Modelos de neuronas y redes neuronales artificiales

TÉCNICAS ESTADÍSTICAS PARA EL APRENDIZAJE II

Máster Universitario en Estadística Computacional
y Ciencia de Datos para la Toma de Decisiones

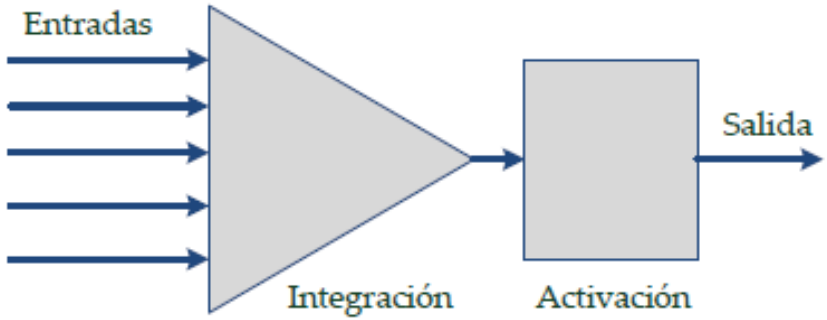
Modelos de neuronas y redes neuronales artificiales

- Neuronas o elementos de procesamiento
- Funciones de activación
- Perceptrón
- Arquitecturas de las redes neuronales artificiales

Neuronas o elementos de procesamiento

Un **modelo** es siempre una **simplificación de la realidad**. Por tanto, nunca será del todo correcto. Pese a ello, algunos modelos nos pueden resultar útiles en la práctica.

El **modelo de neurona artificial** consta de dos etapas:

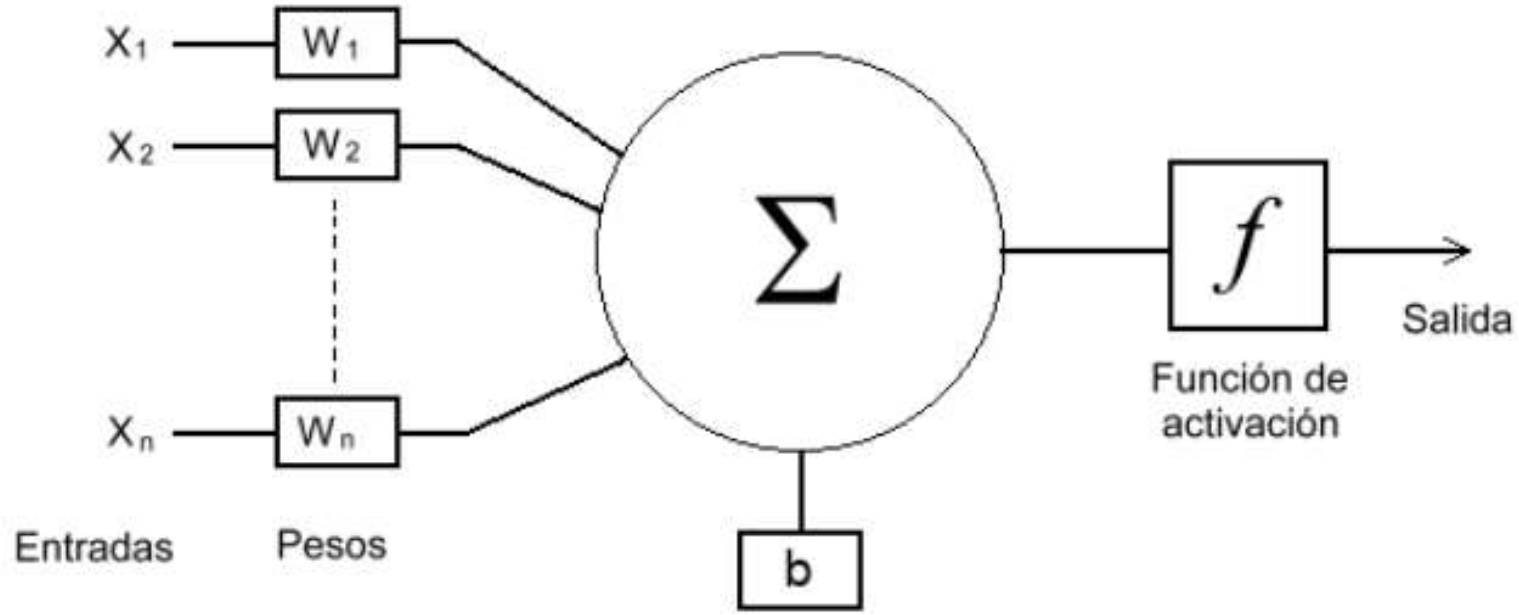


- En una **primera etapa**, se combinan las entradas provenientes de otras neuronas teniendo en cuenta los pesos de las sinapsis. El resultado de esta primera etapa es la entrada neta o excitación de la neurona.
- En la **segunda etapa**, la entrada neta se utiliza para determinar el valor de salida de la neurona, que se propagará a otras neuronas.

Neuronas o elementos de procesamiento

Cada **neurona** se estructura de la siguiente forma:

- Recibe n entradas.
- Pondera estas entradas por unos pesos w.
- Suma el resultado de esta ponderación.
- Le añade un umbral o *bias* b.
- Pasa el resultado por una función f de activación.



Neuronas o elementos de procesamiento

Habitualmente, las redes neuronales artificiales están formadas por conjuntos de neuronas que agruparemos en **capas**, de forma que todas las neuronas de una misma capa compartan ciertas características.

Por convención, denotaremos por x_i a cada una de las n **entradas** recibidas por una capa de neuronas formada por m neuronas.

La **salida de cada neurona** la representaremos por y_j, habitualmente un valor binario o real.

Los **pesos sinápticos** w_{ij} los utilizaremos para modelar las conexiones de entrada a las neuronas, siendo w_{ij} el peso asociado a la sinapsis que conecta la entrada i-ésima con la neurona j-ésima.

Neuronas o elementos de procesamiento

La mayoría de los modelos de redes neuronales artificiales utilizan un **sesgo** externo, **o *bias***, b_j para determinar la entrada neta de la neurona, z_j:

$$z_j = b_j + \sum_{i=1}^{n} w_{ij} x_i$$

Este sesgo es un parámetro adicional de la neurona, igual que los pesos.

A menudo se asume que el sesgo no es más que un peso adicional vinculado a una entrada fija con valor 1.

Si establecemos $w_{0j} = b_j$ y $x_0 = 1$, la expresión anterior se puede simplificar:

$$z_j = \sum_{i=0}^{n} w_{ij} x_i$$

Neuronas o elementos de procesamiento

En el **modelo de neurona artificial**, la generación de la salida de la neurona es responsabilidad de una **fase de activación** independiente de la fase de integración de entradas.

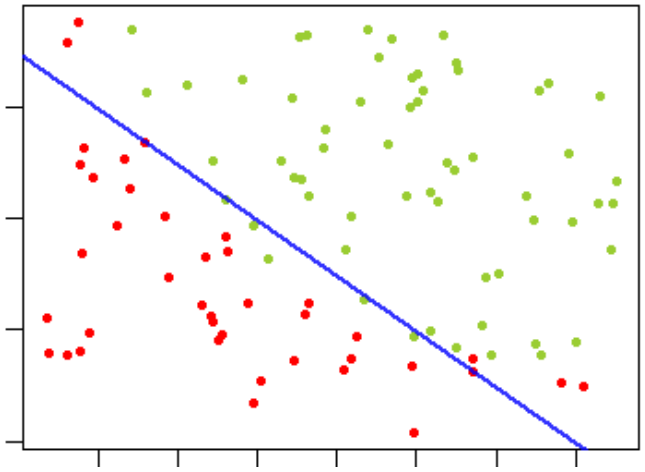
Generalmente, la activación de una neurona artificial no reproduce sin más su entrada neta, sino que aplica algún tipo de **transformación no lineal** a esa entrada neta.

