Clasificación: redes neuronales

Ramon Sangüesa i Solé

PID_00165730

Universitat Oberta de Catalunya
www.uoc.edu

Índice

Introducción

Las redes neuronales son un modelo con cierta historia dentro de la informática. En efecto, en los años cuarenta McCulloch (neurofisiólogo) y Pitts (lógico) propusieron un modelo para intentar explicar el funcionamiento de las neuronas del cerebro humano. Este modelo tenía interés desde el punto de vista de la explicación de un sistema complejo, pero resultó que también se podía utilizar para efectuar cálculos en dominios sin ninguna relación con la fisiología o la anatomía.

Lectura complementaria

Encontraréis más información sobre la primera propuesta de modelo de redes neuronales en la obra siguiente:

W.S. McCulloch; W. Pitts (1943). "A Logical Calculus of Ideas Immanent in Nervous Systems". *Bulletin of Mathematical Biophysics* (núm. 5, pág. 115-137).

En los años cincuenta, se empezaron a implementar modelos de cálculo basados en la idea del perceptrón. Uno de los ejemplos típicos de utilización era en control, intentando mantener en equilibrio una escoba sobre un carro en movimiento. La escoba podía mantenerse en equilibrio a fuerza de compensar los movimientos del carro hacia adelante y hacia atrás. Cuando la escoba empezaba a caerse hacia la izquierda, el carro tenía que moverse hacia la derecha y viceversa. Lo interesante de este modelo era que no había ningún programa explícito de control, sino que "aprendía" por refuerzo positivo o negativo. Cuando un movimiento no conseguía equilibrar la escoba, el sistema neuronal recibía un refuerzo negativo; cuando lo conseguía, recibía uno positivo. De esta manera, el sistema "evolucionaba", "aprendía" y, finalmente, adquiría la capacidad de mantener el sistema en el estado deseado.

Lectura complementaria

Encontraréis más información sobre el perceptrón en la obra siguiente:

F. Rosenblatt (1957). "The Perceptron: a Perceiving and Recognizing Automaton". Report 85-460-1. Ithaca, Nueva York: Cornell Aeronautical Laboratory.

Las **ideas principales de los modelos de redes neuronales** son las siguientes:

1) La red aprende por reajuste mediante la presentación de ejemplos y/o contraejemplos.

2) Los diferentes refuerzos conducen a la red hacia un estado de equilibrio en el que el conocimiento con respecto a la tarea que hay que realizar queda "memorizado" en la fuerza de las conexiones entre neuronas: no hay conocimiento explícitamente interpretable.

La utilidad de los primeros modelos de sistemas neuronales se veía bastante limitada por la capacidad de los ordenadores de la época. Además, Marvin Minsky, en un famoso ataque, demostró algunas de las limitaciones teóricas de los modelos existentes, como, por ejemplo, su incapacidad para efectuar tareas de clasificación de cierta complejidad. Este hecho hizo que la investigación y las aplicaciones de las redes se estancasen durante la década de los setenta. Fue preciso esperar hasta 1982 para que John Hopfield propusiera el método de "propagación hacia atrás" (*backpropagation*), el cual permitió superar algunas limitaciones teóricas y, además, facilitó de una manera extraordinaria la fase de entrenamiento de las redes neuronales.

Lecturas complementarias

Podéis encontrar el artículo de Minsky en el que explica las limitaciones teóricas de las redes neuronales y un artículo sobre el método de propagación hacia atrás, respectivamente, en las obras siguientes:

M. Minsky; S. Papert (1969). *Perceptrons: an introduction to computational geometry*. MIT Press.

J.J. Hopfield (1982). "Neurons with Graded Response Have Collective Computational Properties like those of Two-State Neurons". *Proceedings of the National Academy of Sciences* (núm. 79, pág. 2.254-2.258).

Durante este tiempo las redes neuronales han tenido una expansión considerable, y han ampliado en gran medida su campo de aplicación. Se utilizan predominantemente en problemas de clasificación y de predicción; también tienen aplicaciones en métodos de agregación mediante modificaciones del modelo original, o en desarrollos originados desde otros ámbitos, pero bajo una metáfora conexionista parecida, como por ejemplo los mapas asociativos, de los que no hablaremos.

Lectura complementaria

Encontraréis información sobre los mapas asociativos en:

T. Kohonen (1989). *Self-Organization and Associative Memory* (3ª ed.). Berlín: Springer-Verlag.

Objetivos

Con los materiales asociados a este módulo, el estudiante alcanzará los objetivos siguientes:

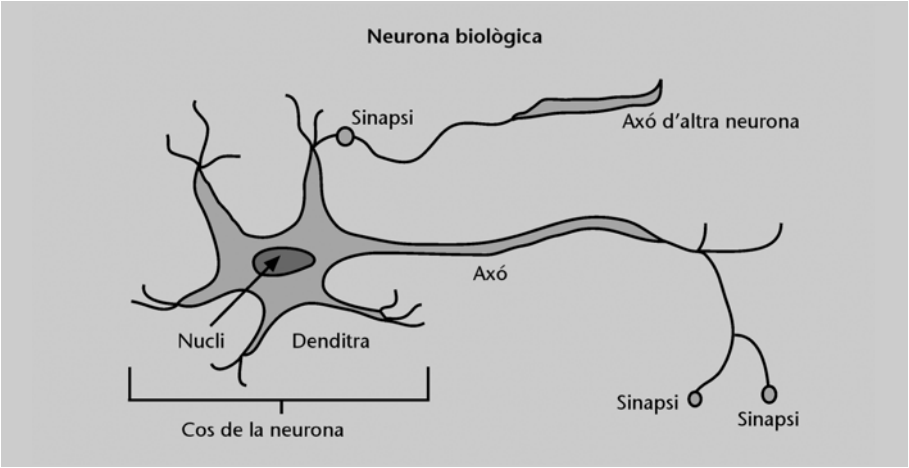
1. Conocer los fundamentos del funcionamiento de las redes neuronales y sus áreas de aplicación.

2. Profundizar en la construcción y entrenamiento de redes *feed-forward*, así como conocer el algoritmo de propagación hacia atrás.

3. Evaluar las ventajas e inconvenientes de estos modelos.

1. ¿Qué son las redes neuronales?

La idea básica de las redes neuronales es conseguir un sistema de cálculo inspirado en la manera en que se conectan entre sí las diferentes neuronas del cerebro humano. Cada neurona tiene un cuerpo principal o **soma**, del que surgen ramificaciones o **dendritas**. Hay una extensión principal, **axón**, que se extiende a una distancia relativamente grande según el tamaño del cuerpo principal de la célula. El contacto de una neurona con otra se produce mediante las **sinapsis**. En éstas, pues, reside la capacidad de comunicación entre neuronas.



En efecto, por medio de complejas reacciones electroquímicas, a través de las dendritas se pasan sustancias químicas transmisoras que hacen aumentar o disminuir el potencial eléctrico de la célula. Cuando este potencial llega a un valor determinado, supera cierto umbral, se genera una señal eléctrica que se transmite a lo largo del axón, llega a las sinapsis y desencadena un proceso parecido en las otras neuronas conectadas.

Neuronas y procesamiento de información

Una neurona se activa o no en relación con las otras neuronas con que está conectada y cuando las señales provenientes de esas neuronas superan un determinado umbral de activación. En ese momento, la neurona altera la señal recibida y la envía hacia las otras neuronas con que se encuentra conectada.

Hay sinapsis excitatorias, que aumentan el potencial, y otras inhibitorias, que lo disminuyen. Lo más interesante, sin embargo, es que las conexiones tienen cierta plasticidad a lo largo del tiempo. Es decir, la fuerza de las conexiones va cambiando a lo largo del tiempo en respuesta a los estímulos recibidos, lo que parece ser la base de la capacidad de aprendizaje del cerebro.

Evidentemente, acabamos de hacer una simplificación extraordinaria, que capta, no obstante, el funcionamiento general de los diferentes sistemas de redes neuronales artificiales.

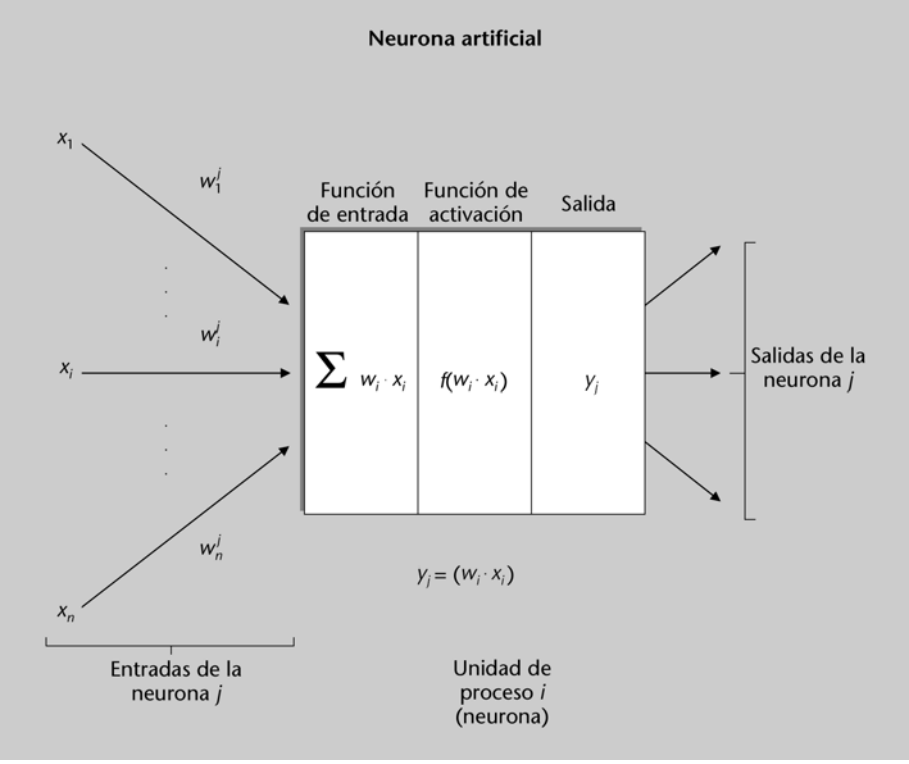
En un intento de imitar la capacidad de aprendizaje a largo plazo de las neuronas biológicas, una **red neuronal** consiste en la interconexión de un conjunto de unidades elementales, llamadas **neuronas**, y que no son más que procesadores muy primitivos.

Cada uno de estos procesadores aplica una función determinada a los valores de sus entradas procedentes de las conexiones con otras neuronas, y así se obtiene un valor o conjunto de valores nuevos que se convierten en la salida (o salidas). Este valor calculado se envía a las otras neuronas que se encuentran comunicadas con una determinada.

Así pues, las entradas desempeñan el papel de las conexiones sinápticas que llegan a una neurona y las salidas, el de las conexiones sinápticas que salen del axón.

El cuerpo de la neurona es la unidad de procesamiento que aplica una función a las entradas. La importancia de la información recibida o transmitida a partir de las conexiones o hacia las conexiones queda reflejada mediante el uso de varios factores o pesos para cada conexión.

A continuación, presentamos el esquema de una neurona artificial:



En la figura anterior tenemos que interpretar la expresión w_i^j *como el peso o la importancia del valor* X_i *que llega a la neurona j procedente de la neurona i.*

***W* proviene del inglés *weight*, 'peso' en castellano.**

Cada neurona combina los valores de entrada, aplicando sobre ellos una **función de activación** que modula el valor de entrada para generar otro valor de salida que se propaga a las conexiones de la neurona *j* con otras neuronas.

Finalmente, también es posible que las salidas puedan utilizarse de nuevo como entradas, yendo más allá del flujo unidireccional de entrada hacia la salida que se asocia con el modelo básico de red neuronal, dando lugar a las redes recurrentes.

