor bias

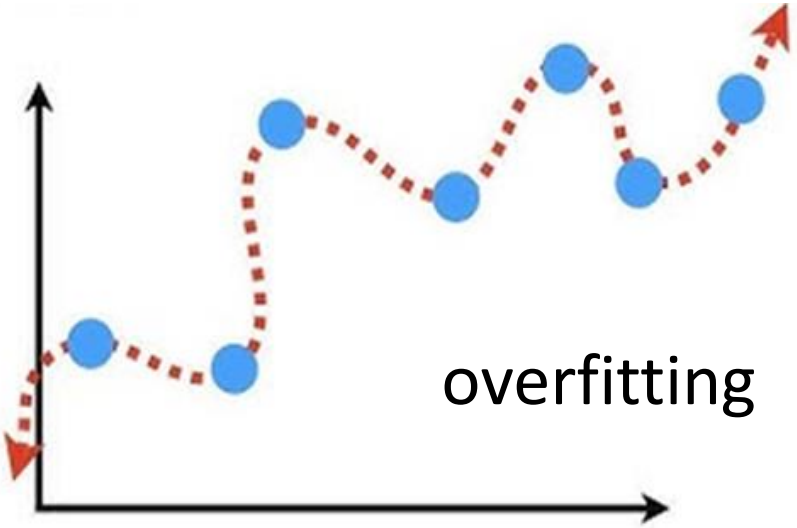


Regresión lineal

Varianza

Es la *sensibilidad* del modelo a loa datos de entrada.

Mayor sensibilidad

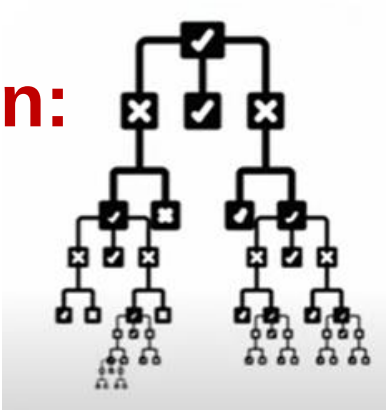


overfitting

Mayor varianza

Árboles de decisión:

alta flexibilidad



Bias – Variance Tradeoff

(Compensación bias – varianza)

Bias - Varianza

Problemas	**Alta Bias** (Performance de entrenamiento es bajo)	**Alta Varianza** (Performance de validación es bajo)
Causas	Underfitting – sub-entrenamiento	Overfitting Sobre entrenamiento
Soluciones	a) Entrenar más b) Aumentar la complejidad del modelo. c) Probar una arquitectura del modelo diferente	a) Aumentar los datos b) Usar regularización. c) Probar una arquitectura del modelo diferente

Bagging y boosting

Tanto el *bagging* como el *boosting* son procedimientos generales para la reducción de la varianza de un método estadístico de aprendizaje.

La idea básica consiste en combinar métodos de predicción sencillos (débiles), es decir, con poca capacidad predictiva, para obtener un método de predicción muy potente (y robusto). Estas ideas se pueden aplicar tanto a problemas de regresión como de clasificación.

Son muy empleados con árboles de decisión: son predictores débiles y se generan de forma rápida. Lo que se hace es construir muchos modelos (crecer muchos árboles) que luego se combinan para producir predicciones (promediando o por consenso).

Algoritmos de bagging

Combinación de algoritmos simples.

Ej: Random Forest que combina árboles de decisión para formar el Random Forest.

«El principal objetivo intrínseco de los algoritmos de bagging es el de la reducción de la varianza»

Los métodos de bagging son métodos donde los algoritmos simples son usados en paralelo.

El principal objetivo de los métodos en paralelo es el de aprovecharse de la independencia que hay entre los algoritmos simples.

Algoritmos de boosting

Algunos ejemplos de estos algoritmos son el XGBoost o el AdaBoost.

«El principal objetivo de los algoritmos de boosting es el de la reducción del sesgo»

En los algoritmos de boosting, los modelos simples son utilizados secuencialmente, es decir, cada modelo simple va delante o detrás de otro modelo simple.

El principal objetivo de los métodos secuenciales es el de aprovecharse de la dependencia entre los modelos simples.

El rendimiento general puede ser mejorado haciendo que un modelo simple posterior le de más importancia a los errores cometidos por un modelo simple previo.

Norma de un vector

La norma de un vector nos da una medida de que tan grande es. La distancia euclídena que conocemos también se denomina norma ℓ_2 .
En general la norma ℓ_p se calcula como:

$$\|x\|_p = \left(\sum_{i=1}^{n} |x_i|^p \right)^{1/p}$$

Donde las normas ℓ_1 (p=1) y ℓ_2 (p=2) son las más conocidas.

$$\|x\|_1 = \sum_{i=1}^{n} |x_i|$$

$$\|x\|_2 = \sqrt{\sum_{i=1}^{n} |x_i|^2}$$

Norma de una matriz

Podemos definir la norma Frobenius que se comporta como una norma ℓ_2 sobre los elementos de la matriz.

$$\|X\|_F = \sqrt{\sum_{i=1}^{m}\sum_{j=1}^{n} x_{ij}^2}$$

Regularización

En general, una forma de reducir el overfitting es utilizando datasets mas numerosos y con mayor variedad de datos. Cuando esto no es posible, es necesario recurrir a otras técnicas. La **regularización** actúa sobre los parámetros del modelo restringiendo los valores que pueden tomar y con ello, la complejidad del mismo.

En particular, si tomamos la norma del vector de pesos, se puede aplicar un término de penalización para que se tenga en cuenta la función de pérdida en conjunto con dicha norma, de modo que la nueva función a optimizar minimice ambos términos simultaneamente.