¿Por qué no lineal? Porque cuando construyamos redes con múltiples capas, si las neuronas fuesen siempre lineales, el resultado final del cálculo realizado por la red seguiría siendo una función lineal de las entradas.

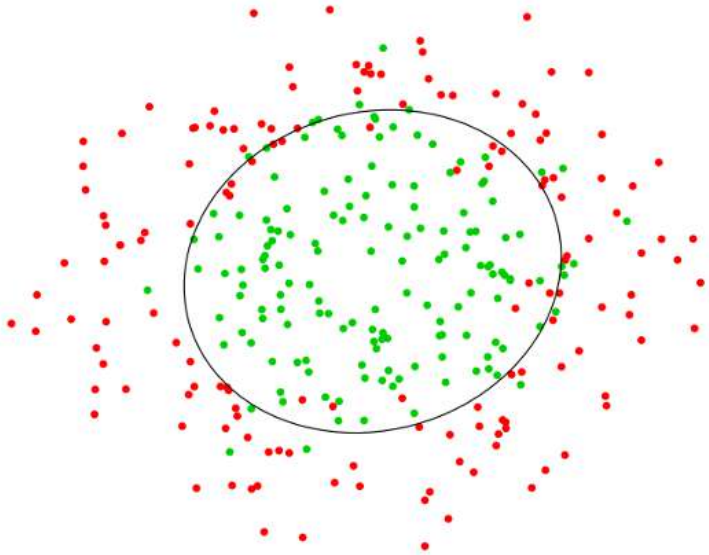
Como la mayor parte de los problemas reales incorporan algún tipo de no linealidad, nos interesará que nuestras redes sean capaces de representar esas no linealidades.

Neuronas o elementos de procesamiento

Cualquier red neuronal compuesta únicamente por elementos lineales siempre se podría sustituir por una red formada por una única capa de neuronas lineales. Las **neuronas lineales**, obviamente, sólo pueden representar funciones lineales.



Al añadir una función de activación no lineal y combinar el resultado de las distintas neuronas capa tras capa lo que obtenemos es un **clasificador no lineal** con el que podemos hacer discriminaciones como la siguiente:



Neuronas o elementos de procesamiento

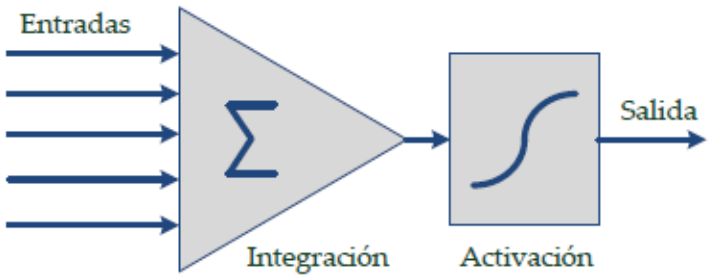
En la **etapa de integración** de entradas, una neurona artificial combina las diferentes entradas x_i con sus pesos para determinar su entrada neta, z_j:

$$z_j = \sum_i w_{ij} x_i$$

En la **etapa de activación**, una neurona artificial utiliza el valor asociado a su entrada neta para generar una salida y_j. El modelo más habitual genera una salida de tipo numérico a partir de la entrada neta de la neurona:

$$y_j = f(z_j) = f\left(\sum_i w_{ij} x_i\right)$$

donde la función f es la función de activación de la neurona, usualmente no lineal.



Neuronas o elementos de procesamiento

Cuando diseñamos una red neuronal artificial compuesta por varias capas de elementos de procesamiento idénticos, el uso de la **notación vectorial** puede simplificar determinadas expresiones.

Por ejemplo, en vez de definir la salida de una única neurona y_j, podemos emplear la notación vectorial para definir simultáneamente las salidas de todas las neuronas de una capa de neuronas:

$$y = f(z) = f(Wx)$$

donde y es el vector de salida de la capa, de tamaño m (el número de neuronas de la capa).

Asumimos que existe una entrada fija, $x_0 = 1$, y los sesgos van representados en la matriz de pesos W de tamaño $m \times (n+1)$.

Neuronas o elementos de procesamiento

Funciones de activación

Perceptrón

Arquitecturas de las redes neuronales artificiales

Funciones de activación

Los modelos de neuronas utilizados en redes neuronales artificiales combinan sus entradas usando pesos que modelan sus conexiones sinápticas y, a continuación, le aplican a la entrada neta de la neurona una **función de activación** o transferencia.

La entrada neta de la neurona recoge el nivel de estímulo que la neurona recibe de sus entradas y **es la función de activación la que determina cuál es la salida de la neurona**.

Como veremos más adelante, la **derivada de la función de activación** desempeña un papel fundamental.

Recurriremos a funciones de activación cuya derivada esté definida para cualquier valor para poder calcular la pendiente en un punto dado. De esta forma, durante el entrenamiento, podremos saber **cómo ajustar los pesos para reducir el error**.

Funciones de activación

A grandes rasgos, las funciones de activación se pueden clasificar en funciones de activación discretas y continuas:

Funciones de activación discretas

La salida de la neurona es discreta: sólo puede tomar un conjunto finito de valores.

Normalmente, se utilizan dos valores, por lo que hablamos de **neuronas binarias** cuando la salida es 0 ó 1, mientras que hablamos de **neuronas bipolares** cuando su salida puede ser -1 ó +1.

En los primeros modelos de redes neuronales artificiales, las funciones de activación eran discretas.

Funciones de activación continuas

La salida de la neurona puede tomar cualquier valor dentro de un intervalo. Generalmente, el rango de ese intervalo está limitado, o bien al intervalo [0, 1] o bien al intervalo [−1, 1].

Dada una función de activación f[0,1] con un rango definido, el intervalo [0, 1], es fácil crear una función de activación con el rango deseado [a, b]. Basta con aplicar una transformación lineal del tipo:

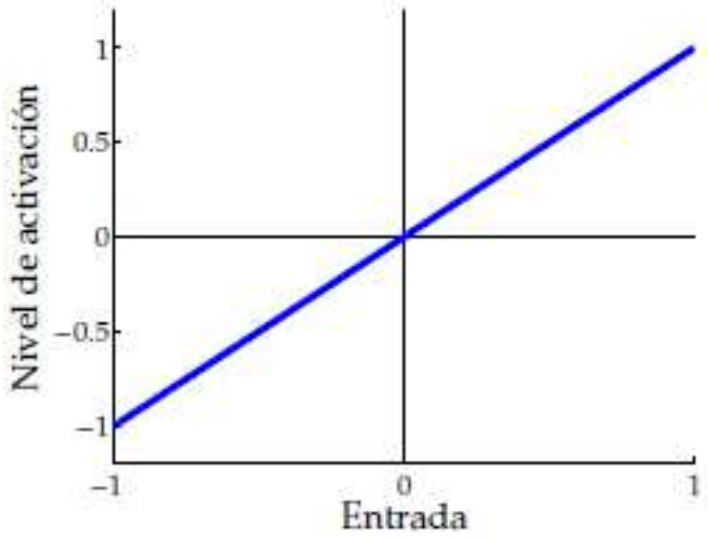
$$f_{[a,b]}(z) = a + (b - a) f_{[0,1]}(z)$$

Dentro de las funciones de activación continuas, se pueden utilizar tanto **funciones de activación lineales** (p.ej. la función identidad) como **funciones no lineales**, que son las más utilizadas ya que son las que dotan a una red neuronal multicapa de su capacidad de aproximador universal.

