CAPÍTULO 4.

Redes neuronales densamente conectadas

De la misma manera que cuando uno empieza a programar en un lenguaje nuevo existe la tradición de hacerlo con un *print Hello World*, en Deep Learning se empieza por crear un modelo de reconocimiento de números escritos a mano. Nos aprovecharemos de este ejemplo para adentrarnos paulatinamente en los conceptos básicos de las redes neuronales, reduciendo todo lo posible conceptos teóricos, con el objetivo de ofrecer al lector o lectora una visión global de un caso concreto para facilitar la lectura de los capítulos posteriores, donde se profundiza en diferentes aspectos del área.

4.1. Caso de estudio: reconocimiento de dígitos

En este apartado, presentamos los datos que usaremos para nuestro primer ejemplo de redes neuronales: el conjunto de datos MNIST, que contiene imágenes de dígitos escritos a mano.

El conjunto de datos MNIST, que se puede descargar de la página The MNIST database[85], está formado por imágenes de dígitos escritos a mano. Este conjunto de datos contiene 60 000 elementos para entrenar el modelo y 10 000 adicionales para testearlo, y es ideal para adentrarse por primera vez en técnicas de reconocimiento de patrones sin tener que dedicar mucho tiempo al preproceso y formateado de datos —ambos pasos muy importantes y costosos en el análisis de datos[86], y de especial complejidad cuando se está trabajando con imágenes—. Este

[85] The MNIST database of handwritten digits. [*online*]. Disponible en: http://yann.lecun.com/exdb/mnist [Consulta: 30/12/2019].

[86] Sin duda, este paso de «limpiar» y «preparar» los datos es la fase más costosa en tiempo y la más desagradecida con datos reales.

conjunto de datos solo requiere pequeñas transformaciones que comentaremos a continuación.

Este conjunto de imágenes, originales en blanco y negro, han sido normalizadas a 20 × 20 píxeles, y conservan su relación de aspecto. En este caso, es importante notar que las imágenes contienen niveles de grises como resultado de la técnica de *anti-aliasing*[87], usada en el algoritmo de normalización (reducir la resolución de todas las imágenes). Posteriormente, las imágenes se han centrado en 28 × 28 píxeles —se calcula el centro de masa de estos y se traslada la imagen con el fin de posicionar este punto en el centro del campo de 28 × 28—. Un grupo representativo de las imágenes de este conjunto de datos se presenta en la Figura 4.1:

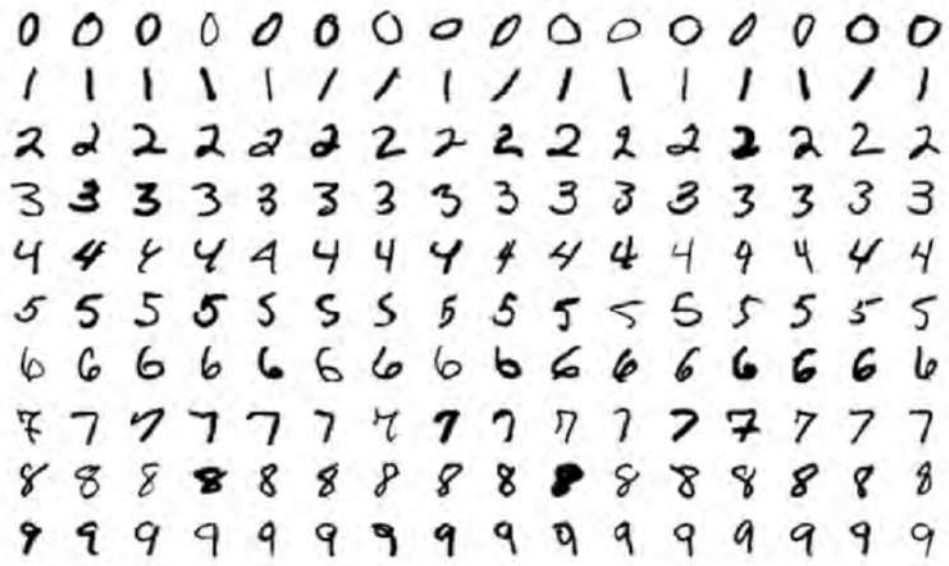

Figura 4.1 *Ejemplos de imágenes que conforman el conjunto de datos MNIST*[88].

Estas imágenes, de entrada, se representan en una matriz con las intensidades de cada uno de los 28 × 28 píxeles con valores entre [0, 255]. Por ejemplo, la imagen de la Figura 4.2 (la octava del conjunto de entrenamiento) se representa con la matriz de puntos de la Figura 4.3:

[87] Wikipedia, (2016). *Anti-aliasing* [*online*]. Disponible en: https://es.wikipedia.org/wiki/Antialiasing [Consulta: 10/12/2019].

[88] Imagen obtenida de Wikipedia: https://en.wikipedia.org/wiki/MNIST_database.

