2022

24 / 11 / 2022

Practica 2

SISTEMAS INTELIGENTES

ADRIAN UBEDA TOUATI 50771466R

UNIVERSIDAD DE ALICANTE | Gr. 03 - 3 teoría

Contenido

Parte 1: Aprende las bases de un MLP

I1) 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	$\overline{(a \wedge b \wedge c \wedge d)} \vee (\bar{b} \wedge \bar{d}) \vee (\bar{a} \wedge \bar{c} \wedge d) \vee (\bar{a} \wedge c \wedge d) \vee (\bar{a} \wedge b \wedge \bar{c} \wedge d)$

- **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.

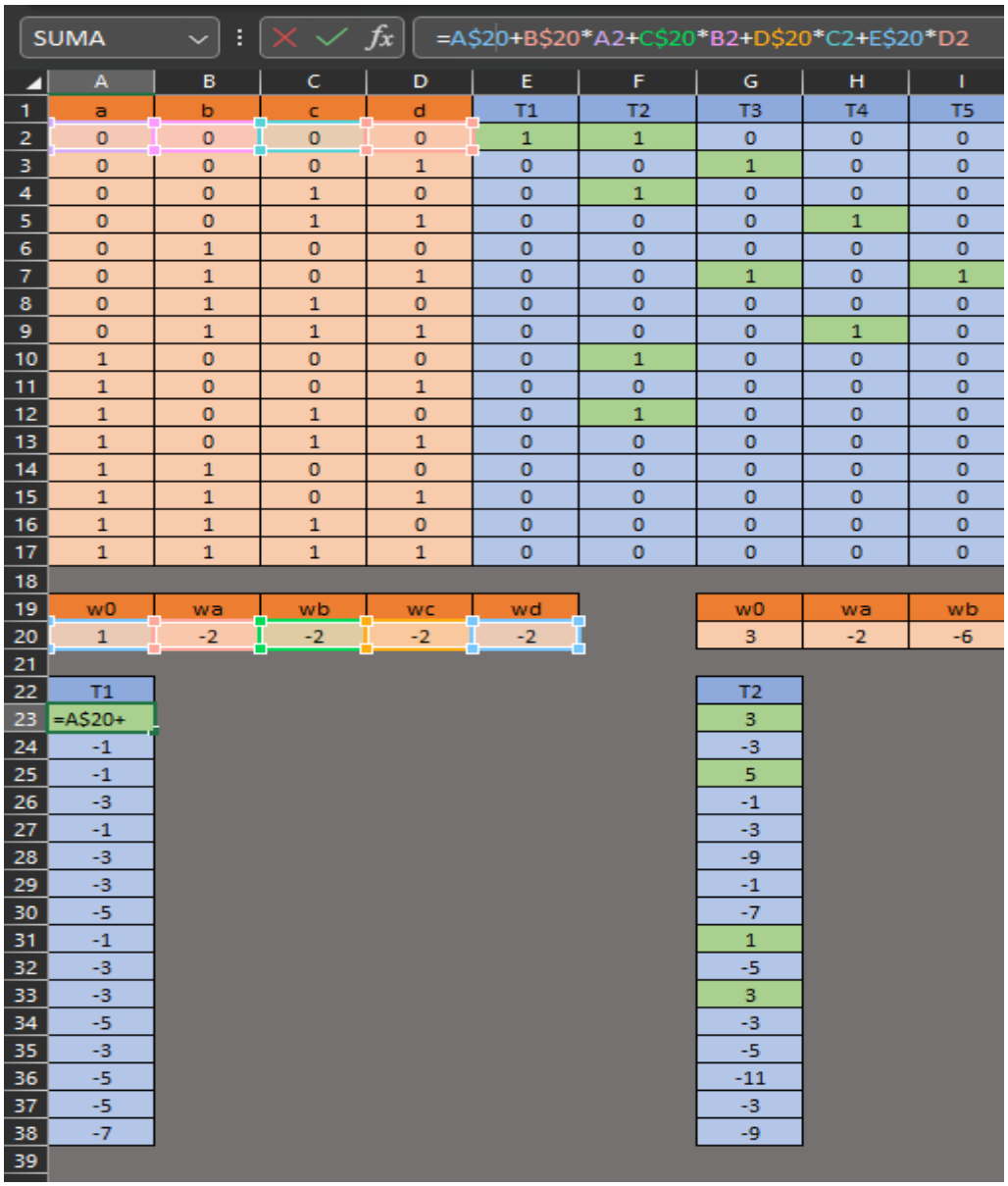
Primero he implementado la tabla de verdad de cada uno de los términos, siendo estos 5

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.

