

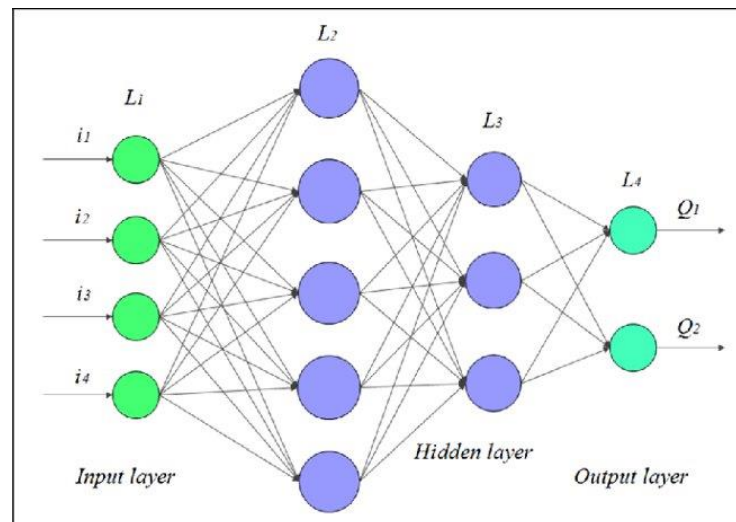
Conceptos base de las redes neuronales artificiales

Introducción

- Las redes neuronales artificiales con representaciones digitales de redes neuronales biológicas que están presentes en nuestro cerebro
- Estas redes intentan imitar su comportamiento, basándose en una regla empírica de "las neuronas que se activan juntas están juntas"
- Está matemáticamente demostrado que estas redes neuronales artificiales estructuralmente capaces de aproximar su comportamiento al de cualquier función multivariable lineal o no lineal

Representación

- En general, las redes neuronales artificiales se representan como grafos normalmente no cíclicos que suelen tener una aplicación biyectiva entre pares de conjuntos de nodos
- Existen diversas implementaciones de redes neuronales artificiales, pero las más comunes son las alimentadas hacia delante, en las cuales una capa es un conjunto de nodos, y están formadas por una capa de entrada a la izquierda, un conjunto de capas intermedias en medio (mínimo una), y una capa de salida
- En estas redes, los nodos de la capa de entrada no se consideran neuronas, pero en el resto de las capas sí

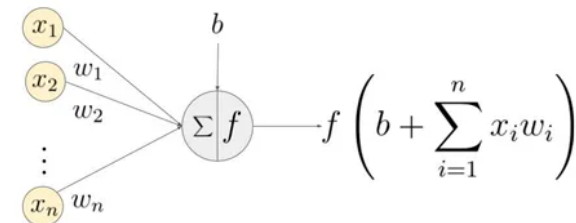
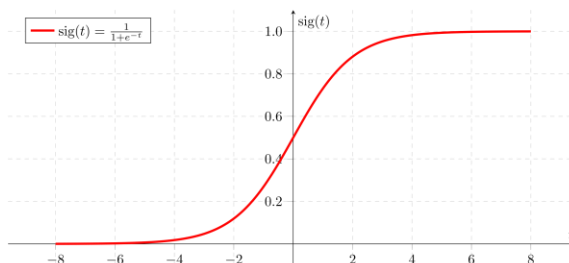


Neuronas

- Como mencionamos, excepto en la capa de entrada de una red neuronal artificial alimentada hacia delante, el resto de los nodos de las otras capas se consideran neuronas. El motivo es porque solamente reciben los valores de un ejemplo como entrada, sin modificarlo
- En el resto de capas, disponemos de las neuronas. Cada una de ellas tiene un valor asociado, llamado bias, que **al crear una red al principio, se suele inicializar con un valor de 0.**
- Aparte, existe una aplicación biyectiva entre cada par de capas, de derecha a izquierda, conectando todos los nodos entre sí por aristas. Cada arista tiene asociado un valor llamado peso, que **al crear una red al principio, se suele inicializar con un valor aleatorio**, que puede ser simplemente un valor entre un rango en concreto o un valor aleatorio en una distribución normal. Hay muchas técnicas de inicialización de estos valores aleatorios, pero la idea clave es que es un valor aleatorio. Si los valores de todos los pesos de la red fuesen iguales, esta nunca aprendería

Neuronas

- Estas neuronas siempre tendrán de una a **varias aristas de entrada, y una arista de salida**
- Aparte, cada neurona tendrá asociada una función de activación, la cual permitirá a la red aprender relaciones no lineales.
- ¿Cómo funcionan estas neuronas?: Primero reciben los valores de salida de sus neuronas/nodos anteriores, después calculan el sumatorio de multiplicar cada valor de salida de esas neuronas/nodos por el peso de la arista que conectan ambos nodos. Después a ese valor le suma el propio valor de bias de dicha neurona, recordemos que se inicializaba a 0 al crear la red. Finalmente, dicho valor se aplica a la función de activación. EL resultado será el valor de salida de dicha neurona al recibir el "impulso" de los nodos de su capa anterior
- Existen varias funciones de activación, algunas no sólo toman un valor...



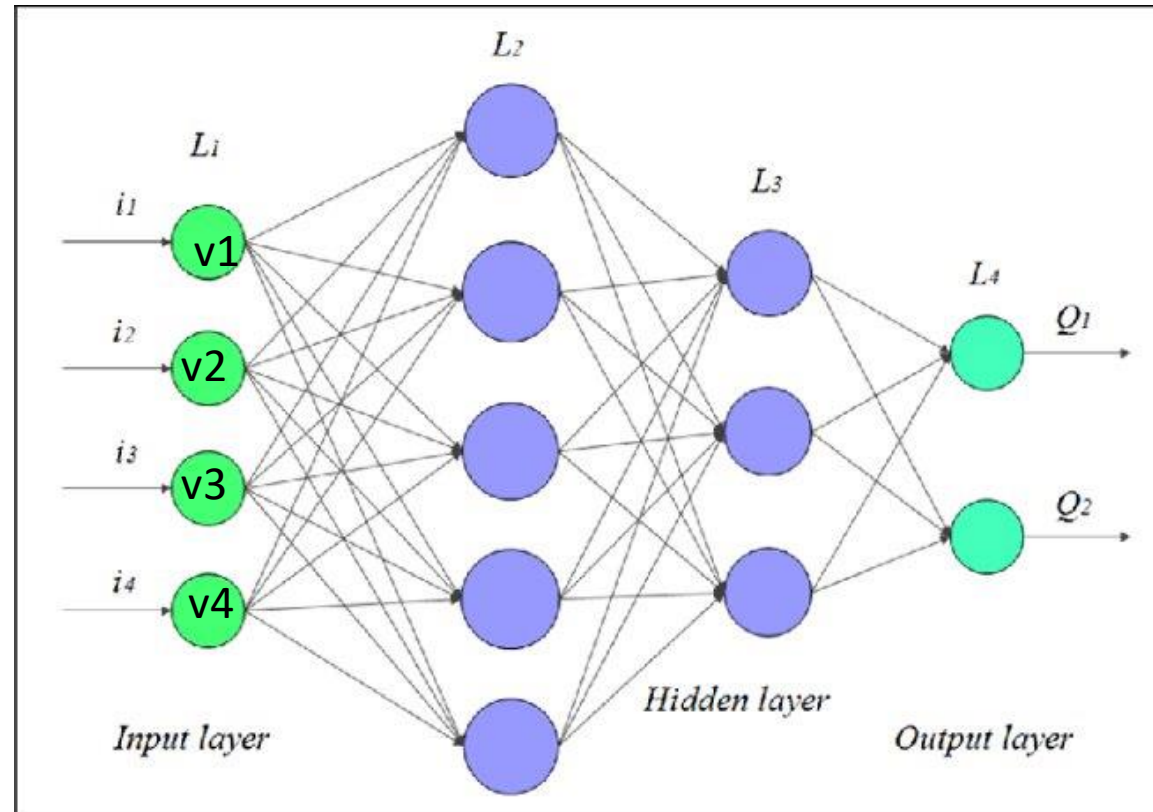
La red predice: propagación hacia delante