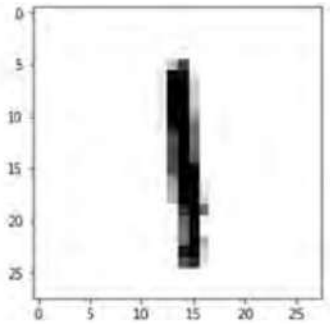

Figura 4.2 *Imagen de la octava muestra del conjunto de datos de entrenamiento.*

```
[[  0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0]
 [  0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0]
 [  0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0]
 [  0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0]
 [  0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0]
 [  0   0   0   0   0   0   0   0   0   0   0   0   5  63 197   0   0   0   0   0   0   0   0   0   0   0]
 [  0   0   0   0   0   0   0   0   0   0   0   0  20 254 230  24   0   0   0   0   0   0   0   0   0   0]
 [  0   0   0   0   0   0   0   0   0   0   0   0  20 254 254  48   0   0   0   0   0   0   0   0   0   0]
 [  0   0   0   0   0   0   0   0   0   0   0   0  20 254 255  48   0   0   0   0   0   0   0   0   0   0]
 [  0   0   0   0   0   0   0   0   0   0   0   0  20 254 254  57   0   0   0   0   0   0   0   0   0   0]
 [  0   0   0   0   0   0   0   0   0   0   0   0  20 254 254 108   0   0   0   0   0   0   0   0   0   0]
 [  0   0   0   0   0   0   0   0   0   0   0   0  16 239 254 143   0   0   0   0   0   0   0   0   0   0]
 [  0   0   0   0   0   0   0   0   0   0   0   0   0 178 254 143   0   0   0   0   0   0   0   0   0   0]
 [  0   0   0   0   0   0   0   0   0   0   0   0   0 178 254 143   0   0   0   0   0   0   0   0   0   0]
 [  0   0   0   0   0   0   0   0   0   0   0   0   0 178 254 162   0   0   0   0   0   0   0   0   0   0]
 [  0   0   0   0   0   0   0   0   0   0   0   0   0 178 254 240   0   0   0   0   0   0   0   0   0   0]
 [  0   0   0   0   0   0   0   0   0   0   0   0   0 113 254 240   0   0   0   0   0   0   0   0   0   0]
 [  0   0   0   0   0   0   0   0   0   0   0   0   0  83 254 245  31   0   0   0   0   0   0   0   0   0]
 [  0   0   0   0   0   0   0   0   0   0   0   0   0  79 254 246  38   0   0   0   0   0   0   0   0   0]
 [  0   0   0   0   0   0   0   0   0   0   0   0   0   0 214 254 150   0   0   0   0   0   0   0   0   0]
 [  0   0   0   0   0   0   0   0   0   0   0   0   0   0 144 241   8   0   0   0   0   0   0   0   0   0]
 [  0   0   0   0   0   0   0   0   0   0   0   0   0   0 144 240   2   0   0   0   0   0   0   0   0   0]
 [  0   0   0   0   0   0   0   0   0   0   0   0   0   0 144 254  82   0   0   0   0   0   0   0   0   0]
 [  0   0   0   0   0   0   0   0   0   0   0   0   0   0 230 247  40   0   0   0   0   0   0   0   0   0]
 [  0   0   0   0   0   0   0   0   0   0   0   0   0   0 168 209  31   0   0   0   0   0   0   0   0   0]
 [  0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0]
 [  0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0]
 [  0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0]]
```

Figura 4.3 *Matriz con las intensidades de cada uno de los 28 × 28 píxeles con valores entre [0, 255] que representa la octava muestra del conjunto de datos de entrenamiento.*

Además, el conjunto de datos dispone de una etiqueta para cada una de las imágenes, que indica qué dígito representa (entre el 0 y el 9), es decir, a qué clase corresponde. En este ejemplo, vamos a representar esta etiqueta con un vector de 10 posiciones, donde la posición correspondiente al dígito que representa la imagen contiene un 1, y el resto son 0. Este proceso de transformar las etiquetas en un vector de tantos ceros como número de etiquetas distintas, y poner un 1 en el índice que le corresponde a la etiqueta, se conoce como *one-hot encoding,* y lo presentaremos más adelante con mayor detalle.

4.2. Una neurona artificial

Para continuar acompañando al lector o lectora en este proceso de aproximación a las redes neuronales, ha llegado el momento de hacer una breve explicación intuitiva sobre cómo funciona una neurona. Pero, antes, será conveniente presentar una breve introducción a la terminología básica (basada en el Machine Learning), que nos facilitará la exposición de las ideas de este capítulo y nos permitirá mantener un guion de presentación de los conceptos básicos de Deep Learning de manera más cómoda y gradual a lo largo del libro.

4.2.1. Introducción a la terminología y notación básica

Basándonos en el caso de estudio que nos ocupa, hablaremos de muestra (o dato de entrada, o ejemplo, o instancia u observación indistintamente, dependiendo del autor en español) cuando en inglés hablamos de *item*, eso es, cuando nos referimos a uno de los datos que conforman el conjunto de datos de entrada —en este caso, una de las imágenes de MNIST—. Cada una de estas muestras tiene unas determinadas características (o atributos o variables indistintamente), que en inglés usualmente referenciamos como *features*.

Con etiqueta de clase o *label* nos referimos a lo que estamos intentando predecir con un modelo.

Un modelo define la relación entre las características y las etiquetas del conjunto de muestras y presenta dos fases claramente diferenciadas para el tema que nos ocupa:

- Fase de entrenamiento o aprendizaje (*training* en inglés), que es cuando se crea o se «aprende» el modelo, a partir de mostrarle las muestras de entrada que se tienen etiquetadas; de esta manera se consigue que el modelo aprenda iterativamente las relaciones entre las características y las etiquetas de las muestras.
- Fase de inferencia o predicción (*inference* en inglés), que se refiere al proceso de hacer predicciones mediante la aplicación del modelo ya entrenado a muestras de las que no se dispone de etiqueta y que se quiere predecir.

Para que la notación sea simple a la vez que eficaz, en general se usan algunos de los términos básicos de álgebra lineal básica. Una notación simple para el modelo que expresa una relación lineal entre características y etiquetas para una muestra de entrada podría ser la siguiente:

$$y=wx+b$$

donde:

- y es la etiqueta de una muestra de entrada.
- x representa las características de la muestra.

- w es la pendiente de la recta y que, en general, llamaremos peso (o *weight* en inglés) y es uno de los dos parámetros que tendrá que aprender el modelo durante el proceso de entrenamiento para poder usarlo luego para la inferencia.
- b es lo que llamamos sesgo (o *bias* en inglés). Este es el otro de los parámetros que deben ser aprendidos por el modelo.

Aunque en este modelo simple que hemos representado solo tenemos una característica de entrada, en el caso general cada muestra de entrada puede tener varias características. En este caso cada una con su peso w_i. Por ejemplo, en un conjunto de datos de entrada en el que cada muestra presenta tres características (x_1, x_2, x_3), la relación algebraica anterior la podríamos expresar de la siguiente manera:

$$y=w_1x_1+w_2x_2+w_3x_3+b$$

es decir, el sumatorio del producto escalar entre los dos vectores (w_1, w_2, w_3) y (x_1, x_2, x_3) y luego la suma del sesgo:

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

Siguiendo la notación algebraica habitual en Machine Learning —que nos será útil a lo largo del libro para simplificar las explicaciones— podríamos expresar esta ecuación como:

$$y = w_1 x_1 + w_2 x_2 + \cdots + w_n x_n + b = \mathbf{w}^T \cdot \mathbf{x} + b$$

Para facilitar la formulación, el parámetro sesgo b a menudo se expresa como el peso w_0, asumiendo una característica adicional fija de $x_0 = 1$ para cada muestra, con lo cual la anterior formulación la podemos simplificar como:

$$y = w_0 x_0 + w_1 x_1 + w_2 x_2 + \cdots + w_n x_n = \mathbf{w}^T \cdot \mathbf{x}$$

En realidad, en nuestro ejemplo de caracteres MNIST podemos considerar a cada bit una característica y, por tanto, x representa a las n = 784 (correspondiente a 28 x 28) características y la y es la etiqueta de una muestra de entrada que tendrá por valor una clase entre la 0 y la 9.

