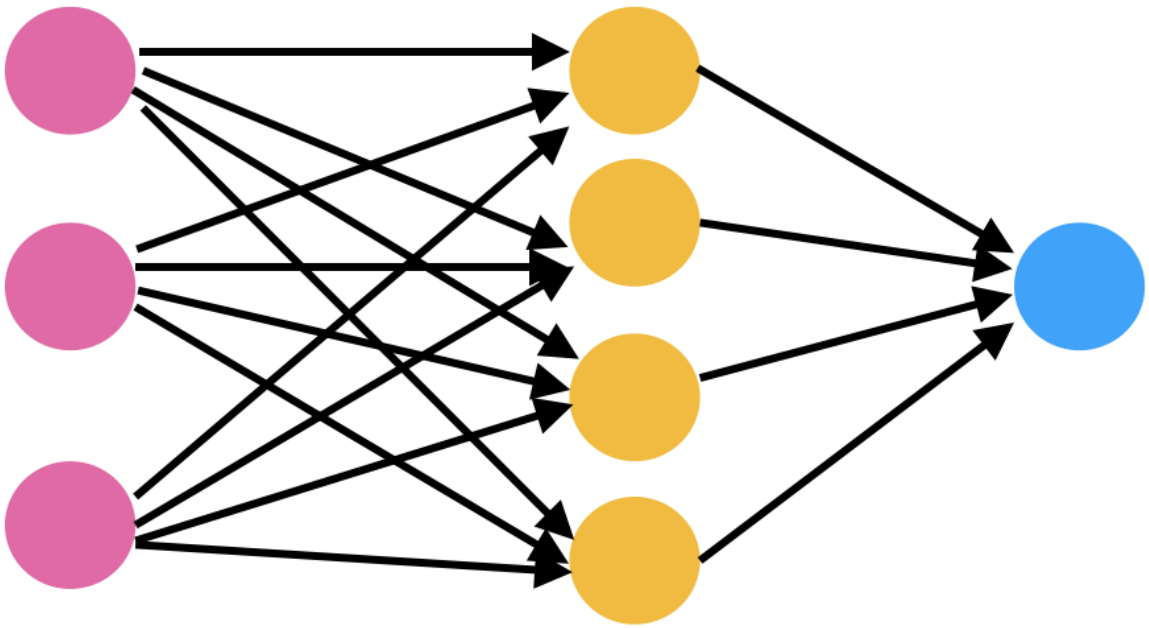
MEMORIA DE INTELIGENCIA COMPUTACIONAL:

REDES NEURONALES



Adrián de la Torre Rodríguez

ÍNDICE

1. Introducción

En esta memoria se va a explicar los pasos seguidos para desarrollar una red neuronal y cómo se ha entrenado para resolver un problema. El objetivo es entrenar una red neuronal de varias capas para que pueda reconocer dígitos escritos a mano. La base de datos usada para su entrenamiento y test se trata de *"The MNIST database of handwritten digits"* de Yan LeCunn [1].

Base de datos

Para el entrenamiento de la red cuenta con dos archivos (archivos de entrenamiento). El primero de 60.000 imágenes de 28x28 píxeles en las cuales aparece un dígito numérico entre 0 y 9. También tiene otro archivo de 60.000 etiquetas de números entre 0 y 9 que corresponden por orden de posición a las imágenes anteriormente explicadas. Para la comprobación del entrenamiento realizado en la red neuronal existen otros dos archivos de las mismas características(archivos test) pero esta vez con 10.000 elementos.

