Máster de Visión Artificial

Asignatura: Aplicaciones Industriales

Práctica puntuable – Reconocimiento Manuscrito

Orión García Gallardo

INTRODUCCIÓN

La escritura a mano ha persistido como medio de comunicación y almacenamiento de información a lo largo de los años incluso con la introducción de las nuevas tecnologías. Debido a la gran cantidad de transacciones que se hacen con la escritura a mano, el reconocimiento manuscrito de caracteres tiene vital importancia. En los últimos años se han conseguido resultados alentadores en el reconocimiento de textos tanto on-line [1–3] como off-line [4-6]. El reconocimiento de texto genérico parece todavía una tarea larga [7], sin embrago, se han llevado a cabo interesantes investigaciones en campos menos ambiciosos como reconocimiento de direcciones postales en sobres [8-9] o la cantidad legal en un cheque bancario[10-11].

La siguiente práctica implementa un sistema para el reconocimiento de números manuscritos. Para la realización de este trabajo se ha hecho uso de unos ficheros de ejemplo facilitados por la base de datos del NIST [12].

Este documento se divide en varias secciones. En la primera de ellas se describe el método desarrollado. A continuación se explica las pruebas realizadas sobre la muestra y las mejoras que se han introducido al ver los resultaos. Y por último se comenta las conclusiones obtenidas y los flancos que quedan abiertos después de la finalización de esta práctica.

DESCRIPCIÓN DEL MÉTODO DESARROLLADO

Entre las múltiples opciones a la hora de implementar esta práctica se opta por diseñan una red neuronal [13]. Las redes neuronales son una tecnología usada en inteligencia artificial que simula la forma en la que funciona el cerebro humano. Una serie de celdas que computan cálculos trabajan en paralelo con el objetivo de producir un resultado. Estas celdas son capaces de aprender por ellas mismas y, si una o más celdas de la red fallan, pueden reponerse relativamente bien.

Para la implementación de esta red neuronal se hace uso de las funciones que aporta la librería Encog [12]. Encog es un framework de dominio público de redes neuronales e inteligencia artificial disponible para Java, .Net y Silverlight. Encog contiene clases para crear una amplia variedad de redes y para normalizar y procesar datos para estas redes. Encog permite entrenar las redes neuronales usando diferentes técnicas, en esta práctica se usa la retropropagación (backpropagation). La retropropagación consiste en propagar el error hacia atrás, es decir, de la capa de salida hacia la capa de entrada, pasando por las capas ocultas intermedias y ajustando los pesos de las conexiones con el fin de reducir dicho error.

Para esta práctica se elige una red neuronal sin capas ocultas, lo cual puede provocar que sea poco tolerante al ruido en el entrenamiento, pero permite que la red sea mucho más sencilla y que el cálculo por iteración sea muchos más rápido. Cabe destacar que el sistema es

configurable y permitiría añadir fácilmente capas ocultas a la red neuronal tan solo añadiendo unos valores al xml de configuración.

Uno de los hándicaps a los que se hace frente al crear esta red neuronal es la cantidad de pixeles que tienen las imágenes a procesar. Las muestras de la base de datos NIST tiene imágenes de 28x28 pixeles lo que se traduce a 784 neuronas de entradas en la red. Con el objetivo de reducir este número de entradas y simplificar la red se dividen estas imágenes en regiones para posteriormente codificarlas a un número real entre 0 y 1. Posteriormente será el valor de cada una de estas regiones lo que se pasará como entrada a la red neuronal. Para simplificar el proceso se establece que estas regiones sean cuadradas, es decir, que tengan el mismo número de filas que de columnas. Así, por ejemplo, para reducir las imágenes a 7x7, o 49 entradas en la red neuronal, se dividen las imágenes en regiones de 4 filas y 4 columnas. Para la codificación de estas regiones se emplea la siguiente fórmula:

Sea a_{jk} el valor en escala de grises de la imagen en la fila j columna k.

El valor de la región 0 viene dada por:

$$Code_0 = a_{00} * 256^0 + a_{01} * 256^1 + \ldots + a_{10} * 256^4 + \ldots + a_{jk} * 256^{(4*j+k)} + \ldots + a_{33} * 256^{15}$$

Análogamente se realizan las mismas operaciones para las otras 48 regiones y se normalizan los valores entre 0 y 1 de la siguiente manera:

$$Region_i = Code_i / (256^{16} - 1)$$

De esta manera cada región viene determinada por un valor entre 0 y 1 generado a partir de los valores de los pixeles que lo forman y la identificaría de manera inequívoca.