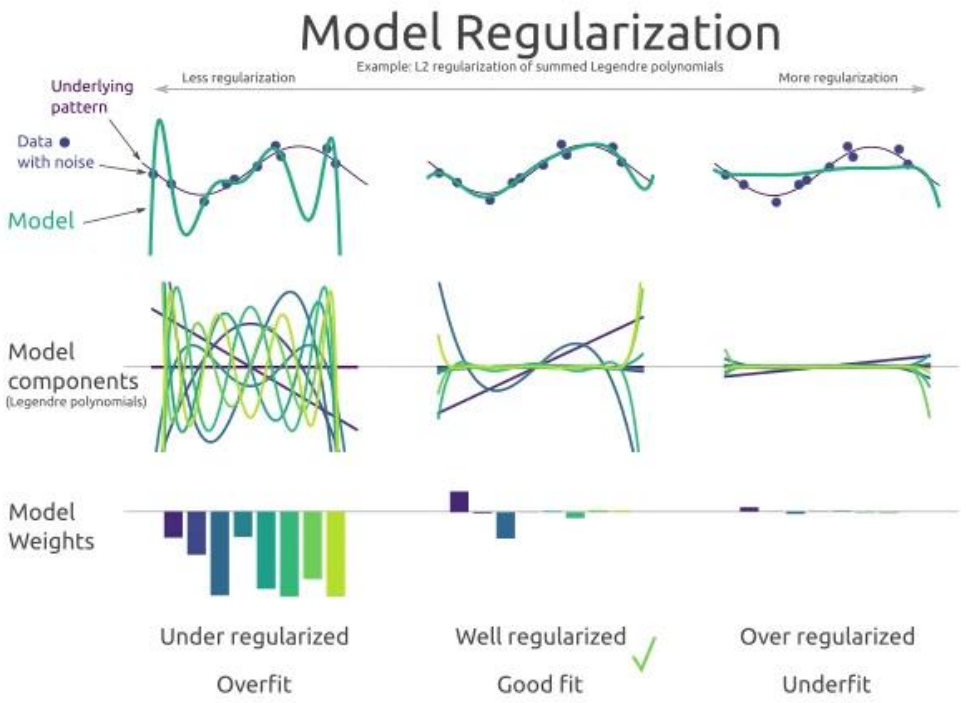
Puesto que ahora la minimización no se aplica solamente sobre el error de entrenamiento, se reduce el overfitting consiguiéndose parámetros de un modelo **menos complejo** que generalice mejor.

Regularización

La idea radica en que modelos más complejos son más fáciles de sobre-entrenar por lo que se busca encontrar soluciones *más simples* donde se aumente el sesgo y se disminuya la varianza.

Los dos tipos de regularización más usados son:

- Regularización Ridge / ℓ_2 / weight decay / regularización cresta
- Regularización Lasso / ℓ_1



Regularización L2 / Ridge / Weight Decay

Se añade un término de penalización proporcional a la suma de los cuadrados de los pesos. Nuestra función de pérdida original para modelos lineales era:

$$L(w,b) = \frac{1}{n}\sum_{i=1}^{n} l^{(i)}(w,b) = \frac{1}{2n}\sum_{i=1}^{n} (\boldsymbol{w}^T \boldsymbol{x}^{(i)} + b - \hat{y}^{(i)})^2$$

Y ahora la pérdida con regularización es:

$$J = L + \lambda \|w\|^2 = \mathrm{L} + \lambda \sum_{i=1}^{n} |w_i|^2$$

Donde λ es un hiperparámetro de regularización que controla la severidad de la penalización.

Regularización L1 / Lasso

Similar al weight decay, con la diferencia que se toma solo el valor absoluto de los pesos y no el valor al cuadrado. La función a optimizar es:

$$J = L + \lambda \|w\| = \mathrm{L} + \lambda \sum_{i=1}^{n} |w_i|$$

El valor de λ puede encontrarse por validación cruzada. λ=0 resulta en un modelo sin límites para los pesos, con lo que valores mayores tenderán a restringir en mayor medida la complejidad del mismo.

Así como antes, la actualización de los parámetros se encuentra tomando el gradiente de la función objetivo. Dependiendo de la implementación, los términos de bias no suelen regularizarse.

*Lasso = Least absolute shrinkage and selection operator

Comparación entre L1 y L2

- La regularización ℓ_2 o Ridge puede disminuir el valor de los pesos (pero no hacerlos 0) por eso el nombre *weight decay*. Puesto que penaliza las componentes mas grandes del vector de pesos, tiende a crear modelos con pesos más distribuidos entre las features, lo que lo hace más robusto ante datos con ruido (se espera que una perturbación leve en un dato no debería cambiar significativamente la salida).

- La regularización ℓ_1 o Lasso tiende a concentrar los pesos en un número reducido de features al ser capaz de reducir los weights a 0. Esto permite realizar selección de características para encontrar las features más significativas para el modelo, eliminando aquellas cuyos pesos sean casi nulos.

Normalización del dataset

Problema: los features contenidos en el dataset contienen rangos de valores muy distintos entre sí. Por ejemplo: edad (18 – 80 años), depósitos en caja de ahorro ($ 500.000 - $ 2.000.000), antigüedad de la cuenta (300 – 1000 días). Durante el entrenamiento, los features con valores más grandes tendrán mayor influencia en la forma que el algoritmo entrena el modelo.

Solución: realizar transformaciones matemáticas que eliminen dichas diferencias entre features. Existen varias soluciones propuestas pero las más populares son la **estandarización Z-score** y la **normalización min-max**.

Normalización del dataset

La normalización, también llamado **feature scaling**, es parte del pre-procesamiento de los datos y es importante ya que hace que la convergencia de algoritmos basados en descenso del gradiente o en la minimización de alguna función objetivo sea mas rápida y mas eficiente.

La normalización Standard score es ampliamente aplicada en el entrenamiento de redes neuronales artificiales, regresión logística, máquinas de soporte vectorial y en algoritmos de aprendizaje no supervisado como k-means clustering.

Muchos de estos algoritmos de ML hacen cálculos basados en distancias euclídenas entre puntos. También es importante cuando hay outliers en los datos y cuando se introducen términos de penalización mediante **regularización**.

Standard score o Z-score

Asumiendo que los valores de un dataset tienen una distribución Gaussiana con media μ y desviación standard σ, se transforman los datos para que tengan una media igual a cero y varianza unitaria.

Entonces para cada dato, la transformación se realiza de la siguiente forma:

$$x_i' = \frac{x_i - \mu}{\sigma}$$

Esta transformación sobre los datos tiene un efecto positivo sobre las etapas posteriores que realiza el modelo, ya que mejora la inicialización de parámetros, previene problemas con funciones de activación no lineales, permite que la actualización de los pesos se haga de forma mas consistente (reduciendo el tiempo de entrenamiento) y evita los sesgos hacia determinadas features.

Reescalamiento: Normalización min-max

Se transforma un rango de valores determinado en un rango [0,1] .

Entonces para cada dato:

$$x_i' = \frac{x_i - x_{min}}{x_{max} - x_{min}}$$

Se pueden tomar variaciones para que los datos obtenidos sean en el rango [-1,1] o un rango cualquiera [a,b].

$$x_i' = a + \frac{(x_i - x_{min})\,(b - a)}{x_{max} - x_{min}}$$

Normalización de vector unitario

Cada ejemplo se considera como un vector, donde se divide cada componente del vector por su norma. Se puede usar cualquier norma, pero las más comunes son las normas L1 y L2.