Las imágenes vienen almacenadas en un array de tamaño 60.000 donde cada elemento de ese array es una matriz de enteros de 28x28. En total se tienen 47040000 elementos para el archivo de imágenes de entrenamiento. Cada elemento de la matriz o pixel es un número entero entre 0 y 255 que representa un valor en la escala de grises siendo 0 totalmente blanco (no hay nada escrito) y 255 totalmente negro (se ha escrito en ese pixel). A continuación se muestran algunas imágenes del conjunto de imágenes de entrenamiento [2]:

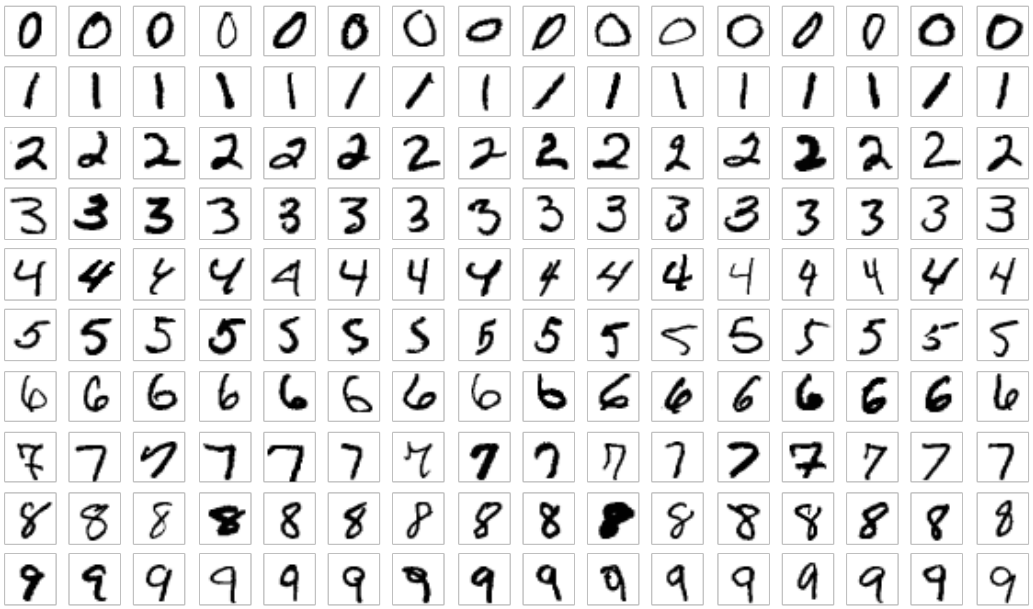


Ilustración 1

Las etiquetas tanto para el conjunto de entrenamiento como para el conjunto de test vienen almacenadas en un array donde cada elemento de ese array es un número entero entre 0 y 9.

Preprocesamiento de datos

Dependiendo del caso se realizarán modificaciones al conjunto de datos, en concreto a las imágenes del conjunto de entrenamiento para que la red neuronal sea capaz de interpretarlos o bien pueda aprender mejor de ellos.

Los tipos de preprocesamiento de datos que se ha hecho a lo largo de la práctica son los siguientes:

- Normalización de los datos: Se han transformado los valores de la imagen de [0,255] a [0,1] simplemente al dividir cada valor entre 255. Esto se ha hecho para que a la hora de aprender no se trabajen con valores tan altos que pueden dar lugar a desbordamiento de variables y porque es conveniente trabajar con valores menos altos para el aprendizaje de la red.
- Cambio de formato: Se han transformado las matrices de las imágenes de 28x28 a vectores de 784 elementos con el objetivo de mejorar su acceso ligeramente.
- Transformaciones en las imágenes: Se ha usado un generador de transformaciones aleatorias en las imágenes el cual las desplaza en algún eje, les aplica zoom, las rota ligeramente, etc. Con esto se pretende que la red generalice mejor al usar un conjunto de entrenamiento más variado.

Redes neuronales

Una red neuronal es un modelo de computación basado originalmente en el funcionamiento del cerebro. Las redes neuronales están formadas por neuronas simples que constituyen una unidad de cálculo en la que reciben una serie de entradas y obtienen una salida (ilustración 2).