Por ahora, podemos considerar que la fase de entrenamiento de un modelo consiste básicamente en ajustar los pesos W con las muestras que tenemos para entrenamiento, de tal manera que cuando usemos este modelo pueda predecir correctamente.

4.2.2. Algoritmos de regresión

Volviendo a lo que se ha introducido en el apartado anterior, es importante hacer un breve recordatorio sobre algoritmos de regresión y clasificación de Machine Learning clásico, ya que son el punto de partida de nuestras explicaciones de Deep Learning.

Podríamos decir que los algoritmos de regresión modelan la relación entre distintas variables de entrada *(features)* utilizando una medida de error, la *loss* (que presentaremos en detalle en el siguiente capítulo), que se intentará minimizar en un proceso iterativo para poder realizar predicciones «lo más acertadas posibles». Hablaremos de dos tipos: regresión logística y regresión lineal.

La diferencia principal entre regresión logística y lineal es en el tipo de salida de los modelos; cuando nuestra salida es discreta hablamos de regresión logística, y cuando la salida es continua hablamos de regresión lineal.

Siguiendo las definiciones del primer capítulo, la regresión logística es un algoritmo con aprendizaje supervisado y se utiliza para clasificar. El ejemplo que usaremos a continuación, que consiste en identificar a qué clase pertenece cada ejemplo de entrada asignándole un valor discreto de tipo 0 o 1, se trata de una clasificación binaria.

4.2.3. Una neurona artificial simple

Para mostrar cómo es una neurona básica, supongamos un ejemplo simple, donde tenemos un conjunto de puntos en un plano de dos dimensiones, y cada punto se encuentra etiquetado como «cuadrado» o «círculo»:

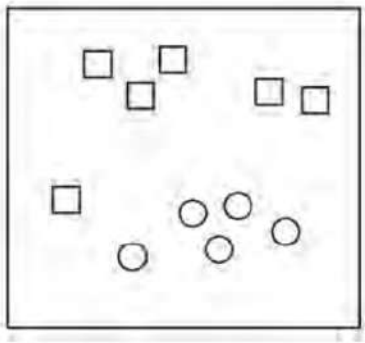

Figura 4.4 *Ejemplo visual de un plano de dos dimensiones que contiene dos tipos de muestras que queremos clasificar.*

Dado un nuevo punto «X», queremos saber qué etiqueta le corresponde (Figura 4.5):

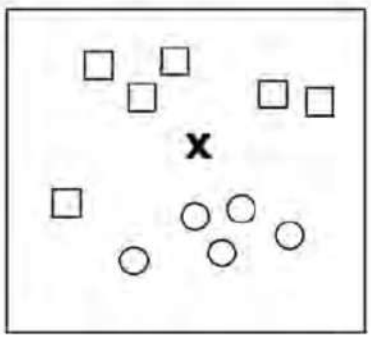

Figura 4.5 *Ejemplo anterior donde se ha añadido un punto que queremos clasificar.*

Una aproximación habitual es encontrar la línea que separe los dos grupos y usarla como clasificador, tal como se muestra en la Figura 4.6:

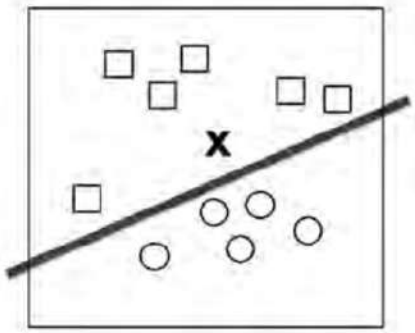

Figura 4.6 *Línea que actúa de clasificador para decidir a qué categoría se asigna el nuevo punto.*

En este caso, los datos de entrada serán representados por vectores de la forma (x_1, x_2) que indican sus coordenadas en este espacio de dos dimensiones, y nuestra función retornará 0 o 1 (que interpretamos como encima o debajo de la línea) para saber si se debe clasificar como «cuadrado» o como «círculo». Como hemos visto, se trata de un caso de regresión, donde la línea (el clasificador) puede ser definida por la recta:

$$y = w_1 x_1 + w_2 x_2 + b$$

Siguiendo la notación presentada en la sección anterior, de manera más generalizada podemos expresar la recta como:

$$y = W * X + b$$

Para clasificar el elemento de entrada X, en nuestro caso de dos dimensiones, debemos aprender un vector de peso W de la misma dimensión que el vector de entrada, es decir, el vector (w_1, w_2) y un sesgo b.

Con estos valores calculados, ahora ya podemos construir una neurona artificial para clasificar un nuevo elemento X Básicamente, la neurona aplica este vector W de pesos —calculado de manera ponderada— sobre los valores en cada dimensión del elemento X de entrada, le suma el sesgo b, y el resultado lo pasa a través de una función no lineal para producir un resultado de 0 o 1. La función de esta neurona artificial que acabamos de definir puede expresarse de una manera más formal:

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

$$y = \begin{cases} 1 \text{ si } z \geq 0 \\ 0 \text{ si } z < 0 \end{cases}$$

Aunque hay varias funciones (que llamaremos «funciones de activación» y presentaremos en detalle en el capítulo 6), para este ejemplo podemos usar una función conocida como función *sigmoid*[89], que retorna un valor real de salida entre 0 y 1 para cualquier valor de entrada:

$$y = \frac{1}{1+e^{-z}}$$

Si se piensa un poco en la fórmula, veremos que tiende siempre a dar valores próximos al 0 o al 1. Si la entrada z es razonablemente grande y positiva, e a la menos z es cero y, por tanto, la y toma el valor de 1. Si z tiene un valor grande y negativo, resulta que para e elevado a un número positivo grande el denominador resultará ser un número grande y, por lo tanto, el valor de y será próximo a 0. Gráficamente, la función *sigmoid* presenta esta forma que se muestra en la Figura 4.7:

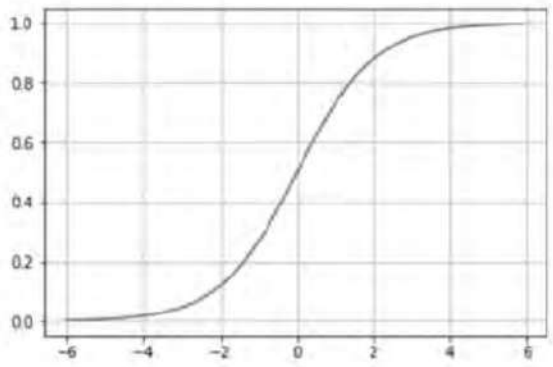

Figura 4.7 *Función de activación* sigmoid.