Si tenemos:

$$x = (v_1, v_2, v_3)$$

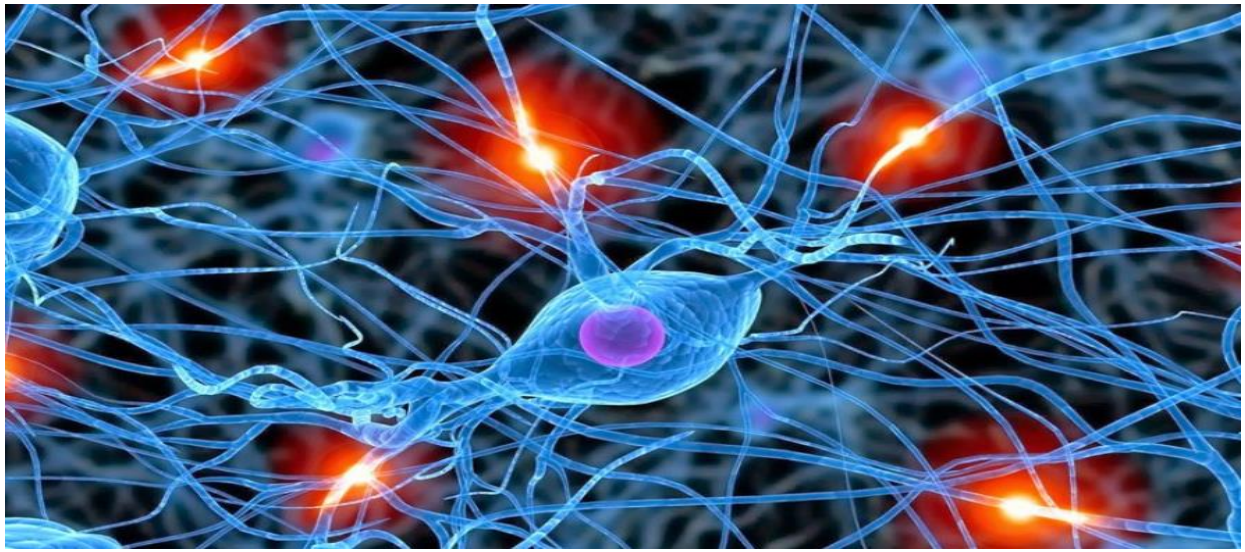
Entonces la versión normalizada es:

$$x' = \frac{x}{\|x\|} = \left(\frac{v_1}{(|v_1|^p + |v_2|^p + |v_3|^p)^{1/p}}, \frac{v_2}{(|v_1|^p + |v_2|^p + |v_3|^p)^{1/p}}, \frac{v_3}{(|v_1|^p + |v_2|^p + |v_3|^p)^{1/p}}\right)$$

Redes Neuronales Artificiales

Las redes neuronales, en su esencia, son modelos artificiales y simplificados del cerebro humano, que es el ejemplo más perfecto de un sistema que adquiere conocimiento a través de la experiencia.

Las redes neuronales imitan la función básica de las neuronas biológicas en la comunicación y el procesamiento de información. Son una nueva forma de procesar información y datos.



Redes Neuronales Artificiales

¿Cómo afrontar problemas que no se pueden desglosar en algoritmos, como la clasificación de objetos por características comunes?

Los humanos son capaces de resolver situaciones complejas basándose en su experiencia acumulada. La inteligencia artificial (IA) surgió como un intento de comprender y emular aspectos de la inteligencia humana en las máquinas. Las redes neuronales, en particular, se inspiraron en la capacidad humana de memorizar y asociar hechos.

RNA: cronología histórica

Desde las teorías iniciales de Warren McCulloch y Walter Pitts en 1943 hasta los avances en aprendizaje y arquitecturas de redes, la evolución de las redes neuronales ha sido impresionante.

Alan Turing, el Congreso de Dartmouth, Donald Hebb, Frank Rosenblatt, Karl Lashley y muchos otros influyeron en el desarrollo de la teoría de redes neuronales.

Cada uno de estos hitos contribuyó al resurgimiento de las redes neuronales en la década de 1980, y a partir de entonces, se han convertido en una herramienta vital en campos como el procesamiento de imágenes, el aprendizaje automático y la inteligencia artificial.

RNA: cronología histórica

(1912-1954) Alan M. Turing
Uno de los pioneros de la Inteligencia Artificial. Creó un test para determinar si una máquina puede exhibir inteligencia similar a un humano. Escribió "Computing Machinery and Intelligence" en 1950.

CAPTCHA
Completely Automated Public Turing test to tell Computers and Humans Apart

RNA: cronología histórica

1943

McCulloch (neurobiólogo) y Pitts (estadístico) publican una teoría sobre la forma de trabajar de las neuronas biológicas Representan una neurona biológica mediante un modelo computacional

1949

Hebb relaciona psicología con fisiología. La conexión entre dos neuronas, se refuerza si ambas se activan repetidamente

1950

Alan M. Turing
Test para máquinas

1958

Rosenblatt: Desarrollo del Perceptrón, la implementación en hardware de la neurona

1959

Widrow y Hoff Introducen el algoritmo LMS y lo utilizan para entrenar el **Adaline**