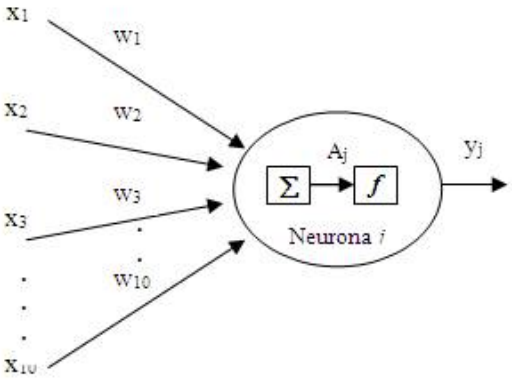


Ilustración 2

No voy a entrar en detalles técnicos del funcionamiento de una neurona artificial porque la memoria no trata de eso. Sí se va a explicar, sin embargo, que se han usado redes neuronales de una o más capas para el reconocimiento de los dígitos explicados en la sección anterior. Una capa no es más que un conjunto de neuronas el cual obtiene un conjunto de entradas de la anterior capa y genera un conjunto de salidas que envía a la siguiente capa.

Para esta práctica se han usado dos versiones de redes neuronales:

- Versión del perceptrón: Esta versión consiste en una red neuronal de una capa basada formada por perceptrones de activación lineal.

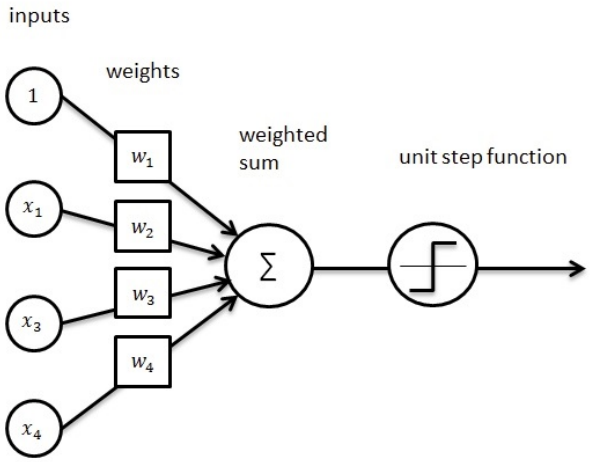


Ilustración 3

- Versión multicapa: Red neuronal con una capa oculta y otra externa de activación no lineal y entrenamiento por *back propagation*.

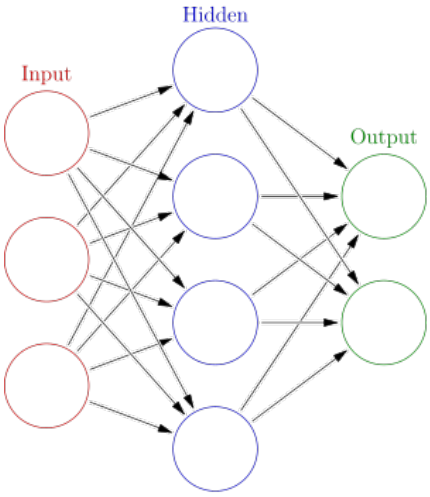


Ilustración 4

Entrenamiento

El entrenamiento de la red neuronal consiste en variar los parámetros (pesos) de las neuronas en base al error que han generado para intentar disminuirlo. Para la primera versión del perceptrón se ha usado un entrenamiento donde se ajustaban los pesos linealmente. Para la versión multicapa se ha usado back propagation y drop out.

El entrenamiento se ha realizado con el conjunto de datos de entrenamiento (60000 elementos). El objetivo es que la red neuronal aprenda a reconocer dígitos distintos a los del conjunto de entrenamiento sin que se equivoque.

Test

Una vez se ha entrenado la red neuronal el siguiente paso es comprobar cuán bien reconoce dígitos ajenos al conjunto de entrenamiento. Para ello se ha usado el conjunto de datos de test (10000 elementos) y la red ha intentado predecir qué dígito representa cada imagen. Se han contado el número de aciertos y errores realizados por la red y se ha obtenido un porcentaje de acierto o precisión. El objetivo es crear una red cuyo porcentaje de acierto sea al rededor del 99%.