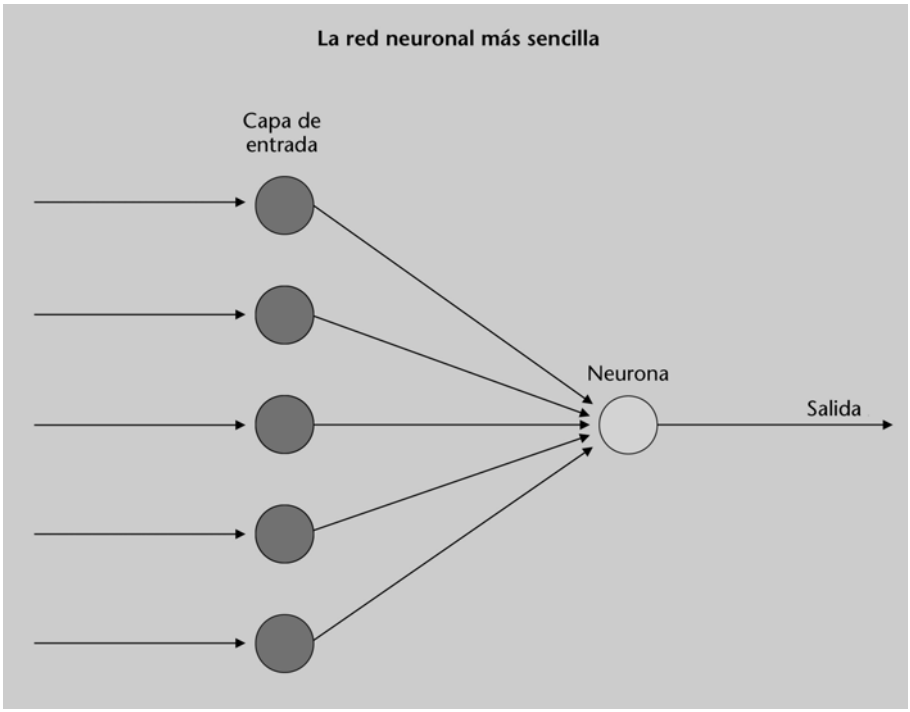
1.1. Topologías

Las diferentes **topologías** de una red neuronal; es decir, la manera de organizar las distintas unidades en capas diferentes, así como las distintas funciones de activación que pueden utilizarse ofrecen diferentes arquitecturas de redes neuronales que difieren entre sí en cuanto a los tipos de problemas a los que pueden aplicarse y en cuanto a la calidad de los resultados que se pueden obtener.

Una arquitectura...

... de red neuronal es su topología global más las funciones de activación utilizadas en cada neurona.

Una de las redes más simples que se pueden dar es la que presentamos en la figura siguiente:



En esta red se genera una única salida a partir de cinco entradas. En principio, una neurona de este tipo sólo permite calcular funciones relativamente sencillas y consigue efectuar una tarea equivalente a la técnica estadística conocida como *regresión no lineal*.

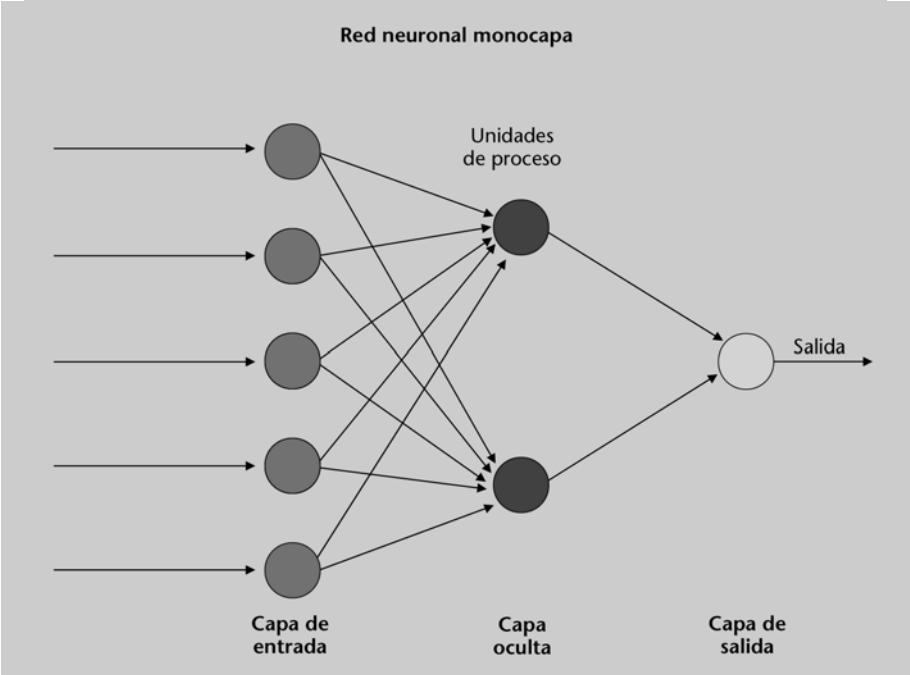
Topologías

Las neuronas pueden conectarse de varias maneras organizándose en capas.

Una red neuronal suele estar formada por una o más capas ocultas.

Un segundo nivel de complejidad es el que representan las **redes monocapa**, en las que aparece una serie de unidades de procesamiento que forman lo que se conoce como la **capa oculta**. Las tareas que permite efectuar una red de este

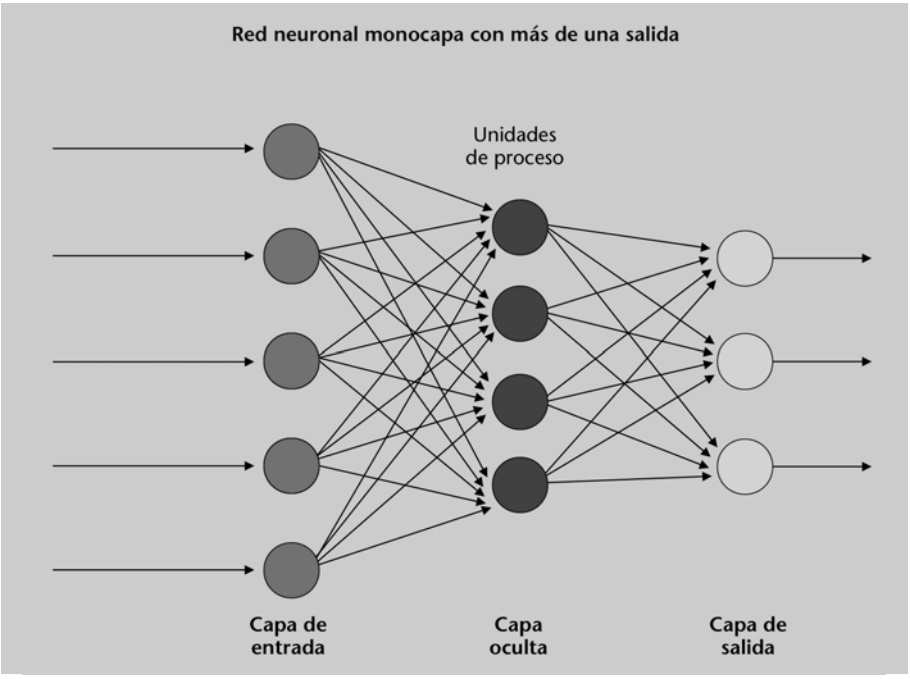
tipo son más complejas: pueden clasificar patrones de entrada más complicados o realizar predicciones sobre dominios de dimensionalidad más alta.



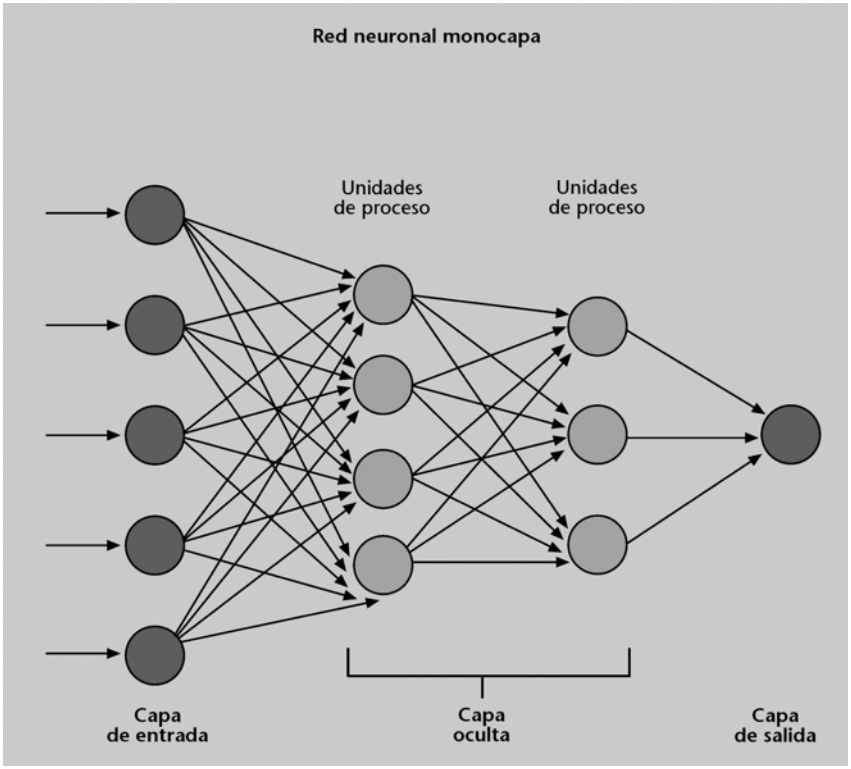
La capa oculta puede tener un número más elevado de unidades de procesamiento, hecho que garantiza el procesamiento de patrones todavía más elaborados y un mejor comportamiento de clasificación. Todo esto, en definitiva, mejora la separabilidad entre clases; el problema es que aumenta el riesgo de la sobreespecialización.*

*** En inglés, *overfitting*.**

En las topologías anteriores hemos descrito "capas de salida" formadas por una única neurona. Evidentemente, el número de neuronas en la capa de salida puede ser mayor:



Las topologías se pueden complicar introduciendo más capas ocultas:



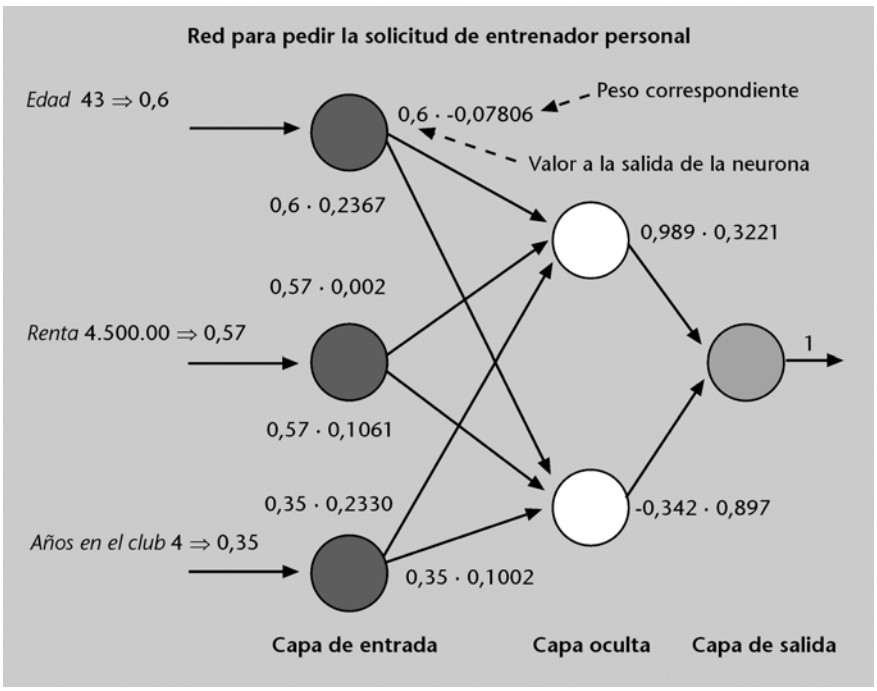
Propiedades de las redes

En función de la topología y las funciones de activación utilizadas, las redes neuronales tienen más o menos poder predictivo, calidad de clasificación y capacidad de separación. También varía su tendencia a la sobreespecialización.

Aunque las redes neuronales han sido aplicadas a tareas de diversa índole, aquí nos centraremos en su presentación de la aplicación a la tarea de clasificación.

Ejemplo de red monocapa para el caso de Hyper-Gym

Aquí tenemos como ejemplo una red monocapa con una salida para el caso de Hyper-Gym. La tarea que realiza es predecir si un cliente solicitará el servicio de entrenador personal en función de su edad, su renta y los años que lleva en el club. Fijaos en que todos los valores que utilizamos son numéricos. La salida tiene dos valores posibles 0 (no lo solicitará) o 1 (sí lo solicitará). Los pesos son ficticios.