Función identidad

La función de activación lineal más sencilla que podemos imaginarnos para una neurona artificial es la función identidad:

$$y_j = z_j = \sum_i w_{ij} x_i$$



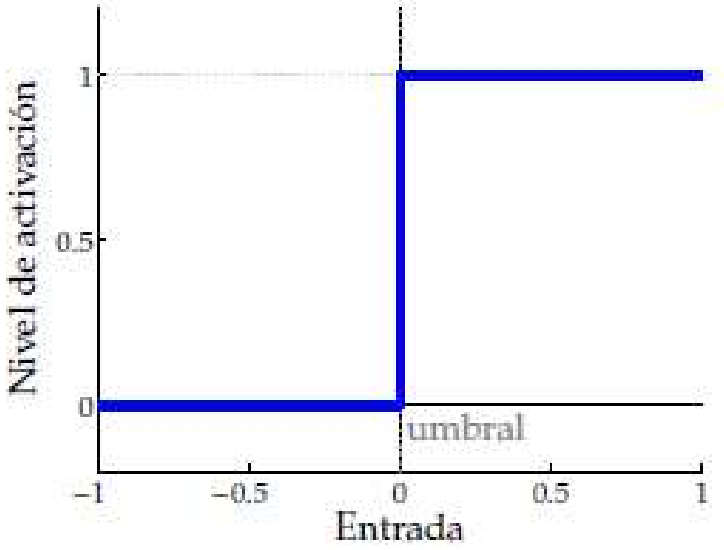
Como hemos comentado, la característica principal de las neuronas lineales es que, si conectamos varias capas de neuronas lineales en serie, el resultado final será siempre equivalente a una única capa de neuronas lineales.

Las capas de neuronas lineales actúan como simples clasificadores lineales.

Función escalón o función umbral

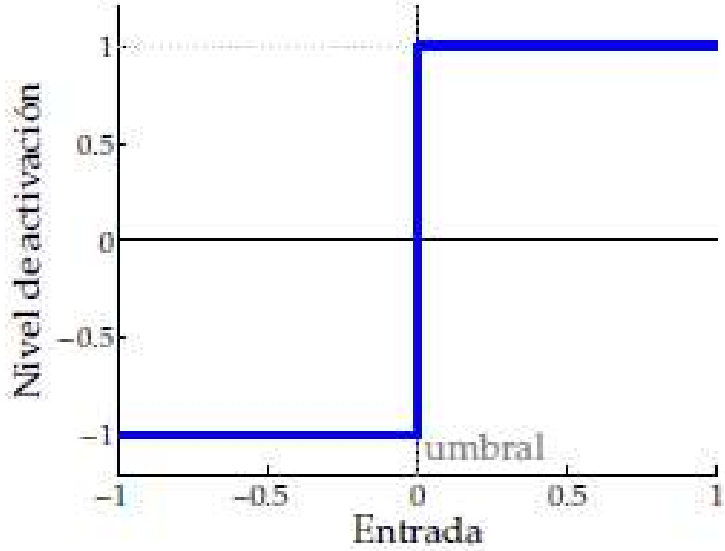
La primera función no lineal que se utilizó para modelar neuronas artificiales es la función usada en el modelo neuronal de McCulloch y Pitts (1943):

$$y_j = 1_{z_j \geq 0} = \begin{cases} 1 \ \ si \ \ z_j \geq 0 \\ 0 \ \ si \ \ z_j < 0 \end{cases}$$



Se trata de una función de activación binaria de la que podemos derivar una versión bipolar: la **función signo**.

$$y_j = \mathrm{sgn}(z_j) = \begin{cases} 1 \ \ si \ \ z_j \geq 0 \\ -1 \ \ si \ \ z_j < 0 \end{cases}$$



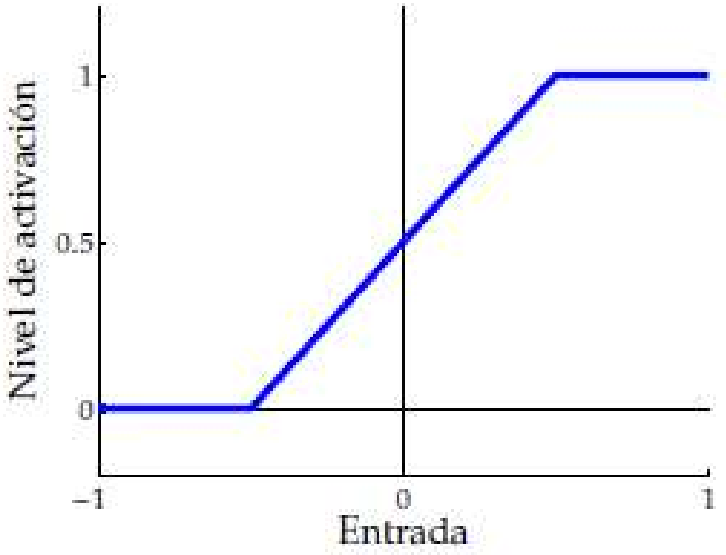
Función lineal con saturación

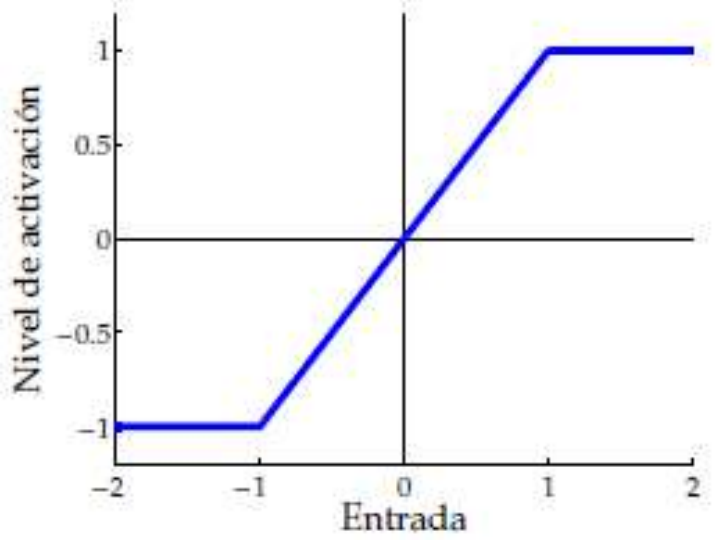
Si queremos disponer de una función de activación continua, podemos diseñar fácilmente una función lineal de activación a trozos que limite el rango de salida de la neurona:

$$y_j = \begin{cases} 1 & si \;\; z_j > \frac{1}{2} \\ z_j + \frac{1}{2} & si \;\; -\frac{1}{2} \leq z_j \leq \frac{1}{2} \\ 0 & si \;\; z_j < -\frac{1}{2} \end{cases}$$

O una versión bipolar:

