Programación de redes neuronales con libfann

JUEGO CEREBRAL

3, 4, 8, 11...? Una red neuronal puede completar esta serie sin conocer el algoritmo subyacente, mediante una especie de sentido intuitivo virtual. Os mostramos cómo las redes neuronales resuelven problemas mediante la simulación y el comportamiento del cerebro humano. **ANDREAS ROMEYKE**

Si buscamos una ruta en un mapa, nuestros ojos irán casi directamente hasta la solución más eficiente. El cerebro humano es capaz de tomar decisiones sin prestar mucha atención a los algoritmos de optimización o los cálculos de distancias. Este método intuitivo está muy alejado de los ordenadores digitales. Los programas convencionales de ordenador suelen operar mediante soluciones matemáticas, lo que los hace inefi-

cientes para tareas como la predicción y reconocimiento de patrones. Un tipo de programa experimental conocido como *Red Neuronal Artificial*, *RNA*, resuelve este problema intentando que el ordenador funcione de manera parecida a la del cerebro humano.

Una red neuronal artificial simula un conjunto de células nerviosas conectadas mediante rutas ponderadas. Uno de los usos de estas redes neuronales en las que ha tenido éxito es el campo del reconocimiento de caras. Una red neuronal puede reconocer una cara en base a un conjunto de píxeles coloreados, a pesar del ruido o la distorsión, de manera similar a un humano. Otras aplicaciones de la tecnología de las redes neuronales incluyen el reconocimiento óptico de caracteres o las predicciones de la actividad del Sol o de la variación del precio de las acciones de bolsa.

Kiyoski Takahase Segundo, Fotolia

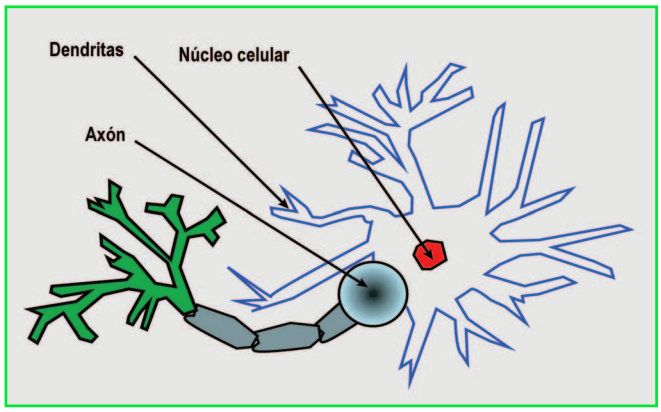

Figura 1: Los neurocientíficos fundamentan la capacidad del cerebro humano para reconocer patrones o predecir estados de sistema que son difíciles de calcular en la disposición ramificada de las células nerviosas.

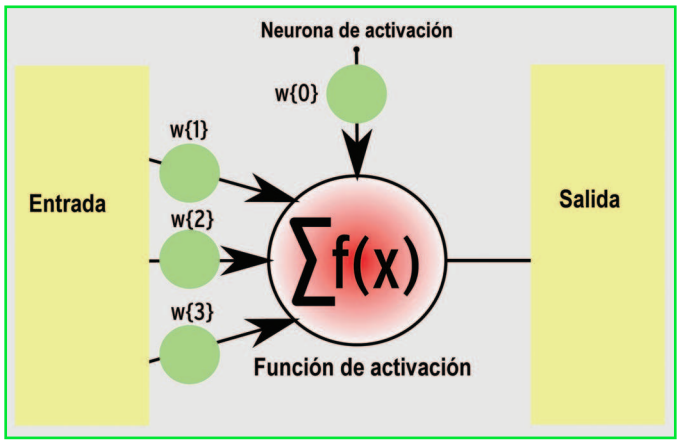

Figura 2: Al igual que su contraparte natural, las neuronas artificiales calculan la suma de los potenciales de actuación de las neuronas enlazadas a ella y pasan la señal a otras neuronas a través de conexiones ponderadas. La modificación de los diversos factores de la conexión influirá en el comportamiento de la red neuronal.

En este artículo daremos un repaso a algunos de los principios básicos de las redes neuronales y presentaremos la librería libfann, desarrollada en software libre, que podemos usar para construir nuestras propias aplicaciones de redes neuronales.