En el xml de configuración se puede introducir el número de entradas de la red neuronal. Se ha de tener en cuenta que el número de neuronas elegido ha de ajustarse a una posible división de las imágenes en regiones cuadradas. Las posibles resoluciones serían: 28x28 (784 entradas), 14x14 (196 entradas), 7x7 (49 entradas), 4x4 (16 entradas), 2x2 (4 entradas) y 1x1 (1 entrada). Sin embargo, a la hora de ejecutar el sistema sólo se permitiría bajar hasta una resolución de 4x4 (16 entradas) porque con menores resoluciones los números $Code_i$ y $Region_i$ se saldrían de rango. Una vez determinado este número de entradas el sistema construye las regiones automáticamente y de manera opaca al usuario.

En cuanto a las salidas la representación que se ha realizado es más sencilla. Como la salida es un número entre 0 y 9 la red neuronal se representa con 10 salidas, poniéndose a 1 la correspondiente al número que se quiera representar. Así por ejemplo si la imagen representa el 5 el resultado correcto sería que la quinta salida valiese uno y el resto estuviese a cero.

El sistema permite dos tipos de ejecución: modo entrenamiento y modo test. En el primer modo lo primero que se lleva a cabo es, haciendo uso del algoritmo anteriormente explicado, la adaptación de la base de datos NIST de entrenamiento a la configuración de entradas seleccionada. A continuación se pasa a la creación de la red neuronal con las librerías de Encog. Esta red se puede reconstruir a partir de una existente en un fichero xml. Si este no existiese la red neuronal se crearía desde cero con las entradas que se seleccionaron en el

archivo de configuración. Cuando la red es creada desde cero se construyen también las capas intermedias con el número de neuronas que se indiquen en el archivo de configuración. Seguidamente se crea un entrenamiento de tipo retropropagación. Para crear este tipo de entrenamiento se hace uso del conjunto de datos de entrenamiento, que se tranformó a las entradas seleccionadas, del error de aprendizaje (learning error) y del momento (momentum) que se indiquen en el fichero de configuración. A partir de este instante empieza el entrenamiento iterándose tanta veces como se haya indicado en la variable de configuración. Finalmente se acaba mostrando cual ha sido el error mínimo del entrenamiento.

El modo de test se puede combinar con el entrenamiento para que se haga seguidamente a este o se puede configurar para que lea una red previamente entrenada. El primer paso de este modo es muy parecido al anterior y consiste en transformar la base de datos NIST de test para que tenga la configuración de entradas seleccionada. Finalmente, usando las librerías de Encog, se calcula el error obtenido con este conjunto de datos para la red y se muestra por pantalla.

PRUEBAS REALIZADAS SOBRE LA MUESTRA

Para realizar las pruebas se han tenido en cuenta tanto el número de entradas en la red neuronal como el aprendizaje del error en cada iteración. Como muestra la tabla (Tabla 1) se ejecuta el sistema durante unos 300 segundos aproximadamente y se registran los resultados obtenidos con las distintas configuraciones.

Input	Error Learn	Time (secs)	Iterations	Error (%)
784	0.00001	300	89	7,1973
196	0.00001	264	300	4,2693
49	0.00001	274	1000	4,7734
196	0.0001	300	348	2,1137
196	0.001	300	323	2,1820
196	0.0005	300	339	1,9428
196	0.00075	300	319	2,1719
196	0.00025	300	318	1,9286

Tabla 1. Entrenamiento de la red neuronal

Con la configuración que da mejores resultados se hace un entrenamiento más largo para observar cual sería un óptimo relativo del sistema. Seguidamente se ejecutan los datos de test para ver cuál es la precisión de la aplicación al intentar reconocer los números de imágenes de datos no entrenados (Tabla 2).

Input	Error Learn	Training Time (secs)	Iterations	Test Error (%)
196	0.00025	6146	6000	1,6395

Tabla 2. Resultados de la muestra de test sobre la red entrenada.