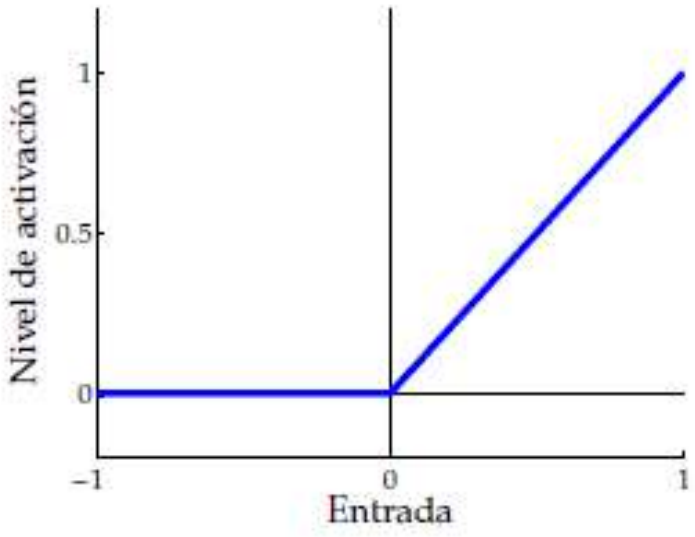
$$y_j = \begin{cases} 1 & si \;\; z_j > 1 \\ z_j & si \;\; -1 \leq z_j \leq 1 \\ -1 & si \;\; z_j < -1 \end{cases}$$





Función de activación lineal rectificada

Utilizada en las unidades lineales rectificadas o ReLU [*REctified Linear Units*] muy habituales en *deep learning:*



$$y_j = f_{relu}(z_j) = \begin{cases} z_j & si \;\; z_j \geq 0 \\ 0 & si \;\; z_j < 0 \end{cases}$$

La derivada de una función lineal rectificada es trivial y coincide con la función escalón utilizada por las neuronas de McCulloch y Pitts:

$$\frac{d}{dz} f_{relu}(z) = 1_{z \geq 0} = \begin{cases} 1 & si \;\; z \geq 0 \\ 0 & si \;\; z < 0 \end{cases}$$

Función de activación sigmoidal

Usualmente, nos interesará que la función de activación de una neurona sea, además de **no lineal, estrictamente creciente, continua y derivable**.

Las funciones sigmoidales satisfacen todos esos requisitos, lo que las hace especialmente útiles en redes neuronales que se entrenan usando ***backpropagation***.

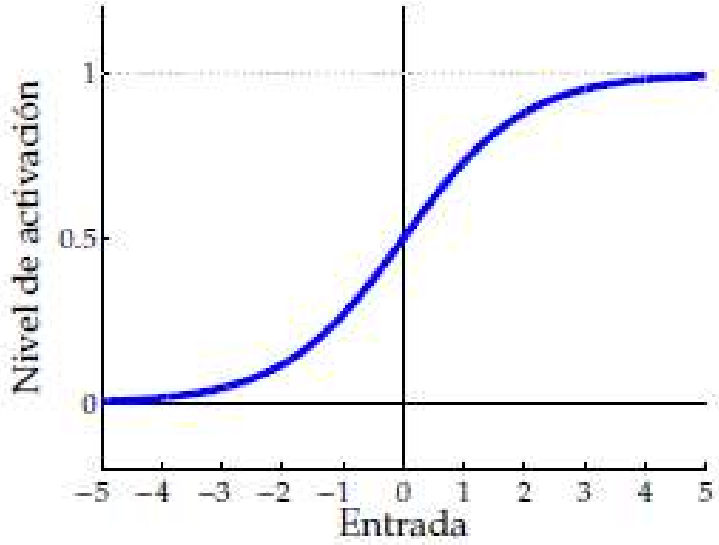
Backpropagation es un **algoritmo de propagación de errores** que, dados los errores observados en la capa de salida, propaga esos errores hacia atrás en la red para ir ajustando sus parámetros internos. La forma en que se ajustan esos parámetros depende de la derivada de la función de activación de las neuronas, de ahí nuestro interés en que la función sea derivable.

Funciones de activación sigmoidal

Existen distintas funciones sigmoidales. Todas con su característica forma de 'S':

Función logística

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



La función logística es simétrica, en el sentido de que:

$$\sigma(-z) = 1 - \sigma(z)$$

Más adelante, veremos también la función *softmax*, una extensión de la función logística para la capa de salida de una red neuronal diseñada para resolver problemas de clasificación con múltiples categorías .

Función logística

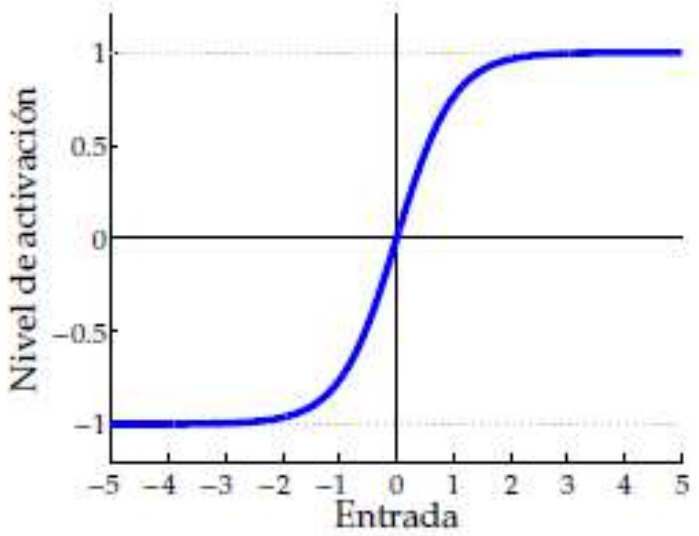
La función logística es una sigmoide "binaria", con rango [0, 1], cuya derivada es:

$$\frac{d}{dz}\sigma(z) = \sigma(z)\,(1\text{-}\,\sigma(z))$$

Como vemos, la relación entre el valor de la función (z) y su derivada nos permite calcular el valor de esta última con **sólo dos operaciones aritméticas**, una resta y una multiplicación, una vez que conocemos el valor de $\sigma(z)$, sin necesidad de realizar llamadas a funciones más costosas desde el punto de vista computacional.

Tangente hiperbólica

Función sigmoide con rango [-1, 1]:



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

La tangente hiperbólica está estrechamente relacionada con la función logística:

$$tanh(z) = 2\,\sigma(2z) - 1$$

La derivada de la tangente hiperbólica en un punto también se puede expresar directamente en términos del valor de la función en ese punto:

$$\frac{d}{dz} tanh(z) = \big(1 + tanh(z)\big)\big(1 - tanh(z)\big) = (1 - tanh^2(z))$$

Perceptrón

McCulloch y Pitts publicaron, en 1943, el **primer modelo computacional de una red neuronal**[1].

Su idea de interpretar el cerebro como un ordenador compuesto de múltiples unidades interconectadas daría lugar al desarrollo de las redes neuronales artificiales.

Sin embargo, el modelo neuronal de McCulloch y Pitts **carecía de un algoritmo de aprendizaje** que permitiese ajustar automáticamente los parámetros de la red neuronal.

No fue hasta más de una década después, en 1957, cuando Frank Rosenblatt publicó un artículo en el que se describía el **primer modelo de red neuronal artificial capaz de aprender**: el **perceptrón**[2].

[1]Warren S. McCulloch y Walter Pitts. A Logical Calculus of the Ideas Immanent in Nervous Activity. The Bulletin of Mathematical Biophysics, 5(4): 115–133, 1943.

[2]Frank Rosenblatt. The perceptron: A perceiving and recognizing automaton. Cornell Aeronautical Laboratory, Buffalo, New York, Report 85-60-1, 1957.

Perceptrón

Los perceptrones simples **permiten reconocer patrones**.