1962

Rosenblatt:
Teorema de convergencia del perceptrón

1969

Minsky y Papert
Limitaciones matemáticsa del Perceptrón: problema del XOR

70's

Época negra de las redes neuronales

1982

Hopfield
Red más plausible con el modelo biológico

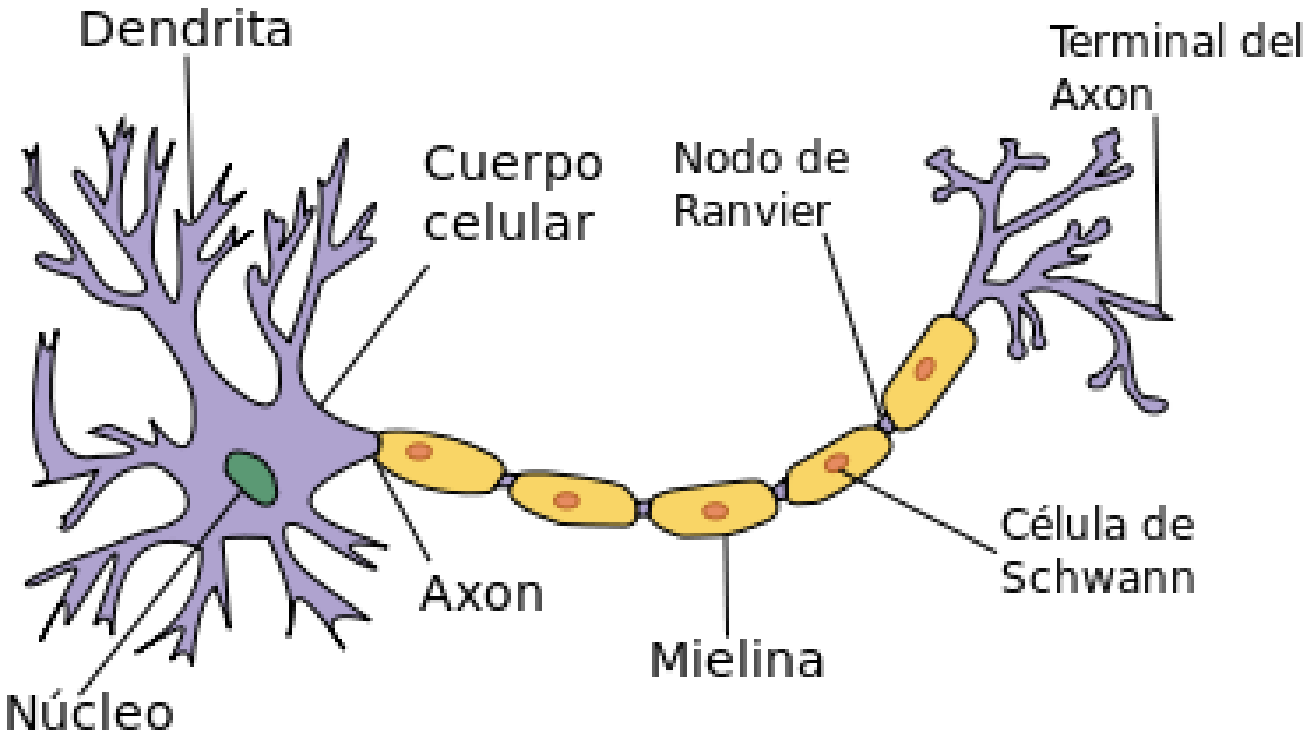
1986

Rumelhart y McClelland
Algoritmo **backpropagation** para entrenar el Perceptrón multicapa

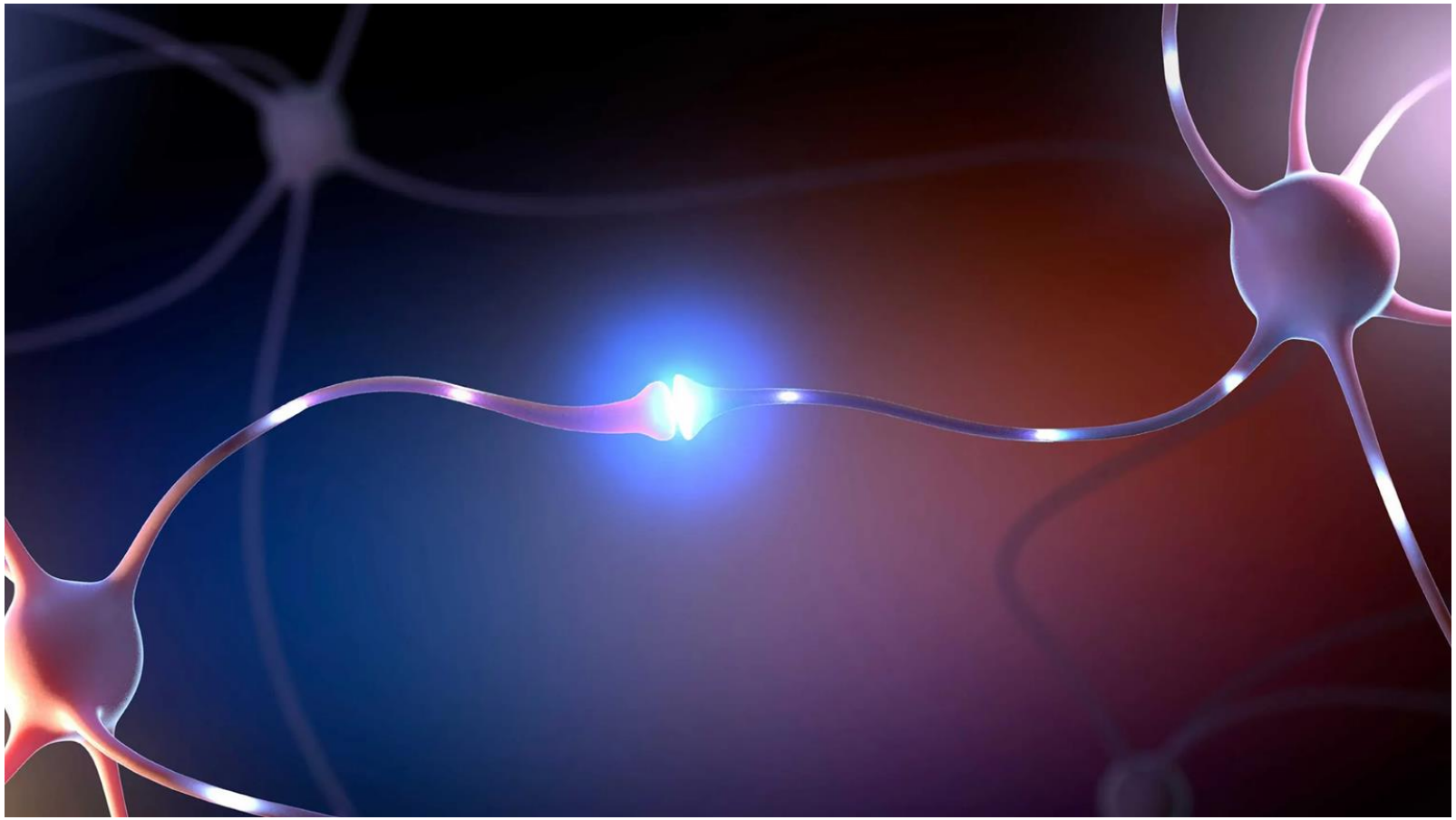
Siglo XXI

Grandes DB, computadoras +rápidas y ágiles y técnicas de ML +avanzadas: IA.
Deep Learning; Big Data; Inteligencia artificial generativa (ChatGPT)