Observad que todos los valores que se utilizan son números reales entre 0 y 1. Este hecho implica que todos los valores de la base de datos deben convertirse en numéricos y, después, ser normalizados, a fin de asegurar que quedan entre 0 y 1. Otros tipos de neuronas actúan con valores entre –1 y 1.

Lectura complementaria

Encontraréis más información sobre las redes de Hopfield y las máquinas de Boltzmann en la obra siguiente:

D.E. Rumelhart; J.L. McLelland (ed.) (1986). *Parallel Distributed Processing*. MIT Press.

A estas clases de topologías todavía pueden añadirse muchas más, por ejemplo, las que reutilizan todas las salidas de una capa o una parte para la capa anterior (redes recurrentes). Las más típicas son las redes de Hopfield, que tienen conexiones simétricas, y las máquinas de Boltzmann.

1.1.1. Funcionamiento global de una red neuronal

A continuación, estudiaremos cómo funciona una unidad mínima, o neurona, y qué funciones de activación se utilizan con más asiduidad.

Podemos expresar las ideas que hay detrás de las redes neuronales de manera matemática, dándonos cuenta de que se trata de obtener una función global que, a partir de un conjunto de valores de entrada, $x_1, \ldots, x_n$ nos calcule un valor de salida y o un conjunto de valores de salida. De momento, para simplificar, supondremos que el valor de salida es único.

$$y = f(x_1,\ldots,x_n)$$

En esta expresión, f representa la red neuronal.

Una red neuronal...

... puede considerarse una función en la que los pesos de cada enlace son los coeficientes sobre cada entrada, incluidas las que conectan las neuronas ocultas.

Apreciad que una red neuronal no representa una función fija, es decir, no siempre es la misma. Por el contrario, precisamente el interés de las redes es que esta función es adaptable, según los datos de entrenamiento que se le suministran (las $x_1, \ldots, x_n$) y los mecanismos de aprendizaje y ajuste que la misma red incorpora.

Visto de este modo, resulta interesante darse cuenta de que las redes nos permiten efectuar una operación de predicción básica: a partir de los valores $x_1, \ldots, x_n$ observados, podemos obtener un valor y hasta ahora desconocido. Cuando hacemos que y sólo tome un rango limitado de valores discretos, también podemos clasificarlo. Un conjunto determinado de valores $x_1, \ldots, x_n$ estará asociado con el valor de y que corresponda a la clase 1, o a la 2, etc.

Los problemas de los métodos de predicción y clasificación están también presentes en las redes neuronales. Veremos cómo se pueden resolver a fuerza de diseñar detalladamente los distintos aspectos de su construcción.

Funcionamiento de una neurona

Como hemos dicho, las redes neuronales corresponden a la interconexión de varias unidades de procesamiento elementales o *neuronas*.

Cada **neurona** combina los valores procedentes de sus conexiones de entrada en una o más conexiones de salida. La **función de activación** realiza la combinación.

1.2. Funciones de activación

Toda **función** puede descomponerse en dos partes:

a) La **función de combinación.** A partir de todos los valores de entrada calcula un único valor.

b) La **función de activación** propiamente dicha. Adopta el valor calculado y lo pasa a la conexión de salida.

Función de activación

Resulta de la aplicación de la función de combinación y la de transferencia.

1.2.1. Funciones de combinación

Como hemos afirmado al principio, cada conexión de entrada tiene un peso determinado que refleja su importancia o estado. El valor que llega por medio de la conexión debe combinarse con su peso y el de todas las conexiones de entrada para dar un único valor.

Para un conjunto de n conexiones de entrada en el que cada una tiene un peso w_i, las funciones siguientes son las más utilizadas para combinar los valores de entrada.

1) La función suma ponderada: $\sum_{1}^{n} x_i \cdot w_i^j$

2) La función máximo: $max(w_1^j, ..., w_n^j)$

3) La función mínimo: $min(w_1^j, ..., w_n^j)$

Unidades lógicas

Con los umbrales adecuados y pesos unitarios de entrada, las neuronas con función escalón pueden servir para implementar las funciones lógicas AND, OR y NOT.

Comprobadlo con umbrales $\alpha = 1{,}5$, $\alpha = 0{,}5$ y $\alpha = -0{,}5$.

4) La función AND: aplicable cuando los valores de entrada son 0 ó 1. Es una implementación de la función lógica AND ($\wedge$), que da *falso* cuando algún valor de la entrada es falso. La asignación de valores numéricos a las constantes de verdad acostumbra a ser, típicamente, *verdadero* = 1 y *falso* = 0.

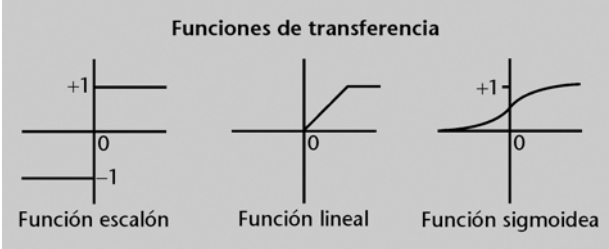
5) La función OR: aplicable cuando los valores de entrada son 0 ó 1. Es una implementación de la función lógica OR ($\vee$), que da *verdadero* cuando algún valor de la entrada es cierto. La asignación de valores numéricos a las constantes de verdad suele ser *verdadero* = 1 y *falso* = 0.

La utilización de una función u otra, en principio, está en relación con la aplicación final y con lo que deba aprender la red neuronal; sin embargo, en conjunto, la suma ponderada suele ser la más utilizada en la mayoría de las herramientas comerciales.

1.3. Funciones de transferencia

Las **funciones de transferencia** toman el valor calculado por la función de combinación y lo modifican antes de pasarlo a la salida.

Aquí tenemos las gráficas de algunas funciones de transferencia:



1) Función escalón:

$$salida(x) = \begin{cases} 1 \text{ si } x \geq \alpha \\ -1 \text{ si } x < \alpha \end{cases}$$

donde α es un valor umbral. Esta función limita los valores a 0 ó 1 y hace que la neurona actúe como un dispositivo puramente binario.

2) Función lineal:

$$salida(x) = \beta x$$

La salida es ahora una función lineal de la entrada y no está limitada a valores binarios. Esta función puede complicarse hasta ser una combinación lineal de las entradas.

3) Función sigmoidea:

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

Esta función admite y devuelve valores reales; sin embargo, en vez de ser un separador "fuerte", es un separador "suave" de las salidas. El valor de ρ influye en la forma de la curva. Los valores altos la aplanan y los bajos la acercan al escalón.

La función sigmoidea (podéis ver la gráfica en la figura anterior) tiene propiedades interesantes. Es una función no lineal; es decir, que para pequeñas variaciones en las entradas puede generar grandes variaciones en las salidas, con lo que refleja en buena medida el comportamiento de las neuronas biológicas. Observando el gráfico de la función podemos entender su comportamiento. Cuando los pesos de todas las entradas poseen valores pequeños, la función sigmoidea genera un valor igualmente pequeño. Dentro de este rango de valores, la función presenta un comportamiento casi lineal. A medida que los pe-

sos adoptan valores mayores, el valor resultante se aproxima al máximo o al mínimo. De manera que se pasa de un modelo lineal a uno no lineal.

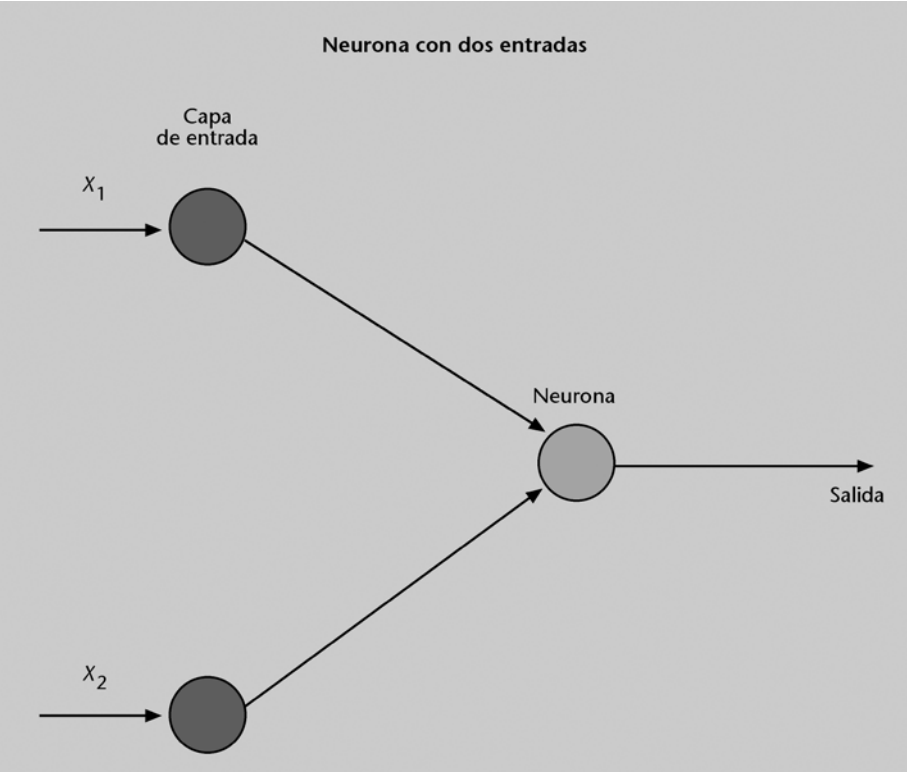
Estas funciones adoptan una importancia crucial en el comportamiento de la red. Cada una permite un determinado tipo de operación.

Dado que las funciones que se representan en caso de utilizar sigmoideas son no lineales, el conjunto de una red neuronal se comporta como una función no lineal compleja. Si los pesos que se aplican a las entradas se consideran los coeficientes de esta función, entonces el proceso de aprendizaje se convierte en el ajuste de los coeficientes de esta función no lineal, de manera que se aproxime a los datos que aparecen en el conjunto de entrada.

1.3.1. Propiedades de las redes según las funciones de transferencia utilizadas

Para entender las limitaciones de los diferentes tipos de redes según las funciones y topologías utilizadas, será preciso que estudiemos con más detalle su comportamiento, es decir, la tarea de clasificarlas o predecirlas. Utilizaremos términos geométricos para aclarar más los diferentes aspectos que hay que tener en cuenta.

Para simplificar las cosas, nos basaremos en la suposición de que tenemos una neurona (¡atención: *neurona*, y no *red neuronal*!) con sólo dos entradas, x_1 y x_2:



Podemos decir que la neurona actúa sobre un espacio de observaciones bidimensionales, puesto que, en realidad, las dos entradas representarán los valores de dos atributos. Para facilitar las cosas, caracterizaremos el comportamiento de la neurona mediante una tabla que para cada combinación de valores de entrada nos determine cuál es el valor con que se activará y qué salida dará. Por ejemplo:

X_1	X_2	Valor de activación	Salida
0	0	0	0
0	1	1	0
1	0	1	0
1	1	2	1

¿Qué función es ésta?

La función que hay detrás de esta tabla no es más que la tabla de verdad de la conjunción lógica convencional (AND) si interpreta que el 0 corresponde al valor *falso* y el 1, al valor *verdadero*.

El espacio de observaciones tiene tantas dimensiones como atributos se utilicen para describirlas. En el caso de Hyper-Gym, $n = 13$ y, por lo tanto, el espacio de observaciones tiene trece dimensiones, por lo que visualizarlo resulta un poco difícil.

La neurona clasifica las diferentes combinaciones de valores de entrada en dos clases: la clase 0 y la clase 1. El umbral, o valor de activación, indica bajo qué condiciones debemos asignar una observación a una clase o a otra.