Desde el punto de vista del aprendizaje automático, son **clasificadores binarios**: funciones que pueden decidir si una entrada dada pertenece a una clase específica o no.

Para ser más precisos, los perceptrones son un tipo de **clasificador lineal**: clasificadores cuya frontera de decisión, la que separa los ejemplos de una clase de los de otra, viene dada en forma de línea recta en dos dimensiones, de plano en tres dimensiones o de hiperplano en general.

Por tanto, un perceptrón sólo será capaz de clasificar correctamente clases que sean linealmente separables.

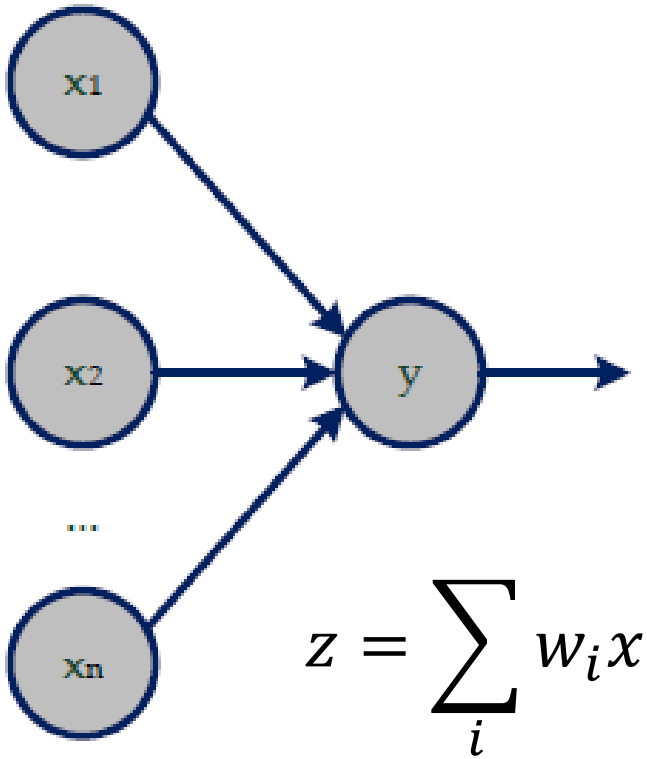
Perceptrón

Matemáticamente, la frontera de decisión definida por un clasificador lineal es de la forma:

$$b + \sum_i w_i x_i = 0$$

o, si consideramos el umbral, sesgo o *bias* como un peso más, asociado a una entrada fija $x_0 = 1$:

$$\sum_i w_i x_i = 0$$



Si z se halla por encima del umbral de activación de la neurona, ésta se activa y se decide que el vector de entrada corresponde a un ejemplo de la clase objetivo, habitualmente denominada **clase positiva ($y = 1$).** Si no es así, el vector de entrada no provoca la activación de la neurona y el ejemplo se asocia a la **clase negativa ($y = 0$)**.

Perceptrón

¿Cómo se realiza el **ajuste de los pesos** del perceptrón?

Para cada ejemplo del conjunto de entrenamiento:

- Si la salida del perceptrón es correcta, se dejan los pesos tal cual.
- Si la unidad de salida incorrectamente da un cero (un falso negativo), se añade el vector de entrada al vector de pesos.
- Si la unidad de salida incorrectamente da un uno (un falso positivo), se resta el vector de entrada del vector de pesos.

Se recorre varias veces el conjunto de entrenamiento hasta que el algoritmo converge, la tasa de error baja por debajo de un umbral establecido por el usuario o no se aprecia una mejora significativa en la precisión del clasificador tras un recorrido completo.

A cada recorrido del conjunto de datos durante el entrenamiento de una red neuronal se le suele denominar **época.**

Perceptrón

Conforme avanza el aprendizaje, podemos **evaluar la tasa de error** del clasificador en cada época utilizando, por ejemplo, el error cuadrático medio [*MSE: Mean Squared Error*] :

$$MSE = \frac{1}{n}\sum_{j=1}^{n}(t_j - y_j)^2$$

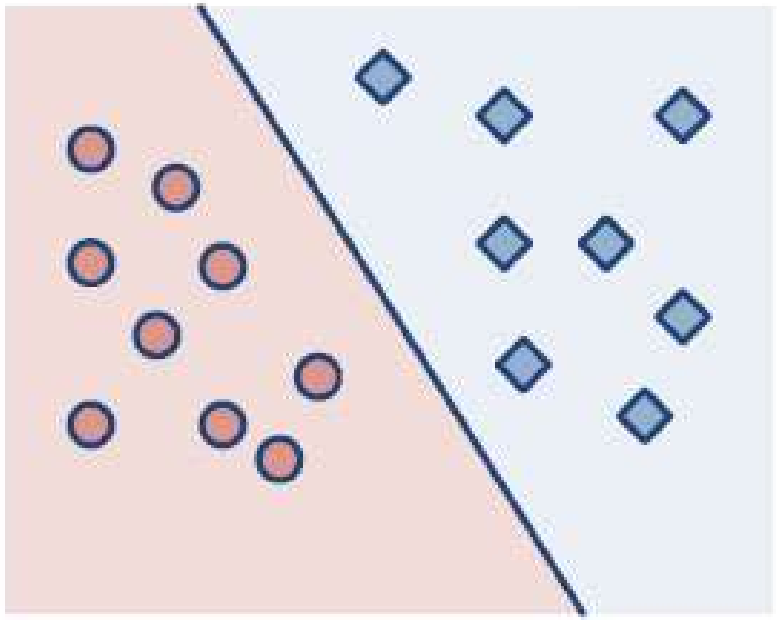
donde y_j es la salida del perceptrón y t_j la salida deseada [*target*].

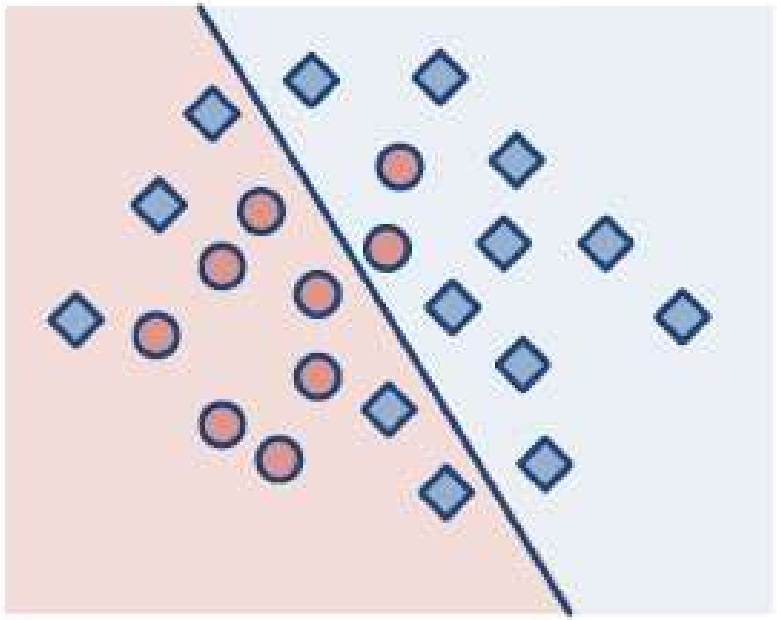
El **algoritmo de aprendizaje** del perceptrón garantiza encontrar un conjunto de pesos que proporcione la respuesta correcta, si tal conjunto existe.

