

2022

24 / 11 / 2022

# Practica 2

SISTEMAS INTELIGENTES

ADRIAN UBEDA TOUATI 50771466R

## Contenido

Parte 1: Aprende las bases de un MLP .....	2
I1) Resolviendo una función booleana mediante MLP .....	2
A) .....	2
B) .....	5
I2) Modelar, entrenar y probar la red en Keras .....	8
A) .....	8
B) .....	9
C) .....	9
D) .....	11
I3) Analizar el entrenamiento y comparar con la red ajustada a mano .....	12
A) .....	12
B) .....	13
C) .....	14
Parte2: Entrena un MLP mediante Deep Learning usando Keras .....	15
II1) Procesamiento de los datos .....	15
A) .....	15
B) .....	16
II2) Implementa la red en keras .....	20
C) .....	20
D) .....	20
II3) Prueba el modelo .....	22
E) .....	22

## Parte 1: Aprende las bases de un MLP

### II) Resolviendo una función booleana mediante MLP

A)

A) Diseña una red MLP con funciones de activación sigmoidea que resuelva tu función booleana. **Importante: no puedes simplificar la función, debes computarla con todos los términos  $T_n$  que contenga (aunque por ejemplo tengas alguno duplicado)**

La función que tenemos que enfrentar es:

**771466	$(a \wedge b \wedge c \wedge d) \vee (\bar{b} \wedge \bar{d}) \vee (\bar{a} \wedge \bar{c} \wedge d) \vee (\bar{a} \wedge c \wedge d) \vee (\bar{a} \wedge b \wedge \bar{c} \wedge d)$
----------	--

- Debes calcular los pesos y umbrales de activación que debe tener la red. Puedes utilizar una calculadora (o Python, que es lo que recomendamos) pero no una librería neuronal para su cálculo.

Para el cálculo de los pesos y umbrales, he utilizado Excel.

Aunque parezca una herramienta extravagante, me ha facilitado mucho el calculo de los pesos y umbrales, ya que cuando modificamos un valor, el resto de los valores se modifican al instante sin necesidad de una ejecución.

Primero he implementado la tabla de verdad de cada uno de los términos, siendo estos 5

a	b	c	d	T1	T2	T3	T4	T5
0	0	0	0	1	1	0	0	0
0	0	0	1	0	0	1	0	0
0	0	1	0	0	1	0	0	0
0	0	1	1	0	0	0	1	0
0	1	0	0	0	0	0	0	0
0	1	0	1	0	0	1	0	1
0	1	1	0	0	0	0	0	0
0	1	1	1	0	0	0	1	0
1	0	0	0	0	1	0	0	0
1	0	0	1	0	0	0	0	0
1	0	1	0	0	1	0	0	0
1	0	1	1	0	0	0	0	0
1	1	0	0	0	0	0	0	0
1	1	0	1	0	0	0	0	0
1	1	1	0	0	0	0	0	0
1	1	1	1	0	0	0	0	0

Antes de calcular los pesos me pregunte como seria la estructura de la red neuronal, como sabemos tenemos 5 términos por lo que utilizaremos una neurona para cada termino. Después todos estos términos se juntan con un or, para implementar esta fusión, he puesto una última neurona que se activara si al menos una de 5 neuronas se activa.

## PRACTICA 2

Una vez la tabla de verdad echa, he procedido a calcular los pesos para la primera neurona.

He ingresado la siguiente formula en cada fila del primer término:

SUMA										=A\$20+B\$20*A2+C\$20*B2+D\$20*C2+E\$20*D2												
	A	B	C	D	E	F	G	H	I													
1	a	b	c	d	T1	T2	T3	T4	T5													
2	0	0	0	0	1	1	0	0	0													
3	0	0	0	1	0	0	1	0	0													
4	0	0	1	0	0	1	0	0	0													
5	0	0	1	1	0	0	0	1	0													
6	0	1	0	0	0	0	0	0	0													
7	0	1	0	1	0	0	1	0	1													
8	0	1	1	0	0	0	0	0	0													
9	0	1	1	1	0	0	0	1	0													
10	1	0	0	0	0	1	0	0	0													
11	1	0	0	1	0	0	0	0	0													
12	1	0	1	0	0	1	0	0	0													
13	1	0	1	1	0	0	0	0	0													
14	1	1	0	0	0	0	0	0	0													
15	1	1	0	1	0	0	0	0	0													
16	1	1	1	0	0	0	0	0	0													
17	1	1	1	1	0	0	0	0	0													
18																						
19	w0	wa	wb	wc	wd																	
20	1	-2	-2	-2	-2																	
21																						
22	T1											T2										
23	=A\$20+											3										
24	-1											-3										
25	-1											5										
26	-3											-1										
27	-1											-3										
28	-3											-9										
29	-3											-1										
30	-5											-7										
31	-1											1										
32	-3											-5										
33	-3											3										
34	-5											-3										
35	-3											-5										
36	-5											-11										
37	-5											-3										
38	-7											-9										
39																						

Lo que se busca es que los términos en verde sean positivos y los azules negativos, para cuando se aplique la función sigmoidea, de  $>0.5$  si positivo o  $<0.5$  si negativo.