Modelo de Funcionamiento Natural

Las redes neuronales artificiales simulan la estructura del cerebro. Una red neuronal modela el efecto de un conjunto de neuronas que influyen unas en otras mediante un gran número de conexiones. Damos diferentes pesos a las conexiones neuronales, que representan las fibras de los nervios del cerebro, para generar un valor dado como respuesta a un patrón específico de las neuronas de entrada. Las conexiones entre las neuronas reciben un ajuste fino mediante un proceso denominado *training* (entrenamiento). Mediante este proceso de entrenamiento la red neuronal aprende a asociar patrones específicos de entrada con valores específicos de salida. Si el entrenamiento ha tenido éxito, nuestro cerebro artificial será capaz de descubrir soluciones que no se le presentaron específicamente como ejemplos.

La Figura 1 muestra una célula nerviosa en un modelo natural de la neurona utilizada en RNAs. Las células nerviosas comprenden un núcleo celular y dendritas que salen de ella. Estas dendritas transportan los impulsos eléctricos al núcleo de las células. Si la suma total de estos impulsos supera un umbral predefinido (potencial de acción), la neurona se activa, enviando impulsos a las células a las que está conectada.

Una neuronal artificial simula las propiedades de su hermana natural: Suma los potenciales de sus dendritas, aplica una función fija de activación y pasa el resultado a las células a las que está conectada (véase la Figura 2). Los enlaces a otras neuronas reciben un peso para atenuar o amplificar la señal transmitida por esa ruta.

La *función de activación* define el umbral en el cual se activará la neurona. Por debajo de este valor la neurona no enviará ninguna señal. Esta función a menudo es una simple función de umbral que devuelve un *1* si la suma de todas las salidas está por encima de un valor específico. La función de activación se suele representar como una neurona separada conocida como *on neuron* o neurona de activación. Podemos así asignar el peso a esta neurona del mismo modo que a los enlaces de otras neuronas.

Diseño

El proceso de entrenamiento adapta la red neuronal ante una situación específica. Sin embargo, a nivel estructural, el desarrollador también debe elegir una topología para la red neuronal que refleje el uso para el cual se está diseñando. Los distintos tipos de enlaces entre las neuronas dan como resultado redes con características diferentes [2].

Una de las topologías de red más sencillas y mejor estudiadas por la comunidad

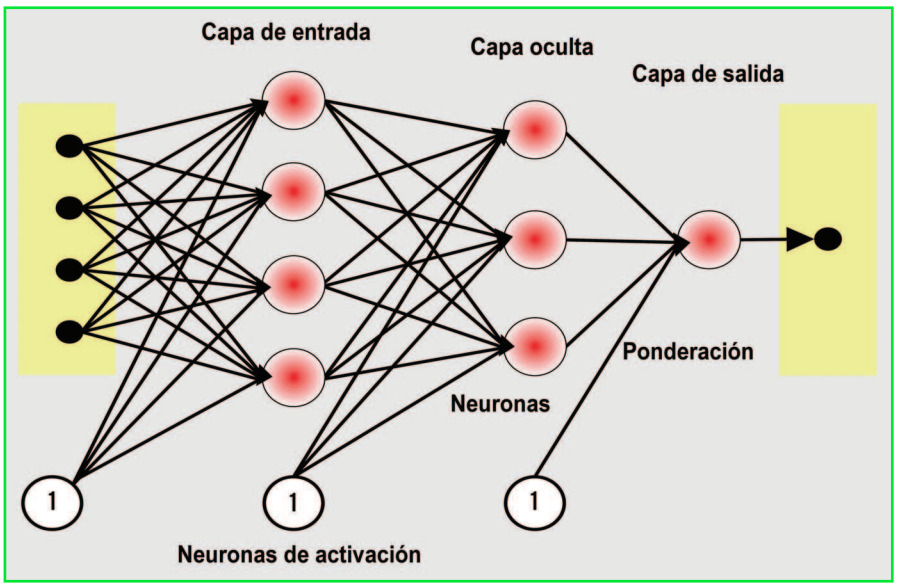

Figura 3: El perceptrón multicapa, que permite a los potenciales propagarse desde las entradas a las salidas sin bucles de retroalimentación, es la estructura de red neuronal artificial más sencilla y mejor estudiada.

científica es el modelo feed forward Multi-Layer Perceptron (MLP) [3]. Este modelo divide la red en capas separadas. Esta red no tiene retroalimentación, en otras palabras: el potencial de actuación simplemente se propaga de izquierda a derecha (véase la Figura 3).

