Reconocimiento de lengua de signos usando redes neuronales

Juan Antonio Cano Salado
Manuel Caballero Sánchez

Trabajo realizado.

Se ha realizado una implementación en Java del perceptrón simple y de una red neuronal con número de entradas, salidas y capas ocultas configurable.
Disponemos de los siguientes algoritmos de entrenamiento: regla delta para entrenamiento del perceptrón simple, algoritmo de retropropagación con momentum y descenso por gradiente.

Problemas resueltos.

Hemos resuelto 4 de los 5 problemas propuestos, en concreto:

- Distinguir una mano abierta de una mano cerrada.
- Distinguir una mano izquierda de una mano derecha.
- Distinguir número de dedos levantados de una mano derecha.
- Distinguir imágenes de las distintas vocales en el alfabeto del lenguaje de signos.

Para el último problema era necesaria la obtención de un conjunto de prueba y entrenamiento muy amplio además de que para algunos de los símbolos era necesario que el sistema fuera capaz de reconocer el movimiento de los dedos.

Proceso de obtención del conjunto de entrenamiento.

Para obtener el conjunto de entrenamiento hemos tomado fotografías de cada símbolo.
Las fotografías están hechas sobre un fondo blanco para resaltar los detalles. Hemos capturado alrededor de 10-15 fotografías de cada símbolo. Posteriormente hemos usado dos tercios de las mismas para el conjunto de entrenamiento y el resto para el de prueba.

Las fotografías están almacenadas usando el formato PNG, con una resolución de 30x30 y en escala de grises.

Estructuras de red utilizadas.

Todas las estructuras probadas han tenido 900 entradas (cada una es el nivel de gris de cada pixel de la imagen de 30x30, normalizado entre 0 y 1), y tantas salidas como grupos tenga el conjunto de entrenamiento utilizado.

Comenzamos experimentando con el perceptrón simple. Para el problema que tenemos que resolver no es adecuado usar una única capa de perceptrones (una red neuronal sin capas ocultas) ya que no se consigue un ajuste aceptable. Utilizando redes neuronales con alguna capa oculta se consiguen resultados mucho mejores (incluso una rapidez de convergencia mayor). La decisión final fue utilizar una única capa oculta con pocas neuronas.

Entrenamiento.

Para resolver cada uno de los problemas propuestos hemos probado distintos parámetros de entrenamiento y algoritmos. Los parámetros de entrenamiento han sido muy importantes para aumentar la velocidad de convergencia de los algoritmos de entrenamiento, en especial el factor de aprendizaje.

Para entrenar cada red neuronal hemos utilizado el siguiente método:

- Suponemos que queremos entrenar una red de manera que clasifique entre n tipos de objetos distintos. Todas las fotografías de un mismo tipo están bajo un mismo directorio, por ejemplo *"train/right/vowels/a/"* (todas las fotografías de entrenamiento que representan la vocal "a" con la mano derecha)

- Nuestra aplicación recorre las fotografías presentes en ese directorio, asignando a cada una de ellas una salida esperada común (ya que representan el mismo símbolo). Esta salida se compone de n valores de 0 a 1, donde un 1 en la posición i-ésima indica que esa instancia pertenece a la clase de objetos i y un 0 que no pertenece.

- Una vez tenemos cada instancia con su salida esperada generada pasamos a entrenar la red.

El método de entrenamiento utilizado ha sido el de retropropagación por descenso por el gradiente (*batch*), que realiza una actualización de pesos cada vez que se ha propagado el delta de cada una de las instancias de entrenamiento. Se ha escogido este frente al delta (*incremental*) porque realiza menos actualizaciones de los pesos de la red.

Con la red neuronal entrenada, se interpretan los valores de salida de nuevas instancias de la siguiente manera: para una salida de n valores donde el valor i-ésimo es próximo a 1 consideramos que la red clasifica la nueva instancia como de la clase de objetos i.

Hemos observado que para este problema no es necesaria una estructura de la red muy compleja para conseguir buenos resultados en la clasificación de nuevas instancias.

Generación de los pesos iniciales y factores de aprendizaje.

La generación de los pesos iniciales de cada nodo de la red neuronal se hace usando valores aleatorios pequeños, por tratarse de algoritmos de búsqueda local siempre es bueno cambiar el punto de partida para evitar mínimos locales en el inicio.