La fórmula sigue

$$w \cdot x + b$$

Siendo w0 el b es decir el peso de la propia neurona, y  $w \cdot x$  la entrada multiplicada por el peso del camino

## PRACTICA 2

Y vamos cambiando los pesos hasta conseguir el resultado deseado

w0	wa	wb	wc	wd		w0	wa	wb	wc	wd
1	-2	-2	-2	-2		3	-2	-6	2	-6
T1						T2				
=AS20+						3				
-1						-3				
-1						5				
-3						-1				
-1						-3				
-3						-9				
-3						-1				
-5						-7				
-1						1				
-3						-5				
-3						3				
-5						-3				
-3						-5				
-5						-11				
-5						-3				
-7						-9				

w0	wa	wb	wc	wd		w0	wa	wb	wc	wd
-1	-4	-1	-4	3		-6	-6	-1	5	5
T3						T4				
-1						-6				
2						-1				
-5						-1				
-2						4				
-2						-7				
1						-2				
-6						-2				
-3						3				
-5						-12				
-2						-7				
-9						-7				
-6						-2				
-6						-13				
-3						-8				
-10						-8				
-7						-3				

w0	wa	wb	wc	wd
-4	-6	3	-6	3
T5				
-4				
-1				
-10				
-7				
-1				
2				
-7				
-4				
-10				
-7				
-16				
-13				
-7				
-4				
-13				
-10				

B)

B) Implementa en python una función `y = forward((a,b,c,d))` que reciba como parámetro **una tupla** de componentes  $(a, b, c, d)$  y calcule la fase *forward* de tu red. **Solo se pueden utilizar las librerías** `math` y `numpy` de python para este apartado.

- Esta función debe llamarse `forward` y debe recibir y retornar los parámetros indicados. No debe imprimir nada por la salida estándar

Para saber si la neurona se activa o no utilizare la función decisión que sigue la formula

- **Definimos:**  $z = w \cdot x + b$

$$\sigma(z) = \frac{1}{1 + e^{-z}}$$

```
def decision(x,w,peso):
    resultado = 0

    for i in range(len(x)):
        resultado = resultado + x[i]*w[i]

    resultado = resultado + peso
    funcion = 1/(1+math.exp(-resultado))

    return funcion
```

Ahora hare la función forward:

```

Adrian Ubeda Touati *
def forward(a,b,c,d):

    x = [a,b,c,d]
    #Primer nodo
    w = [-2,-2,-2,-2]
    b = 1
    primerNodo = decision(x,w,b)

    #Segundo nodo
    w = [-2,-6,2,-6]
    b = 3
    segundoNodo = decision(x,w,b)

    # Tercer nodo
    w = [-4, -1, -4, 3]
    b = -1
    tercerNodo = decision(x, w, b)

    # Cuarto nodo
    w = [-6, -1, 5, 5]
    b = -6
    cuartoNodo = decision(x, w, b)

    # Quinto nodo
    w = [-6, 3, -6, 3]
    b = -4
    quintoNodo = decision(x, w, b)

    #Or
    x=[primerNodo,segundoNodo,tercerNodo,cuartoNodo,quintoNodo]
    w = [2, 2, 2, 2, 2]
    b = -2

    return decision(x, w, b)

```

Donde para cada nodo, establecemos x como los valores de entrada, w los pesos de cada camino y b el peso de la neurona.

## PRACTICA 2

- Si incluimos tu fichero de esta parte `p2base.py` debemos de poder gastar tu función. Verifícalo:

Para probar si la red está bien configurada he creado una función prueba:

En esta función llamo al método `forward` con cada valor posible, he almacenado en la columna aplicación cada resultado

a	b	c	d	Teoria	Aplicación
0	0	0	0	VERDADERO	True
0	0	0	1	VERDADERO	True
0	0	1	0	VERDADERO	True
0	0	1	1	VERDADERO	True
0	1	0	0	FALSO	False
0	1	0	1	VERDADERO	True
0	1	1	0	FALSO	False
0	1	1	1	VERDADERO	True
1	0	0	0	VERDADERO	True
1	0	0	1	FALSO	False
1	0	1	0	VERDADERO	True
1	0	1	1	FALSO	False
1	1	0	0	FALSO	False
1	1	0	1	FALSO	False
1	1	1	0	FALSO	False
1	1	1	1	FALSO	False

Por lo que la red queda verificada



## I2) Modelar, entrenar y probar la red en Keras

A)

Crea una rutina que evalúe tu función booleana para todo el dominio (todas las combinaciones de  $a, b, c, d$ ). Guarda los resultados en dos vectores  $X$  (entrada) e  $Y$  (salida booleana de la función).

He aprovechado la función prueba hecha anteriormente para almacenar las entradas y los resultados en las 2 listas

```
def prueba():
    global Y
    global X
    Y.append(forward(0, 0, 0, 0) > 0.5)
    X.append((0, 0, 0, 0))
    Y.append(forward(0, 0, 0, 1) > 0.5)
    X.append((0, 0, 0, 1))
    Y.append(forward(0, 0, 1, 0) > 0.5)
    X.append((0, 0, 1, 0))
    Y.append(forward(0, 0, 1, 1) > 0.5)
    X.append((0, 0, 1, 1))
    Y.append(forward(0, 1, 0, 0) > 0.5)
    X.append((0, 1, 0, 0))

    Y.append(forward(0, 1, 0, 1) > 0.5)
    X.append((0, 1, 0, 1))
    Y.append(forward(0, 1, 1, 0) > 0.5)
    X.append((0, 1, 1, 0))
    Y.append(forward(0, 1, 1, 1) > 0.5)
    X.append((0, 1, 1, 1))
    Y.append(forward(1, 0, 0, 0) > 0.5)
    X.append((1, 0, 0, 0))
    Y.append(forward(1, 0, 0, 1) > 0.5)
    X.append((1, 0, 0, 1))

    Y.append(forward(1, 0, 1, 0) > 0.5)
    X.append((1, 0, 1, 0))
    Y.append(forward(1, 0, 1, 1) > 0.5)
    X.append((1, 0, 1, 1))
    Y.append(forward(1, 1, 0, 0) > 0.5)
    X.append((1, 1, 0, 0))
    Y.append(forward(1, 1, 0, 1) > 0.5)
    X.append((1, 1, 0, 1))
    Y.append(forward(1, 1, 1, 0) > 0.5)
    X.append((1, 1, 1, 0))

    Y.append(forward(1, 1, 1, 1) > 0.5)
    X.append((1, 1, 1, 1))
```