- Proceso en el cual se pasan unos valores de entrada a la red que se cargan en los nodos de entrada, y después se van activando las neuronas de las capas siguientes, de izquierda a derecha, hasta llegar a la capa de salida de la red
- Cuando la red recibe dicha entrada y hace la propagación hacia delante, esperamos que en las neuronas de la capa de salida se encuentren valores que se esperarían obtener como resultado del aprendizaje de la red, ya sea supervisado (tenemos idea de cómo deben ser los valores y esperamos que así sean) y no supervisado (los valores obtenidos estarán relacionados con los datos con los que se entrenó y podemos tener idea de cómo pueden ser, pero no sabemos exactamente cómo van a ser)
- Recordemos que al principio, los valores de los bias son 0, y los de los pesos, valores aleatorios, por lo que la red no hará ninguna buena predicción seguramente

La red predice: propagación hacia delante

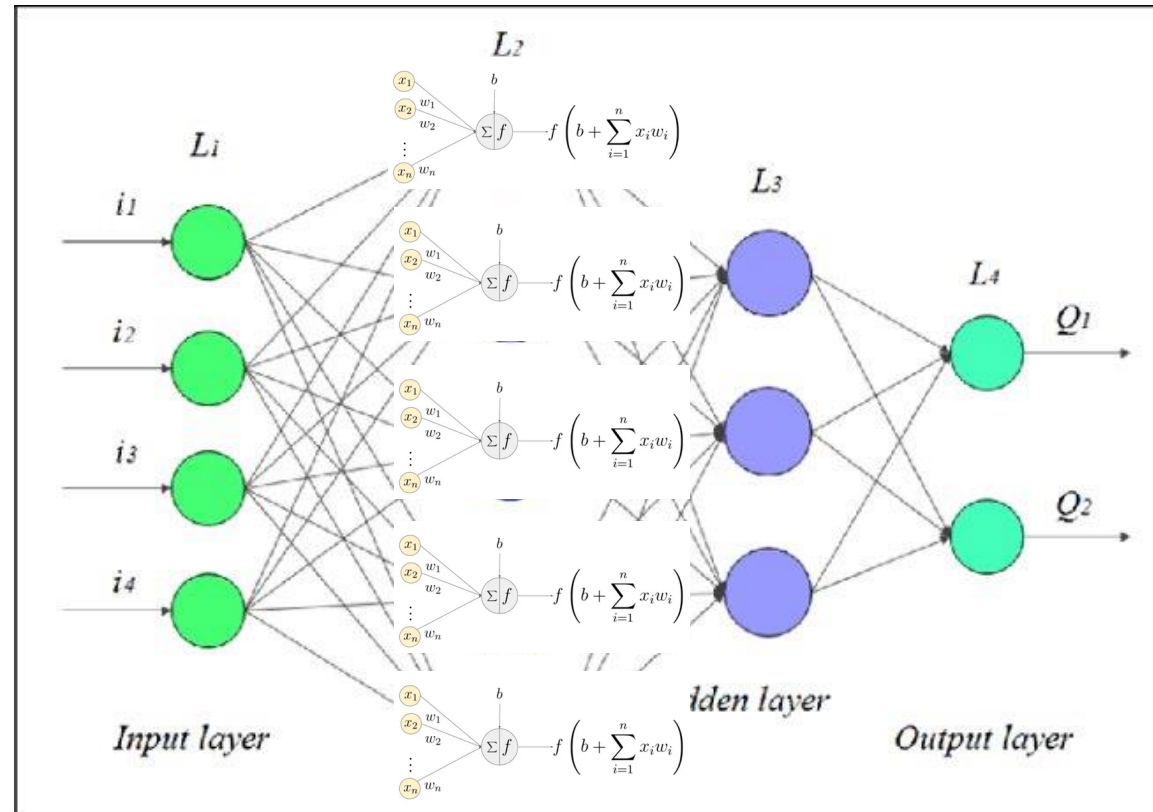
- Primero cargamos los valores de entrada en la capa de entrada y los propagamos hacia la capa L2

Ejemplo de conjunto de
entrenamiento casa
v1 = metros cuadrados
v2 = nº habitaciones
v3 = nº baños
v4 = precio



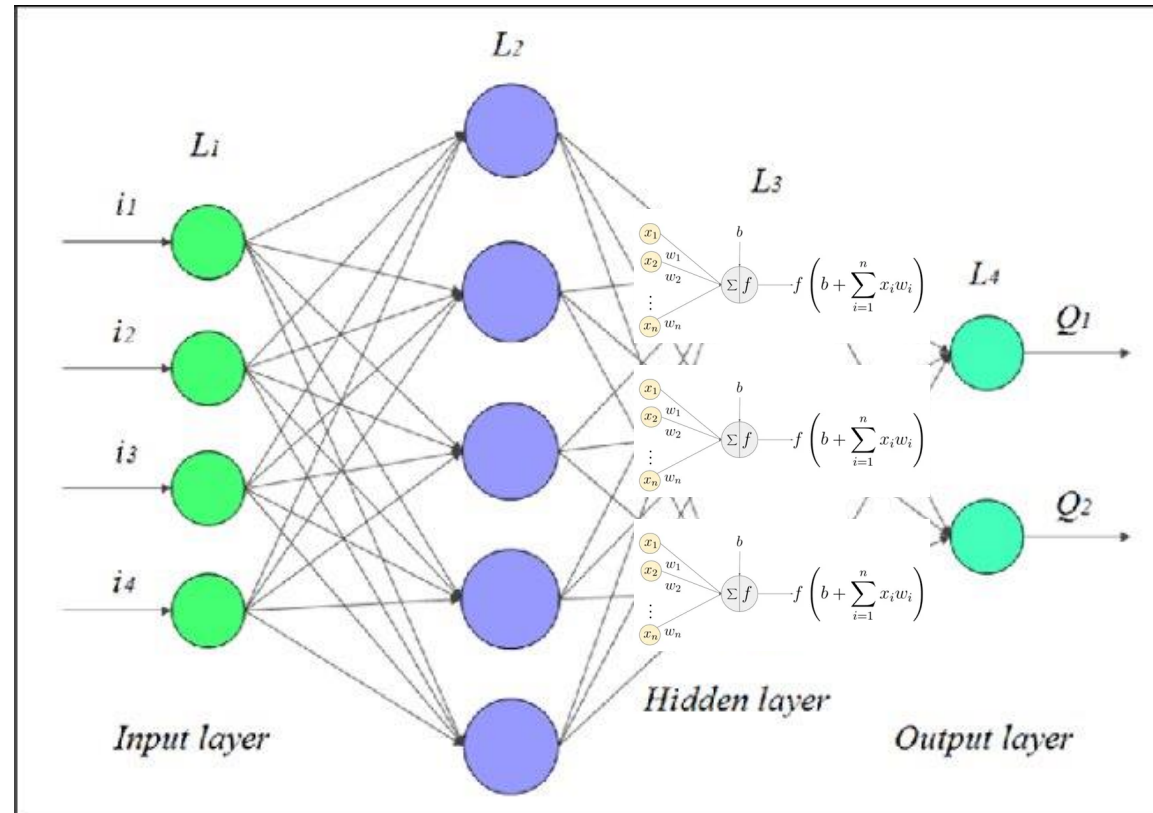
La red predice: propagación hacia delante