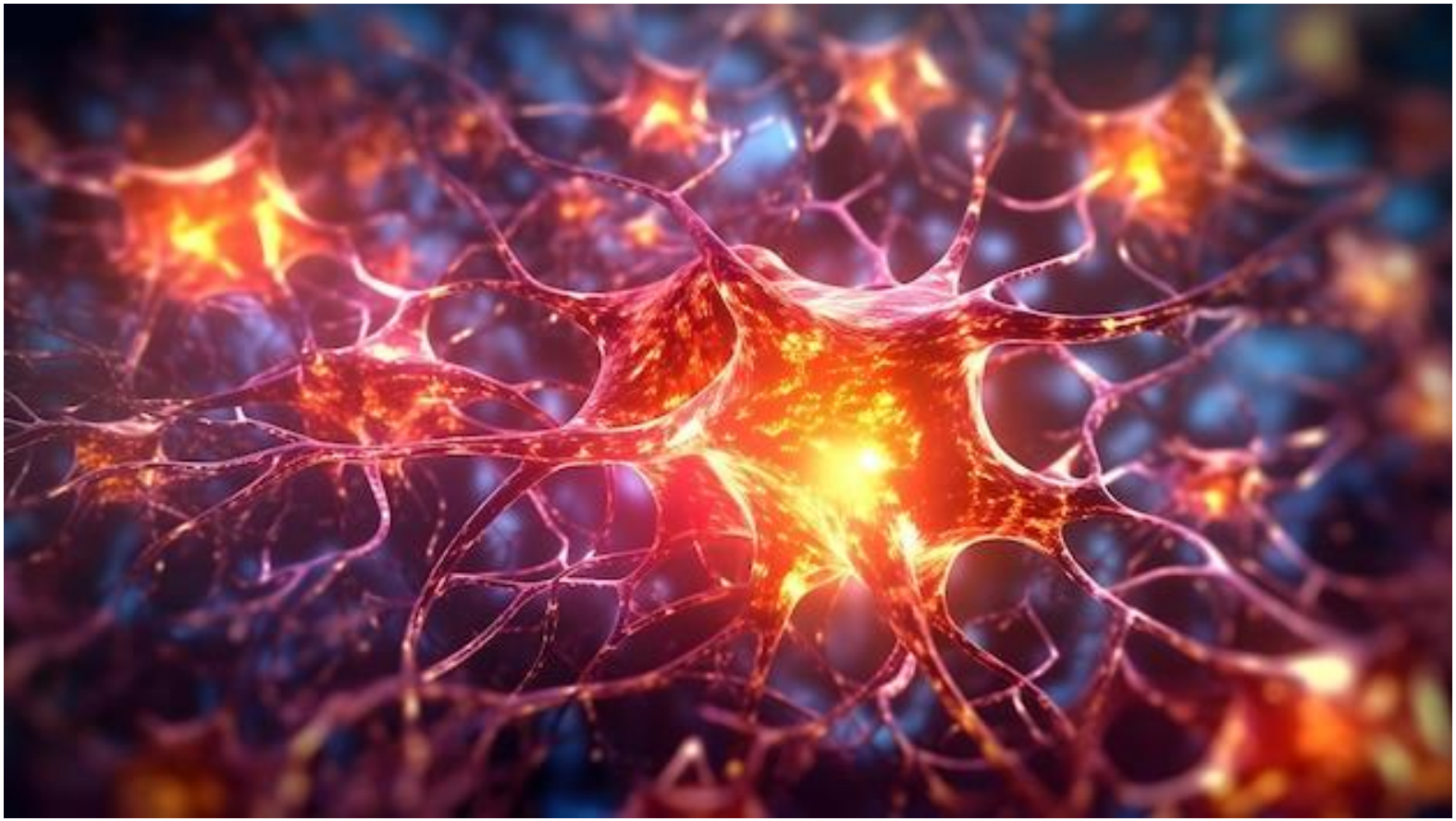
RNA: origen biológico



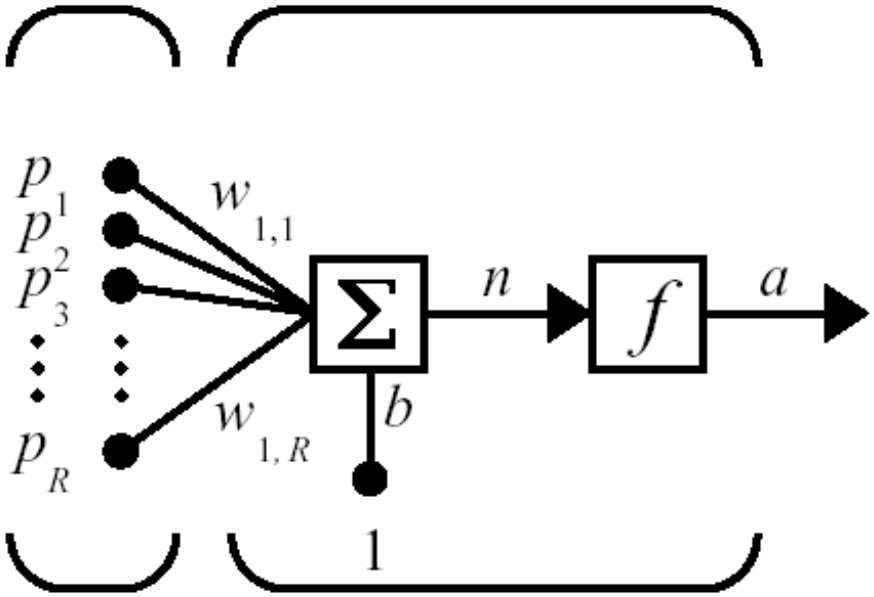
RNA: origen biológico - sinápsis



Red neuronal biológica



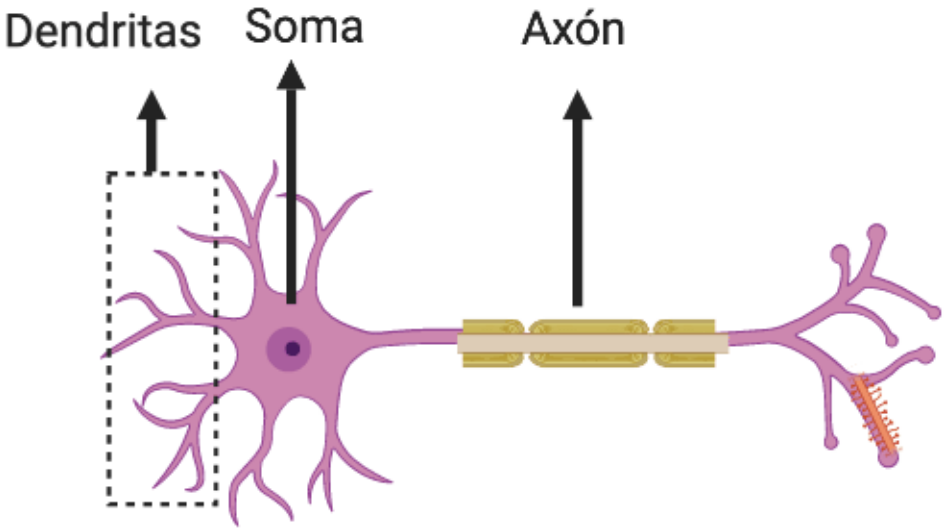