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:



	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

w0	wa	wb	wc	wd
1	-2	-2	-2	-2

w0	wa	wb
3	-2	-6

T1
=A$20+
-1
-1
-3
-1
-3
-3
-5
-1
-3
-3
-5
-3
-5
-5
-7

T2
3
-3
5
-1
-3
-9
-1
-7
1
-5
3
-3
-5
-11
-3
-9

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*x la entrada multiplicada por el peso del camino

Y vamos cambiando los pesos hasta conseguir el resultado deseado

w0	wa	wb	wc	wd
1	-2	-2	-2	-2

T1
=A$20+
-1
-1
-3
-1
-3
-3
-5
-1
-3
-3
-5
-3
-5
-5
-7

w0	wa	wb	wc	wd
3	-2	-6	2	-6

T2
3
-3
5
-1
-3
-9
-1
-7
1
-5
3
-3
-5
-11
-3
-9

w0	wa	wb	wc	wd
-1	-4	-1	-4	3

T3
-1
2
-5
-2
-2
1
-6
-3
-5
-2
-9
-6
-6
-3
-10
-7

w0	wa	wb	wc	wd
-6	-6	-1	5	5

T4
-6
-1
-1
4
-7
-2
-2
3
-12
-7
-7
-2
-13
-8
-8
-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.

– 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 echa 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 ser de 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

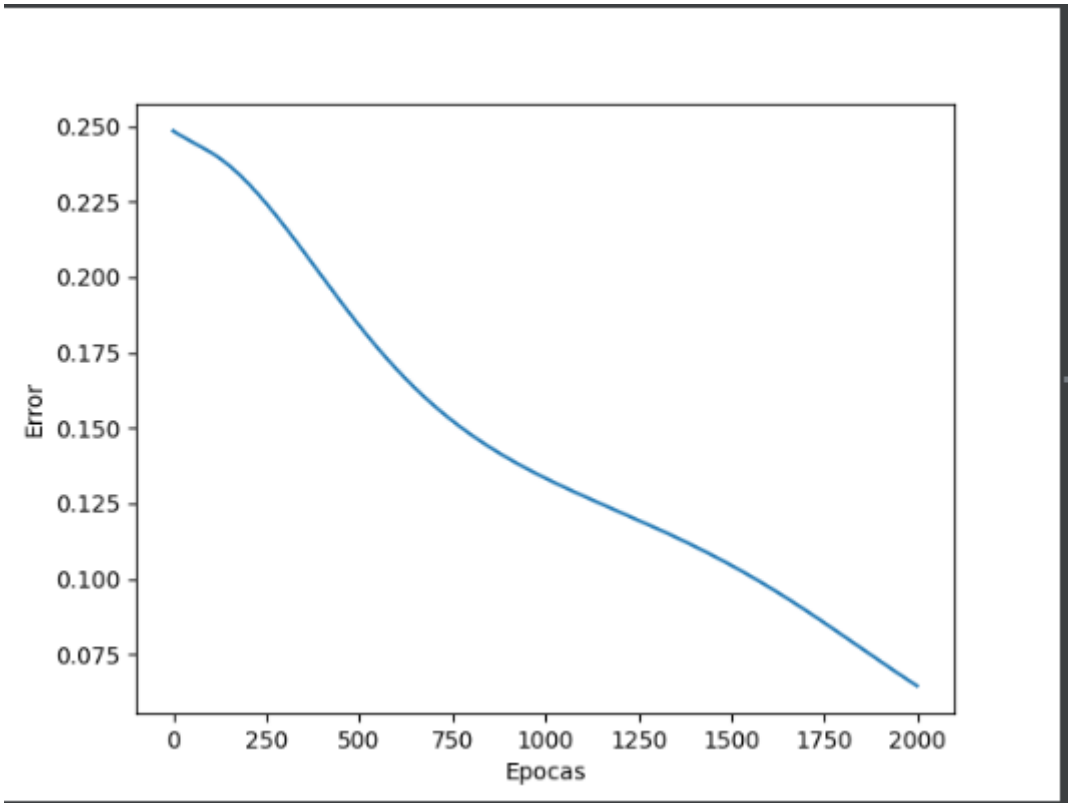
Antes de ponerlo a entrenar, modificamos el código para que así podamos ver el error que se genera por épocas, usaremos la librería matplotlib para poder representar una curva que nos ayudara a entender el entrenamiento

```
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 delco digo 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

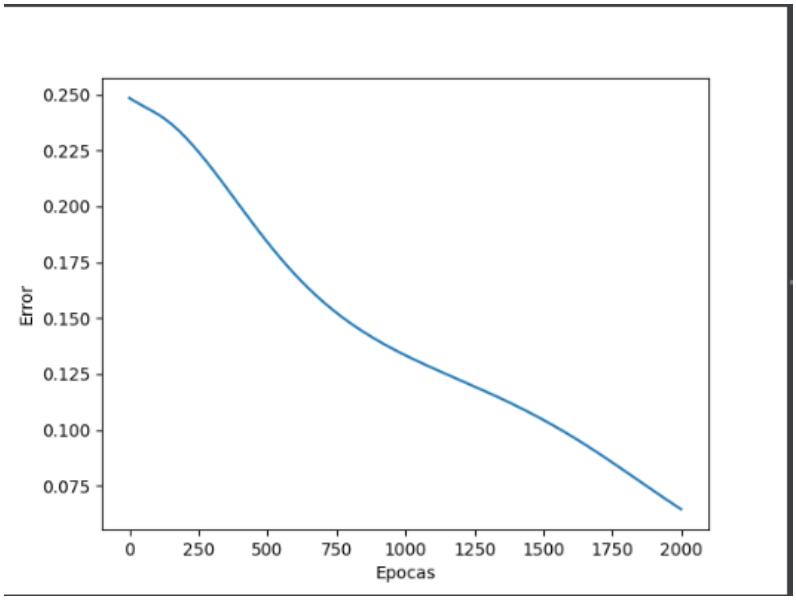
```
1/1 [==============================] - 0s 83ms/step
(0, 0, 0, 0) [[ True]]
1/1 [==============================] - 0s 35ms/step
(0, 0, 0, 1) [[ True]]
1/1 [==============================] - 0s 35ms/step
(0, 0, 1, 0) [[ True]]
1/1 [==============================] - 0s 37ms/step
(0, 0, 1, 1) [[ True]]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