- Ahora calculamos el valor de salida de las neuronas de la capa L2, y las propagamos hacia L3



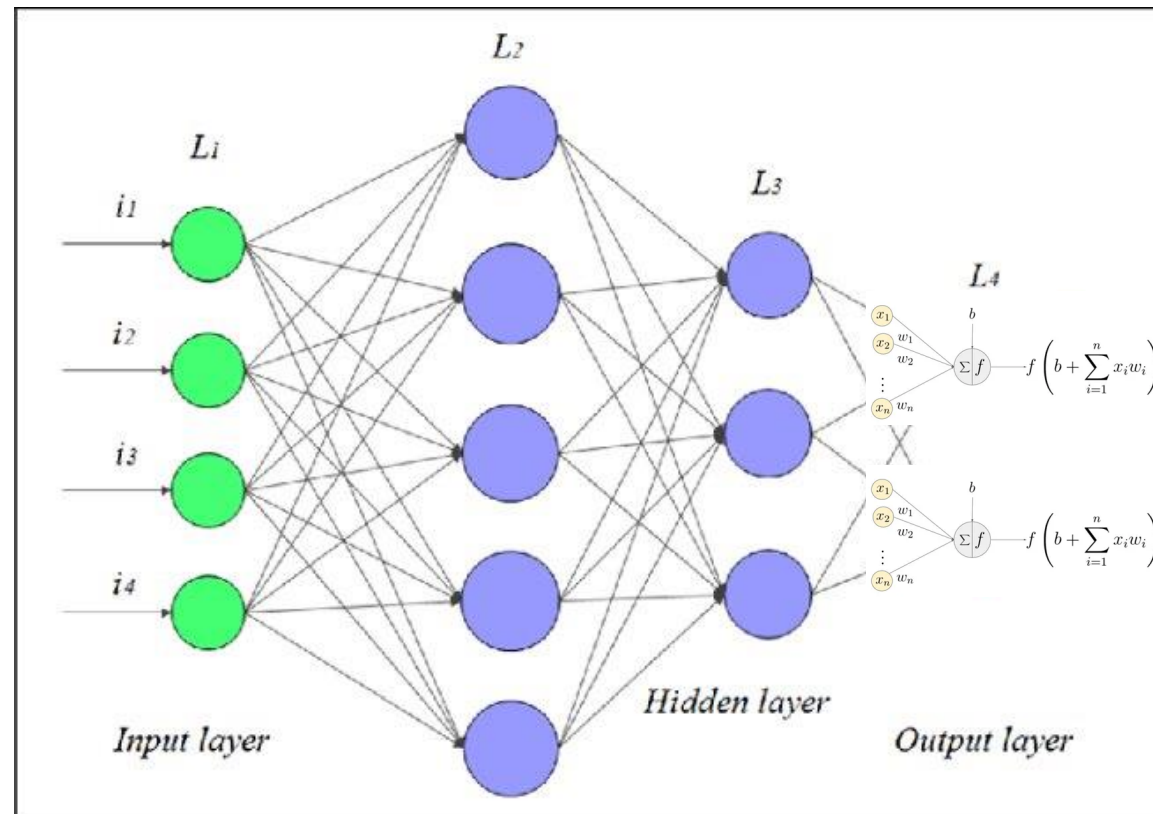
La red predice: propagación hacia delante

- Se hace lo mismo en la capa L3 hacia L4



La red predice: propagación hacia delante

- Finalmente se obtienen las salidas de las neuronas en la capa L4. Estos valores son los de la salida de la red.



Cómo las redes "aprenden"

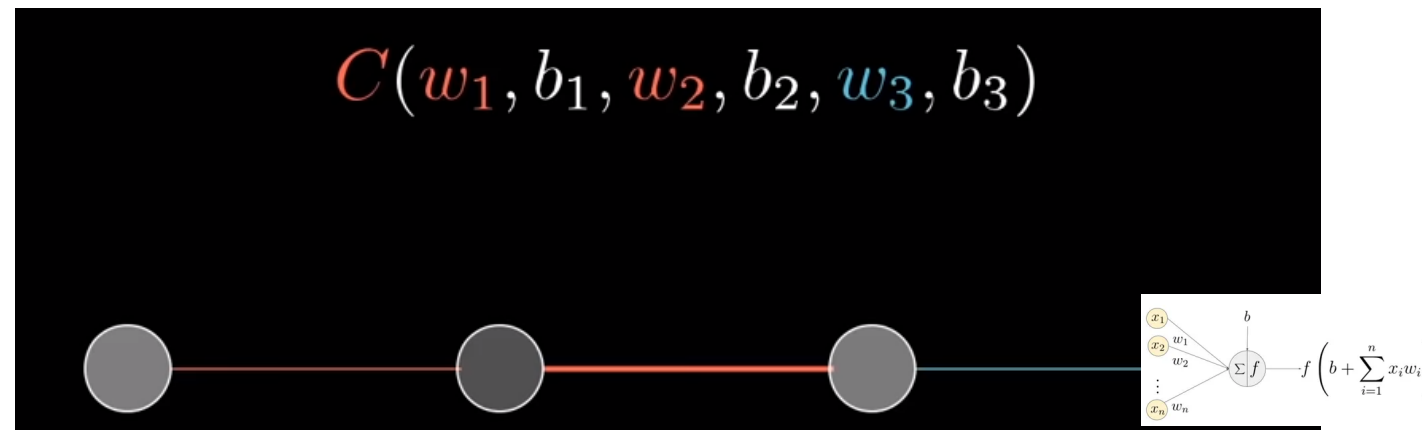
- Para que una red aprenda, se pueden utilizar técnicas supervisadas para calcular un error cometido por la red con respecto a valores de salida esperados, o técnicas no supervisadas que indiquen si un resultado es mejor o peor pero no de manera explícita
- Una vez se calcula este error, se propaga desde la capa de salida de la red hacia las capas anteriores de la misma, restando un número obtenido para cada peso o bias del aprendizaje, el cual "ajusta" los valores de estos pesos y bias para que, al recibir de nuevo dicho ejemplo, la red proporcione una mejor predicción

Cómo las redes "aprenden" (aprendizaje supervisado)

- En este vídeo está bastante bien explicado: <https://www.youtube.com/watch?v=tleHLnjs5U8>
- En el caso de que una red aprenda supervisada implica que se conocen los valores de salida que, una vez se entrene el modelo, esperamos obtener. Con ello, podemos inferir el error que comete la red sin aprender al tratar de hacer una predicción, y con ello, ir ajustando el modelo
- La idea principal consiste en primero hacer propagación hacia delante de un ejemplo del conjunto de entrenamiento a través de la red, después calcular una métrica de error cometido en cada neurona de la capa de salida de la red mediante una función de pérdida, y con ella calcular el cambio de valor que debe experimentar los valores de los biases de dichas neuronas y de los pesos de las aristas que conectan la capa de salida con la capa oculta anterior. Finalmente, con este error, calcular los errores que cometieron las neuronas de la capa anterior, y repetir el proceso hasta llegar a la capa de entrada.