En las siguientes secciones se va a explicar con detalle el desarrollo e implementación de los algoritmos de las redes neuronales y los resultados obtenidos.

2. Primera versión (Perceptrón)

Esta versión ha sido íntegramente desarrollada por mí salvo las funciones para leer datos de la base de datos de MNIST. El lenguaje de programación usado ha sido java y el entorno de programación usado ha sido Eclipse para Windows 10.

Esta red neuronal consiste en una clase llamada “Entrenador” la cual contiene 10 objetos de la clase “Neurona” y se encarga de leer los datos de MNIST, aportarlos a las neuronas y, en base a las salidas que estas proporcionan, alterar los pesos de las neuronas.

Preprocesamiento

3. Segunda versión

Al principio se intentó desarrollar una versión en java de una red neuronal multicapa pero se quedó en la etapa de back propragation debido al tiempo necesario para su implementación y a que era muy posible que no consiguiera los mismos resultados que usando librerías profesionales.

Esta versión se ha desarrollado en Python usando la librería de keras con tensorflow como backend [3]. La creación del modelo, incluyendo las capas elegidas y el entrenamiento han sido elegidos por mí. Sin embargo todas las funciones usadas son las de la librería de keras y por lo tanto han sido implementadas por terceros. El proyecto se ha desarrollado en Ubuntu 18.04 y no hace uso de la GPU por lo que los tiempo de ejecución son más elevados que si se usara.

El proyecto consiste en un script de python que usa el modelo *Sequential* de keras el cual permite añadirle distintos tipos de capas ordenadamente. Se han añadido dos capas al modelo:

- Capa oculta: Capa de 784 neuronas las cuales tienen 784 entradas y una salida por neurona con activación “relu”.
- Capa externa: Capa de 10 neuronas con 784 entradas cada una y una salida de activación “softmax”.

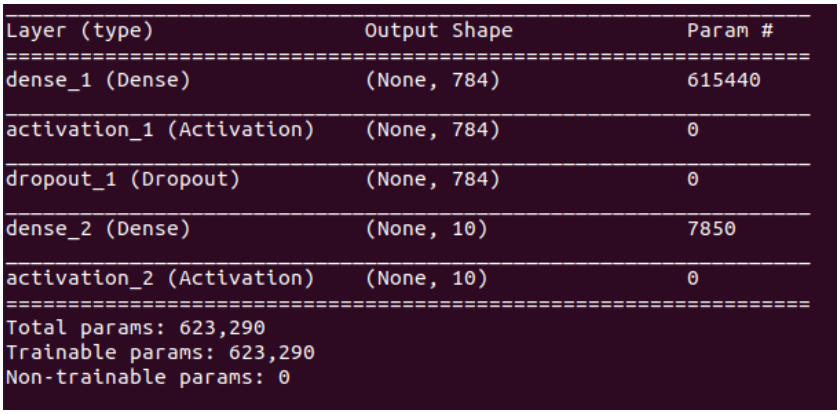


Ilustración 5: Estructura de la red neuronal

Al principio se uso un modelo mucho mas complejo que incluía varias capas convolutivas y max pooling pero el tiempo de cómputo necesario era tan alto que se descartó.

Preprocesamiento

En todos los modelos desarrollados en esta versión se ha usado la normalización de las imágenes, es decir se han pasado de valores de [0,255] a [0,1].