I3) Analizar el entrenamiento y comparar con la red ajustada a mano

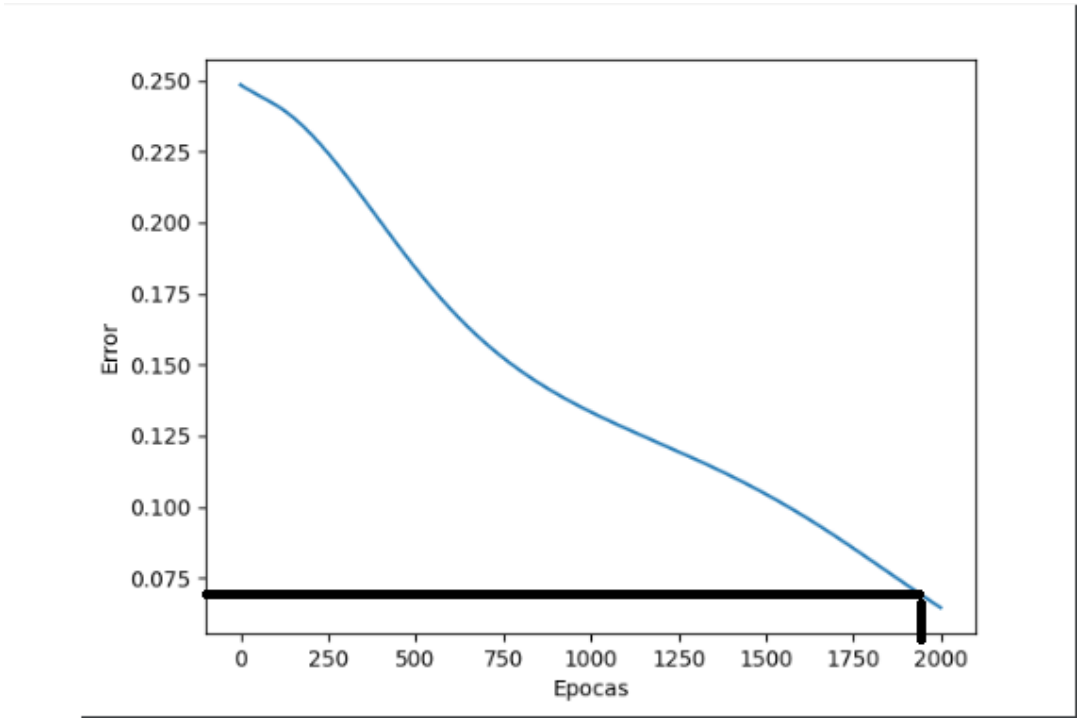
A)

¿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 perdida, esta función de perdida 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