La capacidad de una red neuronal, como la posibilidad de reconocer patrones o predecir valores, es resultado de la estructura interna de la red.

Las siguientes operaciones cambian las características de una red neuronal artificial:

- añadir nuevas conexiones o eliminar existentes
- modificar los pesos de los enlaces entre neuronas
- modificar los valores umbral de las neuronas
- añadir o eliminar neuronas

El entrenamiento proporciona los pesos adecuados para resolver un problema específico. En el caso de reconocimiento de caracteres, la entrada podría ser un mapa de bits o un fragmento de un texto y los caracteres coincidentes. En el caso del comportamiento de la bolsa o de la actividad del Sol, se usan datos históricos para entrenar a la red neuronal (véase la Figura 6). Una función de aprendizaje compara los ejemplos de entrada con los valores objetivo y modifica los enlaces de las neuronas, ajustando los pesos hasta que la reacción de la red se ajusta al objetivo.

Dentro de la Mente

Usaremos un sencillo ejemplo para describir lo que ocurre en una red neuronal en la fase de aprendizaje. Imaginemos que queremos una red con cuatro neuronas para predecir el valor medio de dos números (véase la Figura 4). A la izquierda de la figura, los números *0.1* y *0.3* alimentan a las neuronas de entrada. Los enlaces entre neuronas tienen pesos aleatorios en un primer momento. La función de activación, que define la manera en la que reacciona una neurona ante su entrada, es $f(x) = x$. Las redes neuronales artificiales más potentes necesitan obviamente funciones más complejas, pero este sencillo ejemplo es adecuado para explicar el principio subyacente.