Si suponemos que la entrada se combina según:

$$\sum_{1}^{2} w_i x_i = w_1 x_1 + w_2 x_2 \quad (1)$$

mediante cierta manipulación algebraica podremos ver qué tipo de función en el espacio bidimensional estamos determinando. Igualándola al umbral activación:

$$\sum_{1}^{2} w_i x_i = w_1 x_1 + w_2 x_2 = \alpha \quad (2)$$

vamos a ver qué línea nos determina:

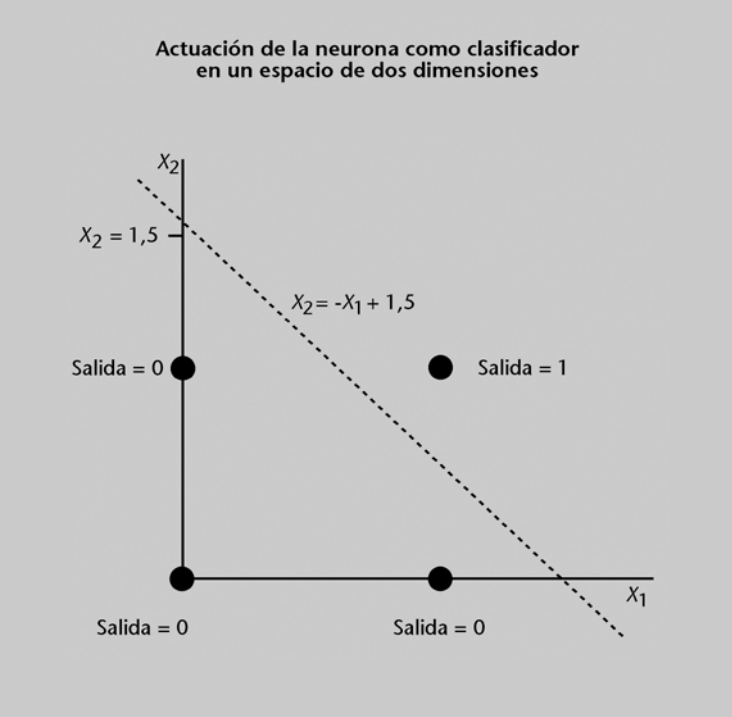
$$w_1 x_1 + w_2 x_2 = \alpha$$

$$x_2 = -\left(\frac{w_1}{w_2}\right) x_1 + \left(\frac{\alpha}{w_2}\right) \quad (3)$$

O sea, la ecuación de una recta con pendiente α y que corta el eje x_2 en el punto $\beta = \frac{w_1}{w_2}$. En nuestro ejemplo tenemos que los dos pesos son iguales a 1, por lo tanto la pendiente es –1 y el punto de intersección con el eje x_2, 1,5:

$$x_2 = \alpha x_1 + \beta w_2 \quad (4)$$

Gráficamente:



Los puntos que se encuentran a la izquierda y por debajo de la línea pertenecen a la clase 0, y los que están por encima y a la derecha, a la clase 1.

Así pues, todo esto indica que para esta función (definida por la tabla) la neurona actúa como **separador lineal** entre clases. Es decir, sólo puede clasificar correctamente conjuntos de observaciones que estén separados en el espacio bidimensional por una recta del tipo que hemos descrito en la ecuación (4).

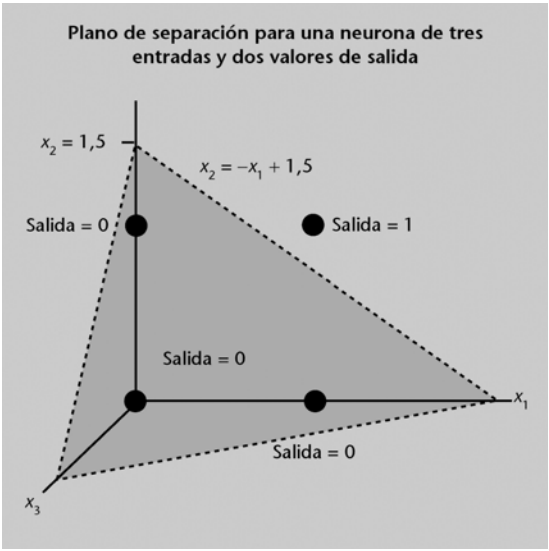
Una neurona simple es un separador lineal...

... que permite clasificar observaciones dentro de regiones linealmente separables.

Si la neurona tuviera tres entradas, x_1, x_2, x_3, la ecuación resultante definiría un plano en el espacio de tres dimensiones. En un espacio de *n*-dimensiones, al resolver la ecuación:

$$\sum_{1}^{n} w_i x_i = w_1 x_1 + \ldots + w_n x_n = \alpha \quad (5)$$

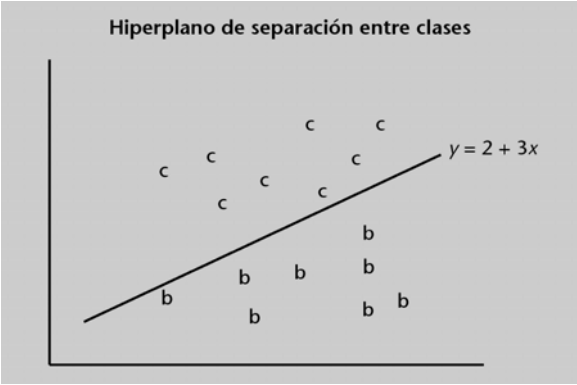
obtendríamos lo que se conoce como un **hiperplano** en el espacio *n*-dimensional; si queréis, obtendríamos una manera de recortar este espacio.



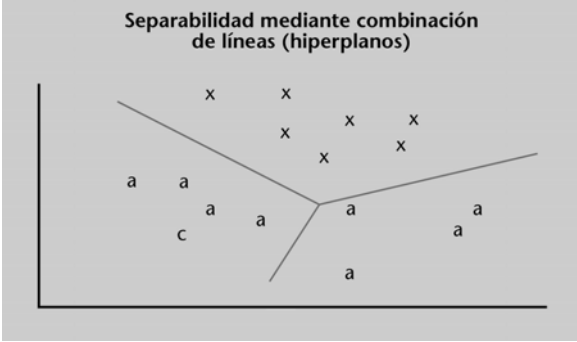
El problema de ciertas redes neuronales sencillas (como el perceptrón, basado en una función escalón) es que sólo pueden clasificar de forma correcta aquellos conjuntos de observaciones que son linealmente separables; es decir, que tienen las observaciones ubicadas dentro de los hiperplanos definidos por ecuaciones parecidas a las que resultan de resolver ecuaciones como la (5).

Ejemplo en dos dimensiones de clases separables y no separables linealmente

Aquí podemos ver gráficamente el problema de la separabilidad o no separabilidad de clases en un caso de dos dimensiones:



Estas dos clases pueden separarse mediante una línea que tiene una expresión en forma de ecuación lineal. En un espacio tridimensional tendríamos un plano expresado como combinación lineal de las otras dos variables; en un espacio *n*-dimensional, como una combinación lineal de $n - 1$ variables.



En esta figura tenemos una muestra gráfica del problema de la separabilidad no lineal. En el caso de las funciones lineales, esto explica que las neuronas simples no puedan realizar clasificaciones a veces muy fáciles de obtener. Este hecho fue remarcado por Minsky en un artículo que tuvo un efecto demoledor y que, como ya hemos indicado, condenó la investigación en el campo de las redes neuronales al ostracismo durante muchos años.

2. El perceptrón

El perceptrón es una estructura propuesta por Rosenblatt en 1962 que está formada por varias unidades A (*association units*) o entradas que se conectan a neuronas que utilizan una función escalón.

El perceptrón

Es una conjunto de neuronas que utilizan funciones escalón como funciones de activación, conectadas a una unidad de salida.

¿Cómo aprende un perceptrón? Podemos decir que el aprendizaje de cada neurona de una estructura de este tipo persigue dos objetivos:

- Ajustar el conjunto de valores de entrada w_1, ..., w_n.
- Ajustar el valor de activación α.

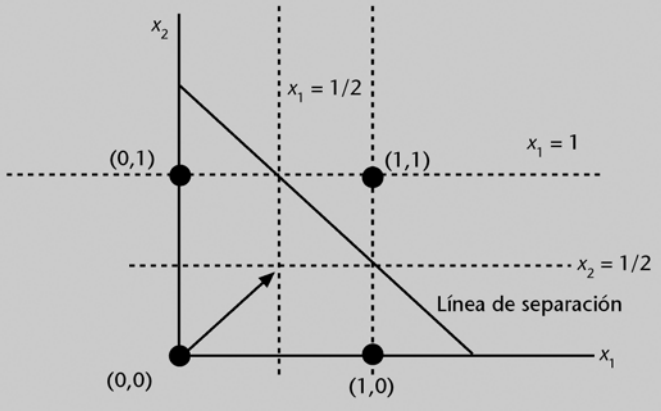
Se pretende conseguir que las observaciones sean clasificadas correctamente. Por lo tanto, en la fase de entrenamiento supervisado de una red neuronal sencilla como ésta, lo que se le suministra a la red son pares (observación, clase) de ejemplos correctos.

Si tenemos n entradas x_1, ..., x_n, y como posibles clases, la 0 y la 1, podemos recurrir a la notación vectorial para simplificar las expresiones y decir que tenemos un vector n-dimensional de entradas $\mathbf{x}$ y un vector n-dimensional de pesos $\mathbf{w}$. El efecto de la multiplicación de los valores de entrada por los pesos correspondientes se expresa como el producto $\mathbf{wx}$ y el comportamiento de la unidad neuronal, como:

$$y = 0 \text{ si } \mathbf{wx} \geq 0$$
$$y = 1 \text{ si } \mathbf{wx} < 0$$

Haciendo $\mathbf{wx} = 0$, definimos el hiperplano de separación entre las dos clases:

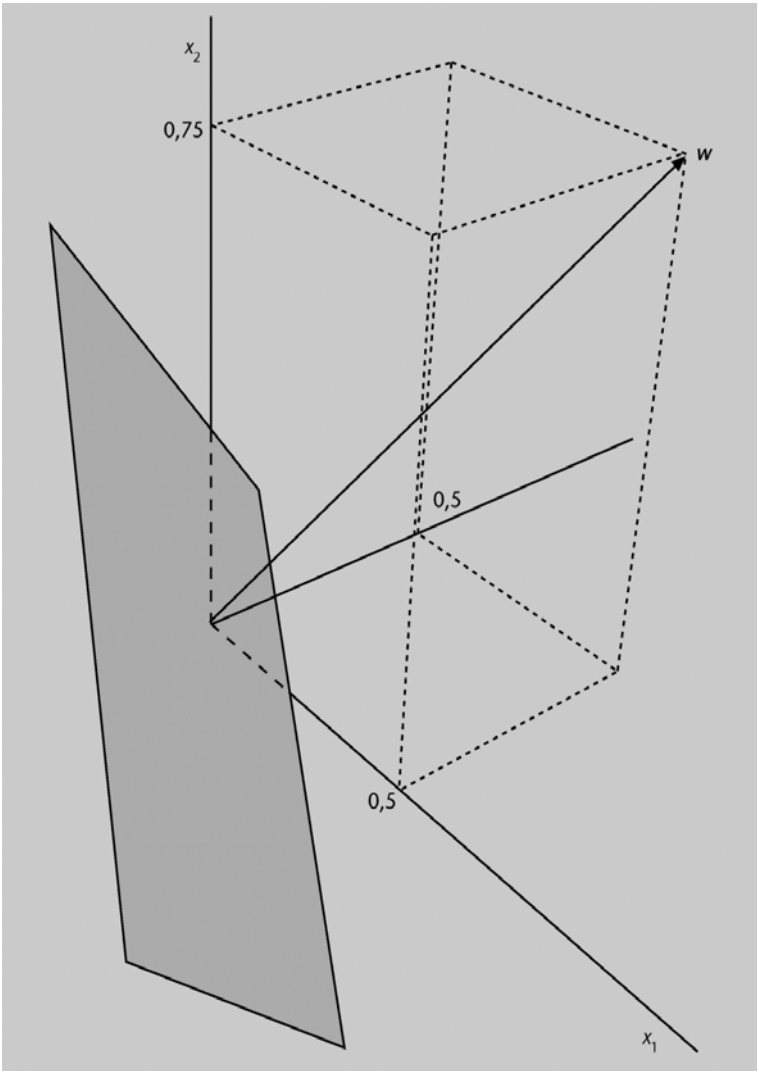
Hiperplano de separación entre clases: caso 2D



De hecho, estamos considerando el umbral de cada unidad como un peso más. En consecuencia, trabajamos con un vector de pesos extendido. De esta manera, para generar el valor que corresponde a la clase 1 pediremos que el valor de activación (la combinación lineal de valores de entrada) sea positivo o igual a cero, y para obtener el valor 0 deberemos hacer que el valor de activación sea negativo.

Las propiedades del producto de vectores nos muestran que el vector de pesos **w** es ortogonal al hiperplano de separación de clases y que este último pasa por el origen del espacio de observaciones:

¿Qué relación hay entre el vector de pesos y el plano de separación?



En la figura hemos mostrado el plano de separación con un vector de pesos **w** = (0,5, 0,5, 0,75) en el que la tercera entrada es la correspondiente al umbral.

