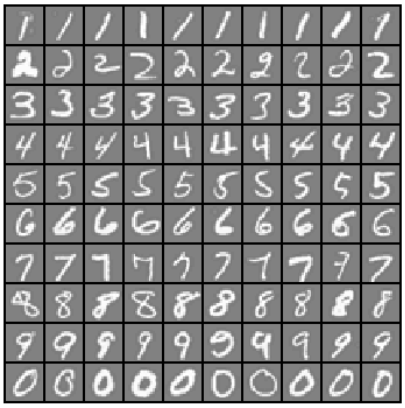
Práctica 5. Redes Neuronales en Keras

Objetivo

El objetivo es resolver mediante redes neuronales un problema real de clasificación: el reconocimiento de dígitos manuscritos del *dataset* MNIST. La figura muestra un ejemplo de los mismos. Cada muestra es una imagen de 28x28 píxeles. Como atributos para la clasificación utilizaremos directamente los niveles de intensidad de los 28x28 píxeles. Utilizaremos Keras, una API sencilla para Redes Neuronales, que permite entrenar y ejecutar redes neuronales utilizando Tensorflow como *back-end*.



Estudio previo

1. Descarga Anaconda https://www.anaconda.com/download/ e instálalo en tu computador. Anaconda es una distribución de Python para cálculo científico, que contiene un manejador de paquetes y la IDE Spyder.
2. Desde el navegador de Anaconda, ve a Environments, y busca los paquetes que contengan "tensor" con la opción "search packages" "all". Selecciona Keras y Tensorflow, y acepta para que instale ambos paquetes y sus dependencias.
3. Para comprobar tu instalación, abre el IDE Spyder, carga el fichero P5_demo.py y ejecútalo.
4. Lee la documentación de Keras en https://keras.io/ para ver cómo definir y entrenar una red, el tipo de capas que pueden usarse, algoritmos de entrenamiento, etc.

Desarrollo de la práctica

Copia a tu directorio de trabajo los ficheros proporcionados, y comprueba que funcionan correctamente en Keras. El programa P5_demo.py carga los datos x_train e y_train (60.000 muestras), que se usarán para entrenamiento y validación, y **los datos x_test e y_test (10.000 muestras) que sólo deben utilizarse para comprobar el funcionamiento final** de la red entrenada.

El objetivo de la práctica es encontrar una red que tenga el menor error de clasificación posible. Ve construyendo una tabla que refleje las diferentes redes que has probado, y los resultados que obtienen. **Al final de la práctica deberás presentar esta tabla, discutida y comentada adecuadamente**.

Como mínimo deberás probar redes de los siguientes tipos:

1. **Perceptrón**. El fichero P5_demo.py define y entrena una red con una única capa de perceptrones. Prueba distintas variaciones con distintos algoritmos de entrenamiento, funciones de activación, etc. Si ves que se produce sobreajuste, prueba alguna técnica para reducirlo.

2. **Perceptrón multi-nivel con una capa oculta.** Define una red para reconocimiento de patrones con una capa oculta, e intenta mejorar los resultados con las técnicas anteriores, y también cambiando en número de neuronas de la capa oculta.

3. **Perceptrón multi-nivel con dos capas oculta.** Define una red para reconocimiento de patrones con dos capa oculta, e intenta mejorar los resultados con las técnicas anteriores.

4. **Red Convolucional.** En el fichero mnist_cnn.py tienes un ejemplo de una red convolucional para este problema. Comprueba su funcionamiento.

Notas
1. Prueba distintos algoritmos de entrenamiento y distintas funciones de activación, y observa si mejora la convergencia (tiempo de entrenamiento, y error obtenido)
2. Utilizar validation_split=0.1 permite analizar si se está produciendo sobre-ajuste, pero desaprovecha un 10% de los datos de entrenamiento. En algunos casos puede ser interesante re-entrenar la red unas pocas épocas con todos los datos de entrenamiento (sin validation_split) para mejorar su tasa de aciertos.
3. En redes complejas, el dropout es una buena técnica para reducir el sobreajuste, mira el ejemplo de mnist_cnn.py

A entregar (en Moodle, dentro de un fichero .zip)
- Programa `P5.py`, que ejecute el entrenamiento de las distintas redes que has probado (al menos la mejor de cada tipo) y vaya mostrando los resultados obtenidos. Si programas funciones auxiliares, entrégalas también.
- Memoria de la práctica en un fichero `P5.pdf` con los resultados de todos los apartados, su interpretación y las conclusiones que hayas obtenido.