Universidad de Granada

Inteligencia Computacional

Máster en Ingeniería Informática

Reconocimiento óptico de números manuscritos en la base de datos MNIST con redes neuronales

Autor
Pedro Manuel Gómez-Portillo López

Escuela Técnica Superior de Ingenierías Informática y de Telecomunicación

Granada, 26 de noviembre de 2018

Reconocimiento óptico de números manuscritos en la base de datos MNIST con redes neuronales

Pedro Manuel Gómez-Portillo López

Resumen

Las redes neuronales son una herramienta muy potente que pueden ser entrenadas para resolver todo tipo de problemas.

En esta práctica se trabajará con ellas para reconocer caracteres escritos a mano. Haciendo uso de la base de datos MNIST de números manuscritos y del framework de desarrollo Keras, se configurará y entrenará una red neuronal para que, al evaluarla, reconozca el mayor número de dígitos posibles.

Palabras clave: *MNIST, reconocimiento óptimo de caracteres, deep learning, keras*

El formato de la documentación de este trabajo ha sido basado en la plantilla LaTeX de https://github.com/erseco.

Índice general

Capítulo 1

Introducción

Las redes neuronales son un modelo computacional basado en un gran conjunto de neuronas individuales de forma aproximadamente análoga al comportamiento observado en los axones de las neuronas en los cerebros biológicos.

Actualmente existe una gran cantidad de programas que hacen uso de las redes neuronales para toda clase de aplicaciones; desde traducción de texto a clustering de datos.

Una de estas aplicaciones es el reconocimiento óptico de elementos. Concretamente, la base de datos MNIST[1] es un conjunto de imágenes con números escritos a mano. Esta base de dato tiene un conjunto de 60,000 imágenes, con un tamaño estándar de 28*28 píxeles cada una, y es la que se utilizará en esta práctica. En la figura 1.1 puede verse un ejemplo de los números almacenados en esta base de datos.

Figura 1.1: Ejemplo de números en la base de datos MNIST

Concretamente, se utilizará esta base de datos para entrenar una red neuronal y aplicarla al reconocimiento de estos caracteres, intentado acertar en el mayor número posible.

En esta práctica se podría optar o bien por desarrollar una red neuronal desde 0 o bien por utilizar un framework o conjunto de librerías ya

[1] `http://yann.lecun.com/exdb/mnist/`

desarrollado. Esta decisión se basará y se justificará en el siguiente capítulo.

1.1. Entorno de desarrollo

Mi ordenador personal, en el cual se ha realizado esta práctica, cuenta con las siguientes características.

- **Sistema operativo**. Ubuntu 18.04.1 LTS (Bionic Beaver)
- **Procesador**. Intel Core i7-6700HQ Quad-Core 2.6GHz
- **Cache**. 6M
- **Memoria RAM**. 8GB DDR4 2,133MHz
- **Tarjeta gráfica**. Sí, pero no recodida por el sistema operativo, por lo que no podrá usarse.
- **Disco duro**. SanDisk SSD Plus 256GB

Capítulo 2

Estado del arte

Antes de elegir lenguaje de programación a usar se realizó una búsqueda de los frameworks para desarrollo de aplicaciones basadas en deep learning disponibles, y de este modo tener razones para poder justificar esta elección.

A continuación se presentan los frameworks más importantes que hay en el mercado.

2.1. Keras

Keras[1] es una librería de código abierto escrita en Python, diseñada específicamente para hacer experimentos con redes neuronales.

Lo interesante de este framework es que funciona como **front-end** de la librería que utilice por debajo, es decir, que redirige las llamadas a sus funciones a las de la librería con la que le digamos que trabaje. Keras puede ejecutarse sobre MXNet, DL4J, TensorFlow, o Theano, algunas de las que hablaremos más adelante.

2.2. Pytorch

Pytorch[2] es un framework para Python que permite el prototipado y desarrollo rápido de aplicaciones deep learning, especialmente centrado en la aceleración por GPU.

La característica principal de Pytorch es que utiliza grafos computacionales dinámicos por cuestiones de eficiencia y optimización, ya que según su API estos grafos se paralelizan especialmente bien en una tarjeta gráfica.

[1] `https://keras.io/`
[2] `https://pytorch.org/`

2.3. TensorFlow

TensorFlow[3] es es un framework de código abierto desarrollado por Google que, según su página web, *se utiliza para realizar cálculos numéricos mediante diagramas de flujo de datos; los nodos de los diagramas representan operaciones matemáticas y las aristas reflejan las matrices de datos multidimensionales (tensores) comunicadas entre ellas.*

Este framework trabaja a un nivel más bajo que Keras o Pytorch y, aunque es uno de los más populares y usados por empresas como Dropbox, su uso parece ser recomendado para proyectos más grandes y complejos que éste.

2.4. Scikit-learn

Scikit-learn[4] es un framework de código abierto escrita en Python utilizada por empresas como Spotify que, según su página web, es *simple, eficiente y accesible, perfecta para para técnicas de análisis y minería de datos.* Está escrita sobre otras librerías de Python como *NumPy*, *SciPy*, y *matplotlib*.

2.5. Theano

Theano[5] es un frameworks de bajo nivel, como TensorFlow, y es otro de los soportados por Keras. Según su página web permite *definir, optimizar y evaluar expresiones matemáticas, especialmente las que trabajan con matrices multi-dimensionales.*

Aún así, Theano no está especialmente centrado en el deep learning, por lo que en lugar de usar esta librería tiene más sentido usar frameworks como Keras que se apoyen en el para realizar tareas de *deep learning*.