B)

Diseña una red idéntica al MLP anterior en keras. Utiliza como función de coste/pérdida/error MSE y Adam como algoritmo de optimización. Tu código será parecido a este, ajustando el número de capas y las neuronas por capa (debes sustituir el '?'):

He modificado un poco el código dando este el siguiente resultado:

```
def entrenando():
    model = keras.Sequential(
        [
            layers.Dense(units=5, input_shape=[4], activation="sigmoid"),
            layers.Dense(units=1, activation="sigmoid"),
        ]
    )

    model.compile(loss="mean_squared_error", optimizer="adam")
```

Tenemos 5 nodos entrelazados que reciben 4 valores de entrada que después se asocian con 1, esta red es la que hemos realizado con anterioridad a mano

C)

Entrena la red. Eso lo debes hacer mediante la función `model.fit` de keras. Determina que `batch_size` es conveniente utilizar y el número de épocas (epochs) para que el entrenamiento sea fructífero.

```
model.fit(X, Y, epochs=2000, batch_size=16, verbose=False)

#model.predict([Y[0]])
for tubla in X:
    print(f"{tubla} {model.predict([tubla]) > 0.5}")
```

El `batch_size` es el tamaño de la muestra, este tamaño puede tomar como máximo 16 ya que solo tenemos 16 valores para entrenar la red

El número de épocas debe ser el suficiente para que el error de la red sea bajo, probaremos con 2000

También hemos puesto `verbose` a `False` para no ver los mensajes generados por el entrenamiento

## PRACTICA 2

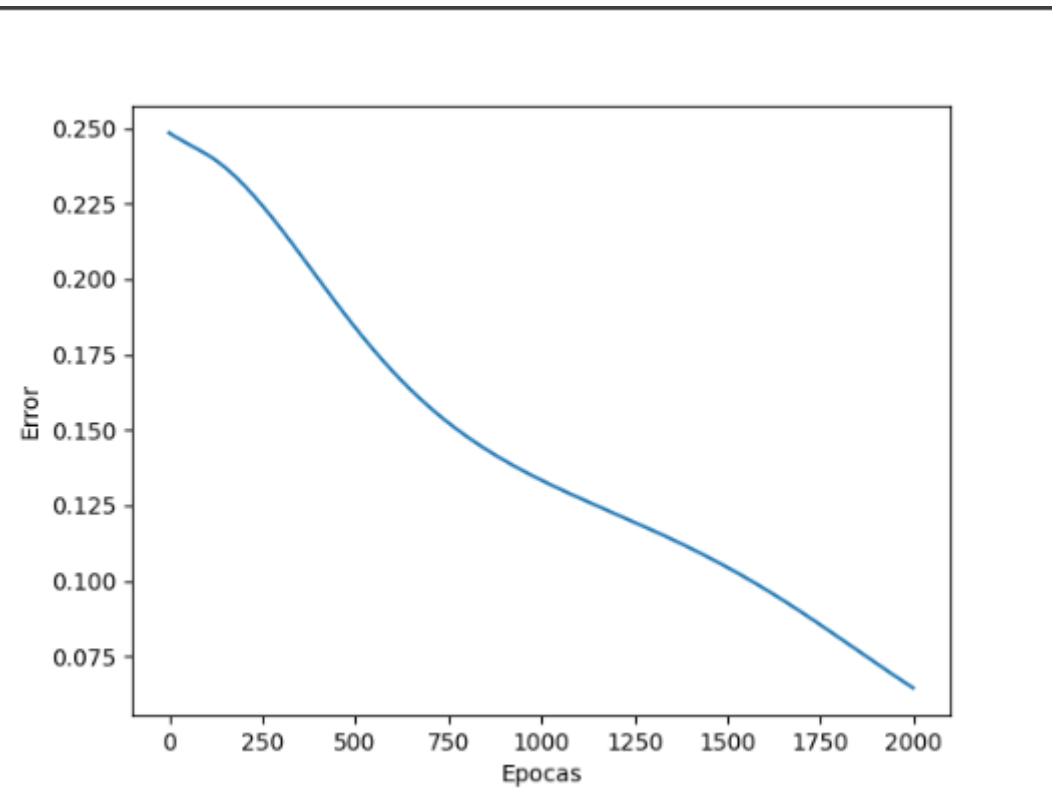
Antes de ponerlo a entrenar, modificamos el código para que así podamos ver el error que se genera por épocas, usaremos la librería matplotlib para poder representar una curva que nos ayudara a entender el entrenamiento

```
historial = model.fit(X, Y, epochs=2000, batch_size=16, verbose=False)

#model.predict([Y[0]])
for tubla in X:
    print(f"{tubla} {model.predict([tubla]) > 0.5}")

plt.xlabel("Epocas")
plt.ylabel("Error")
plt.plot(historial.history["loss"])
plt.show()
```

Si ejecutamos el código nos genera la siguiente grafica



Con 2000 epochs, ya conseguiríamos un error relativamente bajo  $< 0.075$

D)

Comprueba cómo funciona la red con tus conjuntos  $X, Y$ . Debes utilizar la función de keras `model.predict` y determinar cuál es la tasa de acierto de tu red para tu función booleana.

