
UNIVERSIDAD
DE GRANADA

# Inteligencia Computacional

## Práctica 1

## Reconocimiento óptico de caracteres

## MNIST

Curso 2018-2019

Máster en Ingeniería Informática

Miguel de Oliveira Dias Gonçalves

# Índice

# 1. Introducción

Las redes neuronales consisten en sistemas de software y/o hardware basados en la operación de las neuronas humanas[1]. Una neurona artificial es un modelo matemático, dónde hay una serie de entradas, un peso por cada entrada que cambiará durante su entrenamiento, una función de activación teniendo en cuenta las entradas y sus pesos, y una salida[2].

Una red neuronal consiste así en una combinación de neuronas organizadas por capas, dónde cada capa recibe datos, los procesa y los entrega a la capa siguiente. La primera capa es la capa de entrada de los datos, la última es la capa de salida de los datos. Su entrenamiento se realiza utilizando grandes cantidades de datos, dónde cada entrada está acompañada de la salida pretendida de modo a que la red pueda ajustar sus pesos teniendo en cuenta si las salidas obtenidas son o no las pretendidas[3].

Un uso de las redes neuronales es en el procesamiento de imágenes. La base de datos MNIST es de uso popular en el entrenamiento de dichas redes neuronales, estando registradas las mejores tasas de error para diversos tipos de redes y preprocesamientos de datos. Consiste en 60.000 imágenes de dígitos manuscritos para entrenamiento y otras 10.000 imágenes para prueba, normalizadas de modo a que todas tengan un tamaño de 28x28 pixeles, así como sus etiquetas correspondiendo a dígitos de 0 a 9. Tasas de error tabladas varían desde 12% para clasificadores lineales de 1 capa sin preprocesamiento, hasta una tasa de 0,21% obtenida con una junción de 5 redes convolucionales[4] [5].

El objetivo de esta práctica es evaluar la prestación de diversos tipos de redes neuronales, enseñados en las clases prácticas de la asignatura, al clasificar los dígitos manuscritos de la base de datos MNIST. Existe la posibilidad de implementar con código los diferentes algoritmos, o de utilizar bibliotecas externas enfocadas en eficiencia que ya los implementan a cambio de una reducción en la nota máxima en la práctica. El principal criterio para evaluar los resultados obtenidos será su tasa de error sobre el conjunto de prueba de la base de datos MNIST.

[1] https://searchenterpriseai.techtarget.com/definition/neural-network
[2] https://en.wikipedia.org/wiki/Artificial_neuron
[3] https://searchenterpriseai.techtarget.com/definition/neural-network
[4] https://en.wikipedia.org/wiki/MNIST_database
[5] http://yann.lecun.com/exdb/mnist/

# 2. Perceptron 1-capa

## 2.1. Implementación y algoritmo

Se ha empezado la práctica con una red neuronal de 1 capa con 10 neuronas binarias con umbral, cada una correspondiendo a un dígito de 0 a 9, entrenadas usando el algoritmo de entrenamiento Perceptron. Los mejores resultados se han obtenido (aunque por un escaso margen) inicializando todos los pesos de la red con valores aleatorios entre -1 y 1, para una tasa de aprendizaje de 0.55 en el caso del conjunto de prueba y 0.95 en el caso del conjunto de entrenamiento.

Cada neurona de la red tiene 785 entradas, una por cada píxel de una imagen de la base de datos MNIST más una entrada fija igual a 1 correspondiente al umbral.

Su función de activación retorna 1 si la soma de los productos de las entradas con los pesos es igual o superior a 0, y retorna 0 en caso contrario. Se verá abajo que dicha función de activación es más eficiente que una que retorne respectivamente 1 y -1 para los mismos casos.

Los pesos durante el entrenamiento son actualizados cada vez que se recibe una nueva imagen sumando a cada peso el resultado de la fórmula

$$l*(d-o)*i$$

*Ecuación 1 - Función de activación usada en la red neuronal de 1 capa que usa el algoritmo Perceptron*

Dónde $l$ es la tasa de aprendizaje; $d$ es el valor de salida deseado de la neurona, es decir, 1 si la neurona debe reconocer el dígito introducido o 0 si no; $o$ es la salida efectiva de la neurona, 1 si la neurona cree que el dígito introducido es lo que debe reconocer y 0 si no; e $i$ es el valor de entrada correspondiente a ese peso.