Esta propiedad es importante: con el fin de garantizar una clasificación adecuada, es preciso que el vector de pesos sea ortogonal al hiperplano de separación entre clases. Si este factor no se cumpliera, entonces la clasificación no sería adecuada.

La propiedad de ortogonalidad es la clave

Comparando la ortogonalidad del vector de pesos con el hiperplano de separación podemos saber el margen de error que cometemos al clasificar, si es que cometemos alguno.

¿Qué sucede cuando la clasificación no es adecuada? Que podíamos suministrar a la red un ejemplo que recoge los valores de los diferentes atributos más

la clase que le corresponde y, en cambio, la unidad neuronal devolvería un valor incorrecto.

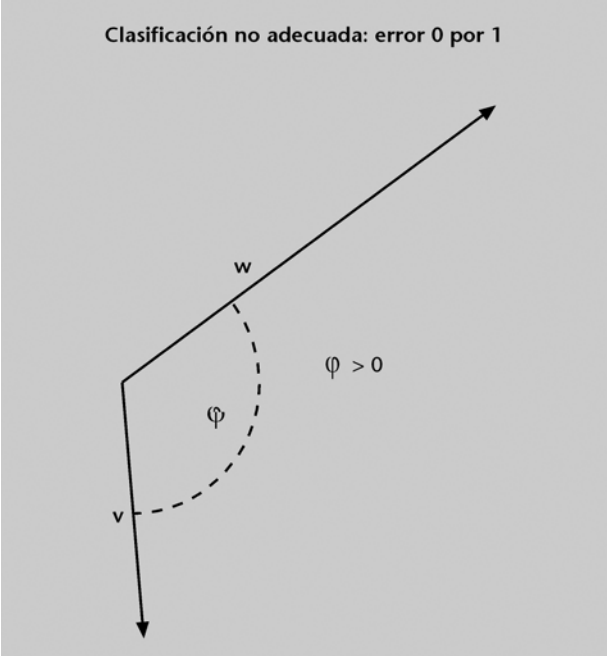
Para compensar y orientar esta situación de manera adecuada, tenemos que lograr que el vector **w** gire de manera que apunte hacia la dirección correcta. Gráficamente, es preciso que lo llevemos hacia la dirección del vector **v**:

Ejemplo de clasificación no adecuada

Un ejemplo de clasificación no adecuada es cuando presentamos el par (observación, clase):

$$\mathbf{v} = (x_1,\ldots,x_n,1)$$

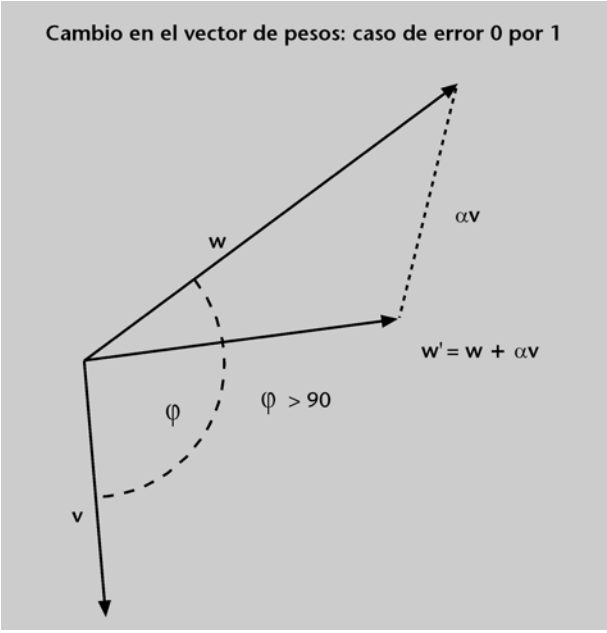
y nos da 0. Para conseguir que el valor de salida en este caso fuera el que le corresponde, el umbral de activación debería haber sido negativo en lugar de positivo.



El problema es que no podemos lograr la rotación de un ángulo tan grande que haga que los ejemplos vistos hasta el momento queden mal clasificados. Por lo tanto, hemos de conseguir modificar la dirección de **w** añadiéndole una fracción de **v**.

$$\mathbf{w}' = \mathbf{w} + \alpha\mathbf{v}$$

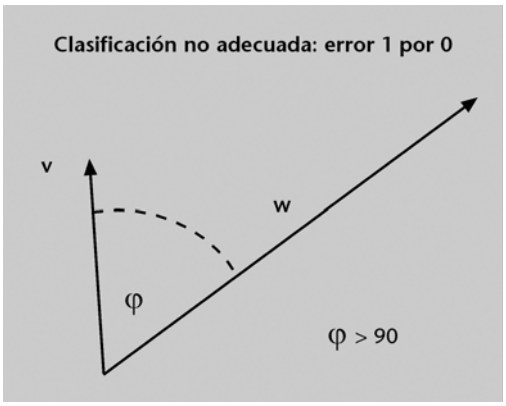
donde $0 < \alpha < 1$. De esta manera, nos queda:



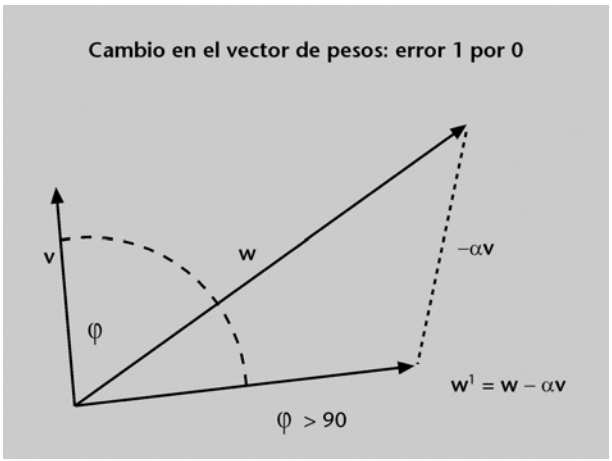
En el caso contrario, cuando la etiqueta de clase del ejemplo de entrenamiento es la clase 0, pero la neurona devuelve 1, nos encontramos con que el umbral de activación ha sido positivo cuando debía haber sido negativo. Por consiguiente, tenemos que rotar en sentido contrario. Ahora debemos hacer rotar **w** alejándolo de **v**. Es decir, le restamos una fracción de **v**:

$$\mathbf{w}' = \mathbf{w} - \alpha\mathbf{v}$$

Gráficamente, a partir de la figura siguiente:



hacemos rotar **w**, de manera que resulta:



En general,

$$\mathbf{w}' = \mathbf{w} + \alpha(c - y)\mathbf{v}$$

donde c es el valor que indica la clase en el ejemplo de entrenamiento e y, el valor de la función de salida.

Esta expresión nos da la clave para desarrollar un sencillísimo algoritmo de aprendizaje para este tipo de redes neuronales simples. Es preciso que expre-

semos individualmente para cada atributo, para cada entrada, la diferencia entre la clase que se le asigna al ejemplo y la que la neurona le ha asignado. Para cada componente del valor de pesos, y utilizando la última ecuación, tenemos lo que ha dado en llamarse la **regla delta**:

$$\Delta w_i = \alpha(c - y)v_i$$

donde α recibe el nombre de **tasa de aprendizaje** o **velocidad de aprendizaje**.

Los perceptrones aprenden utilizando la regla delta.

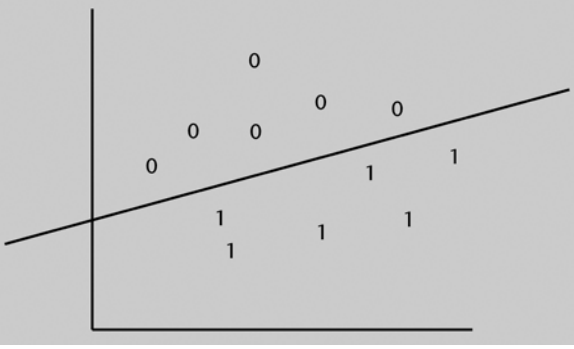
La regla delta puede utilizarse como base para adiestrar este tipo de redes neuronales. Aquí tenemos el algoritmo que resulta:

Repetir
  Para cada ejemplo (**v**,**c**) **hacer**
    Calcular el valor de y cuando la entrada es **v**
      Si y $\neq c$ **entonces**
    Construir un nuevo valor de pesos **w'** = **w'** + $\alpha(c - y)$**v**
    fsi
  fpara
Hasta que $y = c$ para todos los vectores de entrada

Como hemos dicho, para poder utilizar este tipo primitivo de red neuronal hay que asegurarse de que el conjunto de observaciones de entrada puede, efectivamente, subdividirse en clases que se pueden separar en regiones del espacio que se pueden describir por medio de ecuaciones lineales. En otros términos, debemos asegurarnos que las clases son linealmente separables. Si esta propiedad no se da, entonces el error de clasificación no se podrá compensar y siempre tendremos un elevado porcentaje de observaciones clasificadas de manera errónea.

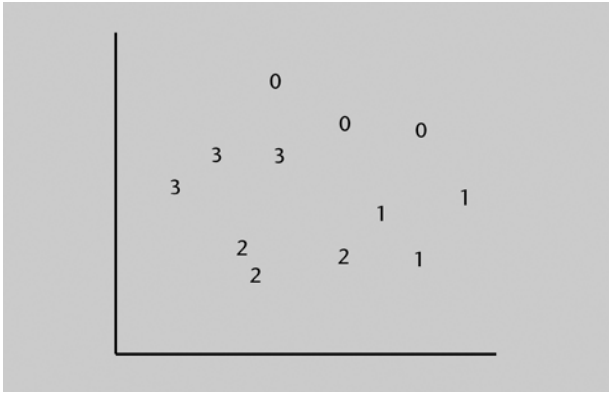
Vamos a ver gráficamente que quiere decir todo esto:

Ejemplo sencillo de clases linealmente separables

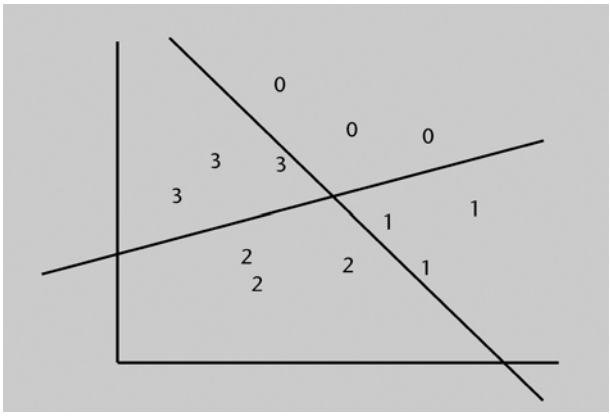


En el ejemplo mostrado, podemos ver que los objetos representados por 0 y los representados por 1 se pueden separar mediante una línea que admite una ecuación lineal para ser descrita. Si queréis, podemos expresarlo diciendo que la frontera entre las dos clases es lineal.

Pongamos un caso un poco más complejo: ¿son linealmente separables las clases que aparecen representadas en la figura siguiente?



Sí, aquí tenemos unas cuantas fronteras lineales que nos resuelven el problema.



Fijaos, no obstante, en que el grupo formado por los elementos de la clase 0 y de la clase 2 conjuntamente no se podrían separar de manera lineal del grupo formado por los elementos de las clases 1 y 3. ¿Qué red resolvería el problema de clasificar las observaciones de entrada en los grupos 0, 1, 2 y 3?

Ataquemos el problema empezado por la clasificación más básica. Si quisiéramos clasificar las diferentes combinaciones de grupos (0,1), (2,3), (0,2), (1,3) que son separables dos a dos, tendríamos suficiente con dos unidades con salidas y_1, y_2 de acuerdo con la tabla siguiente:

Valor de salida	1	0
y_1	(0,1)	(2,3)
y_2	(0,3)	(1,3)

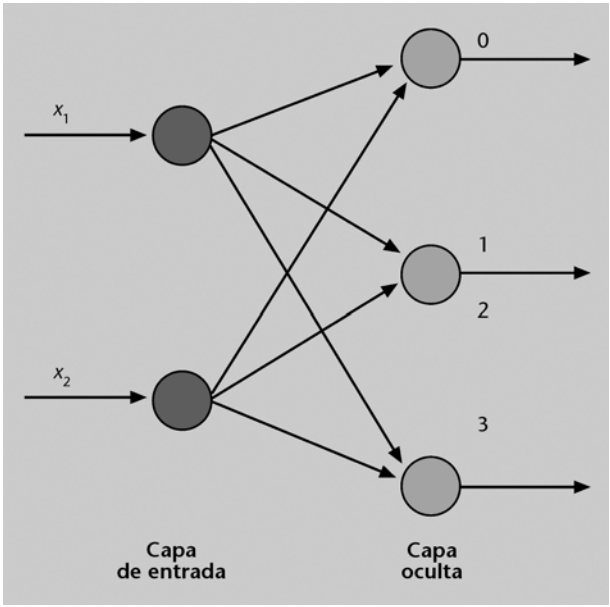
Es decir, la primera unidad, la que tiene por salida y_1, es capaz de separar las clases (0,1) y (2,3). Cuando observa un 0 ó un 1, le asigna a la clase (0,1) y lo

señala con el valor 1 para y_1. Cuando observa un 2 ó un 3, indica que la observación pertenece al grupo (2,3) generando el valor 0.