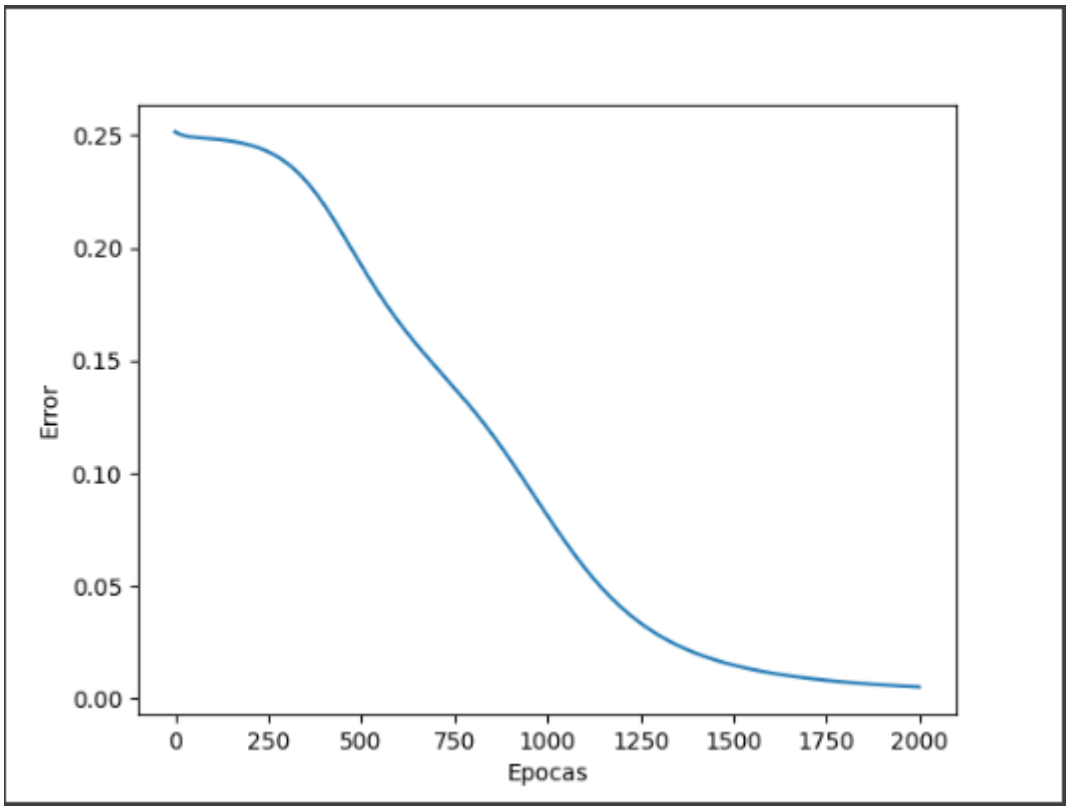
¿Se puede ajustar el LR en Keras? Haz pruebas para intentar minimizar el número de pasos de entrenamiento requeridos. Comenta los resultados obtenidos

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 perdidas.

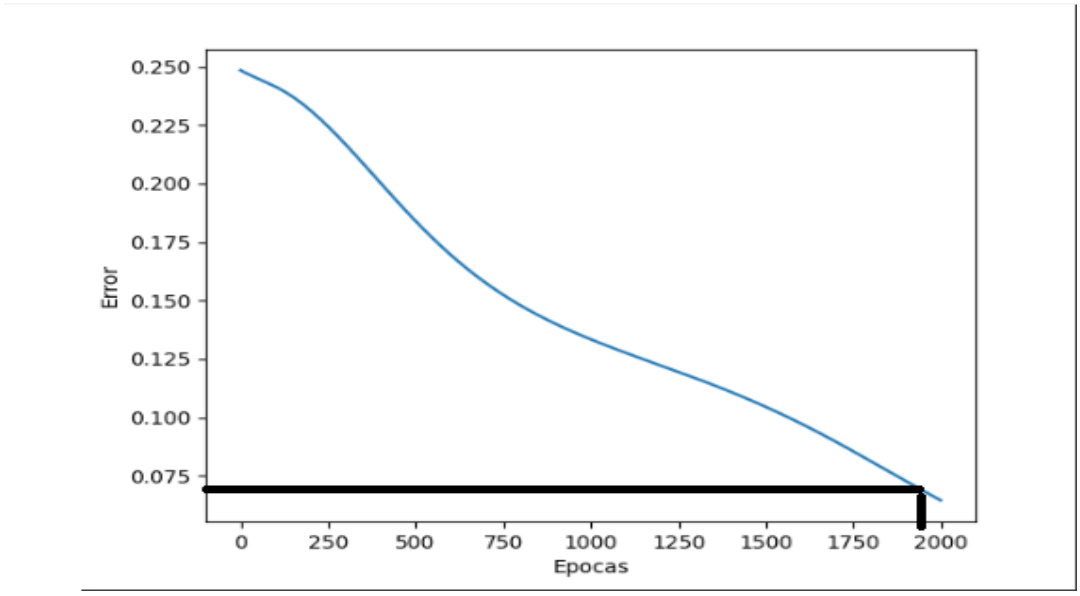
Si queremos minimizar el numero de pasos de entrenamiento requeridos, debemos ir probando diferentes combinaciones de esos 3 factores, en mi caso me enfocare en las capas de nodos.

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



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

II1) Procesamiento de los datos

A)

B)

II2) Implementa la red en keras

C)

D)

II3) Prueba el modelo

E)

II4) Mejora la red (opcional)