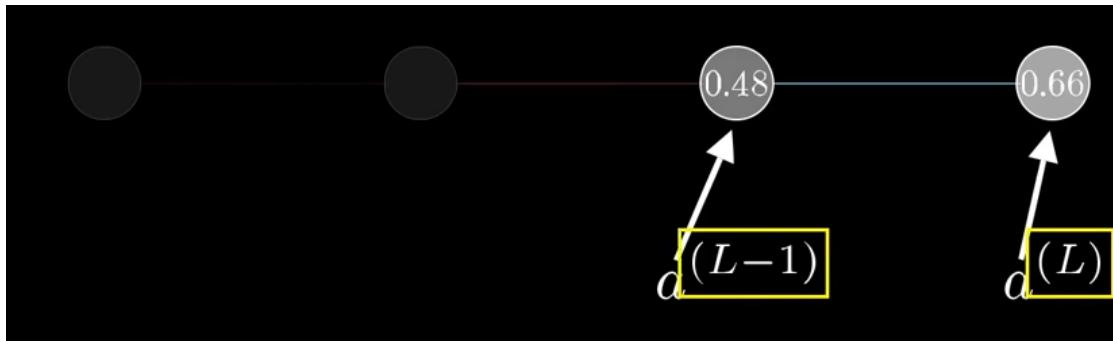
Cómo las redes "aprenden" (aprendizaje supervisado)

- Vamos a reducir la red a un caso en el que cada capa tiene solamente una neurona. Tenemos 3 pesos y 3 biases que ajustar
- Primero hacemos la propagación hacia delante del ejemplo de entrenamiento, como vimos anteriormente



Cómo las redes "aprenden" (aprendizaje supervisado)

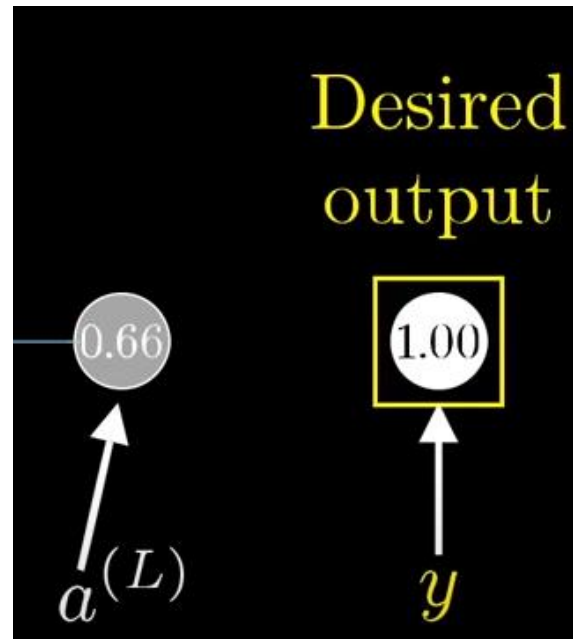
- Ahora, llamemos $a(L)$ al valor de salida de una neurona de la capa de salida, $z(L)$ el resultado que calculó esta neurona sin aplicar la función de activación, $w(L)$ el valor del peso de la arista que conecta la capa actual con la capa anterior
- Ojo, el índice (L) no es una potencia, es sólo un índice



$$z^{(L)} = w^{(L)} a^{(L-1)} + b^{(L)}$$
$$a^{(L)} = \sigma(z^{(L)})$$

Cómo las redes "aprenden" (aprendizaje supervisado)

- Ahora, recordemos que, en el aprendizaje supervisado, calculamos el valor de una función de pérdida para considerar el error cometido en la predicción con respecto a un valor de salida esperado correcto.



Cómo las redes "aprenden" (aprendizaje supervisado)

- Existen varias funciones de pérdida, pero vamos a mostrar, por ejemplo, el error cuadrático. Aparece el coste como C_0 , pero luego utilizaremos derivadas parciales con respecto al valor de salida de la red, solamente se considera la función de pérdida

$$\text{Cost} \rightarrow C_0(\dots) = (a^{(L)} - y)^2$$

- No confundir funciones de pérdida con funciones de coste, las funciones de pérdida evalúan el error de predicción de una sola neurona, las funciones de coste evalúan el error de todas las neuronas de la capa de salida. La función de coste suele ser el sumatorio de los errores de pérdida de las neuronas, y en algunos casos se calcula la media. Por ejemplo, el error cuadrático medio es la media aritmética de todos los errores cuadráticos cometidos

Cómo las redes "aprenden" (aprendizaje supervisado)

- Al aprender, no utilizamos el valor resultado de la función de pérdida directamente, sino que tomamos el resultado de calcular su derivada parcial con respecto al valor de salida de la red.
- ¿Por qué derivadas? El objetivo en el aprendizaje es calcular la derivada parcial de la función de coste con respecto a cada peso y bias de la red, y para propagar el error desde la capa de salida, con respecto a la capa anterior. La analogía directa es calcular lo que varía la función de coste, que indica el error total cometido por la red, con respecto a cada bias, peso y capa anterior, es decir, qué impacto tiene cada uno de estos en el valor final de la función de coste

Cómo las redes "aprenden" (aprendizaje supervisado)

- Al calcular la derivada parcial para cada peso y bias, podemos restar dicho valor a cada peso y bias de la red sin entrenar, y esto es lo que minimizará el error cometido por la red al volver a recibir exactamente el mismo ejemplo de entrada
- Teóricamente, se calculan todas estas derivadas para un ejemplo del conjunto de entrenamiento, los cuales se agrupan en un vector gradiente. Al entrenar el modelo, se suele calcular el vector gradiente para cada uno de los ejemplos de un conjunto de entrenamiento, después para las derivadas del mismo peso o bias se calcula una media aritmética, y finalmente cada media se resta a la red sin entrenar
- En la práctica se hace con subconjuntos del conjunto de entrenamiento, no con todo el conjunto de entrenamiento

Cómo las redes "aprenden" (aprendizaje supervisado)

- Las derivadas que se muestran a continuación se utilizarán después para obtener las derivadas parciales del coste con respecto a los pesos, los biases y la capa anterior de error de la red.

$$\frac{\partial C_0}{\partial a^{(L)}} = 2(a^{(L)} - y)$$

Derivada parcial de la función de coste con respecto al valor de salida de la neurona. Es lo mismo que la derivada parcial de la función de coste con respecto a $a^{(L)}$ $(a^{(L)} - y)^2$

$$\frac{\partial a^{(L)}}{\partial z^{(L)}} = \sigma'(z^{(L)})$$

Al derivar con respecto a $a^{(L)}$, aplicando la regla de la cadena, nos queda derivada_funcion_activacion($z^{(L)}$) * 1

$$\frac{\partial z^{(L)}}{\partial w^{(L)}} = a^{(L-1)}$$

Recordar la expresión de $z^{(L)}$, tenemos $w^{(L)} * a^{(L-1)} + b^{(L)}$, si derivamos con respecto $w^{(L)}$, queda $a^{(L-1)}$

$$\frac{\partial C_0}{\partial b^{(L)}} = 1$$

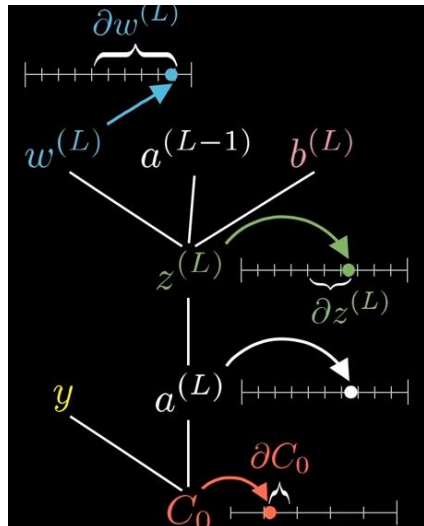
Recordar la expresión de $z^{(L)}$, tenemos $w^{(L)} * a^{(L-1)} + b^{(L)}$, si derivamos con respecto $b^{(L)}$, queda 1

$$\frac{\partial z^{(L)}}{\partial a^{(L-1)}} = \frac{\partial w^{(L)}}{\partial a^{(L-1)}}$$