Esto nos da una primera capa de neuronas que realiza una separación "básica" de las observaciones distinguiéndolas en dos grandes bloques linealmente separables. No obstante, queremos conseguir que ante una observación podamos decir cuál es su clase individual, y no el grupo de clases; por lo tanto, debemos utilizar otra tabla basada en los valores de la salida:

y_1	y_2	Clase
0	0	0
0	1	1
1	0	2
1	1	3

lo que nos da esta estructura de red:



Las unidades de salida han de tener una asignación de pesos tal que su valor de salida sea 1, cuando aparezca la combinación de valores y_1, y_2 que pertenecen a la clase que les corresponde, y 0, en caso contrario.

Capas ocultas

Fijaos en que en la figura hemos introducido una capa oculta de neuronas entre la capa de entrada y la de salida.

Para adiestrar la red, sólo tenemos que actuar sobre la capa formada por las dos unidades con salidas y_1 e y_2. No es necesario que las unidades de salida que estén entrenadas: con la asignación de pesos indicada, si las otras unidades han sido correctamente entrenadas, clasificarán también de forma correcta.

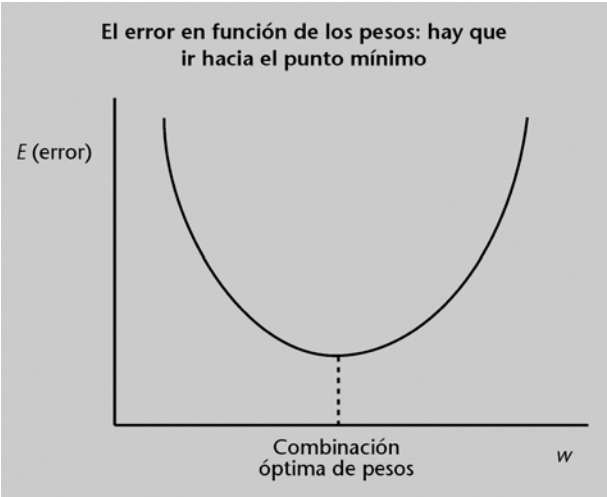
El problema es que para poder construir esta red debemos tener la seguridad de que los dos grupos formados por las clases que se suministran a la capa de dos unidades son linealmente separables. Nos interesa no tener que hacer estas suposiciones y poder adiestrar el conjunto de la red sin tener que preocuparnos por las características de los hiperplanos asociados a cada nodo.

Hay que insistir en el interés de desarrollar métodos de redes neuronales, y en general métodos de clasificación, que permitan definir cualquier tipo de superficie de separación entre clase, y no únicamente hiperplanos definidos por ecuaciones lineales.

Con el fin de llegar a esta manera de proceder, hay que desplazar la operación de corrección hacia los valores de activación, en lugar de hacerlo sobre los pesos. Nuestro objetivo final es la obtención de una red que minimice el error de clasificación.

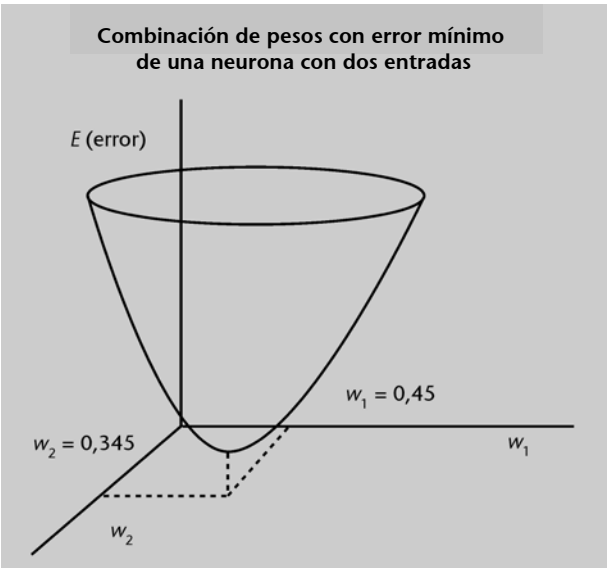
Por lo tanto, consideraremos el error como una función de los pesos e intentaremos llevar la red hacia una configuración de valores de activación que nos garantice que encontramos el mínimo de esta función.

En la figura siguiente vemos gráficamente el citado procedimiento:



Ejemplo de combinación de pesos con error mínimo

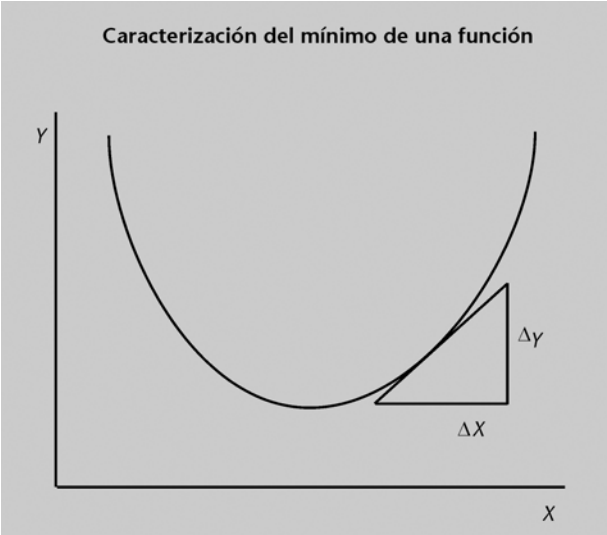
Aquí tenéis una representación para una neurona con dos entradas x_1, x_2 y sus pesos correspondientes w_1 y w_2. La combinación óptima parece estar en $w_1 = 0{,}45$ y $w_2 = 0{,}345$.



Para encontrar el punto mínimo de la función de error, repasamos algunas de las propiedades de las funciones.

¿Cómo podemos encontrar este punto mínimo de la función de error?

Supongamos que tenemos una función $y = f(x)$ y que, aunque no conocemos con exactitud su forma, podemos calcular su pendiente en cualquier punto. No olvidemos que el objetivo es encontrar el mínimo de esta función. ¿Qué caracteriza al mínimo? Observemos la figura siguiente:



Quizá recordemos que la pendiente de la función en un mínimo es 0 (también en un máximo, por cierto). Vamos a ver, sin embargo, cómo evoluciona la pendiente a medida que nos aproximamos hacia un mínimo y no hacia un máximo). La pendiente es la razón:

$$\frac{\Delta y}{\Delta x}$$

En otras palabras, la pendiente en un punto dado es el gradiente de la tangente a la curva de la función en aquel punto. Si Δx es pequeño, entonces Δy se aproxima a la derivada de la función. Podemos expresarlo así:

$$f'(x) = \delta y = \Delta y = \frac{\Delta y}{\Delta x}\Delta x$$

Se cumple que $\delta y < 0$ cuando nos aproximamos al mínimo. Para aproximarnos al mínimo, hay que encontrar un factor α que asegure que $\delta y < 0$; es decir, que encuentre el factor que hace que nos aproximemos hacia el mínimo, y que haga que el gradiente disminuya. Esta técnica se conoce como *descenso de gradiente* y puede extenderse a funciones de n variables. Vamos a ver cómo utilizarla para lograr que una red neuronal aprenda a clasificar.

Debemos aplicar el descenso de gradiente para asegurarnos que llegamos al mínimo de la función de error.

A cada observación *o* le asignamos un error, ε_o, que es función de los pesos que tienen las entradas a la unidad neuronal, $\varepsilon_o = f(w_1,\ldots,w_n)$. Hay varias; por ejemplo podríamos utilizar el error cuadrático:

$$\varepsilon_o = \frac{1}{2}(c-y)^2$$

donde *c* es el valor de clase que acompaña el ejemplo que suministramos a la red neuronal. Hay otras expresiones posibles del **error de clasificación** o **error de predicción**, como comentamos en otro módulo. Consideremos la salida de la unidad y como una función de los pesos de entrada. El error total que genera una unidad es la suma de los errores de todas las observaciones (mejor dicho, de los ejemplos) que le han presentado.

Podéis ver los errores de clasificación posibles en el módulo "Evaluación de modelos", de esta asignatura.

$$Error = \sum_O \varepsilon_o$$

donde *O* representa el conjunto de ejemplos de entrenamiento. Ahora bien, para poder aplicar la técnica de descenso de gradiente, es preciso que la función sobre la que la aplicamos sea una función continua. Consideraremos dos casos:

- La función de la unidad neuronal es la función escalón.
- La función de la unidad neuronal es la sigmoidea.

En el primer caso nos encontramos con una función que no es continua, y en el segundo, con una que lo es.

2.1. Descenso de gradiente con funciones no continuas (ADALINE)

Como ya hemos dicho, en caso de que la función de la unidad neuronal sea la función escalón, no tenemos una función continua y no estamos en condiciones de aplicar el descenso de gradiente. Una solución es considerar que hay que adiestrar la red teniendo en cuenta no las salidas, sino los valores de activación. Esta técnica, conocida como *elementos adaptativos lineales* (*Adaptive Linear Elements*, ADALINE), fue desarrollada en 1962 por Widrow y Hoff, y posteriormente mejorada.

Lecturas complementarias

Encontraréis información sobre la técnica ADALINE en las obras siguientes:

B. Widrow; S. Stearns (1985). *Adaptive Signal Processing*. Prentice Hall.

B. Widrow; J. Winter; T. Baxter (1987). "Learning Phenomena in Layered Neural Networks". *Proceedings of the First International Conference on Neural Nets* (vol. 2, pág. 411 y sig.).

En esta técnica hay que dar con el ejemplo no el valor de clase, sino el valor de activación que ha de tener el nodo para, de ese modo, conseguir una clasificación correcta. Conviene dar números de signo contrario, por ejemplo, –1, 1.

Entonces se puede efectuar un descenso de gradiente utilizando el error cuadrático otra vez, pero definido con respecto al valor de activación en lugar de respecto al de salida. Es decir:

$$\varepsilon_o = \frac{1}{2}(c - a)^2$$

donde a es el valor de activación (resultado de la combinación lineal de las entradas).

La pendiente de la función definida por ε_o con respecto al peso j-ésimo, w_j, es:

$$-(c - a)x_j$$

donde x_j es, recordémoslo, el valor del j-ésimo atributo del ejemplo de entrada $(x_1,...,x_n)$.

La regla de aprendizaje ahora intenta cambiar el peso según la expresión:

$$\Delta w_j = -\alpha(c - a)x_j$$

Entonces el algoritmo de aprendizaje quedaría de la manera siguiente:

Repetir
  Para cada ejemplo (**v**,c) **hacer**
    Calcular el valor de a cuando la entrada es **v**
    Si $a \neq c$ **entonces**
      Construir un nuevo valor de pesos **w**' utilizando $\mathbf{w}' = \mathbf{w} - a\ (c - a)\mathbf{v}$
    fsi
  fpara
Hasta que $y = a$ para todos los vectores de entrada

El problema, si adoptáramos el algoritmo tal como aparece aquí, es que el error casi nunca es igual a cero y, por lo tanto, no acabaríamos nunca de actualizar pesos.

2.2. Descenso de gradiente con funciones continuas: el caso de la sigmoidea

En el caso de que las unidades utilicen la función sigmoidea, podemos aplicar el método de descenso de gradiente, dado que ésta es una función continua. De nuevo, podemos utilizar el valor de la salida para calcular el error. Introducimos un término relacionado con la pendiente de la sigmoidea, su derivada, $\sigma'(a)$. Teniendo esto en cuenta, la regla delta queda ahora:

$$\Delta w_j = \alpha\sigma'(a)(c - y)x_j$$

No lo demostraremos ahora, pero se cumple la igualdad siguiente:

$$\sigma'(a) = \frac{1}{\sigma}\sigma(a)(1 - \sigma(a))$$

Esta regla de aprendizaje puede generalizarse a fin de poder adiestrar más de una capa de neuronas al mismo tiempo.