Dependiendo de si se querían dibujar gráficas o números o si se querían procesar los datos se han usado distintos formatos del conjunto de datos:

- (60000,28,28,1)
- (60000,784)

Para la versión mas compleja se había usado un generador de imágenes que realizaba variaciones aleatorias a estas. El siguiente código ilustra las variaciones que realizaba este generador.

```
datagen = ImageDataGenerator(
        featurewise_center=False,  # set input mean to 0 over the dataset
        samplewise_center=False,  # set each sample mean to 0
        featurewise_std_normalization=False,  # divide inputs by std of the dataset
        samplewise_std_normalization=False,  # divide each input by its std
        zca_whitening=False,  # apply ZCA whitening
        rotation_range=15, # randomly rotate images in the range (degrees, 0 to 180)
        zoom_range = 0.1, # Randomly zoom image
        width_shift_range=0.1,  # randomly shift images horizontally (fraction of total width)
        height_shift_range=0.1,  # randomly shift images vertically (fraction of total height)
        horizontal_flip=False,  # randomly flip images
        vertical_flip=False)  # randomly flip images
```

El objetivo de usar este generador era que el conjunto de entrenamiento fuera más variado y que la red neuronal aprendiera a generalizar más y no hubiera sobre aprendizaje en este conjunto.

Entrenamiento

El entrenamiento usado por la red es *back propagation*, este entrenamiento varía los pesos de las neuronas usando el error cometido por la capa externa y su propagación hacia capas internas [4]. En la siguiente ilustración se puede ver la fórmula usada para calcular las variaciones de pesos para cada capa.

$$\Delta w_{ij}^k = \begin{cases} -\eta x_i^k f'(z_j^k)\,(y_j - t_j) & \text{para las neuronas de salida,} \\ -\eta x_i^k f'(z_j^k)\left(\sum_p \delta_p^{k+1} w_{pj}^{k+1}\right) & \text{para las neuronas ocultas.} \end{cases}$$

Ilustración 6

También se ha usado la técnica de *drop out* que consiste en ignorar ciertas neuronas aleatoriamente en alguna etapa de forward or back propagation con el objetivo de evitar que la red neuronal sobre aprenda, es decir, que aprenda errores.

Se ha hecho un entrenamiento por lotes de tamaño 128 y se han hecho 20 epochs, es decir, se ha recorrido el conjunto de entrenamiento 20 veces. Con el modelo explicado el tiempo de entrenamiento de la red neuronal ronda los 120 segundos.