Finalmente, nos queda saber cómo se puede aprender los pesos W y el sesgo b a partir de los datos de entrada que ya tenemos etiquetados. En el capítulo 6 presentaremos cómo se realiza este proceso de una manera más formal, pero de momento avanzamos una visión intuitiva para comprender el proceso global. Se trata de un proceso iterativo con todos los elementos de entrada, que consiste en comparar el valor de la etiqueta que el modelo predice por cada elemento, «cuadrado» o «redonda», con su valor real. Teniendo en cuenta el error realizado

[89] Wikipedia, (2018). Sigmoid function [*online*]. Disponible en: https://en.wikipedia.org/wiki/Sigmoid_function [Consulta: 29/12/2019].

por el modelo cuando hace una predicción a cada iteración, se ajustan los valores de los parámetros W y b de manera que ayuden a reducir cada vez más el error de predicción a medida que van pasando muestras por el modelo.

4.3. Redes neuronales

En la sección previa hemos introducido un algoritmo de clasificación de regresión logística para adentrarnos de forma intuitiva a lo que es una neurona artificial. En realidad, el primer ejemplo de red neuronal se llama perceptrón, y fue inventado por Frank Rosenblatt hace muchos años.

4.3.1. Perceptrón

El perceptrón[90] es un algoritmo de clasificación equivalente al mostrado en la sección anterior —es la arquitectura más simple que puede tener una red neuronal—, creado en 1957 por Frank Rosenblatt y basado en los trabajos del neurofisiólogo Warren McCullon y el matemático Walter Pitts presentados ya en 1943[91]. Warren McCulloch y Walter Pitts propusieron un modelo muy simple de neurona artificial, basado en una neurona biológica, que consistía en una o más entradas binarias que podían estar activadas o no («encendida» o «apagada») y una salida binaria. La neurona artificial simplemente activa su salida cuando más de un cierto número de sus entradas están activas.

El perceptrón se basa en una neurona artificial ligeramente diferente a la propuesta por Warren McCulloch y Walter Pitts, referenciada también en la literatura como *linear threshold unit* (LTU). Las entradas y salidas ahora son números (en lugar de valores binarios de activación o desactivación) y cada conexión de entrada está asociada con un peso; luego, se aplica una función de activación, como hemos mostrado en la sección anterior. Podemos resumir visualmente su estructura general con el esquema de la Figura 4.8.

---

[90] Wikipedia (2018). Perceptrón [*online*]. Disponible en https://en.wikipedia.org/wiki/Perceptron [Consulta 22/12/2019].

[91] A logical calculus of the ideas immanent in nervous activity. McCulloch, W.S. & Pitts, W. Bulletin of Mathematical Biophysics (1943) 5: 115. https://doi.org/10.1007/BF02478259

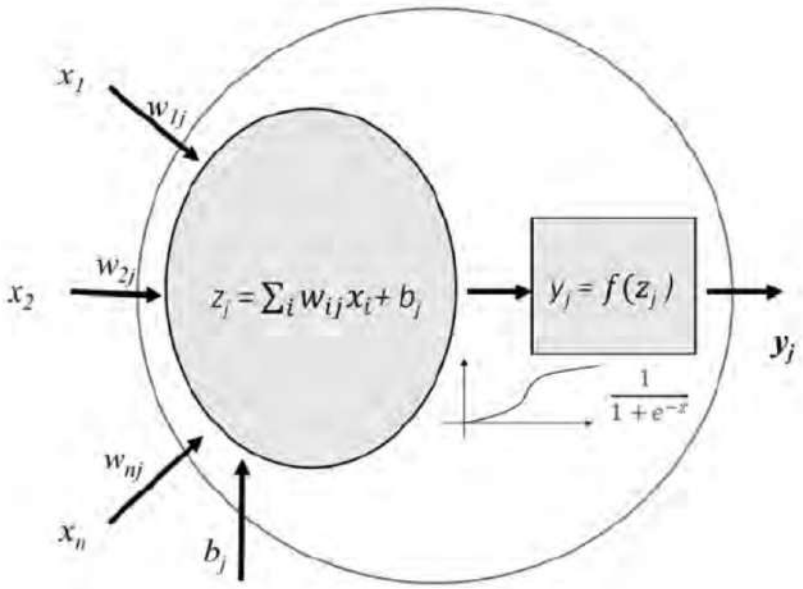

Figura 4.8 *Esquema general de una neurona artificial que computa una suma ponderada de sus entradas y aplica una función de activación a su resultado.*

El perceptrón es la versión más simple de red neuronal porque consta de una sola capa que contiene una sola neurona. Pero, como veremos a lo largo del libro, hoy en día nos encontramos con redes neuronales compuestas de decenas de capas, que contienen muchas neuronas que se comunican con las de la capa anterior para recibir información, y estas a su vez comunican su información a las neuronas de la capa siguiente.

Como veremos en el capítulo 7, hay varias funciones de activación además de la *sigmoid*, cada una con propiedades diferentes. Para el propósito de clasificar números escritos a mano, en este capítulo también les avanzaremos otra función de activación llamada *softmax*[92], que nos será útil para presentar un ejemplo de red neuronal mínima para clasificar en más de dos clases. Por el momento, podemos considerar a la función *softmax* como una generalización de la función *sigmoid* que permite clasificar más de dos clases.

4.3.2. Perceptrón multicapa

Ya hemos avanzado que una red neuronal está compuesta por varios perceptrones como el que acabamos de presentar. Para facilitar la representación gráfica de una red neuronal, podemos usar una forma simplificada de visualizar la neurona presentada en la Figura 4.8 como se indica en la Figura 4.9.

[92] Wikipedia, (2018). Softmax function [*online*]. Disponible en: https://en.wikipedia.org/wiki/Softmax_function [Consulta: 22/02/2018].