Recordar la expresión de $a^{(L-1)}$, tenemos $w^{(L)} * a^{(L-1)} + b^{(L)}$, si derivamos con respecto $a^{(L-1)}$, queda $w^{(L)}$

Cómo las redes "aprenden" (aprendizaje supervisado)

- Estas derivadas se obtienen ya que una variable afecta a la anterior. $a(L)$ afecta al coste, $z(L)$ afecta a $a(L)$, $w(L)$ afecta a $z(L)$
- Por este motivo, para obtener las derivadas que necesitamos, tenemos que aplicar la regla de la cadena, básicamente multiplicar las derivadas en la que una dependa de la otra.



$$\frac{\partial C_0}{\partial w^{(L)}} = \frac{\partial z^{(L)}}{\partial w^{(L)}} \frac{\partial a^{(L)}}{\partial z^{(L)}} \frac{\partial C_0}{\partial a^{(L)}}$$

Derivada de la función de coste con respecto a un peso de la red

$$\frac{\partial C_0}{\partial b^{(L)}} = \frac{\partial z^{(L)}}{\partial b^{(L)}} \frac{\partial a^{(L)}}{\partial z^{(L)}} \frac{\partial C_0}{\partial a^{(L)}}$$

Derivada de la función de coste con respecto a un bias de la red

$$\frac{\partial C_0}{\partial a^{(L-1)}} = \frac{\partial z^{(L)}}{\partial a^{(L-1)}} \frac{\partial a^{(L)}}{\partial z^{(L)}} \frac{\partial C_0}{\partial a^{(L)}}$$

Derivada de la función de coste con respecto a la capa anterior de la red

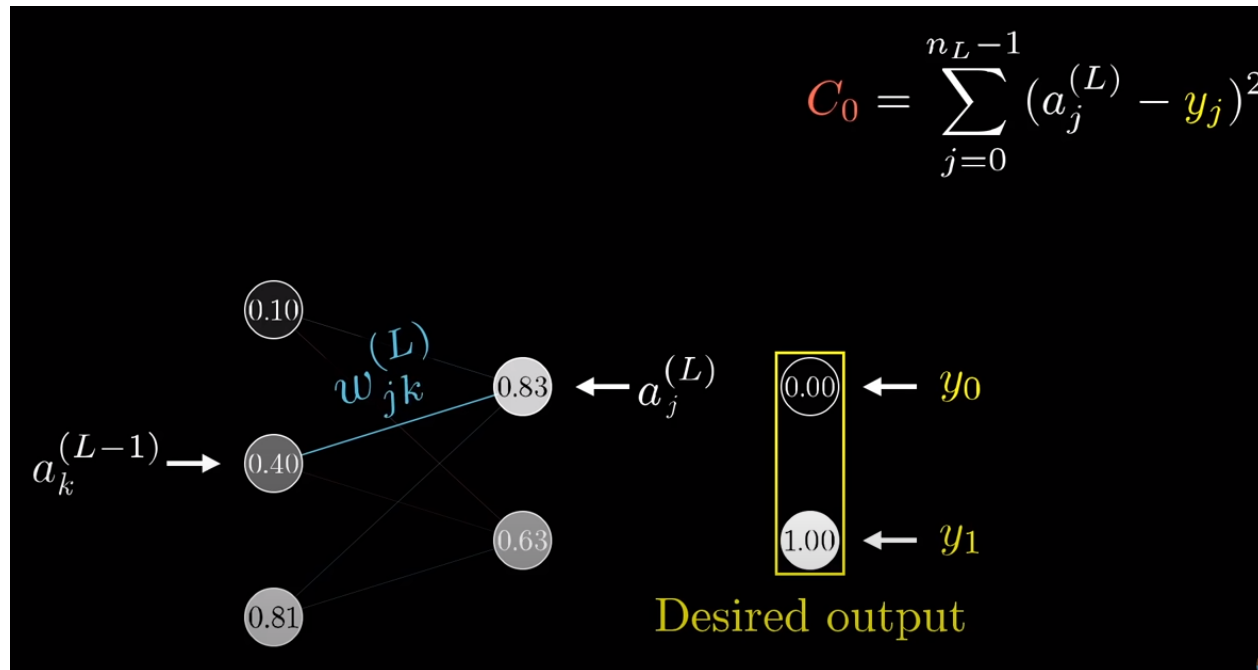
Cómo las redes "aprenden" (aprendizaje supervisado)

- Ahora, para seguir, nos movemos a la capa anterior. Ahora, no se calcula esta derivada parcial así: $\frac{\partial C_0}{\partial a^{(L)}} = 2(a^{(L)} - y)$, sino que esta derivada parcial pasa a ser $\frac{\partial C_0}{\partial a^{(L-1)}}$ calculada en la iteración anterior
- Ahora, en la capa actual, calculamos las 3 mismas derivadas de antes, y volvemos hacia atrás, y seguimos así hasta la capa anterior a la capa de entrada de la red, en la cual sólo calcularemos las derivadas

$\frac{\partial C_0}{\partial w^{(L)}} = \frac{\partial z^{(L)}}{\partial w^{(L)}} \frac{\partial a^{(L)}}{\partial z^{(L)}} \frac{\partial C_0}{\partial a^{(L)}}$ y $\frac{\partial C_0}{\partial b^{(L)}} = \frac{\partial z^{(L)}}{\partial b^{(L)}} \frac{\partial a^{(L)}}{\partial z^{(L)}} \frac{\partial C_0}{\partial a^{(L)}}$, recordemos que la capa de entrada no tiene neuronas, ya que no se modifican los valores que recibe, y por tanto, desde la primera capa oculta, no hace falta calcular el error de su capa anterior, que sería la capa de entrada, ya que esta no modifica el valor de la función de coste como tal

Cómo las redes "aprenden" (aprendizaje supervisado)

- Recordemos que estábamos en una red con una neurona por capa. Para extender al caso general, cambiarán un poco nuestras derivadas, sobre todo $\frac{\partial C_0}{\partial a^{(L-1)}}$
- Para ello, primero hay que considerar estas expresiones:



$$C_0 = \sum_{j=0}^{n_L-1} (a_j^{(L)} - y_j)^2$$

$$z_j^{(L)} = w_{j0}^{(L)} a_0^{(L-1)} + w_{j1}^{(L)} a_1^{(L-1)} + w_{j2}^{(L)} a_2^{(L-1)} + b_j^{(L)}$$

$$a_j^{(L)} = \sigma(z_j^{(L)})$$

Cómo las redes "aprenden" (aprendizaje supervisado)