CONCLUSIONES

- Resumen de resultados obtenidos.
  Según los resultados obtenidos, con este sistema la configuración para la que se obtienen los mejores resultados es 196 entradas con un aprendizaje de error de 0,00025. Esto da un error del 1,92 % si se deja entrenando la red durante 5 minutos y llegando este porcentaje a ser inferior al 1,7 % cuando el entrenamiento es más largo. Incluso se puede observar que para las base de datos de test se obtienen mejores resultados llegando el acierto al 98,36%.
- Mejoras que podrían realizarse.
  La eficiencia del algoritmo presentado podría mejorarse para intentar alcanzar ese 100% ideal. Entre las cosas que se podrían probar para mejorarlo cabe destacar:
  - Introducir capas ocultas en la red neuronal.
  - Introducir en la red más entradas que sean características de los elementos a clasificar:
    - Densidad normalizada: Un valor entre 0 y 1 que mida la proporción de píxeles activos entre píxeles totales de la caja que contiene al carácter.
    - Elementos conexos: Un valor entre 0 y 1 proporcional al número de elementos conexos de la imagen si esté número es menor a tres.
  - Modificación del error de aprendizaje dinámicamente para adaptarse al aprendizaje de cada iteración.
- Problemas que no se sabe cómo solucionar.
  Uno de los problemas principales que tienen las redes neuronales es que es difícil de determinar cuál es el número óptimo de iteraciones. En esta solución se ha entrenado una red hasta 6000 iteraciones porque parecía que se había alcanzado un mínimo y que el aprendizaje no iba a ser mucho mayor por mucho que se continuara ejecutando. Sin embargo, después de este proceso es imposible determinar si el resultado es un mínimo relativo o si en pocas iteraciones más se podría haber llegado a un punto en el que se hubiera reducido drásticamente el error acercándolo al cero ideal.

REFERENCIAS

[1] Bellegarda EJ, Bellegarda JR, Nahamoo D, Nathan K. A probabilistic framework for on-line handwriting recognition. Proc 3rd International Workshop on Frontiers in Handwriting Recognition, Buffalo, NY, 1993; 225–234
[2] Connell S. Online Handwriting Recognition Using Multiple Pattern Class Models. PhD thesis, Michigan State University, East Lansing, MI, May 2000
[3] Jaeger S, Manke S, Reichert J, Waibel A. Online handwriting recognition: The NPEN__ recognizer. Int J Document Analysis and Recognition 2001; 3:169–180
[4] Bozinovic RM, Srihari SN. Off-line cursive script word recognition. IEEE Trans Pattern Analysis and Machine Intelligence, 1989; 22(1):63–84
[5] Bunke H, Roth M, Schukat-Talamazzini EG. Off-line cursive handwriting recognition using hidden markov models. Pattern Recognition 1995; 28(9):1399–1413
[6] Chen MY, Kundu A, Zhou J. Off-line handwritten word recognition using a hidden Markov model type stochastic network. IEEE Trans Pattern Analysis and Machine Intelligence 1994; 16(5):481–496
[7] Kim G, Govindaraju V, Srihari SN. An architecture for handwriting text recognition systems. Int J Document Analysis and Recognition 1999; 2:37–44
[8] Chen MY, Kundu A, Zhou J. Off-line handwritten word recognition using a hidden Markov model type stochastic network. IEEE Trans Pattern Analysis and Machine Intelligence 1994; 16(5):481–496
[9] El-Yacoubi A, Gilloux M, Sabourin R, Suen CY. Unconstrained handwritten word recognition using hidden markov models. IEEE Trans Pattern Analysis and Machine Intelligence 1999; 21(8):752–760
[10] Dimauro G, Impedovo S, Pirlo G, Salzo A. Automatic bankcheck processing: A new engineered system. In: Impedovo S, Wang PSP, Bunke H, eds, Int J Pattern Recognition and Artificial Intelligence, World Scientific, 1997; 467–503
[11] Guillevic D, Suen CY. HMM-KNN word recognition engine for bank cheque processing. Proc International Conference on Pattern Recognition, Brisbane, Australia, 1998; 1526–1529
[12] http://yann.lecun.com/exdb/mnist/
[13] C.G. Looney, Pattern Recognition using Neural Networks: Theory and Algorithms for Engineers and Scientists, Oxford University Press, 1997.
[14] http://www.heatonresearch.com/encog