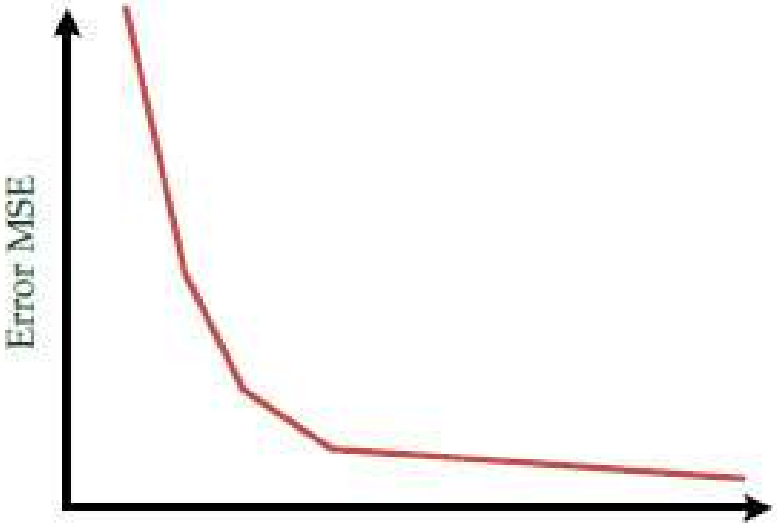
Dado que el perceptrón es un **modelo de clasificación lineal**, será capaz de clasificar correctamente los ejemplos de entrada siempre que las clases sean linealmente separables.

Será incapaz de encontrar un conjunto de pesos adecuado para separar clases no linealmente separables.

Perceptrón





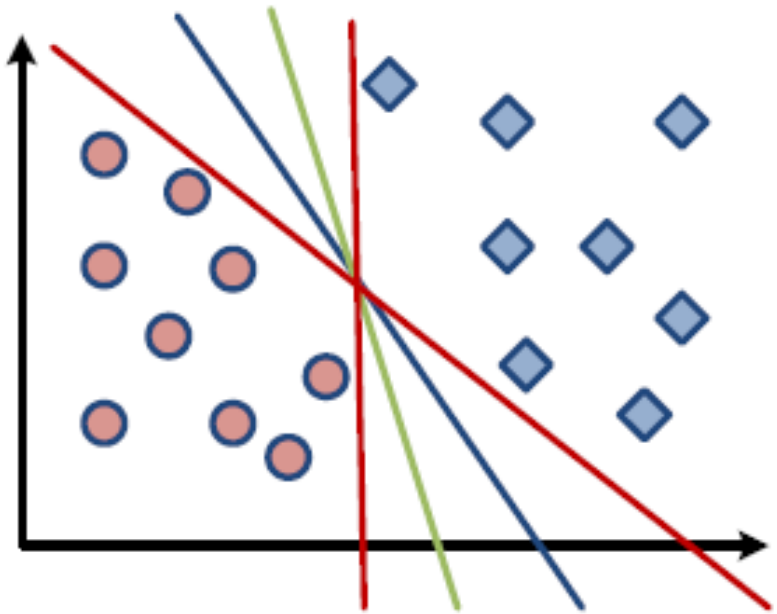


Perceptrón

El **algoritmo de aprendizaje del perceptrón** va corrigiendo los errores conforme se los encuentra, desde cualquier configuración inicial de los pesos, hasta clasificar correctamente todos los ejemplos siempre que las clases sean linealmente separables.

Existe un **número infinito de fronteras de decisión** que nos podrían servir para separar dos clases linealmente separables.

Como consecuencia, el algoritmo de aprendizaje puede que escoja una frontera de decisión que, aun clasificando correctamente todos los ejemplos del conjunto de entrenamiento, **no** resulte **óptima para clasificar datos diferentes** a los del conjunto de entrenamiento.



Perceptrón

Aunque, inicialmente, el perceptrón parecía prometedor, Minsky y Papert, del MIT, publicaron en 1969 un famoso libro[1] en el que analizaban detalladamente las **limitaciones del perceptrón**.

Aunque los resultados de Minsky y Papert hacían referencia a perceptrones simples, exclusivamente, muchos pensaron que esas limitaciones se extendían a todos los modelos de redes neuronales.

Las limitaciones, no sólo del perceptrón, sino también de los sistemas de reconocimiento de voz y traducción automática de la época, hicieron que la Inteligencia Artificial pasase temporalmente de moda, entrando en el período gris conocido como **invierno de la I.A**.

No fue hasta más de diez años después cuando las redes neuronales resurgieron con fuerza, a mediados de los 80, entrenadas utilizando el **algoritmo de propagación de errores** conocido por ***backpropagation***.

[1]Marvin Minsky y Seymour Papert. Perceptrons: An Introduction to Computational Geometry. MIT Press, 1st edition, 1969. ISBN 0262130432

Neuronas o elementos de procesamiento

Funciones de activación

Perceptrón

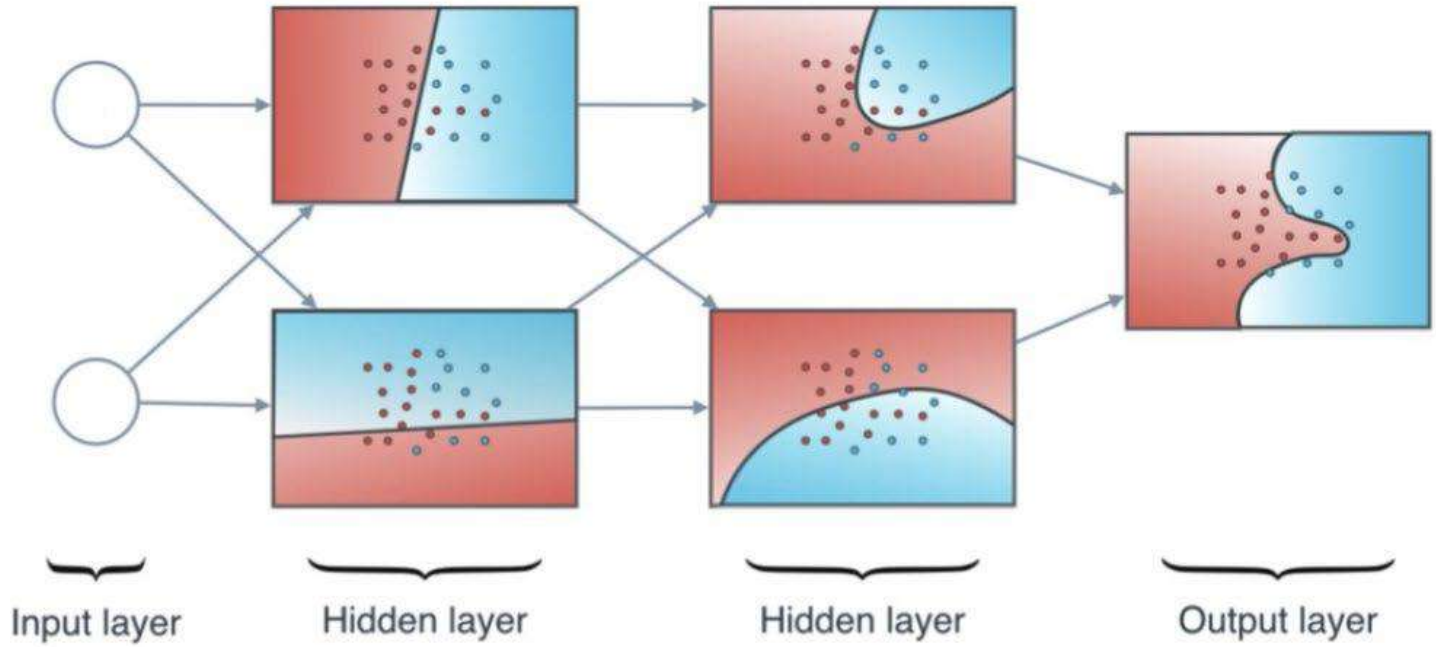
Arquitecturas de las redes neuronales artificiales

Arquitecturas de las redes neuronales artificiales

Una neurona no puede representar toda la información → **Necesitamos una agrupación de neuronas**

¿Cómo podemos combinar colecciones de neuronas para resolver problemas de interés?