Estructura de una RNA

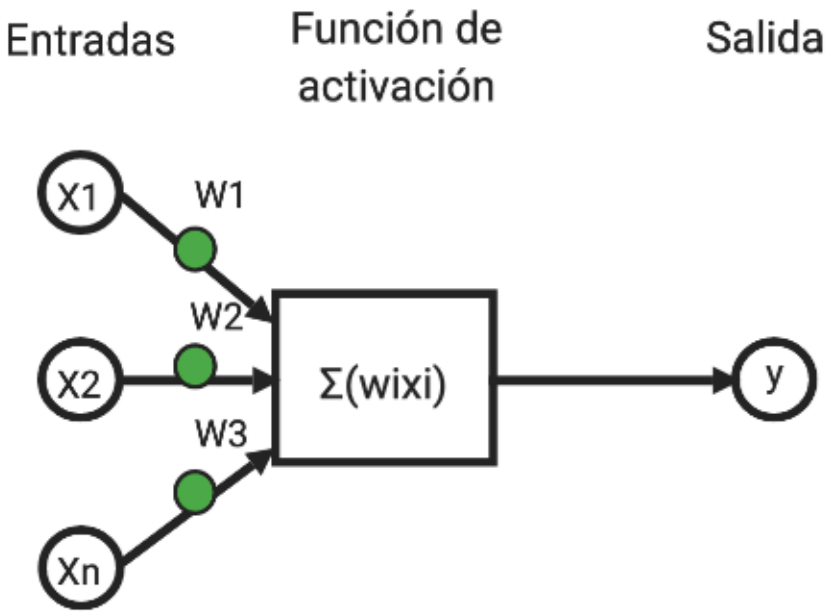


$$a = f(\mathbf{W}\mathbf{p} + b)$$

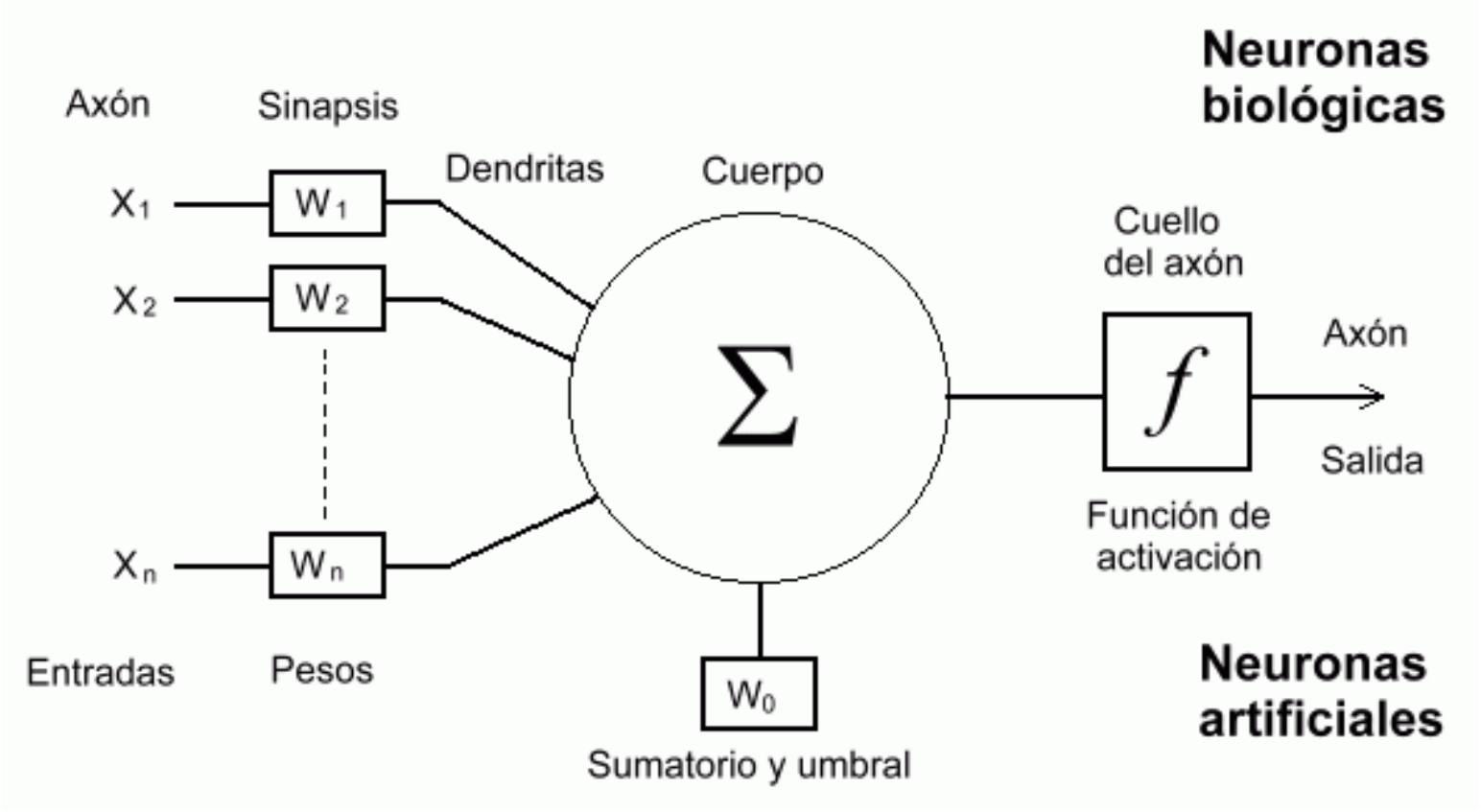
$$a(k) = f(n) = f\left| \sum w_{k \cdot j} x_j + b_k \right|$$