Se ha conseguido una tasa de error de 23.52% sobre el conjunto de prueba y 23.44% sobre el conjunto de entrenamiento.

Para implementar la red se ha usado el lenguaje Java y sus bibliotecas. La implementación del algoritmo es de mi autoría, usando el fichero *MNISTDatabase.java*, creado por el profesor Fernando Berzal colocado en los materiales de la asignatura, para leer y normalizar los datos de la base de datos MNIST y sus etiquetas.

## 2.2. Experimentos realizados

Para determinar la mejor combinación de parámetros se han realizado 6 experimentos correspondientes a igual número de combinaciones. En particular, se han hecho 2 experimentos con todos los pesos inicializados a 0, otros 2 con los pesos inicializados con valores aleatorios entre -0.5 y 0.5, y otros 2 con los pesos inicializados con valores aleatorios entre -1 y 1.

En cada par de experimentos, en uno la salida de la neurona y el valor de salida deseado eran 1 si respectivamente la neurona creía reconocer y se debería reconocer el dígito introducido, y -1 en caso contrario. En el otro, se usaba 0 en lugar del -1.

En cada experimento se ha entrenado la red 9 veces con tasas de aprendizaje entre 0.1 y 0.9, aumentándose 0.1 la tasa entre cada entrenamiento y registrándose la tasa de error sobre el conjunto de prueba después de cada entrenamiento.

| Experimento | Pesos iniciales | Salidas obtenida y deseada | Tasa de error mínima (%) | Tasa de error media (%) | Tasa de error máxima (%) |
|---|---|---|---|---|---|
| 1 | 0 | 1 y -1 | 27,16 | 30,01 | 32,45 |
| 2 | 0 | 1 y 0 | 27,8 | 29,62 | 31,33 |
| 3 | [-0.5;0.5] | 1 y -1 | 27,38 | 29,47 | 32,21 |
| 4 | [-0.5;0.5] | 1 y 0 | 28,13 | 29,79 | 33,69 |
| 5 | [-1;1] | 1 y -1 | 26,72 | 30,33 | 33,89 |
| 6 | [-1;1] | 1 y 0 | 26,05 | 28,85 | 31,68 |

*Tabla 1 - Tasas mínimas, medias y máximas de error sobre el conjunto de prueba para diferentes combinaciones de parámetros*

En el experimento 6, con valores iniciales entre -1 y 1 para los pesos y valores de salida de las neuronas iguales a 1 o 0, se han obtenido las tasas mínima y media de error más bajas, así como la 2ª tasa máxima más basa.

Teniéndose obtenido la mejor combinación de parámetros, se han realizado después para esa combinación 1.000 entrenamientos de la red neuronal, empezándose con una tasa de aprendizaje de 0.001 e incrementándose 0.001 la tasa entre cada entrenamiento, registrándose la tasa de error sobre el conjunto de prueba después de cada entrenamiento.

Se ha obtenido así una tasa de error mínima de 23,52% para una tasa de aprendizaje de 0.551, una tasa media de 27,70%, y una tasa máxima de 38,16% para una tasa de aprendizaje de 0.001 (aunque la 2ª tasa de error más grande sea de 33,94%, para una tasa de aprendizaje de 0.030). Observando el gráfico de dispersión de la ilustración 1 se puede ver que casi

todas las tasas de error medidas (96,2% del total) se sitúan en el intervalo [24;31]%.