El factor de aprendizaje se ha escogido por experimentación, comprobando la rapidez de convergencia del algoritmo de retropropagación. Se comprueba que para una estructura de red neuronal más compleja, se hace necesario tener un factor de aprendizaje más pequeño, ya que hay una mayor influencia de los errores a la salida en la retropropagación de los deltas.

Distintos criterios de parada.

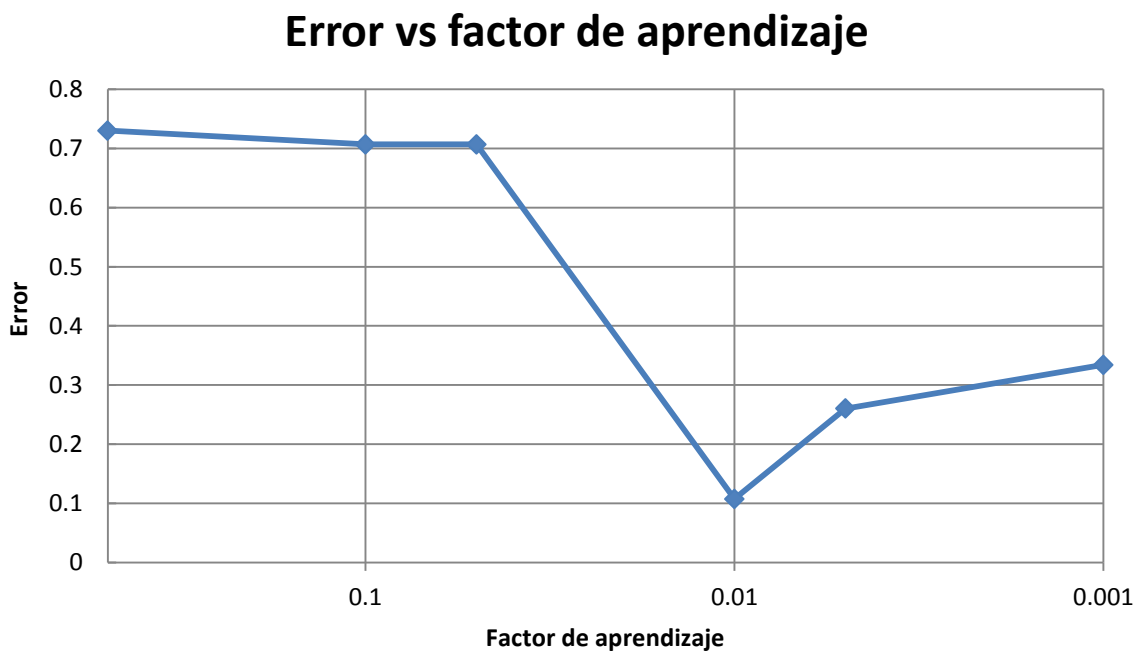
Para todos los algoritmos de entrenamiento usamos el mismo criterio de parada: tenemos tanto un límite de iteraciones como una cota máxima del error que comete la red al clasificar el conjunto de entrenamiento. Por tanto, la red neuronal se entrena durante el número fijado de iteraciones como máximo, parando antes si se consigue el error especificado.

Resultados.

Los resultados obtenidos tras el entrenamiento son bastante satisfactorios para haber usado una estructura tan simple para la red neuronal. Las redes son capaces de distinguir con bastante precisión las nuevas instancias, habiendo algunos errores en ocasiones en las que por ejemplo la mano se ha situado un poco más alejada de la cámara.

Consideramos relevante analizar la dependencia que tiene el error final del entrenamiento con el factor de aprendizaje. Para ello, hemos hecho algunas pruebas con diferentes valores del mismo para el primer ejemplo (mano abierta o mano cerrada), con un número fijo de iteraciones.

Este es el resultado:



Se puede observar que para valores altos del factor de aprendizaje, el algoritmo no es capaz de llevar los pesos hacia el mínimo, debido a que los cambios son muy grandes. Con esos factores el algoritmo se queda estancado en ese error, haciendo difícil (si no imposible) alcanzar valores más bajos.

Con valores más bajos del factor se observa una mejora en el error, ya que esta vez sí es capaz de dirigirse a valores bajos. Es necesario prestar atención para no seleccionar un valor demasiado bajo, ya que como se observa en la gráfica es posible obtener velocidades de convergencia más lentas.