Estructura de una RNA





Estructura de una RNA

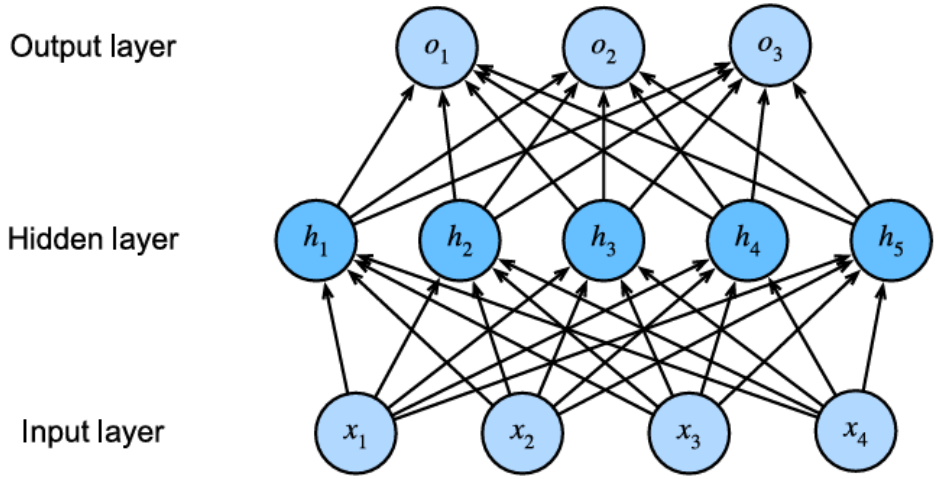


Estructura de una RNA

LAS NEURONAS	
NEURONA BIOLÓGICA	NEURONA ARTIFICIAL
Soma Dendrita Axón Sinapsis	Neurona Entrada Salida Peso

Introducción a las redes neuronales artificiales

Una red neuronal sencilla es el perceptrón multi capa. Además de los nodos de entrada y salida tenemos una o varias capas “ocultas” de neuronas que dan más complejidad al modelo. Estas redes se denominan “fully connected” porque todas las salidas de las neuronas de una capa se conectan con todas las entradas de las neuronas de la capa siguiente.



Introducción a las redes neuronales artificiales

Sin embargo, es interesante notar que añadir mas capas de este tipo, no le añade mayor dimensión matemática al modelo. Puesto que la capa oculta puede entenderse como una transformación lineal afín de las entradas, y la capa de salida (antes de alguna activación final) es una transformación afín de la capa oculta. Se puede demostrar que una transformación afín de una transformación afín es una transformación del mismo tipo, que es lo que podíamos hacer antes de añadir la capa oculta.

$$\mathbf{W} = \mathbf{W}^{(1)}\mathbf{W}^{(2)} \qquad \mathbf{b} = \mathbf{b}^{(1)}\mathbf{W}^{(2)} + \mathbf{b}^{(2)}$$

$$\mathbf{O} = (\mathbf{X}\mathbf{W}^{(1)} + \mathbf{b}^{(1)})\mathbf{W}^{(2)} + \mathbf{b}^{(2)} = \mathbf{X}\mathbf{W}^{(1)}\mathbf{W}^{(2)} + \mathbf{b}^{(1)}\mathbf{W}^{(2)} + \mathbf{b}^{(2)} = \mathbf{X}\mathbf{W} + \mathbf{b}.$$

Rompiendo la linealidad

Para obtener un modelo que no pueda reducirse a un modelo lineal es necesario introducir un elemento que rompa dicha linealidad y así aprovechar la arquitectura de las redes multi-capa. Para ello se utiliza una función no-lineal de activación que se aplique luego de la transformación lineal.

$$\mathbf{H} = \sigma(\mathbf{X}\mathbf{W}^{(1)} + \mathbf{b}^{(1)}),$$
$$\mathbf{O} = \mathbf{H}\mathbf{W}^{(2)} + \mathbf{b}^{(2)}.$$

Esto nos permite generar modelos de mayor complejidad con lo que podríamos aproximar un mayor número de tipos de funciones. La salida de cada capa se calcula en base a las salidas de las anteriores.

$$\mathbf{H}^{(1)} = \sigma_1(\mathbf{X}\mathbf{W}^{(1)} + \mathbf{b}^{(1)}) \qquad \mathbf{H}^{(2)} = \sigma_2(\mathbf{H}^{(1)}\mathbf{W}^{(2)} + \mathbf{b}^{(2)}),$$

Función de activación

Es una transformación matemática que se aplica sobre el valor obtenido a la entrada de la neurona.

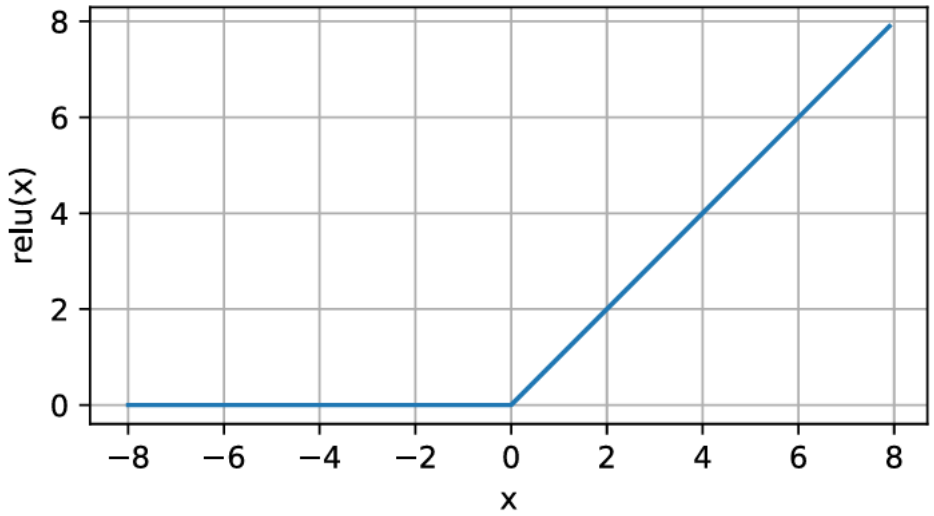
Las funciones de activación deciden si una neurona debe activarse o no calculando la suma ponderada y agregándole un sesgo.

Son operadores diferenciables para transformar señales de entrada en salidas, mientras que la mayoría de ellos agregan no linealidad.