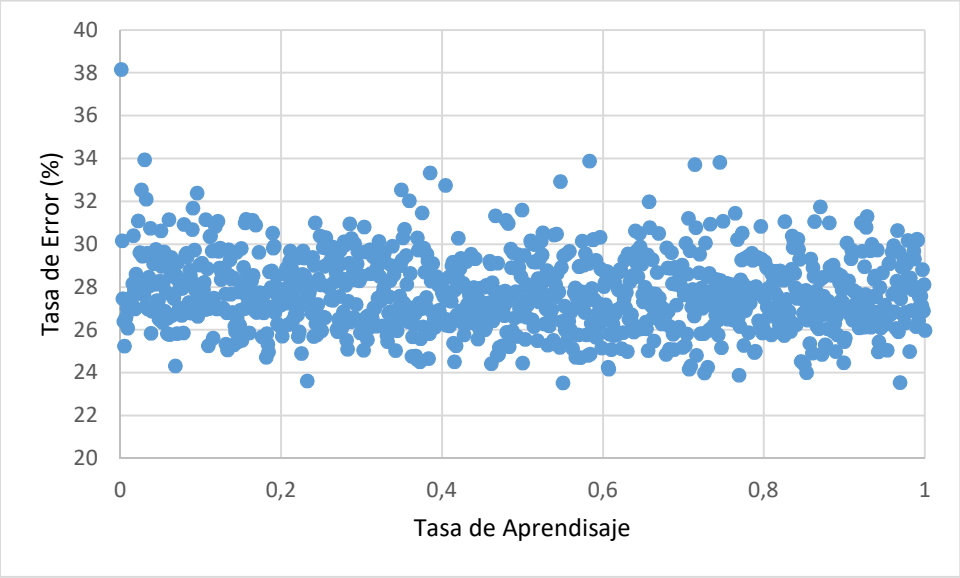


*Ilustración 1 – Gráfico de dispersión de la tasa de error sobre el conjunto de prueba en función de la tasa de aprendizaje de la red, con los pesos de la red neuronal inicializados con valores aleatorios entre -1 y 1, y valores de salida de las neuronas de 1 o 0*

Dividiendo las tasas de error en clases de 0.5%, se puede observar que la categoría con más observaciones es [26.5;27.0]%, con 133 observaciones o 13,3% de los entrenamientos realizados. Un total de 55,1% de las observaciones se sitúan en el intervalo [26;28,5[%.

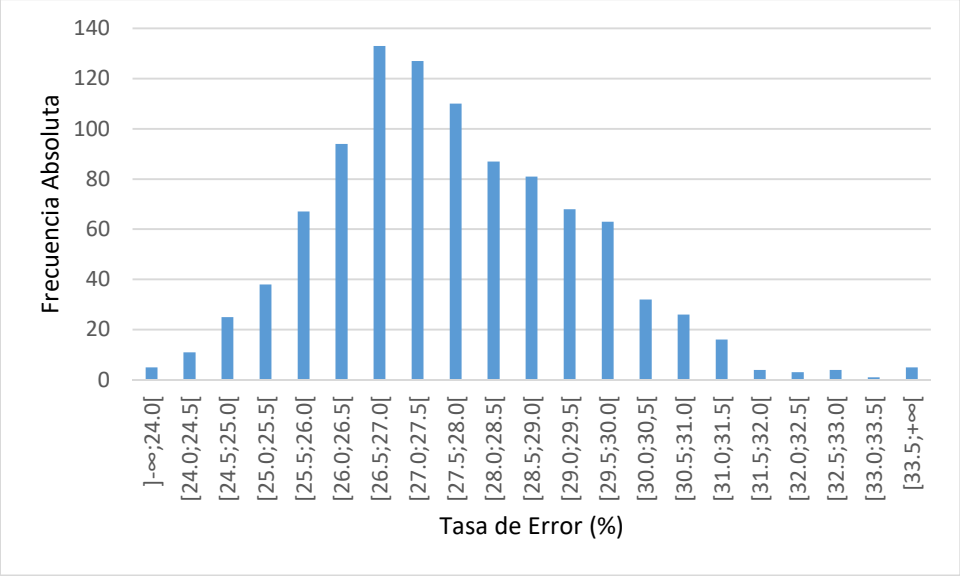


*Ilustración 2 – Gráfico de barras de la frecuencia absoluta de la tasa de error en cada entrenamiento de la red agrupadas en clases de 0.5%, en las mismas condiciones de la Ilustración 1*

Para obtener la tasa de error sobre el conjunto de entrenamiento se han realizado 100 experimentos con los mismos valores iniciales de los pesos y

de salida de las neuronas, empezándose con una tasa de aprendizaje de 0.01 e incrementándose la tasa en 0.01 entre cada entrenamiento. Se ha obtenido una tasa mínima de error de 23,44% para una tasa de aprendizaje de 0.95.