3. Redes con capas múltiples: retropropagación

En el caso de disponer de una red neuronal multicapa, no nos interesa tener que construir "a mano" la asignación de pesos de las capas ocultas. Nos interesa tener en cuenta el error total expresado valorando todos los pesos, tanto los de los nodos de salida como los de las capas ocultas. La diferencia entre nodos ocultos y nodos de salida es que en los primeros no podemos saber *a priori* cuáles son los valores de salida correctos (no tenemos ningún control), mientras que en los segundos, sí. Los únicos valores que pueden ayudarnos y que podemos utilizar para asociar a cada ejemplo son los de los nodos de salida. ¿Cómo podemos adiestrar también los nodos internos?

Suponiendo que tengamos nodos que utilizan la función sigmoidea, sabemos que para el nodo *j*-ésimo la expresión de la regla de la delta es:

$$\Delta w_i^j = \alpha\sigma'(a^j)(c^j - y^j)x_i^j$$

donde, recordémoslo:

a) $-(c^j - y^j)$ representa una medida del error que se produce en el nodo *j*-ésimo.

b) $\sigma'(a^j)$ indica el factor o la velocidad con que la función de ese nodo puede afectar al error (indica en realidad la pendiente de la función):

- Si ese factor es pequeño, nos encontramos muy cerca de los extremos de la S de la sigmoidea correspondiente y el cambio en el valor de la activación no afecta demasiado a la salida.

- En cambio, si es grande, nos hallamos en un lugar donde un pequeño cambio en la entrada genera un gran cambio en la salida.

c) Finalmente, x_i^j indica hasta qué punto la *i*-ésima entrada del *j*-ésimo nodo ha afectado al error. Si es 0, no podemos decir que la culpa del error corresponda a la entrada y no debemos tocar el peso (el cambio en el valor del peso debe ser 0). Si es grande, entonces sí podemos decirlo, y tenemos que cambiar el valor del peso proporcionalmente.

La delta que corresponde al nodo *j*-ésimo puede denotarse así:

$$\delta^j = \sigma'(a^j)(c^j - y^j)$$

Y, finalmente:

$$\Delta w_i^j = \alpha\delta^j x_i$$

El problema que aparece ahora con los nodos ocultos es determinar qué parte del error total se debe a cada uno de los nodos ocultos, cuánta responsabilidad tiene este nodo en el error que se ha producido. ¿Debemos actuar sobre los pesos de las entradas, o el error se debe a otro nodo o grupo de nodos? ¿Cómo efectuamos los cambios? ¿Qué proporción afecta a las tasas de aprendizaje? El problema, en definitiva, es encontrar la combinación adecuada de modificaciones sobre las tasas de aprendizaje, vectores de pesos, etc., del conjunto de nodos responsables que se parezca a la forma en que hemos aprendido a hacerlo para los nodos de salida.

No entraremos aquí a hacer una derivación matemática completa de la regla de retropropagación. Sólo comentaremos las razones que explican su formulación matemática. Vayamos por partes:

Lectura recomendada

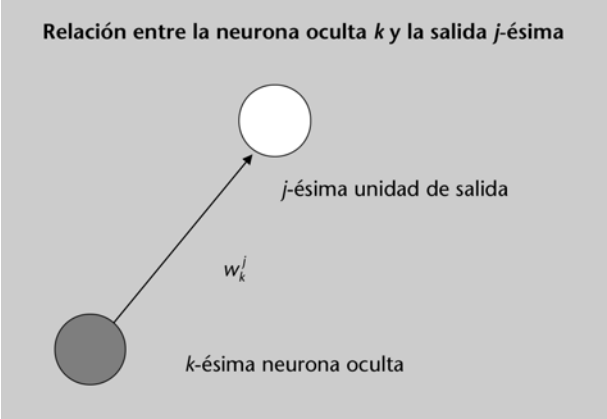
Encontraréis una derivación matemática completa de la regla de retropropagación en la obra siguiente:

D.E. Rumelhart; J.L. McLelland (ed.) (1986). *Parallel Distributed Processing*. MIT Press.

1) La contribución del valor *i*-ésimo de entrada a la neurona *k*-ésima oculta será:

$$\Delta w_i^j = \alpha \delta^K x_i$$

Vamos a ver cómo se calcula δ^k. Centrémonos en la relación que existe entre la neurona oculta *k*-ésima y la unidad de salida *j*-ésima.



El **efecto en el error del nodo oculto** depende de dos cosas: su influencia sobre la salida del nodo *j*, y la proporción en la que la salida del nodo *j* afecta al error por medio del mismo.

Se supone que cuanto más afecte el nodo *k* al nodo *j*, mayor será el efecto esperado sobre el error. Ahora bien, surge aquí cierto "razonamiento hacia atrás" o "efecto de retorno" de *j* sobre *k*, pues esta influencia de *k* sobre *j* sólo tendrá importancia si el nodo *j* afecta al error de su salida (lo cual tiene que ver, por ejemplo, con la pendiente de su función). La influencia de *j* sobre el error es recogida por la delta que corresponde a este nodo, δ^j. La influencia del nodo *k* sobre el nodo *j* queda mediatizada por el peso w_k^j. Ahora bien, las salidas del

nodo oculto k pueden estar conectadas a más de un nodo de la capa de salida, por lo tanto, deberemos tener en cuenta las contribuciones al error de cada una. Así pues, hay que sumar los productos $w_k^j \delta^j$ sobre todos los nodos de salida (todos los valores que puede adoptar j). Asimismo, deberemos tener en cuenta la influencia de la velocidad de aprendizaje del nodo oculto k, $\sigma'(a^k)$. Por lo tanto, la delta que corresponde al nodo oculto k se calcula según la expresión:

$$\delta^k = \sigma'(a^k) \sum_{j \in S_k} \delta^k w_k^j$$

donde S_k son los nodos de salida a los que está conectado el nodo oculto k.

En consecuencia, para calcular los cambios que hay que introducir en los pesos de las entradas del nodo oculto k, tenemos la expresión:

$$\Delta w_i^j = \alpha \sigma'(a^k) \sum_{j \in S_k} \delta^k w_k^j x_i$$

que puede parecer muy complicada, pero detrás de ella subyace una idea relativamente sencilla de la influencia entre nodos ocultos y nodos de salida, y entre el error de salida y los nodos precedentes.

2) Para calcular los ajustes por medio de las diferentes capas de neuronas, en primer lugar debemos hacer un proceso hacia adelante, propagando la influencia de las entradas por medio de los nodos ocultos, y después hacia atrás, para reconsiderar los cambios en los pesos originados a causa de los nodos de salida a los cuales están conectados los elementos de las capas ocultas. Ésta es la base del algoritmo de entrenamiento por retropropagación*. El esquema general es el siguiente:

*** En inglés, *backpropagation*.**

a) Propagación hacia adelante. Introducir el ejemplo y obtener las salidas.

b) Propagación hacia atrás. Introducir en las unidades de salida los valores correctos y propagar hacia atrás para calcular las deltas de los nodos ocultos.

Con un poco más de detalle la secuencia es la siguiente:

1) Introducir el ejemplo en la capa de entrada.

2) Hacer que las unidades ocultas calculen las salidas que corresponden al ejemplo.

3) Hacer que las unidades de salida calculen su salida según lo que reciben del paso anterior.

4) Introducir el valor de clasificación correcta en las unidades de salida.

5) Calcular las deltas de los nodos de salida mediante la expresión:

$$\delta^j = \sigma'(a^j)(c^j - y^j)$$

6) Adiestrar cada nodo de salida siguiendo el método de descenso de gradiente con la fórmula siguiente:

$$\Delta w_i^k = \alpha\sigma'(a^k) \sum_{j \in S_k} \delta^k w_k^j x_i$$

7) Calcular la delta de cada nodo oculto mediante la expresión:

$$\Delta w_i^k = \alpha\delta^k x_i$$

8) Para cada nodo oculto, utilizar la δ calculada en el paso anterior para entrenar la red según el método de descenso de gradiente.

3.1. Velocidad de aprendizaje

La **velocidad de aprendizaje** viene determinada por el parámetro α. El valor que tenga este parámetro posee una gran influencia sobre la manera en que evoluciona el proceso de aprendizaje. Si se hace crecer mucho, el proceso se vuelve inestable, lo cual significa que la red alterna entre configuraciones que se aproximan al mínimo, pero no se acaba de estabilizar en torno a éste.

Recordemos que el esquema general de los métodos de construcción de redes neuronales es el siguiente:

Repetir
  Para cada ejemplo de entrenamiento
    Entrenar la red con el ejemplo
  fpara
Hasta que el error sea aceptable

Recordemos también que su topología presenta varias capas diferenciadas: la capa de entrada, una o más capas ocultas, y la capa de salida. Este hecho plantea una serie de cuestiones en cuanto a la construcción de estas redes:

Podéis ver las diferentes topologías de las redes neuronales en el subapartado 1.1 de este módulo.

1) "*Error aceptable*". La medida de error en predicción se adopta sobre la desviación entre la salida real y la predicha, como ya hemos comentado. Por nor-

Podéis ver la medida de error en predicción en el subapartado 1.5.1 del presente módulo.

ma general, cuesta mucho menos hacer que una red se estabilice en torno a un error de 0,1 que sobre un error de 0,0000001. No siempre se puede asegurar que esta medida de error signifique una medida proporcionalmente equivalente en la evaluación del modelo.

2) Dimensión de la capa de entrada. La dimensión de la capa de entrada se fija inicialmente en el número de atributos que se consideran. Las técnicas de selección de atributos permiten reducir el número de elementos de entrada sin perder una calidad excesiva en la capacidad de predicción del modelo.

3) Topología y dimensión de las capas ocultas. El problema de encontrar la topología general óptima de la red no es trivial. Incluso en este problema se encuentra el dimensionamiento de la capa o las capas ocultas. Por desgracia, existe una cierta tendencia a tomar estas dos decisiones por el método de "prueba y error".

Si bien las dimensiones de las capas de entrada y de salida vienen dadas en gran medida por las características del problema (número de atributos de entrada y números de clases de salida), no hay una regla tan clara para delimitar el número de nodos que tienen que conformar la o las capas ocultas.

Lectura complementaria

Podéis encontrar más información sobre la sobreespecialización en la obra siguiente:

T. Krushke (1988). "Creating Local and Distributed Bottleencks in Hidden Layers of Back-Propagation Networks". En: D. Touretsky; G. Hinton; T. Sejnowski (ed.). *Proceedings of the 1988 Connectionist Models Summers Schools*. Morgan Kaufmann.

Cuantos más nodos compongan, más observaciones diferentes puede reconocer y clasificar la red. Pero también aumenta el peligro de sobreajuste (*overfitting*). En el peor de los casos, podemos acabar cubriendo exactamente las observaciones del conjunto de entrenamiento.

Por una parte, las redes con menos nodos obligan, en general, a aprender menos pesos, a fijar menos parámetros y, por lo tanto, reducen el riesgo de sobreespecialización.

Por otra parte, contar con más unidades ocultas puede tener sus ventajas. En efecto:

Lecturas complementarias

Podéis ver las dos ventajas mencionadas sobre el hecho de tener muchas unidades ocultas en las obras siguientes:

D.E. Rumelhart; J.L. McLelland (ed.) (1986). *Parallel Distributed Processing*. MIT Press.

D.C. Plaut; G.E. Hinton (1987). "Learning Sets of Filters Using Backpropagation". *Computer Speech and Language* (vol. 2, pág. 35-61).

a) Hay menos posibilidades de caer en un mínimo local; es decir, no poder salir de una configuración de pesos que parece óptima, pero sólo lo es en una zona del espacio de búsqueda (Rumelhart, 1986).

b) El tiempo necesario de aprendizaje varía inversamente con el número de unidades ocultas (Plaut, 1987).

Una regla obtenida a partir de la experiencia nos dice que un tamaño lo suficientemente bueno es el que resulta de poner un número de nodos en las capas ocultas equivalente a unas dos veces el número de nodos de entrada. Sin embargo, ésta no es una regla infalible. De hecho, es más razonable empezar con un número casi igual al de nodos de entrada.