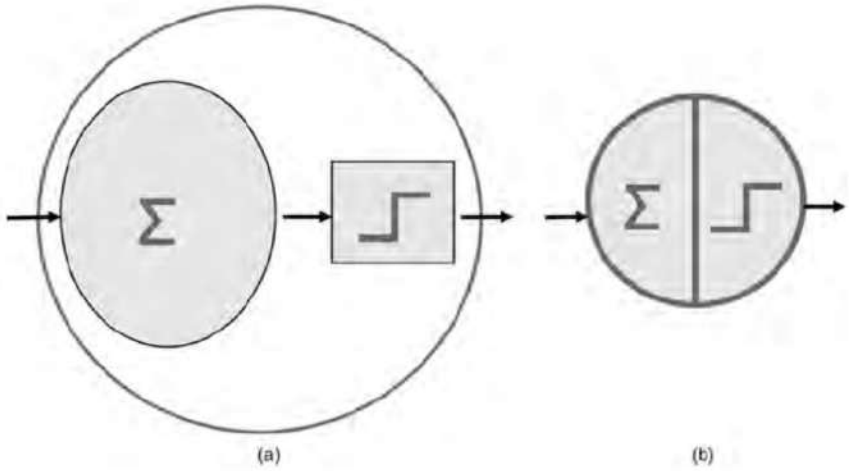

Figura 4.9 *(a) Representación simplificada de una neurona artificial que computa una suma ponderada de sus entradas y aplica a su resultado una función de activación. (b) Representación visual más sintética de las dos etapas de la neurona artificial.*

Cuando todas las neuronas de una capa están conectadas a todas las neuronas de la capa anterior (es decir, sus neuronas de entrada), la capa se denomina capa completamente conectada o capa densa. Las entradas de los perceptrones se alimentan de neuronas especiales llamadas neuronas de entrada, que emiten cualquier entrada con la que se alimenten. Todas las neuronas de entrada forman la capa de entrada. Además, como ya hemos avanzado, se agrega el sesgo b como una entrada adicional fijada a 1 (neurona de sesgo, que genera 1 todo el tiempo). Siguiendo todas estas indicaciones de notación, podemos representar una red neuronal con dos neuronas de entrada y tres neuronas de salida tal como se muestra en la Figura 4.10.

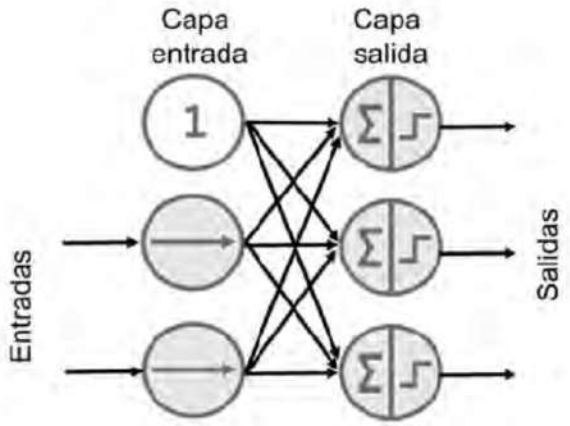

Figura 4.10 *Representación gráfica de una arquitectura de una red neuronal con dos neuronas de entrada, una neurona de sesgo y tres neuronas de salida.*

En la literatura del área nos referimos a un perceptrón multicapa, o *Multi-Layer Perceptron* (MLP), cuando nos encontramos con redes neuronales que tienen una capa de entrada (*input layer*), una o más capas compuestas por perceptrones llamadas capas ocultas (*hidden layers*), y una capa final con varios perceptrones llamada la capa de salida (*output layer*).

Ahora que ya hemos presentado con un poco más de detalle qué es una red neuronal, podemos concretar mejor a qué nos referimos cuando hablamos de Deep Learning. Nos referimos a Deep Learning cuando el modelo basado en redes neuronales está compuesto por múltiples capas ocultas. Visualmente se puede presentar con el diagrama de la Figura 4.11.

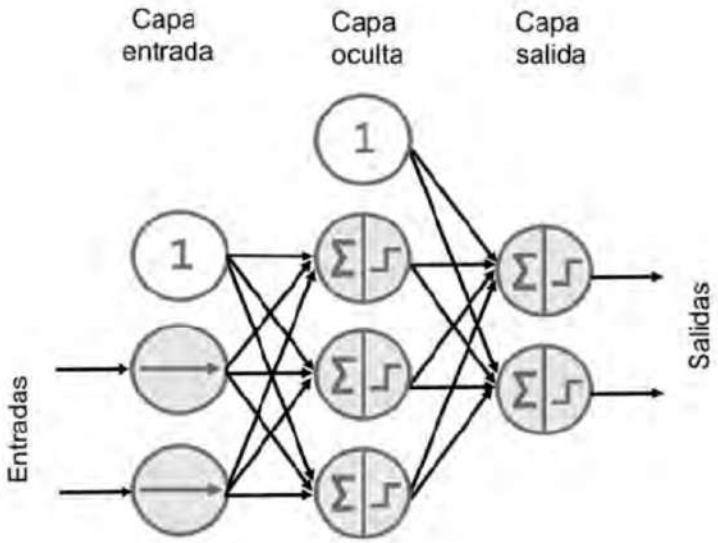

Figura 4.11 *Representación gráfica de un perceptrón multicapa con tres neuronas de entrada, una capa oculta con cuatro neuronas y una capa de salida con dos neuronas.*

Aun así, a veces se habla de Deep Learning en cualquier caso en el que intervienen redes neuronales.

Las capas cercanas a la capa de entrada generalmente se denominan capas inferiores, y las cercanas a las salidas generalmente se denominan capas superiores. Cada capa, excepto la capa de salida, incluye una neurona que representa el valor de sesgo y está completamente conectada a la siguiente capa aunque a menudo son ímplicitas y se obvian en las representaciones gráficas (Figura 4.12).