Utilizaremos **capas de neuronas**, donde cada capa extraerá características cada vez más complejas.



Redes feed-forward

Redes neuronales con **múltiples capas de neuronas**. Las neuronas de cada capa son independientes entre sí y operan en paralelo, lo que facilita su implementación eficiente si disponemos de una GPU.

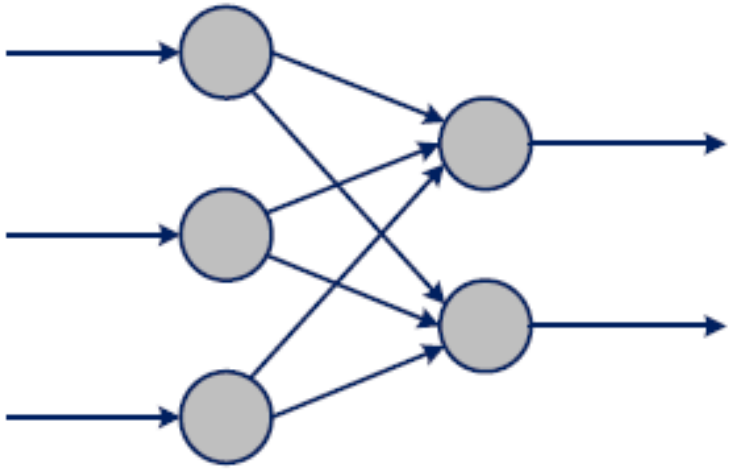
Las distintas capas se conectan entre sí de tal forma que la salida de la capa i se utiliza como entrada en la capa $i + 1$ (carecen de mecanismo alguno de realimentación).

Las capas de una red multicapa se dividen en dos categorías: **capas visibles** (capas de entrada y salida) y **capas ocultas** (capas intermedias).

Nos podemos encontrar con varias situaciones dependiendo del número de capas ocultas que se utilicen:

Redes feed-forward. Redes simples, sin capas ocultas

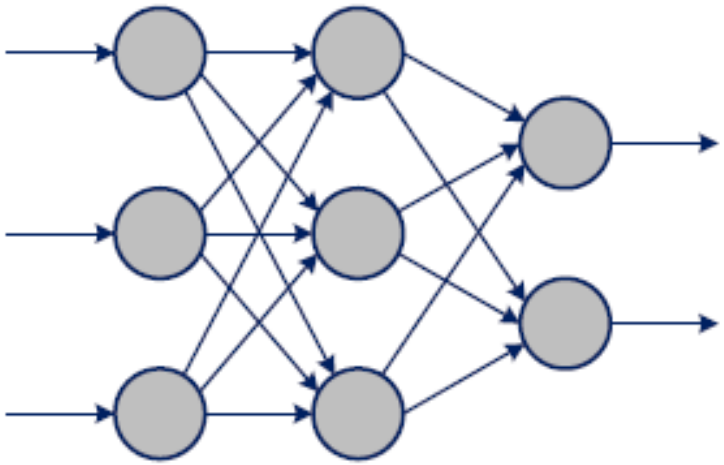
Es el caso más simple de red neuronal: sólo capas de entrada y salida.



Los **perceptrones** son el modelo más conocido de red neuronal sin capas ocultas. Fueron las primeras redes neuronales para las que se diseñó un algoritmo de aprendizaje, en los años 50 del siglo XX.

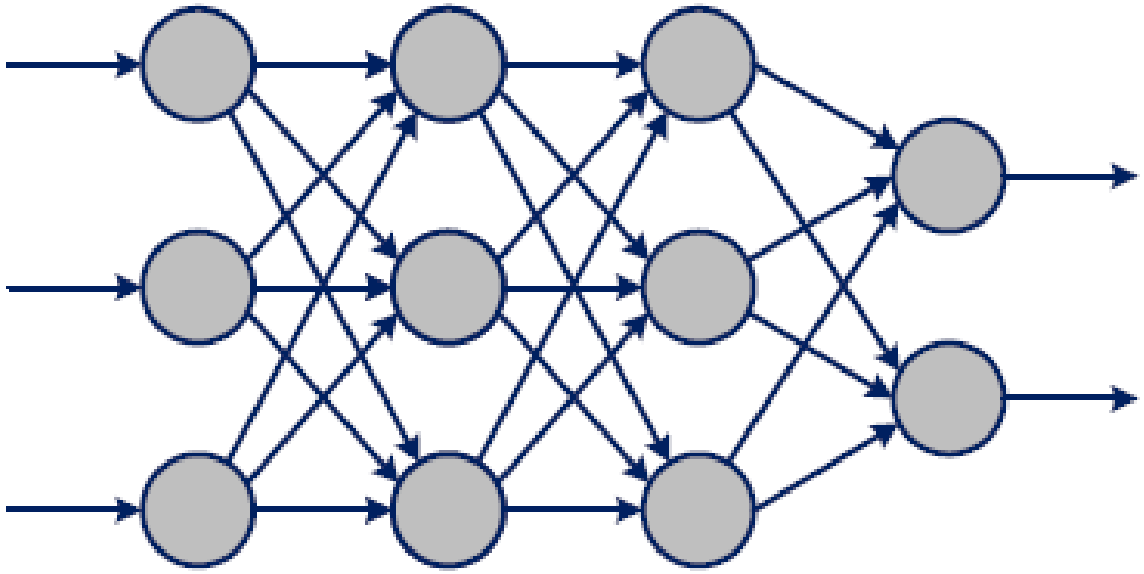
Redes feed-forward. Redes multicapa, con una capa oculta

La presencia de **una única capa oculta** ya dota a la red multicapa de su capacidad de **aproximador universal** (informalmente, su capacidad de aprender cualquier cosa que se pueda aprender).



Este tipo de redes fue muy popular desde los años 80 hasta finales del siglo XX. El hecho de tener capas intermedias no visibles nos obligará a utilizar algoritmos como *backpropagation* para **ajustar** sus **parámetros** internos.