# 3. Perceptron multicapa

## 3.1. Implementación y algoritmo

Se ha creado en Python un Perceptron multicapa utilizando el lenguaje Python, y las bibliotecas Tensorflow y Keras para obtener la implementación de los algoritmos de entrenamiento. Además, se ha utilizado Google Colab para ejecutar el código. La red fue implementada con un número variable entre 1 y 5 de capas ocultas, que pueden ser ajustadas mediante una constante del programa. Es posible ajustar de forma individual el número de neuronas y la función de activación de cada capa oculta de la red. Existe también una capa de entrada y una capa de salida.

La tasa de error más baja sobre el conjunto de prueba, 2,94%, se ha obtenido utilizando 1 capa oculta. La capa de entrada y la capa de salida cuentan con 10 neuronas, una por cada dígito de 0 a 9. La capa oculta utilizada para obtener el mejor resultado tiene 1960 neuronas, correspondientes a 2,5 veces 784, el número de pixeles por imagen de la base de datos MNIST. Las neuronas de las capas ocultas son unidades lineales rectificadas, utilizando así un rectificador como función de activación. Las neuronas de la capa de salida utilizan la función softmax como función de activación.

La red se entrena en una solo época, con una tasa de aprendizaje de 0.001. Se aplica un dropout de 0.2 entre la capa oculta y la capa de salida. Los pesos de las neuronas son inicializados con valores entre -0.05 y 0.05 obtenidos usando una distribución uniforme. Los bias de las neuronas son inicializados con 1. Se utiliza el RMSprop como función de optimización de la red, manteniendo intactos sus valores por defecto incluso su tasa de aprendizaje por defecto de 0.001. La función de pérdida es el Sparse Categorical Crossentropy, también con sus valores por defecto.

El principal tutorial que he utilizado se encuentra en https://colab.research.google.com/github/csc-training/intro-to-dl/blob/master/day1/keras-mnist-mlp.ipynb#scrollTo=JHXVwqyQORr1

aunque también haya utilizado los tutoriales que se pueden encontrar en https://machinelearningmastery.com/build-multi-layer-perceptron-neural-network-models-keras/ y https://www.tensorflow.org/tutorials/.

### 3.2. Experimentos realizados

Para determinar la mejor combinación de parámetros, se han establecido valores por defecto en la red y cambiado solamente uno de cada vez, manteniendo todos los otros constantes. El objetivo fue determinar cuál el valor del parámetro que se cambiaba que proporcionaba la tasa de error sobre el conjunto de prueba más baja.

Los valores por defecto fueron:

- Neuronas de la capa de entrada = 10
- Neuronas de las capas ocultas = 784
- Capas ocultas = 1
- Épocas = 1
- Tasa de aprendizaje = 0.001
- Dropout = 0,2
- Valores iniciales de los pesos = Distribución uniforme entre -0,05 y 0,05
- Valores iniciales de los bias = 1
- Función de activación de las capas ocultas = ReLU
- Función de activación de la capa de salida = Softmax
- Función de optimización = RMSprop
- Función de pérdida = Sparse Categorical Crossentropy

Se han realizado algunos conjuntos de experimentos sobre la red, en cada uno variándose un parámetro distinto de la red y manteniendo los restantes intactos e iguales a los valores por defecto definidos arriba.

Primeramente se ha realizado un experimento con el número de neuronas de todas las capas ocultas. Se empezó por fijar el número de neuronas en 10% de 784, arredondeado por exceso, y entre cada experimento el número de neuronas era incrementado en 10% de 784 hasta llegar a 3*784, igual a 2.352 neuronas por capa. A partir de 50%*784 (392 neuronas), la tasa de error se mantiene casi siempre entre 3% y 4%, aunque con grandes variaciones, exceptuando los 90% con 4,21% de tasa de error, y los 250% con 2,94% de tasa de error y que es la tasa de error más baja obtenida con esta red.