- Las derivadas se expresarían:

$$\frac{\partial C_0}{\partial w_{jk}^{(L)}} = \frac{\partial z_j^{(L)}}{\partial w_{jk}^{(L)}} \frac{\partial a_j^{(L)}}{\partial z_j^{(L)}} \frac{\partial C_0}{\partial a_j^{(L)}}$$

$$\frac{\partial C_0}{\partial b_j^{(L)}} = \frac{\partial z_j^{(L)}}{\partial b_j^{(L)}} \frac{\partial a_j^{(L)}}{\partial z_j^{(L)}} \frac{\partial C_0}{\partial a_j^{(L)}}$$

$$\frac{\partial C_0}{\partial a_k^{(L-1)}} = \sum_{j=0}^{n_L-1} \frac{\partial z_j^{(L)}}{\partial a_k^{(L-1)}} \frac{\partial a_j^{(L)}}{\partial z_j^{(L)}} \frac{\partial C_0}{\partial a_j^{(L)}}$$

- Ahora en $\frac{\partial C_0}{\partial a_k^{(L-1)}}$ se están sumando los resultados que se calculan en todas las neuronas posteriores con respecto de la que se está calculando el error

Cómo las redes "aprenden" (aprendizaje supervisado)

- En resumen, las derivadas quedan (suponiendo que utilizamos la misma función de pérdida y coste):

$L = \text{output layer}$

$$\frac{\partial C_o}{\partial w_{jk}^{(L)}} = \begin{array}{|c|} \hline a_k^{(L-1)} \\ \hline \end{array} \begin{array}{|c|} \hline \sigma'(z_j^{(L)}) \\ \hline \end{array} \begin{array}{|c|} \hline 2 \cdot (a_j^{(L)} - y_j) \\ \hline \end{array}$$

$$\frac{\partial C_o}{\partial b_j^{(L)}} = \begin{array}{|c|} \hline 1 \\ \hline \end{array} \begin{array}{|c|} \hline \sigma'(z_j^{(L)}) \\ \hline \end{array} \begin{array}{|c|} \hline 2 \cdot (a_j^{(L)} - y_j) \\ \hline \end{array}$$

$$\frac{\partial C_o}{\partial a_k^{(L-1)}} = \sum_{j=0}^{n_L-1} \left(\begin{array}{|c|} \hline w_{jk}^{(L)} \\ \hline \end{array} \begin{array}{|c|} \hline \sigma'(z_j^{(L)}) \\ \hline \end{array} \begin{array}{|c|} \hline 2 \cdot (a_j^{(L)} - y_j) \\ \hline \end{array} \right)$$

$L \neq \text{output layer}$

$$\frac{\partial C_o}{\partial w_{jk}^{(L)}} = \begin{array}{|c|} \hline a_k^{(L-1)} \\ \hline \end{array} \begin{array}{|c|} \hline \sigma'(z_j^{(L)}) \\ \hline \end{array} \begin{array}{|c|} \hline \frac{\partial C_o}{\partial a_j^{(L)}} \text{ from previous iteration} \\ \hline \end{array}$$

$$\frac{\partial C_o}{\partial b_j^{(L)}} = \begin{array}{|c|} \hline 1 \\ \hline \end{array} \begin{array}{|c|} \hline \sigma'(z_j^{(L)}) \\ \hline \end{array} \begin{array}{|c|} \hline \frac{\partial C_o}{\partial a_j^{(L)}} \text{ from previous iteration} \\ \hline \end{array}$$

$$\frac{\partial C_o}{\partial a_k^{(L-1)}} = \sum_{j=0}^{n_L-1} \left(\begin{array}{|c|} \hline w_{jk}^{(L)} \\ \hline \end{array} \begin{array}{|c|} \hline \sigma'(z_j^{(L)}) \\ \hline \end{array} \begin{array}{|c|} \hline \frac{\partial C_o}{\partial a_j^{(L)}} \text{ from previous iteration} \\ \hline \end{array} \right)$$

Cómo las redes "aprenden" (aprendizaje supervisado)

- En el caso de utilizar una función de pérdida y de coste diferentes, la derivada parcial en la capa de salida $\frac{\partial C_0}{\partial a^{(L)}}$ será diferente. Por ejemplo, supongamos que la función de pérdida que utilizamos es (siendo y con gorrita $a^{(L)}$, y sin gorrita la salida esperada):

$$\mathcal{L}(\theta) = -[(1 - y) \log(1 - \hat{y}) + y \log \hat{y}]$$

- Tendríamos que $\frac{\partial C_0}{\partial a^{(L)}}$ en la capa de salida sería $= \frac{\hat{y} - y}{(1 - \hat{y})\hat{y}}$

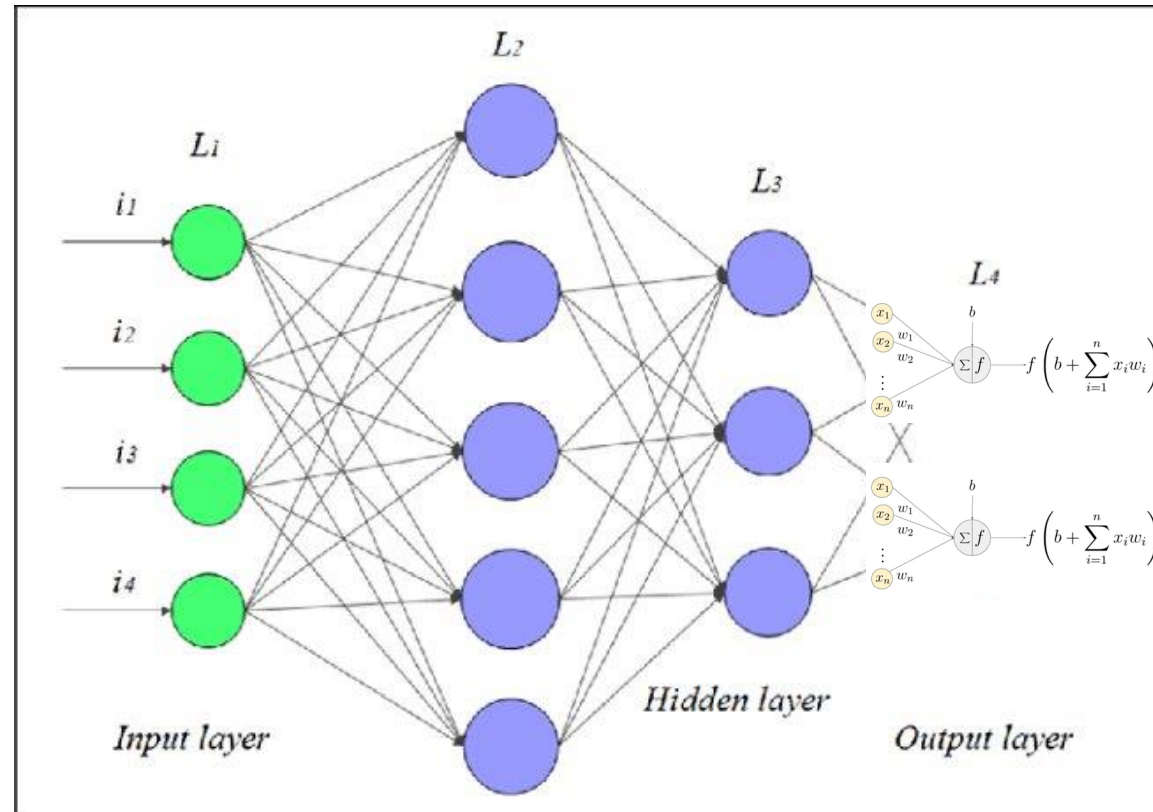
$$\begin{aligned}\frac{\partial \mathcal{L}(\theta)}{\partial \hat{y}} &= \frac{\partial}{\partial \hat{y}} \{-(1 - y) \log(1 - \hat{y}) - y \log \hat{y}\} \\ &= (-)(-1) \frac{(1 - y)}{(1 - \hat{y})} - \frac{y}{\hat{y}} \\ &= \frac{\hat{y}(1 - y) - y(1 - \hat{y})}{(1 - \hat{y})\hat{y}} \\ &= \frac{\hat{y} - y}{(1 - \hat{y})\hat{y}}\end{aligned}$$