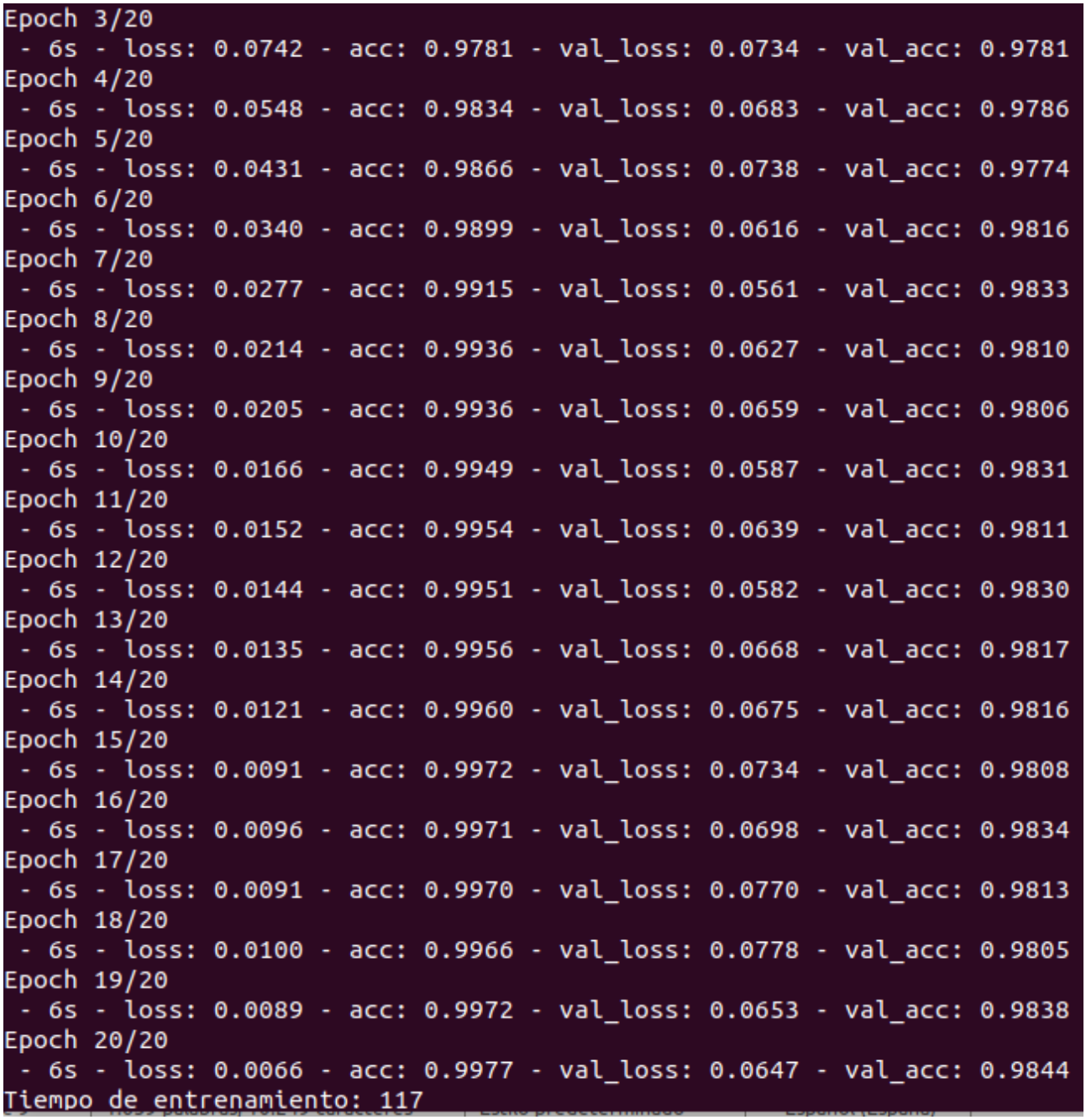


Ilustración 7: Resultado del entrenamiento de la red neuronal

Como se puede ver en la ilustración anterior, se han tardado 117 segundos en entrenar la red neuronal y se ha obtenido un porcentaje de acierto del 98,44%.

Durante la implementación del proyecto se incluyó la funcionalidad para almacenar y leer los datos del modelo como:

- Estructura: Capas de la red neuronal, activación de neuronas y técnicas de entrenamiento.
- Pesos: Los pesos de las neuronas.
- Historia: Valores obtenidos durante la fase de entrenamiento como porcentaje de errores.

Así, si se desea una ejecución rápida del proyecto basta con seleccionar en el código que no se quiere que aprenda la red y esta cargará los datos de la última ejecución.

Test

Este es un resultado de la comprobación del entrenamiento de la red neuronal sobre el conjunto de test:

```
Numero de errores: 156. Porcentaje de error = 1.56%.
```

Ilustración 8

Se puede ver que el resultado es menor que el 1,8 % que es el resultado que se indica en el guión de esta práctica para la red neuronal más compleja.

4. Conclusión

5. Bibliografía

[1] LeCunn, Y., Cortes, C. and J.C. Burges, C. (n.d.). MNIST handwritten digit database, Yann LeCun, Corinna Cortes and Chris Burges. [online] Yann.lecun.com. Available at: http://yann.lecun.com/exdb/mnist/ [Accessed 7 Dec. 2018].

[2] Wikipedia contributors. (2018, November 21). MNIST database. In Wikipedia, The Free Encyclopedia. Retrieved 11:04, December 7, 2018, from https://en.wikipedia.org/w/index.php?title=MNIST_database&oldid=869917056

[3] Keras.io. (2018). Home - Keras Documentation. [online] Available at: https://keras.io/ [Accessed 7 Dec. 2018].

[4] Berzal, F. (2018). Redes Neuronales & Deep Learning. 1st ed. Granada.