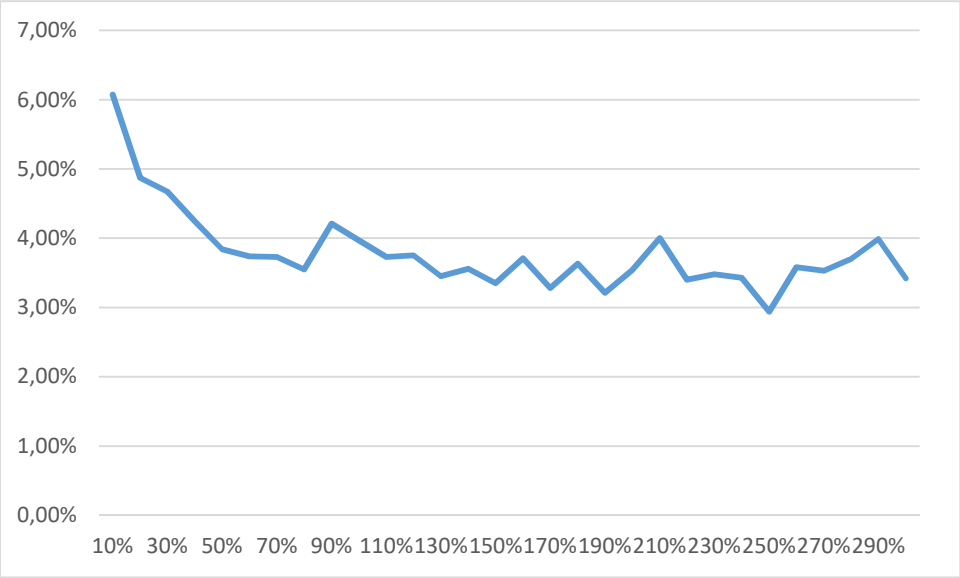


*Ilustración 3 - Gráfico del porcentaje de error en función del número de neuronas, definido como un porcentaje de 784*

| Capas ocultas | 1 | 2 | 3 | 4 | 5 |
|---|---|---|---|---|---|
| Tasa de error | 3,55% | 3,03% | 5,22% | 7,31% | 90,20% |

*Tabla 2 - Tasa de error en función del número de capas ocultas de la red*

Se puede observar en la Tabla 2 el resultado del conjunto de experimentos con el número de capas ocultas, que se ha variado entre 1 y 5 de forma sucesiva. Las mejores tasas de error se han obtenido con 1 y 2 capas ocultas. Con 3 y 4 capas ocultas la tasa de error ya es mayor, y se da un gran salto al introducirse una quinta capa oculta con una tasa de error superior a 90%. Los resultados de estos experimentos muestran que el Perceptron multicapa no funciona bien con un número demasiado grande de capas.

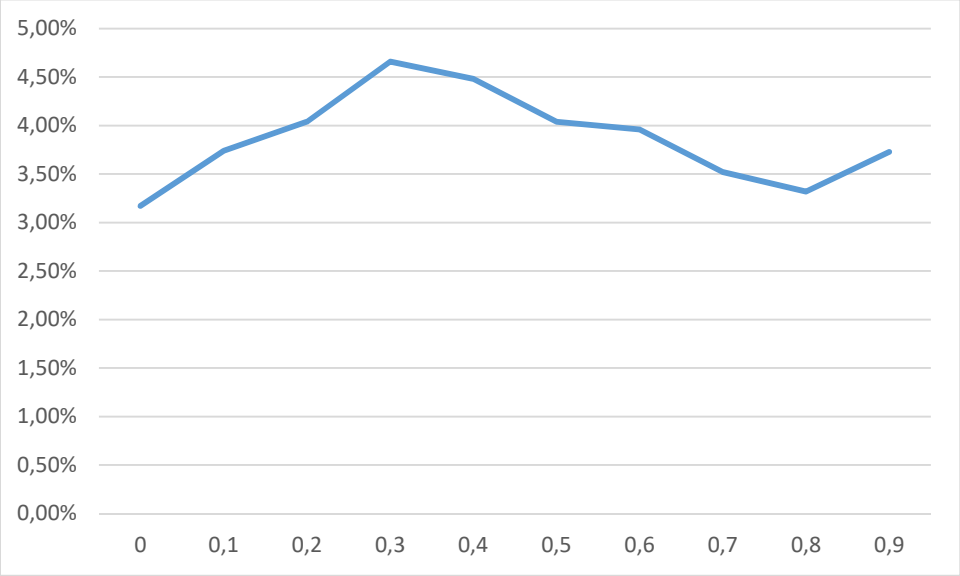


*Ilustración 4 - Gráfico de la tasa de error en función del dropout*

Para observar la influencia del dropout en la tasa de error, se han realizado 10 experimentos donde entre cada uno se incrementaba el dropout en 0,1, empezándose en 0 y terminándose en 0,9. Los mejores resultados se han obtenido con un dropout de 0 (3,17% de tasa de error) y con un dropout de 0,8 (3,32%). A partir del dropout igual a 0, la tasa de error ha subido de forma aproximadamente lineal hasta llegar a un máximo de 4,66% con un dropout de 0,3. Después ha bajado de forma también aproximadamente lineal hasta un dropout de 0,8, volviendo a subir al incrementarse más el dropout.