Cómo las redes "aprenden" (aprendizaje supervisado)

- Esta regla aplica incluso a funciones más peculiares, como Softmax, que su valor depende del resto de valores de salida de las neuronas de la capa de salida, aunque sus derivadas son un poco más enreversadas que acaban en una simplificación concreta al utilizar una función de coste/pérdida concreta, llamada entropía cruzada

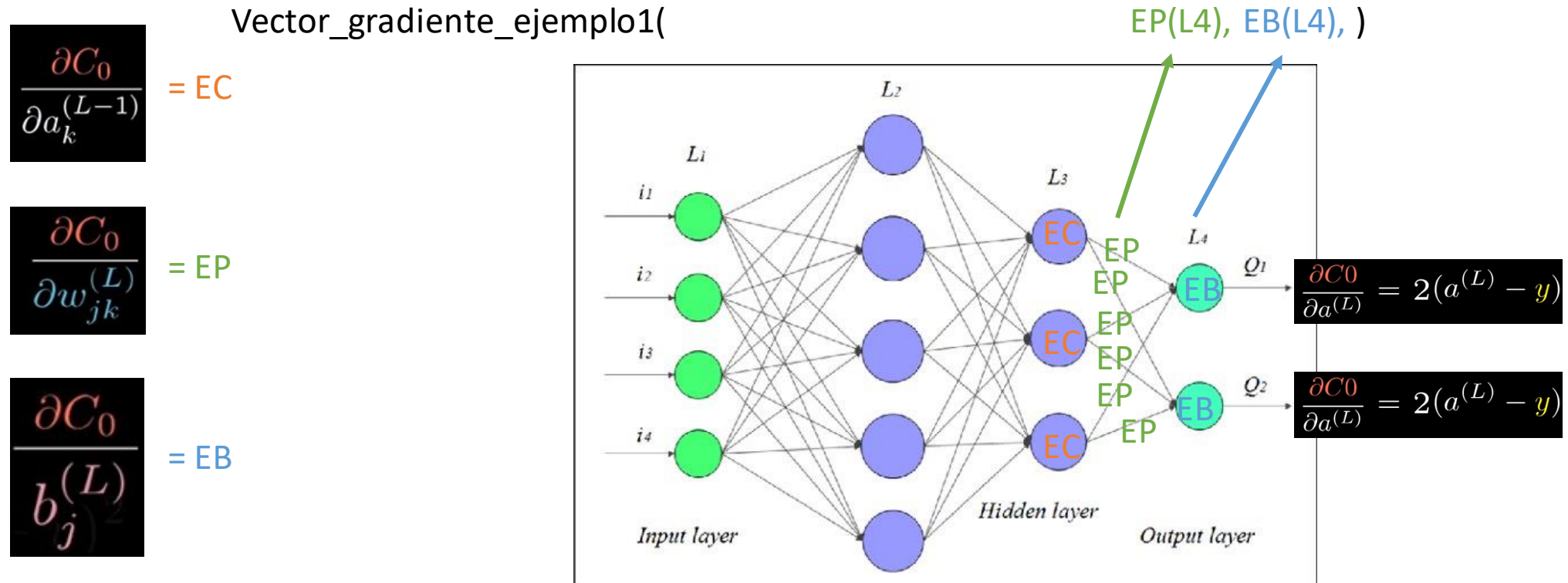
Cómo las redes "aprenden" (aprendizaje supervisado)

- Hacemos la propagación hacia delante y obtenemos las salidas



Cómo las redes "aprenden" (aprendizaje supervisado)

- Calculamos las derivadas parciales de pesos y bias en L4 y las guardamos en el vector gradiente . Los errores de la capa L3 se utilizarán cuando pasemos a esa capa después. Luego nos movemos a L3



Cómo las redes "aprenden" (aprendizaje supervisado)

- Hacemos el mismo proceso, ahora $\frac{\partial C_0}{\partial a^{(L)}}$ no es $2(a^{(L)} - y)$, sino el **EC** que calculamos anteriormente, el $\frac{\partial C_0}{\partial a_k^{(L-1)}}$ de antes. Al terminar, nos movemos a $L2$

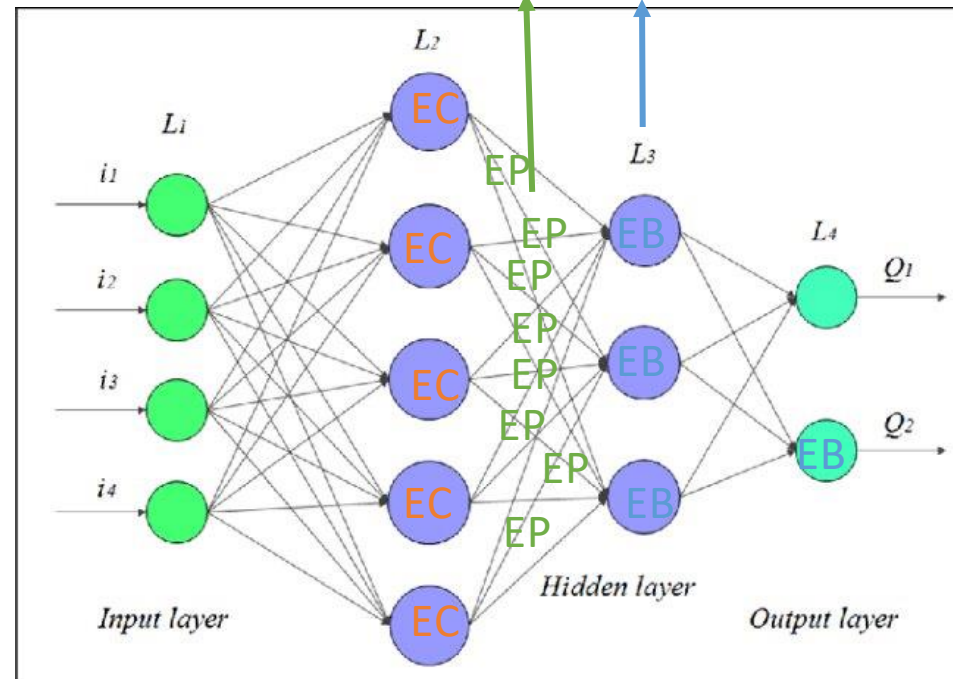
$$\frac{\partial C_0}{\partial a_k^{(L-1)}} = \text{EC}$$

$$\frac{\partial C_0}{\partial w_{jk}^{(L)}} = \text{EP}$$

$$\frac{\partial C_0}{\partial b_j^{(L)}} = \text{EB}$$

Vector_gradiente_ejemplo1(

EP(L3), EB(L3), EP(L4), EB(L4),)