Gracias a esta parte del código vemos los resultados previstos para cada entrada posible:

```
for tubla in X:
    print(f"{tubla} {model.predict([tubla]) > 0.5}")
```

Cuando hacemos predict, podemos ver que el resultado corresponde

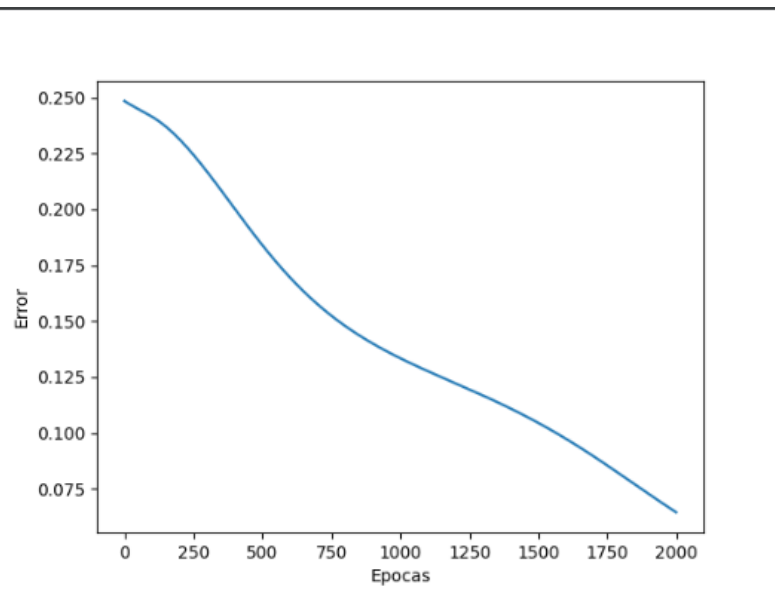
```
1/1 [=====] - 0s 83ms/step
(0, 0, 0, 0) [[ True]]
1/1 [=====] - 0s 35ms/step
(0, 0, 0, 1) [[ True]]
1/1 [=====] - 0s 35ms/step
(0, 0, 1, 0) [[ True]]
1/1 [=====] - 0s 37ms/step
(0, 0, 1, 1) [[ True]]
1/1 [=====] - 0s 36ms/step
(0, 1, 0, 0) [[False]]
1/1 [=====] - 0s 35ms/step
(0, 1, 0, 1) [[ True]]
1/1 [=====] - 0s 36ms/step
(0, 1, 1, 0) [[False]]
1/1 [=====] - 0s 36ms/step
(0, 1, 1, 1) [[ True]]
```

13) Analizar el entrenamiento y comparar con la red ajustada a mano

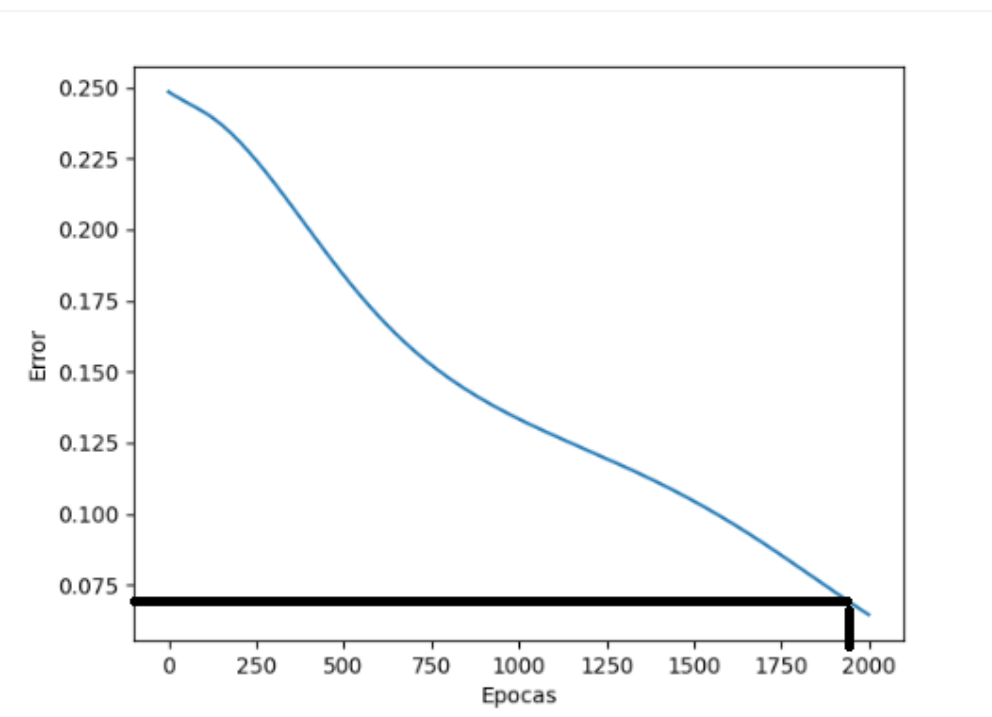
A)

¿Cuántos pasos de entrenamiento necesitas para que aprenda correctamente la función? ¿Te parecen muchos?

Primero veamos la curva generada con el entrenamiento:



Consideremos que la función ha aprendido correctamente si llega al 95% de aciertos es decir  $<0.05$  de error.