Función ReLU

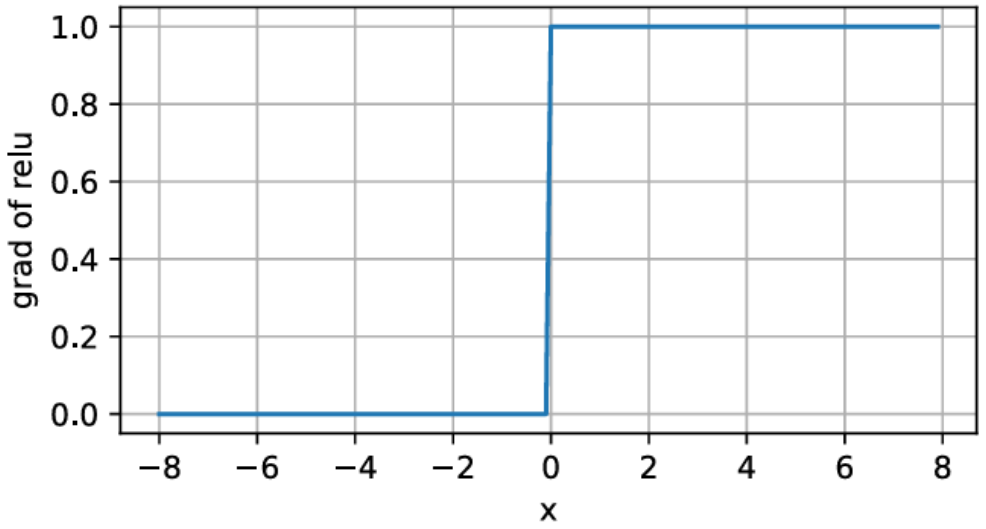
Su nombre viene de Rectified Linear Unit. Se define de la siguiente manera:

$$ReLU\ (x) = \begin{cases} x, si\ x \geq 0 \\ 0, si\ x < 0 \end{cases}$$



Función ReLU

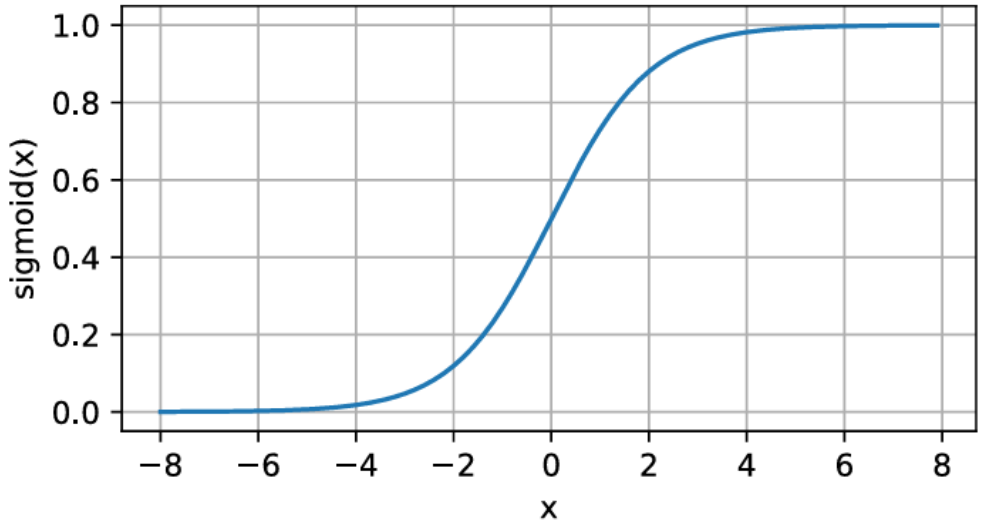
La función no es diferenciable para x = 0, sin embargo se puede considerar que la derivada de la misma es 0 para x = 0. Puesto que la derivada toma solo dos valores (o se desvanece o no) para cualquier entrada, esto hace que sea útil para la optimización y es más estable computacionalmente (evita el problema de los *exploding* y *vanishing gradients*, que veremos más adelante).



Función Sigmoide

Transforma entradas en el dominio de los números reales a salidas contenidas en el rango [0,1]. En la formulación clásica de las redes neuronales, la activación de la neurona dependía de si se llegaba o no a un determinado valor umbral, con lo que la neurona se disparaba o no se disparaba. Cuando el eje cambió hacia el aprendizaje basado en el gradiente, esta función ofreció una alternativa suave y diferenciable a la función escalón.

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

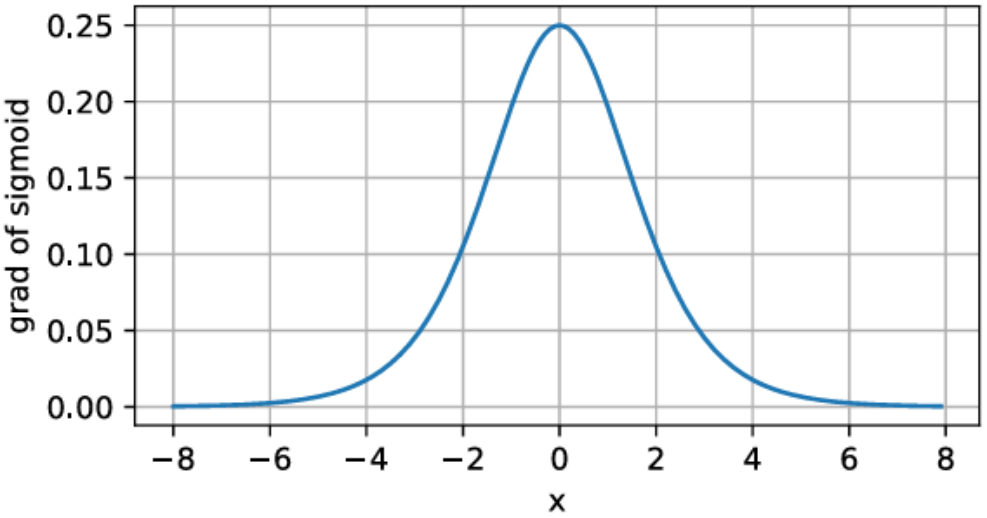


Función Sigmoide

Su derivada es:

$$\frac{\partial}{\sigma x}\sigma(x) = \sigma(x)(1 - \sigma(x))$$

Se utiliza en redes de clasificación binaria y en redes neuronales recurrentes para controlar el flujo de información en el tiempo. Su uso en redes neuronales clásicas se ve limitado por el hecho de que su derivada se desvanece para argumentos de entrada positivos y negativos grandes.



Función Tangente Hiperbólica

Restringe los valores de entrada al rango [-1,1]. Para valores cercanos a 0, se comporta como una función lineal. Su expresión y la de su derivada son las siguientes:

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

$$\frac{\partial}{\sigma x} tanh(x) = 1 - tanh^2(x)$$