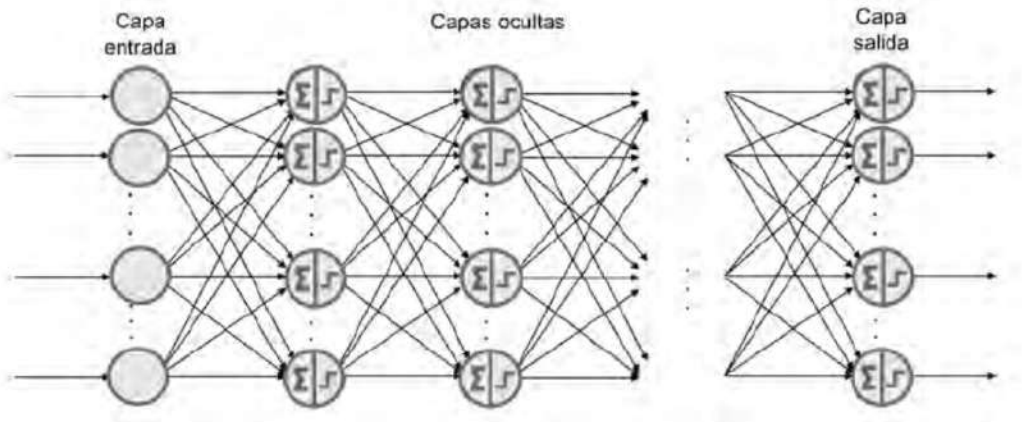

Figura 4.12 *Representación genérica de una red Deep Learning con muchas capas ocultas (donde se han obviado las neuronas de sesgo).*

4.3.3. Perceptrón multicapa para clasificación

Los MLP a menudo son usados para clasificación. Para un problema de clasificación binaria, solo se necesita una neurona de salida que utilice la función de activación logística *sigmoid* como la que hemos visto: la salida será un número entre 0 y 1, que se puede interpretar como la probabilidad estimada de una de las dos clases. La probabilidad estimada de la otra clase simplemente es igual a uno menos el valor que retorna la neurona.

En cambio, cuando queremos clasificar con más de dos clases y, en concreto, cuando las clases son exclusivas —como es el caso de la clasificación de imágenes de dígitos que nos ocupa (en clases de 0 hasta 9)—, se requiere una neurona de salida por cada clase, y debemos usar la función de activación *softmax* que nos garantiza que todas las probabilidades estimadas son entre 0 y 1 y que suman 1 (lo cual es necesario si las clases son exclusivas). Esto se llama clasificación multiclase (ver Figura 4.13), en la que la salida de cada neurona corresponde a la probabilidad estimada de la clase correspondiente.

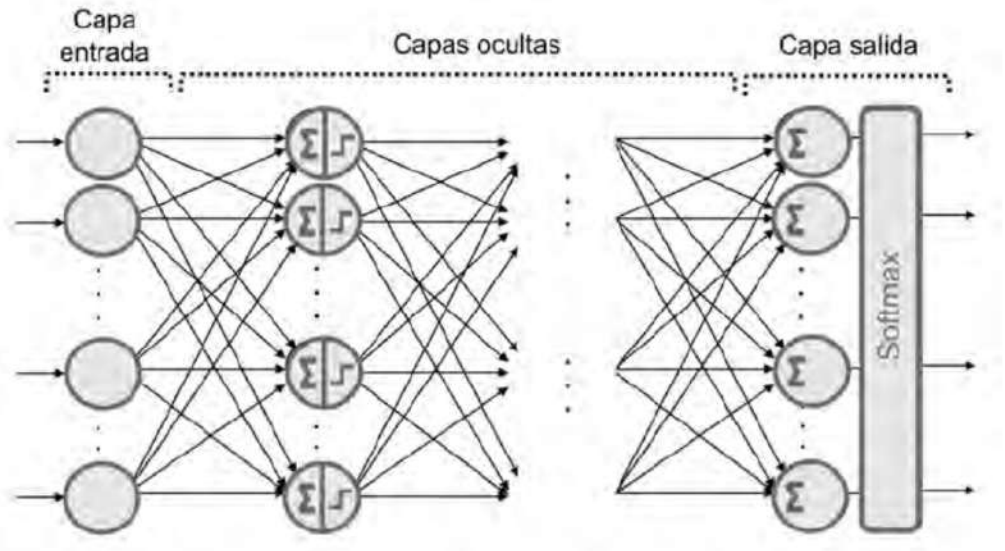

Figura 4.13 *Representación genérica de una red Deep Learning con muchas capas ocultas con una capa de salida con función de activación* softmax.

4.4. Función de activación *softmax*

De una forma muy visual, vamos a explicar cómo podemos resolver el problema de manera que, dada una imagen de entrada, obtengamos las probabilidades de que sea cada uno de los 10 posibles dígitos. De esta manera tendremos un modelo que, por ejemplo, podría predecir en una imagen un nueve; pero solo puede estar seguro en un 80 % de que sea un nueve ya que, debido al dudoso bucle inferior, piensa que podría llegar a ser un ocho en un 5 % de posibilidades, e incluso podría dar una cierta probabilidad a cualquier otro número. Aunque en este caso concreto consideraremos que la predicción de nuestro modelo es un 9, pues es el que tiene mayor probabilidad, esta aproximación de usar una distribución de probabilidades nos puede dar una mejor idea de cuán confiados estamos de nuestra predicción. Esto es bueno en este caso, donde los números son trazados a mano; seguramente en muchos otros casos no podemos reconocer los dígitos con un 100 % de seguridad.

Por tanto, para este ejemplo de clasificación de MNIST, para cada dato de entrada obtendremos como vector de salida de la red neuronal una distribución de probabilidad sobre un conjunto de etiquetas mutuamente excluyentes, es decir, un vector de 10 probabilidades —cada una correspondiente a un dígito—, y que todas estas 10 probabilidades sumen 1 (las probabilidades se expresarán entre 0 y 1).

Como ya hemos avanzado, esto se logra mediante el uso de una capa de salida en nuestra red neuronal con la función de activación *softmax*, en la que cada neurona en esta capa *softmax* depende de las salidas de todas las otras neuronas de la capa, puesto que la suma de la salida de todas ellas debe ser 1.

Pero ¿cómo funciona la función de activación *softmax*? La función *softmax* se basa en calcular «las evidencias» de que una determinada imagen pertenece a una

clase en particular, y luego se convierten estas evidencias en probabilidades de que pertenezca a cada una de las posibles clases.

Para medir la evidencia de que una determinada imagen pertenece a una clase en particular, una aproximación consiste en realizar una suma ponderada de la evidencia de pertenencia de cada uno de sus píxeles a esa clase. Para explicar la idea usaré un ejemplo visual con el número cero.