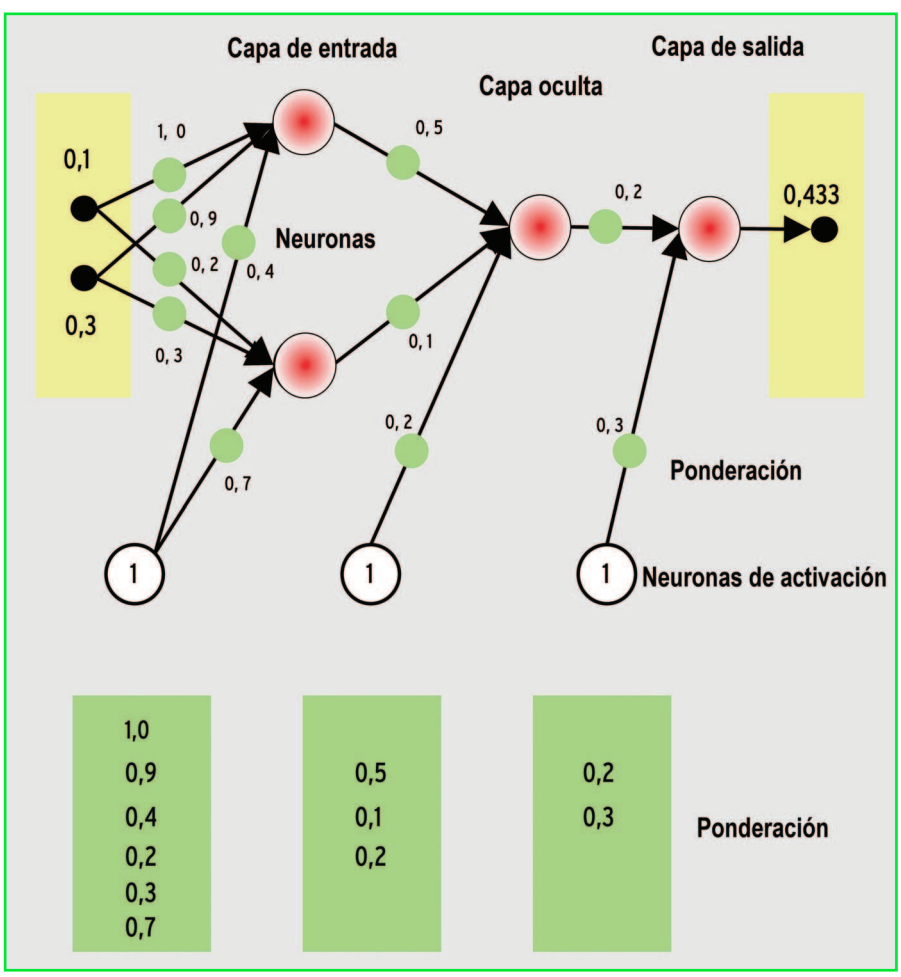

Figura 4: El comportamiento de una red neuronal se define por la ponderación de los enlaces y de las neuronas de activación, que fijan el umbral a partir del cual las neuronas pasarán un estímulo a las siguientes.

Si las neuronas de entrada tienen los valores *0.1* y *0.3*, al ponderar las conexiones obtenemos los siguientes valores: *(0.1*1.0 + 0.3*0.9 + 1.0*0.4) = 0.77* para la primera neurona *N(1.1)*, y *(0.1*0.2 + 0.3*0.3 + 1.0*0.7) = 0.81* para la segunda neurona *N(1.2)*. La neurona situada entre las capas de entrada y salida tienen un valor de *(0.77*0.5 + 0.81*0.1 + 1.0*0.2) = 0.666*. La neurona de salida devuelve un valor de: *0.666*0.2 + 1.0*0.3) = 0.433*, a pesar de que la media correcta entre los números *0.1* y *0.3* es *0.2*.

En otras palabras, la red neuronal no se ha quedado demasiado cerca del valor correcto en esta primera pasada. Para permitir que se acerque al resultado correcto tenemos que modificar la ponderación de los enlaces neuronales. Las contribuciones al error nos permiten descubrir qué pesos de qué neuronas tenemos que corregir. La contribución del error es la raíz cuadrada del valor esperado de salida, menos la raíz cuadrada de los valores obtenidos en las neuronas de salida. Este valor se conoce como valor cuadrático medio (MSE ó MQLE).

Marcha Atrás

Los potenciales de acción generalmente se mueven hacia adelante a través de la red desde la entrada hasta la salida (feed forward). Un método de aprendizaje conocido como propagación hacia atrás va, sin embargo, en dirección contraria: alimenta a toda la red hasta la entrada con el valor del error devuelto por la salida en función de la ponderación de las conexiones individuales. La distribución de los valores del error sobre los nodos de la red proporciona la base para modificar la ponderación. (Los expertos han desarrollado muchas otras técnicas además de la propagación hacia atrás, así como otros métodos que prometen mejores resultados para ciertas tareas).

La Figura 5 muestra cómo la contribución del error se propaga hacia atrás desde la salida hasta la entrada. El potencial de la neurona de salida es la suma de sus dos enlaces: el enlace a la neurona de activación, con un peso de *0.3*, y la conexión a la neurona de la capa subyacente, que tiene un peso de *0.2*. En función de esto, la contribución del error *(0.433 - 0.2 = 0.233)* en

la neurona de salida se distribuye sobre los dos enlaces. La ruta hasta la neurona de activación tiene una cuota del *0.3/(0.2+0.3) = 60%*, mientras que la ruta hasta la neurona subyacente tiene una cuota del *0.2/(0.2+0.3) = 40%*. Este método permite calcular el potencial del error total para cada enlace neuronal.

Por último, un factor fijo de aprendizaje dicta cómo influye una contribución de error a la ponderación. Una buena elección del factor de aprendizaje es un requisito importante para un entrenamiento efectivo. Al igual que muchos otros parámetros de la red, este factor a menudo es desconocido hasta que comenzamos el aprendizaje. Para completar el entrenamiento de una red neuronal artificial tenemos que llevar a cabo un gran número de ciclos de retroalimentación con parejas de valores de entrada y salida para el problema que necesitamos resolver.