# 4. Red convolucional

## 4.1. Implementación y algoritmo

Se ha implementado en Python usando las bibliotecas Tensorflow y Keras una red convolucional con un número variable entre 1 y 5 tanto de capas convolutivas como de capas densas, que pueden ser ajustadas con 2 constantes en el programa. Google Colab fue utilizado para ejecutar el código.

La mejor tasa de error sobre el conjunto de prueba, 0,88%, se ha obtenido con 1 capa de entrada, 2 capas convolucionales ocultas que hacen convolución espacial sobre imágenes, 1 capa oculta de max-pooling para datos espaciales, 1 capa densa oculta y 1 capa densa de salida. La primera capa oculta tiene 32 filtros, la segunda tiene 64. Las dos capas cuentan con un kernel de 3 por 3. La capa de pooling tiene un pool de 2 por 2. La capa densa oculta tiene 128 neuronas, y la capa densa de salida tiene 10 neuronas. Las neuronas de las capas convolucionales y de la capa densa oculta utilizan la función softmax como función de activación. Las neuronas de la capa densa de salida son unidades lineales rectificadas, utilizando así un rectificador como función de activación.

Una vez que las imágenes de la base de datos MNIST son grises, la red solo tiene un canal. La red se entrena en 10 épocas. Se aplica un dropout de 0.25 antes de la capa de max-pooling, y un dropout de 0.5 antes de la capa densa de salida. Los pesos de las neuronas son inicializados con valores entre -0.05 y 0.05 obtenidos usando una distribución uniforme. Los bias de las neuronas son inicializados con 1. Se utiliza como función de optimización de la red Adadelta, que por recomendación del próprio Keras se deja con sus valores por defecto, incluso su tasa de aprendizaje de 1. La función de

pérdida es el Categorical Crossentropy, también con sus valores por defecto.

El principal tutorial utilizado se encuentra en

https://towardsdatascience.com/build-your-own-convolution-neural-network-in-5-mins-4217c2cf964f.

### 4.2. Experimentos realizados

Se han realizado un conjunto de experimentos para observar el impacto del número de capas de la red en la tasa de error. Se ha empezado con 1 capa convolutiva y 1 capa densa, la de salida, y después de cada experimento se ha añadido 1 capa oculta convolutiva y 1 capa oculta densa hasta haber 5 capas convolutivas y 4 densas ocultas.

Los restantes parámetros de la red se han mantenido constantes al largo de los diferentes experimentos:

- Épocas = 1
- Dropout de las capas convolucionales = 0,25
- Dropout de las capas densas = 0,5
- Filtros de la primera capa convolucional = 32
- Filtros de las restantes capas convolucionales (cuando existentes) = 64
- Neuronas de las capas densas ocultas (cuando existentes) = 128
- Neuronas de la capa densa de salida = 10
- Tamaño del kernel de las capas convolutivas = 3 por 3
- Tamaño del pool de la capa max_pooling = 2 por 2
- Valores iniciales de los pesos = Distribución uniforme entre -0,05 y 0,05
- Valores iniciales de los bias = 1
- Función de activación de la capa de salida = ReLU
- Funciones de activación de las restantes capas (exceptuando la capa de max-pooling) = Softmax
- Función de optimización = Adadelta
- Función de pérdida = Categorical Crossentropy

| **Capas** | **1** | **2** | **3** | **4** | **5** |
|---|---|---|---|---|---|
| **Tasa** | 3,24% | 1,53% | 1,74% | 1,96% | 2,25% |

*Tabla 3 - Tasa de error en función del número de capas convolutivas y densas de la red*

Los mejores resultados se han obtenido para 2 capas convolutivas y densas, 1,53% de tasa de error, aunque los resultados con 3 y 4 capas convolutivas se acerquen. Las tasas de error para 5 y 1 capas convolutivas y densas también no son muy grandes, no superando los 3,5%.