Redes feed-forward. Redes profundas [*deep networks*], con varias capas ocultas

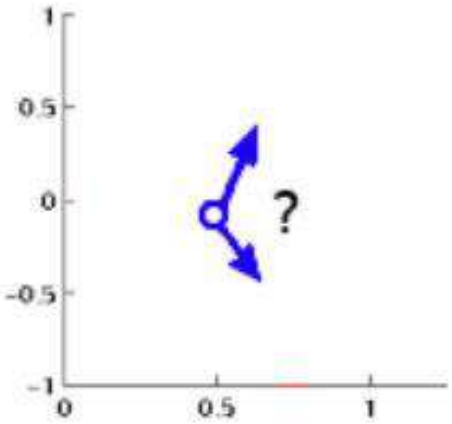


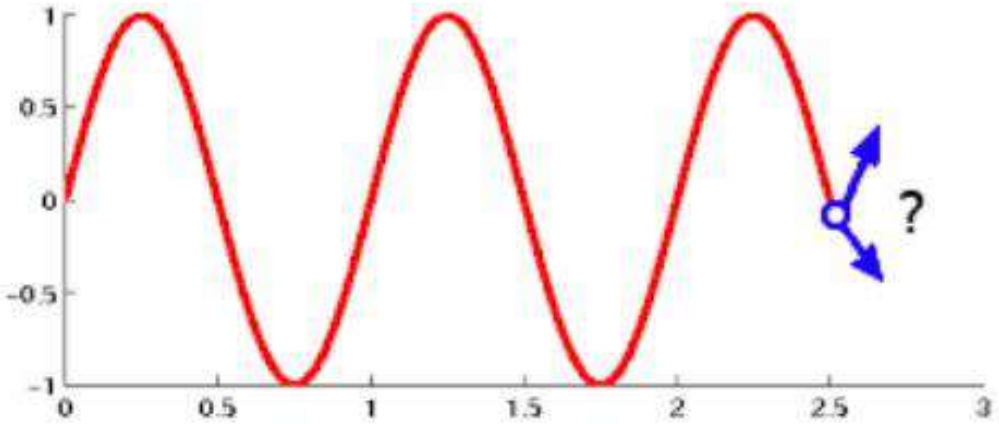
Las redes neuronales actuales suelen incluir **múltiples capas ocultas**.

Redes recurrentes RNN [*Recurrent Neural Network*]

Determinados problemas no se pueden resolver usando solamente la información del instante actual.

Por ejemplo, qué está sucediendo en una película viendo solamente una imagen, el significado de una palabra sin consultar su contexto, o más sencillo, el siguiente valor en una serie basándonos solamente en el valor anterior.



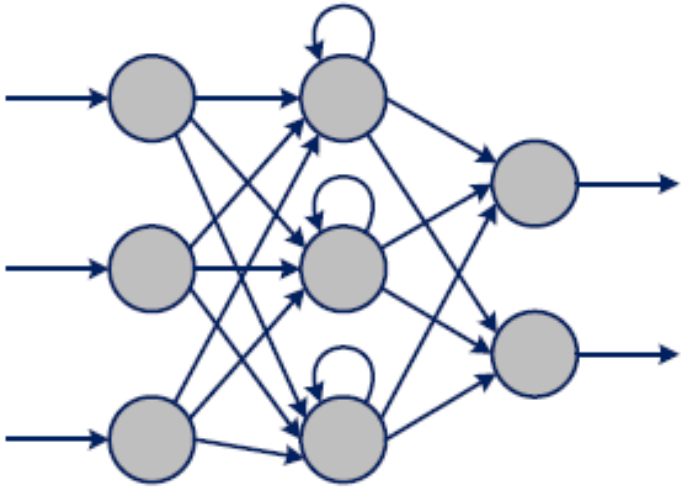


Redes recurrentes

Por su topología, las **redes de tipo feed-forward** no pueden hacer esto: sólo condicionan su salida a la entrada, es decir, para una misma entrada, darán la misma salida, **carecen por completo de memoria**.

Para solucionarlo, en los años 80 del siglo XX, surgieron las Redes Neuronales Recurrentes.

Las neuronas recurrentes son similares a las neuronas normales pero con **bucles recurrentes** que les permiten mantener un **estado** en el tiempo.



Redes recurrentes

Una **neurona de una red recurrente** predice su salida a partir de la entrada y también a partir de su **valor o estado anterior** → el estado de la neurona va cambiando o evolucionando en función de lo que ha visto hasta ese momento.

En algunas tareas solo es necesario **utilizar la información más reciente** para calcular la siguiente predicción.

Por ejemplo, en el caso de las series temporales solo necesitamos consultar los n elementos anteriores, o para aprender a predecir la siguiente palabra en una frase sencilla: "Las nubes están en el ____".

Redes recurrentes. Problema de las dependencias a largo plazo

A veces sí es necesario **consultar más contexto** para obtener la siguiente predicción.

Por ejemplo, para predecir la siguiente palabra en la frase:
"Nací en Francia pero a los 12 años me vine a España, por eso hablo tan bien el _____".

El contexto reciente sugiere que la siguiente palabra es un idioma pero para saber el idioma correcto tenemos que utilizar la información del principio de la frase.

Este es un problema para las redes recurrentes, dado que según aumenta la distancia hasta la información a utilizar, se vuelve más difícil aprender a conectar la información.

Redes convolutivas CNN [*Convolutional Neural Network*]

Empleadas en **visión artificial** para reconocer objetos en imágenes y localizarlos.

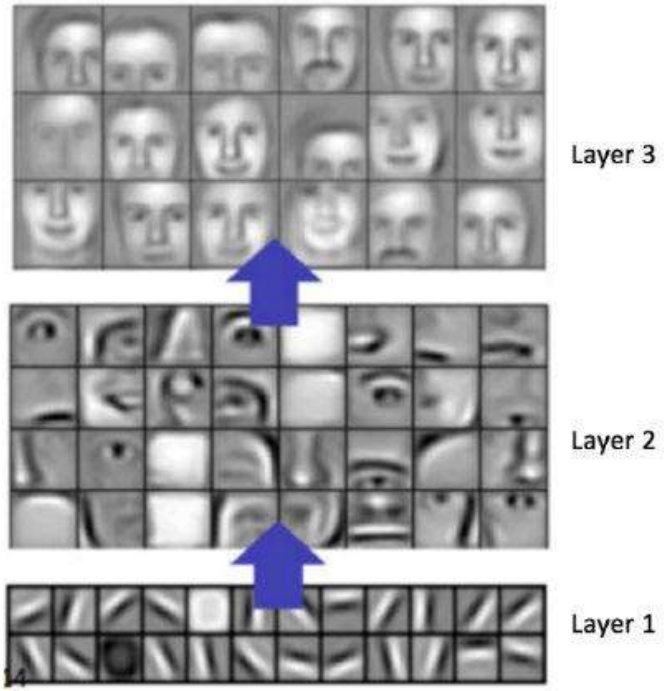
Son redes multicapa en las que algunas capas realizan una **operación de convolución** en lugar de la tradicional multiplicación matricial de entradas por pesos.

Matemáticamente, la convolución es una operación matemática que se realiza sobre dos funciones para producir una tercera que se suele interpretar como una versión modificada (filtrada) de una de las funciones originales.

Redes convolutivas

Las primeras capas tienden a aprender **características simples** (como bordes), mientras que **características más complejas** (formas básicas) se aprenden a medida que pasamos a capas más profundas.

Su éxito en la resolución de problemas de **visión artificial** que, hasta hace poco, se consideraban casi intratables, sirvió para volver a poner de moda las redes neuronales en Inteligencia Artificial. Su explotación comercial dio lugar a lo que hoy entendemos por *deep learning*.



Arquitecturas de las redes neuronales artificiales

Estamos interesados en crear redes neuronales que resuelvan problemas prácticos concretos y, para ello, tendremos que ajustar sus parámetros mediante un proceso de **entrenamiento** → próximo tema

Para terminar este tema, veamos una simulación de diferentes arquitecturas de redes neuronales:

https://playground.tensorflow.org/

Bibliografía

•[1] Fernando Berzal. Redes Neuronales & Deep Learning. Edición independiente.

•[2] François Chollet. Deep learning with Python. Manning Shelter Island.

•[3] Ian Goodfellow, Yoshua Bengio & Aaron Courville. Deep Learning . MIT Press.