Plan de Entrenamiento

Obviamente las ponderaciones nunca deberían ser cero, ya que no habría manera de trazar los errores hacia atrás. Una ponderación demasiado concentrada o demasiado dispersa también tienen un efecto negativo en el proceso de aprendizaje. Para que una red neuronal artificial sea eficiente es preferible que las señales se propaguen a través de toda la red, excepto por algunas áreas específicas de la misma que deban controlar patrones específicos.

En lo que se refiere a aplicaciones prácticas, la sencilla función de activación *f(x) = x* se reemplaza por una tangente hiperbólica o una función sigmoidal. Esto mejora el rendimiento de la red neuronal, ya que estas funciones pueden mapear un rango de entrada infinito hasta valores controlables. La retroalimentación también supone que la función de activación puede invertirse. La ventaja de este esfuerzo extra es que los perceptrones de tres capas son capaces de aprender arbitrariamente, funciones matemáticas, suponiendo que usan funciones de activación no lineales adecuadas.

La Librería FANN

La Fast Artificial Neural Network Library (FANN) es una librería desarrollada bajo software libre que proporciona una interfaz en C para implementar redes neuronales multicapa. Esta librería fue desarrollada en 2003 por Steffen Niessen en la Universidad de Copenhague como parte de sus investigaciones científicas, y aún está bajo desarrollo activo. Libfann es fácil de utilizar, está bien documentada y funcionará en cualquiera de las principales plataformas. La página del proyecto dispone también de un par de ejemplos prácticos que facilitan cómo iniciarse en ella. Aparte de C, existen enlaces para todos los lenguajes de programación más usados. En el momento de escribir este artículo, libfann es una de las implementaciones más rápidas para la simulación con redes neuronales.

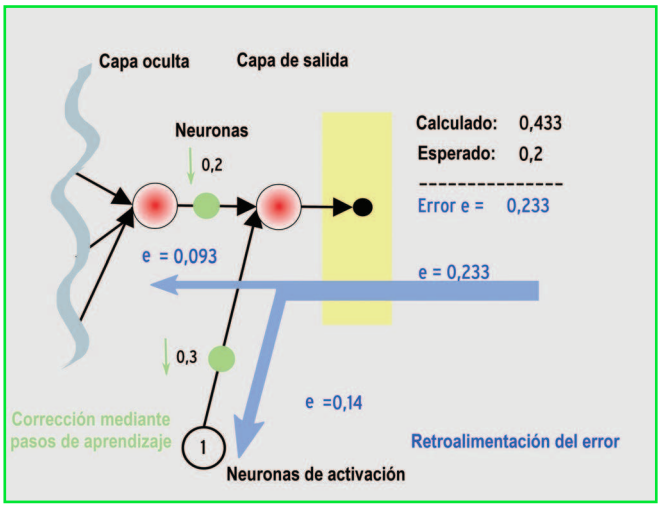

Figura 5: Las redes neuronales aprenden cometiendo errores al adivinar valores específicos, trazando esos errores hacia atrás a través de toda la estructura, y volviendo a ponderar los enlaces individuales a las neuronas en función de la contribución de ese error en comparación con el error total.

La mayoría de las distribuciones de Linux incluyen la versión 1.2 de libfann. Con el comando *aptitude install libfann1-dev* podemos instalarla en Debian. Encontraremos el código fuente en [1], código que podemos compilar de la manera usual: *configure; make; make install*.

Compilar a Medida

Todas las aplicaciones con redes neuronales son diferentes, resultando imposible explorar todos los matices de este complejo campo en un solo artículo. La página del proyecto libfann incluye un manual de referencia con la descripción y notas de uso de las funciones de la librería. En la página Web de *Linux Magazine* [4] encontraremos un programa de ejemplo escrito en C que crea una red neuronal y luego la entrena.

Si le apetece experimentar, algunas de las funciones más importantes de libfann son: *fann_train()*, *fann_run()* y *fann_test()*. *fann_train()* aguarda una estructura de red, siendo*struct fann * ann;* su primer parámetro. Podemos realizar la llamada *ann = fann_create(connection_rate, learning_rate, num_layers,num_input, num_neurons_hidden, num_output);* para crear la estructura. El parámetro *connection_rate* especifica la intensidad de los enlaces entre neuronas. Un valor adecuado suele ser *1.0*. El valor de *learningrate* debería estar entre *0.7* y *0.00001*. El parámetro *num_layers*, y los valores que le siguen, indican a libfann el número de capas de la red y el número de neuronas de cada capa.