Como podemos ver este se consigue por las 1800 – 1900 épocas

Parecen muchas épocas

B)

**¿Qué es el parámetro loss y optimizer de la red?**

```
model.compile(loss="mean_squared_error", optimizer="adam")
```

**loss:** Es el encargado de elegir una función de pérdida, esta función de pérdida es la encargada de calcular la cantidad que un modelo debería tratar de minimizar durante el entrenamiento, en nuestro caso hemos utilizado mean\_squared\_error

**optimizer:** Es el encargado de elegir un algoritmo para calcular los pesos, en nuestro caso hemos utilizado Adam

**¿Crees que afectan al entrenamiento?**

El algoritmo de búsqueda de pesos afecta directamente al entrenamiento, ya que, dependiendo de esta, el tiempo de entrenamiento será mayor o menor.

Pasa lo mismo con el manejo de errores, por lo que las 2 variables son cruciales y afectaran enormemente al entrenamiento.

**¿Qué es el learning rate (LR) en una red neuronal?**

Es el proceso en el que la red aprende y progresa para intentar minimizar el error de los resultados

**¿Se puede ajustar el LR en Keras? Haz pruebas para intentar minimizar el número de pasos de entrenamiento requeridos. Comenta los resultados obtenidos**

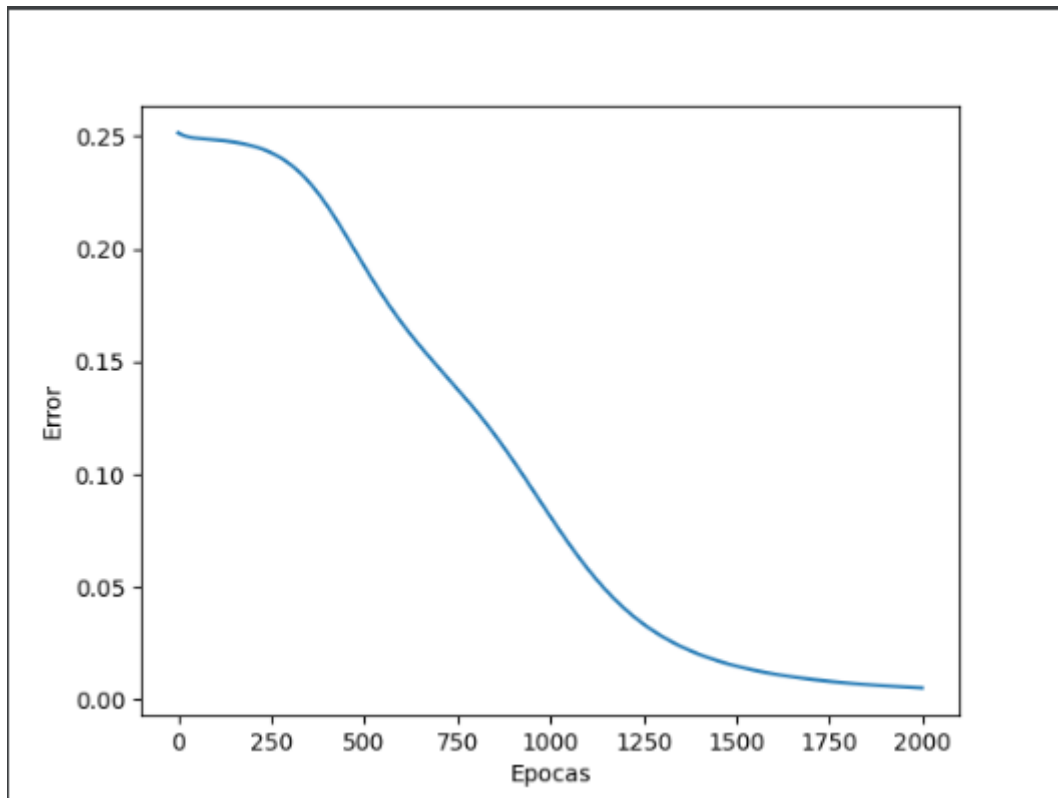
El LR, depende directamente de la red, y se puede ajustar en Keras, pero cambiando la estructura de la red, además del algoritmo de búsqueda y el manejo de pérdidas.

Si queremos minimizar el numero de pasos de entrenamiento requeridos, debemos ir probando diferentes combinaciones de esos 3 factores, en mi caso me enfocare en las capas de nodos.

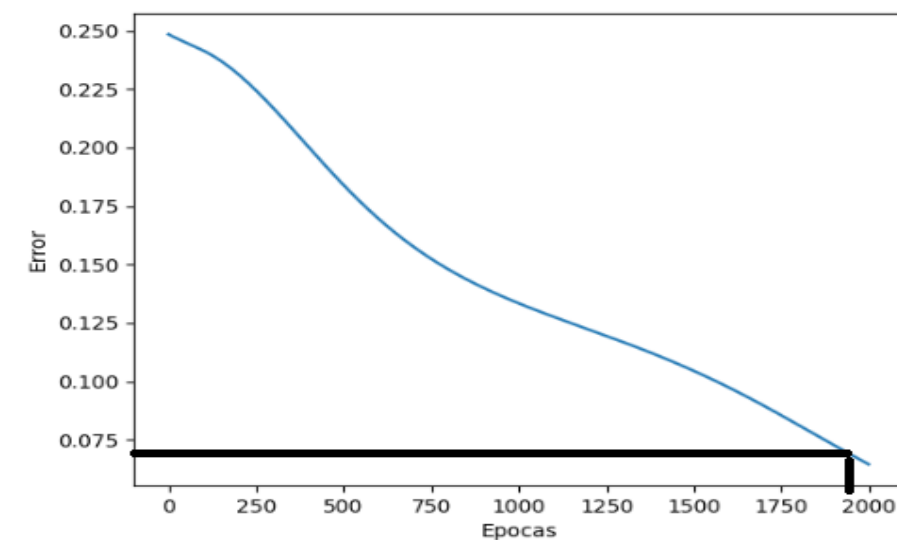
Antes tenia una capa de 5 y otra que era la salida de 1