- Si resulta que la red se está sobreajustando, entonces hay que ir reduciendo progresivamente el número de nodos en la capa oculta.

- Si, por el contrario, la evaluación del modelo indica que la precisión obtenida no es suficiente, quizá sea necesario aumentar el número de nodos.

Cuando se clasifica, empezar con un nodo oculto por cada posible clase suele dar buenos resultados. Hay métodos interesantes para descubrir automáticamente la topología óptima.

Lectura complementaria

Podéis encontrar métodos para descubrir la topología óptima en la obra siguiente:

M. Mezard; J.P. Nadal (1989). "Learning in Feed-Forward, Multilayered Networks; the Tiling Algorithm". *Journal of Physics* (núm. 22, pág. 2.191-2.204).

4) Establecimiento de la tasa de aprendizaje y el momento. Ya hemos visto la importancia que tienen estos dos factores en todo el proceso de entrenamiento de la red. Hay que proceder de manera progresiva, estableciendo primero valores altos para la tasa de aprendizaje, para después reducirla con el fin de poder ajustar la red. En lo referente al momento, contribuye a lograr que la red converja con mayor rapidez hacia una solución estable.

5) Análisis de sensibilidad. Una de las dificultades existentes a la hora de trabajar con redes neuronales es que cuesta mucho saber lo que ocurre durante el proceso de aprendizaje; es decir, es muy difícil dar una interpretación para la evolución de los pesos de la red. El análisis de sensibilidad permite saber la importancia relativa de las diferentes entradas en los resultados de la red. Una posible manera de proceder es la siguiente:

a) Encontrar el valor medio de cada entrada.

b) Medir el valor que presentan las salidas de la red cuando las entradas tienen el valor medio.

c) Medir las salidas cuando se altera cada uno de los valores de entrada.

Para el primer punto debemos tener en cuenta que como los valores de entrada no tienen que seguir una distribución uniforme, el valor medio no tiene por qué ser 0,5. El tercer punto es potencialmente conflictivo, en especial respecto al grado de variación. ¿Cuánto hay que incrementar o decrementar los valores de entrada? Otro aspecto importante es el de probar conjuntamente la modificación de varios valores de entrada, algo que puede ser especialmente importante si sospechamos que algunos subgrupos de valores se encuentran bastante asociados entre sí.

3.2. Problemas de la fase de entrenamiento

Por norma, la construcción de una red se realiza sobre un conjunto de datos de entrenamiento, y su prueba o evaluación, sobre otro conjunto diferente, como acostumbra a hacerse en la mayoría de los problemas de *data mining*.

Por lo tanto, la fase de la preparación de los datos de prueba comparte alguna de las dificultades con otros métodos de clasificación:

a) Cobertura. Hay que elegir un conjunto de datos tales que para cada atributo posible aparezcan todos sus valores. Este requerimiento todavía es más necesario en el caso de las redes.

b) Número de atributos. Saber qué atributos son realmente importantes corresponde al proceso de preparación de datos, en concreto a la selección y reducción de atributos, y puede afrontarse con árboles de decisión.

Podéis ver los árboles de decisión en el módulo "Clasificación: árboles de decisión" de esta asignatura.

3.3. Preparación de datos

En el módulo correspondiente comentamos las técnicas más importantes para preparar los datos en el formato adecuado. En el caso de las redes neuronales, el problema es básicamente que todos los valores se sitúan entre el 0 y el 1, y es preciso transformar valores tanto numéricos como categóricos, discretos o continuos. Este hecho no es especialmente novedoso, y ya hemos comentado las técnicas principales al respecto. Pero sí es nuevo en el caso de las redes la importancia de este paso, que puede tener una gran influencia en el rendimiento y la calidad de la red resultante.

Podéis consultar al respecto el módulo "Preparación de datos" de esta asignatura.

3.4. Interpretación de resultados

Como ya hemos dicho, las redes poseen una salida numérica. En el caso de que se apliquen estrictamente para clasificación, la interpretación de la salida es relativamente sencilla: el valor indica la clase a la que pertenece el objeto clasificado.

Ahora bien, como también hemos señalado antes, las redes no sólo ocultan nodos, sino también conocimiento. El "conocimiento" que contiene una red consta de las relaciones entre nodos y el peso con que se comunica la influencia de un nodo a otro, lo cual carece de traducción sencilla en términos simbólicos.

Conocimiento oculto

En el caso de las redes no contamos con una traducción fácil de la estructura de la red y de los valores de sus pesos en un formalismo más comprensible como, por ejemplo, una regla de clasificación simbólica.

Ya hemos apuntado que el análisis de sensibilidad permite hacerse una idea sobre qué valores de entrada tienen más influencia al obtener determinadas salidas, pero la relación entre una y otra no es tan sencilla como la que se establece, por ejemplo, en una ecuación de regresión lineal.

Finalmente, si las redes se utilizan para efectuar predicción de valores numéricos y no sólo de clases, es necesario un proceso de descodificación de los valores numéricos obtenidos a fin de que éstos sean más comprensibles.

La necesidad del proceso de descodificación

Si queremos predecir el valor de crédito que puede pedirnos un cliente en el futuro, debemos saber, a partir de los valores mínimo y máximo de estas cantidades, a qué equivale la salida de la red. Un 0 o un valor bajo tiene que corresponder al valor mínimo (pongamos un crédito de un millón), y un 1 o un valor alto, al valor máximo (pongamos cien millones). Ahora bien, según la distribución de los valores de los créditos, una salida de 0,5 no necesariamente equivale al valor medio. Nos encontramos, pues, en la situación inversa a la discretización de la que hemos hablado al presentar la preparación de datos.

Las redes que dan resultados numéricos continuos también presentan dificultades de interpretación en el caso de los valores categóricos binarios. En este caso, debemos prever un procedimiento de discretización: cualquier valor que se encuentre por debajo de 0,5 correspondería a la primera categoría, y por encima, a la segunda.

La alternativa consiste en crear una red con dos nodos: uno para la primera categoría o clase y otro para la segunda. Hay que adiestrar la red para que la salida que corresponde al primer nodo sea 1 en presencia de ejemplos de esta categoría y 0, para la otra (y viceversa). Entonces la interpretación es muy sencilla; el problema es que no siempre se consigue obtener un valor tan preciso.

Por último, presentamos un resumen de los pasos que hay que efectuar para construir una red neuronal:

a) Decidir el número de salidas de la red.
b) Decidir el número de atributos de entrada de la red.
c) Proponer una primera topología interna para la red.
d) Preparar los datos.
e) Seleccionar el conjunto de entrenamiento entre los datos.
f) Entrenar la red.
g) Evaluar la red sobre un conjunto de datos diferente del de entrenamiento.
h) En caso de que la red no cumpla los requerimientos de calidad satisfactoria, reconsiderar los pasos anteriores.

4. Ponderación final de las redes neuronales

Empecemos mencionando las ventajas que ofrecen las redes neuronales:

a) Se pueden utilizar en clasificación y predicción numérica y, con las adiciones correspondientes y métodos de interpretación, en la predicción de valores categóricos.

b) Son capaces de enfrentarse con problemas de alta dimensionalidad y con regiones de clasificación delimitados por hiperplanos no lineales.

c) Pueden mezclar el tratamiento de datos numéricos y categóricos.

Pasemos a dejar constancia de los inconvenientes:

a) En cuanto a la preparación de datos de entrada y salida. Hay que asegurarse de que todas las entradas se mapean mediante valores reales situados entre 0 y 1. Ya hemos comentado los problemas que puede comportar este tipo de transformaciones y cómo, en general, afectan al proceso de aprendizaje al que sirven. Debemos asegurarnos de que la transformación mantiene propiedades de distribución adecuadas, etc.

b) Las redes neuronales "ocultan" el conocimiento. Son poco comprensibles en primera instancia. Ya hemos comentado que su estructura y su combinación de pesos son las que guardan el "conocimiento" extraído a partir de los datos. Esta expresión de conocimiento no es fácilmente interpretable según conceptos simbólicos y es preciso transmitir el conocimiento conseguido a usuarios humanos. El análisis de sensibilidad ayuda a hacerse una idea, pero no facilita la traducción a un lenguaje más abstracto. No obstante, se han desarrollado técnicas para "traducir" la topología de las redes a esquemas simbólicos, como listas de reglas de clasificación.

Lectura complementaria

Encontraréis información acerca de las listas de reglas de clasificación en la obra siguiente:

S. Sestito; T.S. Dillon (1994). *Automated Knowledge Acquisition*. Prentice Hall.

Resumen

Las redes neuronales son métodos de representación basados en la analogía con el sistema nervioso. Se fundamentan en la distribución de cálculo entre unidades muy sencillas y la adaptabilidad que surge por la interconexión de varias capas de nodos o unidades neuronales básicas.

Cada unidad recibe entradas, las combina y las transforma en información de salida mediante una función. Esta información de salida se propaga hacia otras unidades. Típicamente, esta estructura interconectada se organiza en capas: una de entrada, una de salida y una o más internas u ocultas.

La función más utilizada es la sigmoidea. El **perceptrón** es la estructura de red más sencilla en que las unidades utilizan funciones escalón. El conjunto de la red puede entenderse como una función no lineal de sus entradas, que hay que ajustar para que alcance unos valores de salida determinados con el mínimo error. Esta función puede servir para predecir valores numéricos o de clase.

El perceptrón se limita a problemas de clasificación sobre clases linealmente separables. Las **redes multicapa** pueden enfrentarse a conjuntos no linealmente separables.

El uso de las redes exige una fase de entrenamiento para llegar a estabilizar los pesos de las conexiones en torno a la región de mínimo error.

El algoritmo más utilizado en redes multicapa es el **método de retropropagación**.

El principal inconveniente de este formalismo reside en la dificultad de comprensión del modelo resultante y su capacidad de explicación. 

Actividades

1. Para el problema propuesto en la actividad 1 del módulo "Extracción de conocimiento a partir de datos" de esta asignatura, ¿os sirven las redes neuronales? ¿Qué método creéis que os resultaría más conveniente?

Bibliografía

Beale, R.; Hilger, T. (1990). *Neural Computing, an Introduction*. Bristol.

Bigus, J. (1996). *Data Mining With Neural Networks*. McGraw-Hill.

Hinton, G.E.; Williams, R.J. (1986). "Learning Internal Representations by Error Propagation". En: D.E. Rumelhart; J.L. McLelland (eds.). *Parallel Distributed Processing*. MIT Press.

Hopfield, J.J. (1982). "Neurons with Graded Response Have Collective Computational Properties like those of Two-State Neurons". *Proceedings of the National Academy of Sciences* (núm. 79, págs. 2.254-2.258).

Kohonen, T. (1989). *Self-Organization and Associative Memory* (3ª ed.). Berlín: Springer-Verlag.

Krushke, T. (1988). "Creating Local and Distributed Bottleencks in Hidden Layers of Back-Propagation Networks". En: D. Touretsky; G. Hinton; T. Sejnowski (eds.). *Proceedings of the 1988 Connectionist Models Summers Schools*. Morgan Kaufmann.

McCulloch, W.S.; Pitts, W. (1943). "A Logical Calculus of Ideas Immanent in Nervous Systems". *Bulletin of Mathematical Biophysics* (núm. 5, págs. 115-137).

Mezard, M.; Nadal, J.P. (1989). "Learning in Feed-Forward, Multilayered Networks; the Tiling Algorithm". *Journal of Physics* (núm. 22, págs. 2191-2204).

Minsky, M.; Papert, S. (1969). *Perceptrons: an introduction to computational geometry*. MIT Press.

Plaut, D.C.; Hinton, G.E. (1987). "Learning Sets of Filters Using Backpropagation". *Computer Speech and Language* (vol. 2, págs. 35-61).

Rosenblatt, F. (1957). "The Perceptron: a Perceiving and Recognizing Automaton". Report 85-460-1. Ithaca: Cornell Aeronautical Laboratory.

Rumelhart, D.E.; McLelland, J.L. (eds.) (1986). *Parallel Distributed Processing*. MIT Press.

Sestito, S.; Dillon, T.S. (1994). *Automated Knowledge Acquisition*. Prentice Hall.

Widrow, B.; Stearns, S. (1985). *Adaptive Signal Processing*. Prentice Hall.

Widrow, B.; Winter, J.; Baxter, T. (1987). "Learning Phenomena in Layered Neural Networks". *Proceedings of the First International Conference on Neural Nets* (vol. 2, pág. 411 y sig.).