Entrenamiento y Pensamiento

Los paralelismos con el pensamiento de los humanos son útiles para comprender lo que ocurre durante el proceso de aprendizaje y para descubrir la fuente de cualquier problema de la red neuronal. Después de todo, una red neuronal intenta emular la estructura del cerebro humano. Si la sesión de entrenamiento se alimenta con datos históricos en orden cronológico, la red podría desarrollar una visión de túnel. En otras palabras, la red neuronal artificial simplemente codificaría la estructura del primer ejemplo, o primera pareja de ejemplos en sus neuronas. Esto afectaría a la capacidad de la red neuronal para manejar nueva información. Un orden aleatorio evitaría la generalización prematura, y de esta manera tener que volver a entrenar la red neuronal después de que una estructura inválida se haya establecido en los enlaces neuronales. El script en Perl [5] asegura el orden aleatorio de la información.

La información que la red no ha visto durante el entrenamiento nos ayuda a valorar la capacidad de abstracción de la red neuronal en un momento dado del mismo. El script Perl [5] divide la información en subconjuntos. El error que aparece aquí se refiere al valor cuadrático medio generalizado, o MQGE. Junto con el error cuadrático medio de aprendizaje (MQLE),

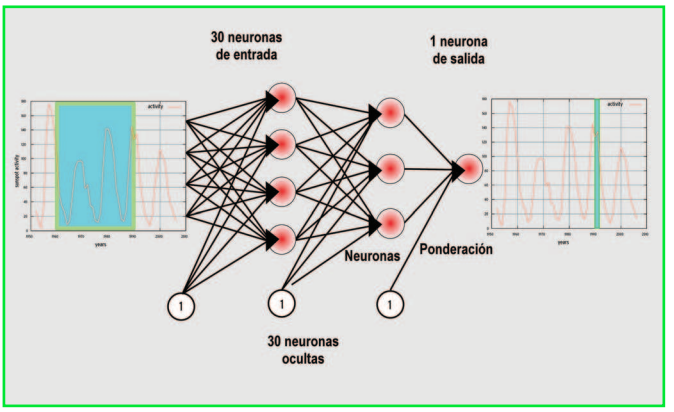

Figura 6: Una red neuronal con tres capas de neuronas aprovecha los datos de la actividad del sol en los últimos 30 años, que sirven para entrenar a las neuronas y poder predecir la intensidad del sol para el año que viene.

indica si la red neuronal está preparada para predecir el futuro, o si necesita aún más entrenamiento. Libfann inyecta los dos valores mediante las funciones *fann-test(ann, inputarray, expected_outputarray)* y *fann_get_MSE()* . Por último, *fann_save(ann, filename)* guarda la estructura de red y los pesos actuales para su posterior uso.

Vida Real

Que el entrenamiento tenga éxito o no depende no sólo de la información y del orden de la información, sino también de que la estructura de la red sea adecuada para la tarea, empezando por la función de activación. Libfann usa la función sigmoidal por defecto, la cual es adecuada para predicción de la actividad solar y otros fenómenos que tienen un rango positivo. Para variaciones del precio de las acciones y otras series temporales que contienen valores negativos necesitaremos *fann_set_activation_function_output(ann, FANN_SIGMOID)*, la función tangente hiperbólica.

El factor de aprendizaje también tiene una importante influencia en el éxito o fracaso del entrenamiento, ya que especifica el efecto de los errores del aprendizaje sobre la ponderación de los enlaces entre las neuronas y en el número de neuronas de la red. El número de neuronas de la capa intermedia debería mantenerse al mínimo en un primer momento. Entre tres y un máximo de 15 neuronas nos servirán para la mayoría de las aplicaciones. Mediante ensayo y error también podemos acercarnos intuitivamente al número adecuado. Para esta sesión de entrenamiento, 500.000 pasos de aprendizaje deberían ser suficientes.