Ahora he puesto una capa de 5 otra de 5 una de 3 y una última de 1

## PRACTICA 2



Como podemos ver esta curva de aprendizaje es mucho mejor que la anterior necesitando 1100 épocas en vez de 1900 como antes



C)

Compara los pesos y bias aprendidos con los que pusiste a mano ¿Se parecen? ¿Ha aprendido la misma función? Justifica tu respuesta. Ten en cuenta que puedes utilizar el siguiente código para acceder a los pesos del modelo una vez aprendido:

No se parecen en nada, la estructura de la red ha sido cambiada, además no tiene porque parecerse ya que hay infinitas soluciones, por lo que comparar los pesos no tiene sentido.

## Parte2: Entrena un MLP mediante Deep Learning usando Keras

### III) Procesamiento de los datos

A)

a) Busca información sobre el conjunto/base de datos MNIST

(<http://yann.lecun.com/exdb/mnist/>) y explica con detalle para que sirve y su importancia. Una vez que hayas comprendido su estructura cárgalo utilizando la librería para *deep learning* keras (<https://keras.io/>). Esta librería ya incluye una directiva sencilla para cargar el conjunto: `keras.datasets.mnist.load_data()` (puedes utilizarla).

La base de datos MNIST almacena una gran cantidad de imágenes de números escritos manualmente, se utiliza para entrenar la red, como tiene muchos ejemplos, el entrenamiento es muy completo y se consigue una tasa de error muy baja.

Ahora estudiemos la directiva que carga el conjunto:

```
(x_train,y_train), (x_test,y_test) = keras.datasets.mnist.load_data()
```

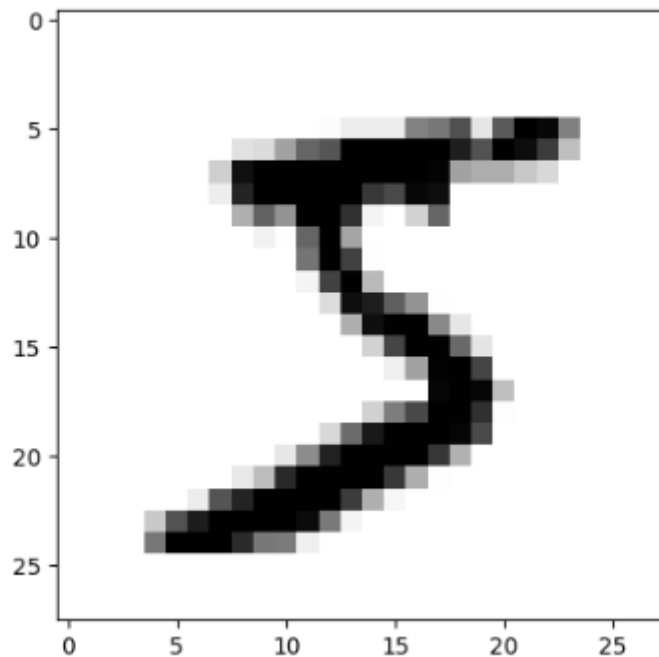
Analicemos su contenido con el siguiente código:

```
print(len(x_train[0]))
print(x_train[0])
plt.figure()
plt.imshow(x_train[0], cmap=plt.cm.binary)
plt.show()
```

Lo que imprime:

```
28
[[ 0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0
  0  0  0  0  0  0  0  0  0]
 [ 0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0
  0  0  0  0  0  0  0  0  0]
 [ 0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0
  0  0  0  0  0  0  0  0  0]
 [ 0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0
  0  0  0  0  0  0  0  0  0]
 [ 0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0
  0  0  0  0  0  0  0  0  0]
 [ 0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0
  0  0  0  0  0  0  0  0  0]
 [ 0  0  0  0  0  0  0  0  0  0  0  0  3  18  18  18 126 136
175 26 166 255 247 127  0  0  0  0]
 [ 0  0  0  0  0  0  0  0 30 36 94 154 170 253 253 253 253 253
225 172 253 242 195 64  0  0  0  0]
```





Como podemos ver, el primer argumento tiene 28 listas que cada una de ellas tiene 28 elementos.

Estos hacen referencia a la cantidad de pixeles que hay en la imagen, y tienen como valor un numero del 0 al 255, que corresponde al tono de gris del píxel siendo 255 totalmente negro y 0 totalmente blanco

El 5 impreso en la gráfica, es el resultado de todos esos valores juntos

B)

**b) Dado el formato de la base de datos MNIST para clasificar dígitos manuscritos contesta las siguientes preguntas de manera detallada y razonada:**

**¿Cuántas neuronas necesitamos en la capa de entrada? ¿Y en la capa de salida?**

En la capa de entrada podríamos asociar un píxel a cada neurona de entrada, en ese caso deberíamos tener  $28 \times 28 = 784$  neuronas de entrada y de salida 10, una neurona para con una cadena que simbolice un número entre 0 y 9

```
model = keras.Sequential(
    [
        layers.Dense(units=128, input_shape=[784], activation="relu"),
        layers.Dense(units=128, activation="relu"),
        layers.Dense(units=10, activation="softmax"),
    ]
)
```

### ¿Cómo dividirás el conjunto de entrenamiento/test/validación?

El conjunto de entrenamiento tiene 60000 imágenes y el de test/validación tiene 10000.