Supongamos que disponemos ya del modelo aprendido para el número cero —más adelante ya veremos cómo se aprenden estos modelos—. Por el momento, podemos considerar un modelo como «algo» que contiene información para saber si un número es de esa determinada clase. En este caso, para el número cero, el modelo aprendido correspondería visualmente al que presentamos en la Figura 4.14.

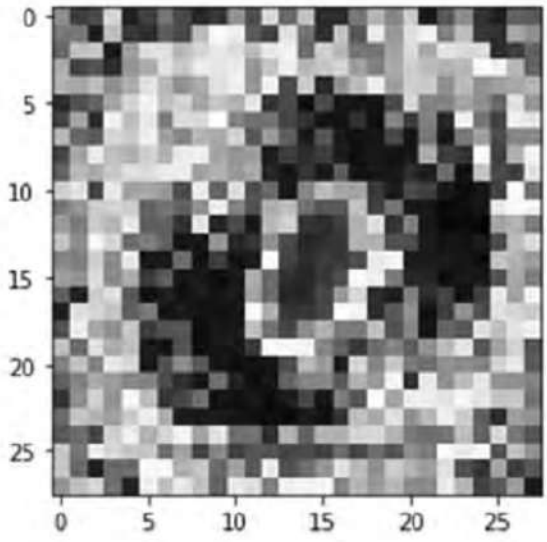

Figura 4.14 *Modelo aprendido correspondiente al número 0.*

En este caso se trata de una matriz de 28 × 28 píxeles, donde los píxeles en rojo (en la edición en blanco/negro del libro es el gris más claro) representa pesos negativos (es decir, reducir la evidencia de que pertenece), mientras que los píxeles en azul (en la edición en blanco/negro del libro es el gris más oscuro) representan pesos positivos (aumenta la evidencia de que pertenece). El color blanco representa el valor neutro.

En realidad, se trata de una representación visual (para facilitar la explicación) de la matriz de parámetros que corresponde al modelo de la categoría cero que ha aprendido la capa de salida para el ejemplo de MNIST. En la Figura 4.15 se puede ver esta matriz de números, y el lector o lectora puede comprobar la equivalencia de los valores en rojo y azul. En el GitHub del libro se presenta el código que muestra cómo se ha obtenido esta matriz visual de los parámetros que ha aprendido la capa de salida para el ejemplo de MNIST (por si el lector o lectora quiere comprobar cómo se ha obtenido y ver las imágenes en color).

```
[[ 22. -21.  11.  -8.   0. -17.  -4. -19.  12.  21. -10.  12. -20. -19.  11. -21.  14. -13. -21. -10.  17.  -5. -11.  15. -16. -19. -21.   4.]
 [ 10.   1.   2.   5.  20.  -5.  16. -19.  -1. -16.  -9.  25.  -9. -11. -48.   7. -10. -14. -15. -11.   6. -11.  17. -21.  20. -17. -20. -18.]
 [  1.  17.  16.   3. -19. -16.   3. -15. -33. -46. -56. -53. -42. -56. -99. -88. -104. -98. -86. -79. -42. -54. -51. -37.  -4.  13.  21.  17.]
 [ -4.  -7.   5.   5. -27.  -9. -17. -22. -12. -29. -46. -33. -50. -41. -31. -50. -13. -44. -82. -49. -66. -55. -78. -52. -34.  -4. -19. -14.]
 [ 11.  -4.  -7.  13. -60. -44. -68. -50. -38. -30. -23. -23.  -1. -10. -21.  13. -24. -13.  -6. -19. -21. -15. -21. -58. -61. -34. -63. -33.]
 [ 16.  -8. -30. -21. -72. -46. -45. -38. -28. -22. -23. -21. -13.   3.  43.  32.  15.  19.  21.  -1. -16.   7. -21. -44. -81. -92. -66. -14.]
 [-21.  10. -31. -49. -46. -33. -36. -36. -10. -36. -15.   4.  -6.   9.  32.  26.  24.  55.  44.  16.  29.   6.  53.  -2. -82. -142. -58. -24.]
 [ -7. -26. -23. -46. -89. -16.  -3. -37. -37. -31.  -1.  18.  17.  24.   9.  16.  28.  42.  36.  38.  22.  16.  16.  19. -50. -107. -89. -12.]
 [ 29. -38. -21. -23. -62. -50. -33. -27. -33.  -6. -14.  -2.   7.  -2.  10.  28.  42.  62.  60.  23.   1.  18.  32.  26. -43. -155. -98. -39.]
 [ -8. -13.  -1. -42. -50. -24. -43. -41. -24.   8.   8.   1.  19.  34.  18.   7.  21.  42.  47.  43.  19.   7.  34.  46.   6. -145. -89. -33.]
 [  2. -10.   9. -27. -32. -16. -10. -11. -10.  -9. -24. -15.  16.   7.   2. -28. -24.  27.  44.  28.  11.  42.  22.  40.  23. -96. -94. -34.]
 [ -9.  -1.   1. -53. -62. -39. -14. -18. -11. -23. -31.  -1. -15. -22. -71. -92. -100. -47.  -7.   9.   8.  19.  54.  40.  60. -71. -95. -49.]
 [ 10.   8. -32. -22. -42. -13.  21.   4.  -7.   5.  17.  13. -14. -65. -112. -156. -118. -64. -31.   8.  23.  25.  57.  68.  81. -17. -39. -27.]
 [  4.   1.   5. -22. -14. -12.  27.  11.  -3.  15.  11.  15.  -4. -68. -132. -165. -133. -56.  -9. -20. -17.  15.  46.  47.  62. -12. -61. -38.]
 [ 12. -24. -12. -38. -38.  29.  24.  57.  27.  12.  21.   5. -36. -103. -152. -139. -112. -70. -13. -22.  -7.  12.  49.  67.  63. -15. -40.  -6.]
 [-18.  -5. -51. -44.   8.  34.  25.  44.  36.  53.  54.   1. -45. -118. -181. -130. -104. -55. -14. -22.  14.  11.  22.  39.  32. -27. -43. -18.]
 [-17. -20. -20. -54.   0.  30.  30.  47.  26.  60.  45.  -9. -72. -130. -129. -99. -56.  -3.   1.  19.  -8.  18.   5.   7.   9. -49. -44. -20.]
 [ 20. -19. -33. -68.   0.  13.  25.  45.  13.  47.  41.  -7. -74. -144. -119. -89. -48.  -7.   2.  -6.  20.   6.  26.   2.   9. -39. -36. -10.]
 [-23.  14. -40. -85. -11.  -1.   4.  34.  17.  62.  73.   8. -73. -92. -92. -38. -12.   2. -18.   3.  -2.  20.  22.   1. -38. -30. -18. -29.]
 [  9.  17. -11. -74. -31.   3.  27.   2.  34.  48.  37.  31.   9. -51. -12. -19.  -8.   7. -33.  -5. -30.  -7.   8. -11. -37. -47. -47. -37.]
 [-22.  -4. -32. -54. -19.  12.  21.  -8.  23.  16.  39.  55.  46.  -3.  11. -16.  -0. -28. -19. -14.  -1.  14. -18. -10. -57. -30.   5.   1.]
 [-12. -25. -30. -42. -11.  -7.  13.   5.  21.  10.  50.  54.  18.  34.   4.  20.  -7. -22. -15. -49.  -7. -30.  -4. -19. -52. -34.   9.  10.]
 [-17.   2.  -5. -31. -75. -15.  20.  -3.  19.  11.  49.  49.  38.  48.  21.  10.  -7. -12. -20. -16. -41. -27.  -7. -17. -18. -21.  -4.  10.]
 [ -3. -13. -26. -56. -77. -46. -24.  -1.  12.  36.  49.  47.  44.  51.  61.  24.   8.   4. -36. -51. -52. -52. -27. -75. -49. -15.   1.  -9.]
 [ 14. -16. -45. -28. -31. -45. -64. -52. -30. -18.   4.  16.  24.   7.   5. -13. -18. -64. -84. -71. -99. -99. -66. -56. -42.  -5.   4.   9.]
 [-22. -14.   3. -15. -29. -46. -51. -75. -103. -105. -129. -119. -125. -121. -142. -119. -136. -117. -102. -86. -76. -80. -30. -65. -34.  -0. -20.  -7.]
 [-13.   6.   3. -11.   3. -52. -21. -32. -73. -100. -87. -98. -89. -85. -98. -72. -93. -69. -82. -39. -53. -62.  -6.   4. -22.   0.  18.  10.]
 [ -9.  15.   3.  15.  -8. -33.  12. -24. -25. -14. -40. -24. -46. -45. -38. -51. -13. -19. -41. -10. -45. -25. -14.  -6.  12.  21. -13. -19.]]
```