Si el número de errores en el aprendizaje no baja de manera continua, la red se ha atascado en un mínimo local, y su rendimiento no mejorará por mucho que continuemos en entrenamiento. En este caso tendremos que reiniciar el entrenamiento con un factor de entrenamiento menor, y probablemente tengamos que cambiar la estructura de la red. Si estudiamos el archivo *fann_save* podremos averiguar por qué no podemos mejorar el rendimiento de la red mediante el entrenamiento. A menudo neuronas concretas con un peso excesivo interfieren en el proceso de aprendizaje.

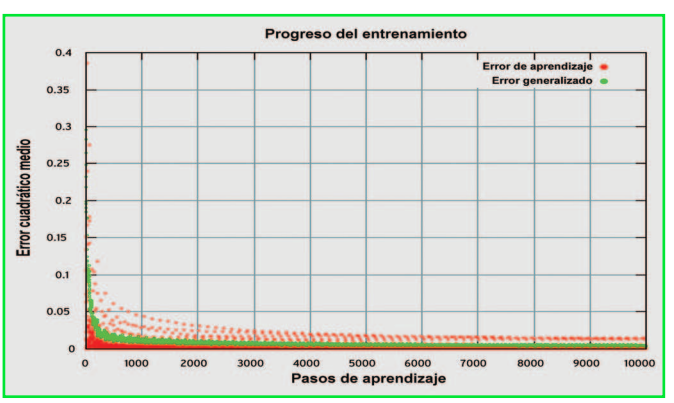

Figura 7: Si el entrenamiento ha tenido éxito, el error de aprendizaje (MQLE) descenderá de forma continua. Al mismo tiempo, la red neuronal continúa mejorando su capacidad para extraer los valores. Esto se representa como un descenso del valor del error generalizado (MQGE).

Si el error en el aprendizaje continúa bajando, como se muestra la Figura 7, ya podemos acudir al error generalizado. Si la curva es suave, no podemos esperar una buena capacidad predictiva. La red ha aprendido los valores de entrenamiento a fuego y se le atragantan los valores desconocidos de entrada. Para cambiar esto tendremos que reducir el número de neuronas ocultas.

Si el error generalizado se mantiene en un nivel consistentemente alto, el número de nodos ocultos es demasiado bajo o la sesión de entrenamiento no ha sido suficientemente intensiva.

La función *fann_load* de Libfann carga una red guardada previamente mediante *fann_save()*. Por otro lado, *fann_run(ann, input)* devuelve el valor procesado por la red entrenada. El script Perl [5] automatiza una prueba de la red neuronal. En esta prueba, la salida de una red correctamente entrenada se acerca bastante a las predicciones.

Conclusiones

Libfann facilita la configuración, el entrenamiento y el uso de las redes neuronales artificiales. El usuario no tendrá que preocuparse de los detalles matemáticos, como invertir la función de activación. Elegir los parámetros adecuados para la tasa de aprendizaje y el número de neuronas intermedias puede llevar algo de práctica y paciencia. El error de aprendizaje, el error generalizado y la comprensión de la saturación de neuronas individuales nos proporcionarán algunas pistas de por qué está fallando el entrenamiento de una red específica. Libfann nos ayudará a averiguar estos valores.

La versión actual 2.0 de la librería libfann extiende el alcance funcional, añadiendo nuevos algoritmos de aprendizaje así como nuevos tipos de neuronas. ■

RECURSOS

[1] Libfann: *http://fann.sourceforge.net*

[2] Tipos de Redes Neuronales: *http://www.neuronalesnetz.de/netztypen.html*

[3] Perceptrón Multicapa: *http://en.wikipedia.org/wiki/Perceptron*

[4] Programa de entrenamiento en C: *http://www.linux-magazine.es/Magazine/Downloads/35*

[5] Script Perl para preparación de los datos: *http://www.linux-magazine.es/Magazine/Downloads/35*

[6] Warren Sarle. FAQ de comp.ai.neural-nets, *http://www.faqs.org/faqs/ai-faq/neural-nets*

[7] Russell, Stuart y Peter Norvig. "Artificial Intelligence: A Modern Approach", 1ª ed. Prentice Hall, 1995.