Las he recogido en estas 2 tuplas:

```
(x_train,y_train), (x_test,y_test) = keras.datasets.mnist.load_data()
```

X contiene las "imágenes" procesadas y Y los números digitales a los que corresponde

Es importante que el conjunto de entrenamiento sea el mayor posible pero no podemos testear imágenes que han sido entrenadas, por lo que es importante encontrar un equilibrio

### ¿Cómo preprocesaras los datos de entrada?

Las imágenes primero cambian de formato a una imagen de 28\*28 pixeles, después se vuelve a color binario, es decir, solo blanco o negro

### ¿Cómo transformarás la imagen para poder entrenarla con una red MLP?

Una vez la imagen preprocesada, tenemos que asociar a cada píxel el número que corresponde su brillo. Después se almacenan en una lista y ya están listas para ser poder realizar el entrenamiento

En el código, hemos tenido que adaptar los valores recogidos, para x\_train y x\_test, hemos tenido que aplanar la lista para pasar de 28 listas de 28 elementos a una sola lista de 784 elementos

Después hemos normalizado el valor para que este entre 0 y 1

```
x_train = x_train.reshape((x_train.shape[0],28*28))
x_train = x_train.astype('float32') / 255.0
```

```
x_test = x_test.reshape((x_test.shape[0], 28 * 28))
x_test = x_test.astype('float32') / 255.0
```

Para y\_test y y\_train, hemos tenido que transformar el numero decimal en una cadena de números que representaran cada posibilidad del 0 al 9

```

def salida(y_train):
    i = 0
    salida = []

    for elemento in y_train:
        if elemento == 0:
            salida.append([1, 0, 0, 0, 0, 0, 0, 0, 0, 0])
        elif elemento == 1:
            salida.append([0, 1, 0, 0, 0, 0, 0, 0, 0, 0])
        elif elemento == 2:
            salida.append([0, 0, 1, 0, 0, 0, 0, 0, 0, 0])
        elif elemento == 3:
            salida.append([0, 0, 0, 1, 0, 0, 0, 0, 0, 0])
        elif elemento == 4:
            salida.append([0, 0, 0, 0, 1, 0, 0, 0, 0, 0])
        elif elemento == 5:
            salida.append([0, 0, 0, 0, 0, 1, 0, 0, 0, 0])
        elif elemento == 6:
            salida.append([0, 0, 0, 0, 0, 0, 1, 0, 0, 0])
        elif elemento == 7:
            salida.append([0, 0, 0, 0, 0, 0, 0, 1, 0, 0])
        elif elemento == 8:
            salida.append([0, 0, 0, 0, 0, 0, 0, 0, 1, 0])
        elif elemento == 9:
            salida.append([0, 0, 0, 0, 0, 0, 0, 0, 0, 1])
        i += 1

    return salida

```

Y después estos valores transformarlos a int32 para que sea compatible en la red neuronal

```

y_train = salida(y_train)
y_train = numpy.array(y_train, dtype=numpy.int32)

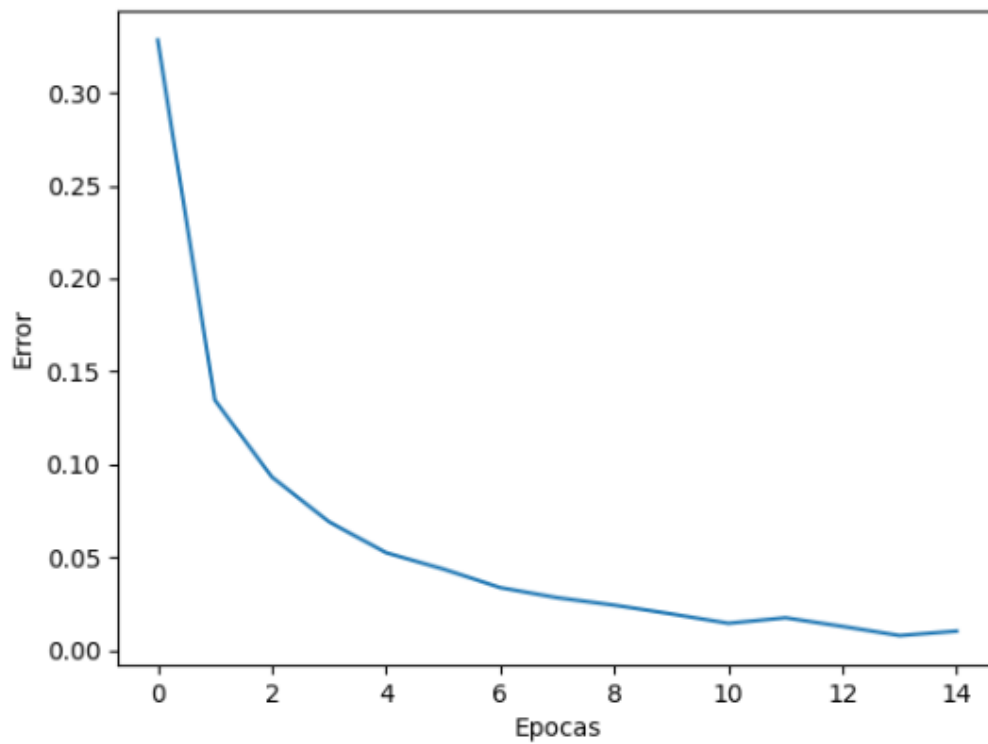
y_test = salida(y_test)
y_test = numpy.array(y_test, dtype=numpy.int32)