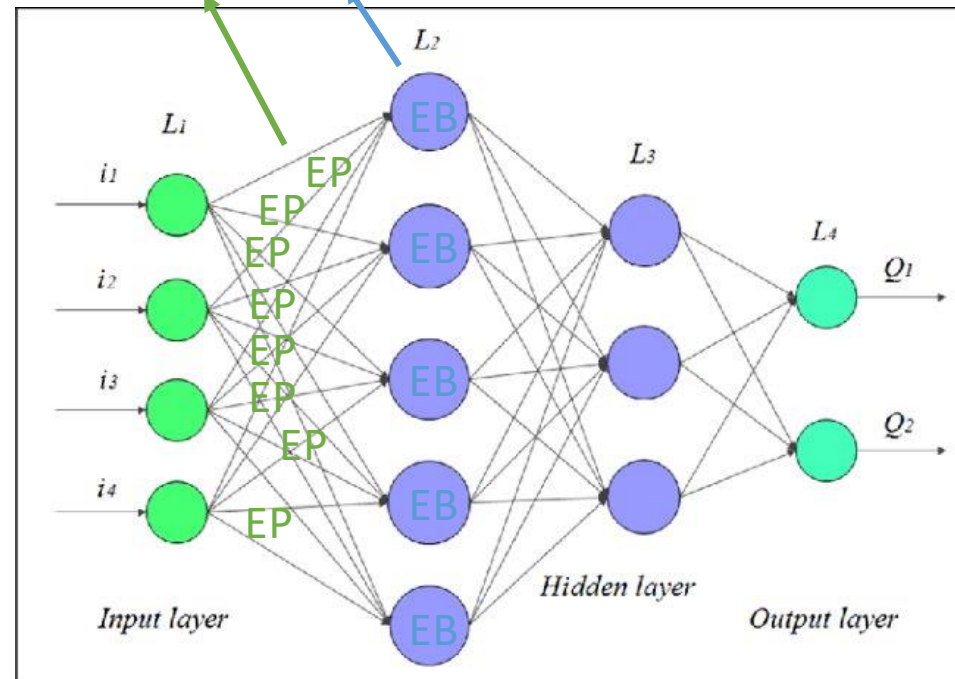
Cómo las redes "aprenden" (aprendizaje supervisado)

- Finalmente, en L2 sólo calculamos los errores de los bias y de los pesos

Vector_gradiente_ejemplo1(EP(L2), EB(L2), EP(L3), EB(L3), EP(L4), EB(L4),)





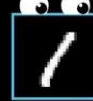

$$\frac{\partial C_0}{\partial w_{jk}^{(L)}} = EP$$

$$\frac{\partial C_0}{\partial b_j^{(L)}} = EB$$



Cómo las redes "aprenden" (aprendizaje supervisado)







- Una vez hemos calculado los valores de corrección de pesos y biases (las derivadas parciales), calculamos la media aritmética de los valores de todos los ejemplos para cada peso y bias. En la imagen sólo están los pesos, pero se hace lo mismo con los biases

							Average over all training data ...
w_0	-0.08	+0.02	-0.02	+0.11	-0.05	-0.14	... → -0.08
w_1	-0.11	+0.11	+0.07	+0.02	+0.09	+0.05	... → +0.12
w_2	-0.07	-0.04	-0.01	+0.02	+0.13	-0.15	... → -0.06
\vdots	\vdots	\vdots	\vdots	\vdots	\vdots	\vdots	\ddots
$w_{13,001}$	+0.13	+0.08	-0.06	-0.09	-0.02	+0.04	... → +0.04

Vector gradiente de todos los ejemplos del conjunto de entrenamiento (vamos a seguir el enfoque teórico)

Cómo las redes "aprenden" (aprendizaje supervisado)

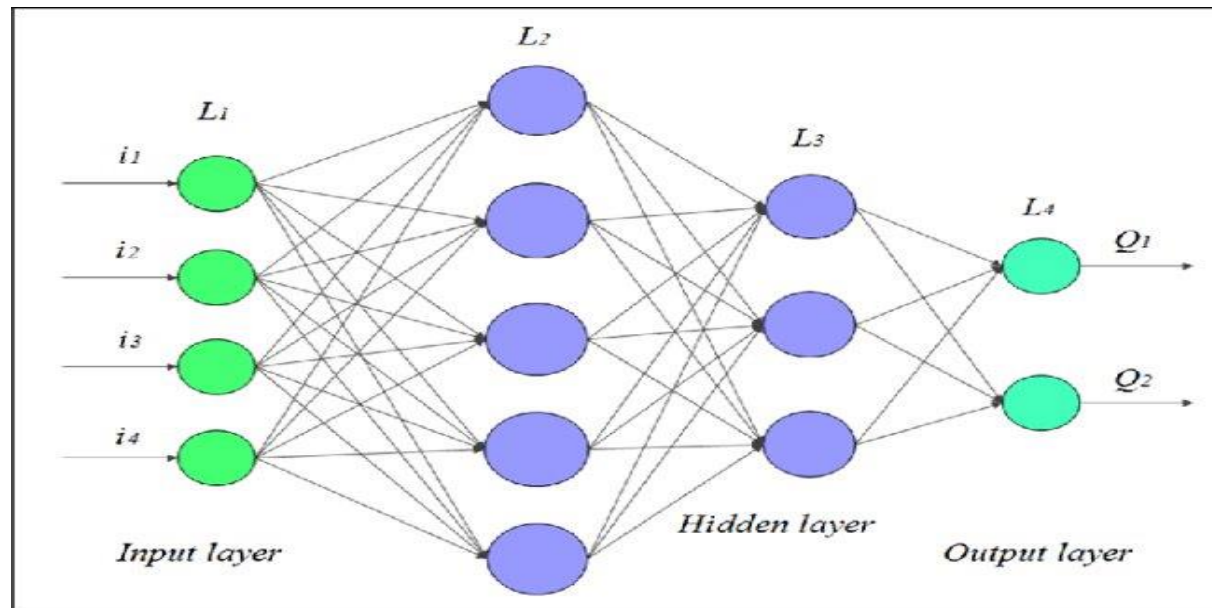
- Después, a cada valor, le multiplicaremos un número, que se considera un hiper-parámetro, llamado tasa de aprendizaje (ta)
- Su valor está comprendido en el rango [1.0,0.0)

							Average over all training data ...
w_0	-0.08	+0.02	-0.02	+0.11	-0.05	-0.14	... → -0.08
w_1	-0.11	+0.11	+0.07	+0.02	+0.09	+0.05	... → +0.12
w_2	-0.07	-0.04	-0.01	+0.02	+0.13	-0.15	... → -0.06
\vdots	\vdots	\vdots	\vdots	\vdots	\vdots	\vdots	\ddots
$w_{13,001}$	+0.13	+0.08	-0.06	-0.09	-0.02	+0.04	... → +0.04

$\ast = ta$
 $\ast = ta$
 $\ast = ta$
...
 $\ast = ta$

Cómo las redes "aprenden" (aprendizaje supervisado)

- Para terminar, se **resta** el valor de cada error de peso o de bias a su correspondiente peso o bias



Vector_gradiente_ejemplo1(EP(L2), EB(L2), EP(L3), EB(L3), EP(L4), EB(L4),)
=
Red entrenada

Cómo las redes "aprenden" (aprendizaje supervisado)

- Esta forma de hacer "aprender" a la red es el algoritmo de optimización del descenso de gradiente. Si en vez de hacerlo para todo el conjunto de entrenamiento se hubiese hecho con un subconjunto del mismo, el algoritmo de optimización pasa a ser el descenso de gradiente estocástico
- Depender de una tasa de aprendizaje fija provoca ciertos problemas en el aprendizaje, como provocar que la red desaprenda en vez de aprender. Es por ello que se suelen utilizar algoritmos de optimización más sofisticados, como puede ser ADAM