Figura 4.15 *Matriz de parámetros correspondiente al modelo de la categoría cero.*

Ahora que tenemos el modelo visual, imaginemos una hoja en blanco encima sobre la que trazamos un cero. En general, el trazo de nuestro cero caería sobre la zona azul (recordemos que estamos hablando de imágenes que han sido normalizadas a 20 × 20 píxeles y, posteriormente, centradas a una imagen de 28 × 28). Resulta bastante evidente que si nuestro trazo pasa por encima de la zona roja lo más probable es que no estemos escribiendo un cero; por tanto, usar una métrica basada en sumar si pasamos por zona azul y restar si pasamos por zona roja parece razonable.

Para confirmar que es una buena métrica, imaginemos ahora que trazamos un tres. Está claro que la zona roja del centro del anterior modelo que usábamos para el cero va a penalizar la métrica antes mencionada puesto que, como podemos ver en la Figura 4.16, al escribir un tres pasamos por encima de zonas rojas. Pero en cambio, si el modelo de referencia es el correspondiente al 3 —como el mostrado en la parte izquierda de la Figura 4.16—, podemos observar que, en general, los diferentes posibles trazos que representan un tres se mantienen mayormente en la zona azul.

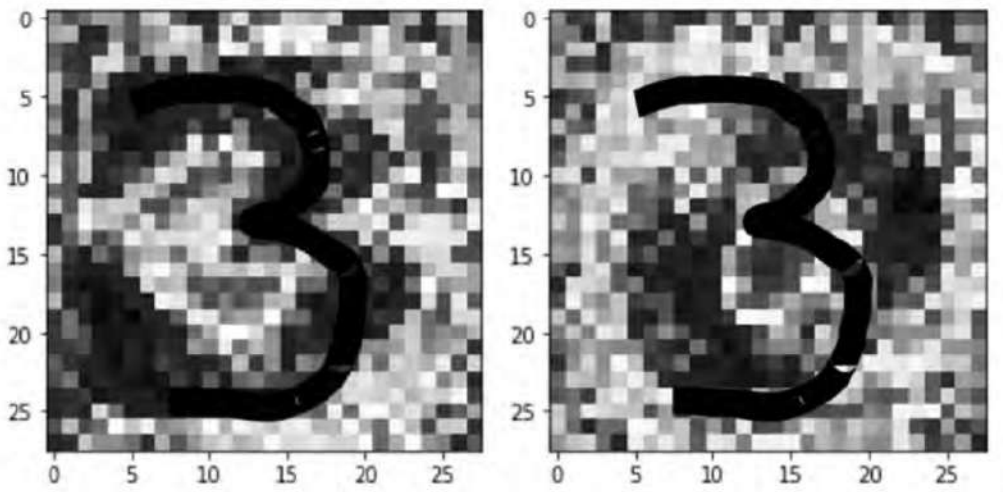

Figura 4.16 *Comparativa del solapamiento del trazado del número 3 con los modelos correspondientes al número 3 (izquierda) y al cero (derecha).*

Una vez se ha calculado la evidencia de pertenencia a cada una de las 10 clases, estas se deben convertir en probabilidades cuya suma de todos sus componentes sume 1. Para ello, *softmax* usa el valor exponencial de las evidencias calculadas y, luego, las normaliza de modo que sumen uno, formando una distribución de probabilidad. La probabilidad de pertenencia a la clase *i* es:

$$Softmax_i = \frac{e^{evidencia_i}}{\sum_j e^{evidencia_j}}$$

Intuitivamente, el efecto que se consigue con el uso de exponenciales es que una unidad más de evidencia tiene un efecto multiplicador y una unidad menos tiene el efecto inverso. Lo interesante de esta función es que una buena predicción tendrá una sola entrada en el vector con valor cercano a 1, mientras que las entradas restantes estarán cerca de 0. En una predicción débil tendrán varias etiquetas posibles, y todas ellas tendrán una probabilidad parecida. En el siguiente capítulo lo analizaremos con más detalle con un ejemplo de código.