```

## PRACTICA 2

Una vez la red compilada y entrenada esto es la gráfica que obtenemos:

```
model.compile(loss="categorical_crossentropy", optimizer="adam", metrics=["accuracy"])  
  
historial = model.fit(x_train, y_train, epochs=15, batch_size=128, validation_split=0.1)  
  
plt.xlabel("Epocas")  
plt.ylabel("Error")  
plt.plot(historial.history["loss"])  
plt.show()
```



## II2) Implementa la red en keras

Implementa la red en keras. Para ello debes utilizar exclusivamente capas de tipo Dense, ya que deseamos desarrollar la red como un *vanilla MLP*. Utilizaremos funciones de activación `relu` para las capas ocultas y `softmax` para la capa de salida. Gastaremos dos capas ocultas con 128 neuronas. Por tanto, el diseño de la red será parecido a:

C)

c) Contesta detalladamente a las siguientes preguntas:

**¿Qué es la función de activación `relu`? ¿Cómo varia de la sigmoidea vista en teoría?**

La función `relu` es una función de activación, permite a la red neuronal buscar más allá de las soluciones lineales y así poder encontrar soluciones más complejas.

La función de activación `relu`, tiene la propiedad que todo valor mayor que 0 conserva su valor, pero todo valor menor que 0 es sustituido por 0

**¿Qué consigue la función de activación `softmax` de la última capa? ¿Se podría usar una función de activación `relu` en la última capa?**

Esta red neuronal es de clasificación, ya que se recoge unos datos, para categorizarlos posteriormente, la función de activación `softmax` se utiliza en redes neuronales de clasificación puesto que nos da un porcentaje que nos indica la probabilidad de que la imagen pertenezca a una categoría, se usara solo en la última capa

No se podría usar `relu` puesto que nos devolvería un numero y no un porcentaje

**¿Qué función de activación crees que es mejor? ¿Por qué crees que se suelen utilizar más `relu` que su alternativa sigmoidea?**

Como el brillo es expresado de 255 a 0, no necesitamos números negativos por lo que con la función `relu` los eliminamos, después como la función `relu` es también más eficiente que sigmoidea

D)

d) Contesta razonadamente a las siguientes preguntas

**Explica que son y para qué sirven los parámetros `batch_size` y `validation_split`.**

`batch_size`: Número de muestras por actualización de gradiente que serán procesadas

`validation_split`: es una fracción de los datos de entrenamiento que se usará como datos de validación que fluctúan entre 0 y 1 en tipo flotante

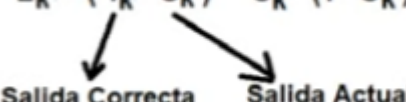
¿Conoces algún otro algoritmo de optimización distinto adam? Comenta brevemente el funcionamiento de otro distinto.

El algoritmo de Backpropagation

Es un algoritmo de ajuste de pesos.

Primero se mira el error en la capa de salida y posteriormente se ajustan los pesos, de igual forma en la capa de salida-1 se mirará el error y se ajustaran nuevamente los pesos

El error en la capa de salida se calcularía de la siguiente forma:

$$E_k = (T_k - O_k) * O_k * (1 - O_k)$$


Salida Correcta      Salida Actual

El error en las capas ocultas se calcularía de esta forma:

$$E_j = O_j * (1 - O_j) * \sum E_k * W_{jk}$$

Y el peso se recalcularía de la siguiente forma:

$$W_{jk} = W_{jk} + L * E_k * O_j$$


peso      %aprendizaje      entrada al nodo K desde J

### II3) Prueba el modelo

Prueba el modelo. Keras permite utilizar cualquier modelo entrenado de manera sencilla.

E)

e) Una vez entrenado el modelo contesta a las siguientes preguntas:

¿Qué error de clasificación obtienes para los datos de test? ¿Y qué valor de pérdida de la función de coste? Para este apartado gasta la función de keras `model.evaluate`

```
print(model.evaluate(x_test, y_test))
```

Nos que nos da como resultado:

```
[0.10578954964876175, 0.9750000238418579]
```

Escoge 1000 elementos al azar del conjunto de entrenamiento y comprueba que tasa de error de clasificación obtienes. ¿Coincide con la métrica anterior?

La métrica anterior era la siguiente:

```
[0.10578954964876175, 0.9750000238418579]
```

Y esta es la obtenida con la del conjunto entrenamiento

```
1000 aleatorios de entrenamiento [0.013327726162970066, 0.9929999709129333]
```

La tasa de error es infinitamente mas pequeña, esto es debido a que la red se ha entrenado con estos valores por los que los tiene interiorizados del entrenamiento, aunque algún ejemplo haya fallado

Escoge 1000 elementos al azar del conjunto de test y comprueba que tasa de error de clasificación obtienes. ¿Coincide con la métrica anterior?

La métrica anterior era la siguiente:

```
[0.10578954964876175, 0.9750000238418579]
```

Y esta es la obtenida con la del conjunto entrenamiento

```
1000 aleatorios de test [0.09408143162727356, 0.9819999933242798]
```

Como podemos observar se asemejan mucho mas que en el anterior caso, esto es debido a que los valores en los 2 casos no han sido utilizados para el entrenamiento, por lo que la red aun los desconocía.