2.6. Lasagne

Lasagne[6] es una librería escrita en Python que nació exclusivamente para permitir usar Theano en tareas de deep learning, y proporcionando una interfaz más amigable. Es una de las principales competidores de Keras, aunque parece no ser tan preferida como esta.

[3] `https://www.tensorflow.org/`
[4] `https://scikit-learn.org/`
[5] `http://www.deeplearning.net/software/theano/`
[6] `https://lasagne.readthedocs.io/`

2.7. DSSTINE

DSSTINE[7] (pronunciado *destiny*), es un framework de código abierto desarrollado por Amazon especialmente pensado para entrenar y desplegar modelos de recomendación.

Además, únicamente permite ejecutarse sobre GPU, aunque permite hacerlo en paralelo usando varias. Por otro lado, su documentación es muy pobre y a veces es necesario bucear en su código fuente para entender qué hace.

2.8. MXNet

MXNet[8] es una librería de uso general desarrollada por Apache. A pesar de asegurarse bastante potente parece estar en una fase muy verde.

2.9. DL4J

DeepLearning4J[9] es una librería distribuida de código abierto escrita en Java y disponible para Java, Python y C++.

Según su página web, se especializa en redes neuronales profundas, y su documentación parece muy completa y está muy bien escrita.

2.10. Microsoft Cognitive Toolkit

CNTK[10] es una librería de código abierto desarrollada por Microsoft Research, la división de investigación de Microsoft que, según su página web, *entrena algoritmos de deep learning para pensar como personas.*

A pesar de lo prometedor que parecía, no parece ser muy popular, ya que en comparación con el resto de frameworks no hay muchos ejemplos por la web, ni siquiera en páginas especializadas como Kaggle[11].

[7] `https://github.com/amzn/amazon-dsstne`
[8] `https://mxnet.incubator.apache.org/`
[9] `https://deeplearning4j.org/`
[10] `https://www.microsoft.com/en-us/cognitive-toolkit/`
[11] `https://www.kaggle.com/`

Capítulo 3

Implementación

Tras realizar el estudio acerca del estado del arte, y basándonos que necesitamos un framework centrado en deep learning que no necesite GPU para ejecutarse, se ha decidido utilizar el lenguaje de programación Python3 con el framework Keras con TensorFlow como back-end.

Cabe destacar que, aunque me hubiera parecido mucho más interesante programar la práctica desde 0, como se proponía en el guión, en lugar de utilizar una librería ya implementada, las restricciones de tiempo y esfuerzo a las que ha tenido que enmarcase esta práctica debido al resto de asignaturas han hecho que finalmente opte por utilizar una framework de desarrollo de aplicaciones *deep learning*.

3.1. Framework y librerías utilizadas

A continuación se presentan los programas más importantes utilizados en esta práctica, así como su versión específica. Se han obviado las dependencias.

- **Python 3.6.6**. Será el lenguaje de programación de la práctica.
- **Keras 2.2.4**. Se utilizará como front-end para trabajar con TensorFlow, ya que actúa como interfaz facilitando su uso.
- **TensorFlow 1.13.0**. Será el corazón de la aplicación
- **Matplotlib 3.0.0**. e utilizará para visualizar el contenido de la base de datos MNIST.

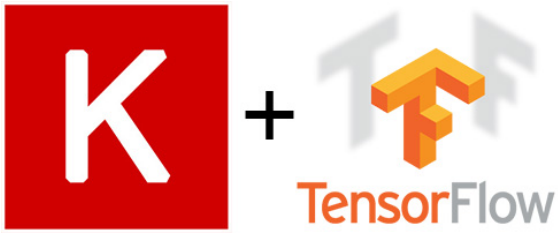

Figura 3.1: Logos de Keras y TensorFlow

Para instalar estas librerías usando el proyecto solo es necesario situarse en el directorio raíz y ejecutar `make install`.

3.2. Detalles comunes de la implementación

A continuación se presentan la información de la implementación del proyecto. Primero se presentará la información común a todas las versiones del proyecto y a continuación se pasará a describir cada versión por separado. Los resultados detallados de cada versión pueden verse en el capítulo 5, mientras que aquí solo se presentará la configuración de cada una.

En total, este proyecto ha pasado por tres versiones, pero es necesario destacar que las versiones no son totalmente sucesivas, sino que entre ellas se realizaron varias subversiones probando diferentes configuraciones de épocas y capas, y no se decidía definir una nueva versión hasta obtener una mejoría suficiente.

Además, se ha utilizado un repositorio en GitHub para realizar el control de versiones de la práctica, cuya dirección es `https://github.com/gomezportillo/mnist`.

Al final de cada versión se ha publicado una release del proyecto, por lo que las tres versiones pueden verse en la correspondiente sección del repositorio.

Todas las versiones utilizan las mismas librerías, por lo que para probar cualquiera de ellas, y habiendo clonado previamente el repositorio e instalado las dependencias, bastaría con situarse en el directorio raiz y ejecutar `git checkout vX.0`, donde x es la versión del proyecto, entre 1 y 3, a la que nos queremos mover, y escribir `make` para ejecutar la práctica en dicha versión.

3.3. Primera versión

Capítulo 4

Resultados

Capítulo 5

Conclusiones

Capítulo 6

Referencias

Capítulo 7

Anexos