IES Pere Maria Orts

Sistemas de Aprendizaje Automático

Practicando con los perceptrones

Autor:
Kenny Berrones

Profesor:
David Campoy Miñarro

iesperemariaorts

GENERALITAT
VALENCIANA

Índice

1. Introducción

En esta práctica realizaremos distintas pruebas para entender el funcionamiento del perceptrón. En primer lugar, un perceptrón es un modelo simple de una neurona artificial que puede realizar operaciones de clasificación binaria.

2. Experimentos

Tendremos que realizar distintas pruebas para ir entendiendo como funciona el perceptrón.

2.1. Actividad 1

En esta parte vamos a entrenar el perceptrón para que aprenda la función lógica AND, esta función se define por la siguiente tabla:

A	B	$A \wedge B$
0	0	0
0	1	0
1	0	0
1	1	1

Cuadro 1: Tabla de verdad de la función AND

Como se ve en el código, por defecto tenemos que entrenaremos el perceptrón con 100 **épocas**, pero hemos ido modificando este valor y el valor con el que el perceptrón acierta correctamente la tabla de verdad ha sido de 5 épocas. El otro parámetro, el **learning rate** hemos probado y hemos visto que no tiene mucha influencia, ya que hemos probado con valores tanto bajos como altos. Respondiendo a la pregunta: **¿Cuántas épocas han sido necesarias?** Como hemos dicho, el valor de épocas con el que el perceptrón acepta correctamente son **5 épocas**. Tras entrenar el perceptrón obtenemos los siguientes pesos:

```
Pesos finales:  [-0.2  0.2  0.1]
```

Ahora vamos a dibujar la red neuronal con estos pesos:

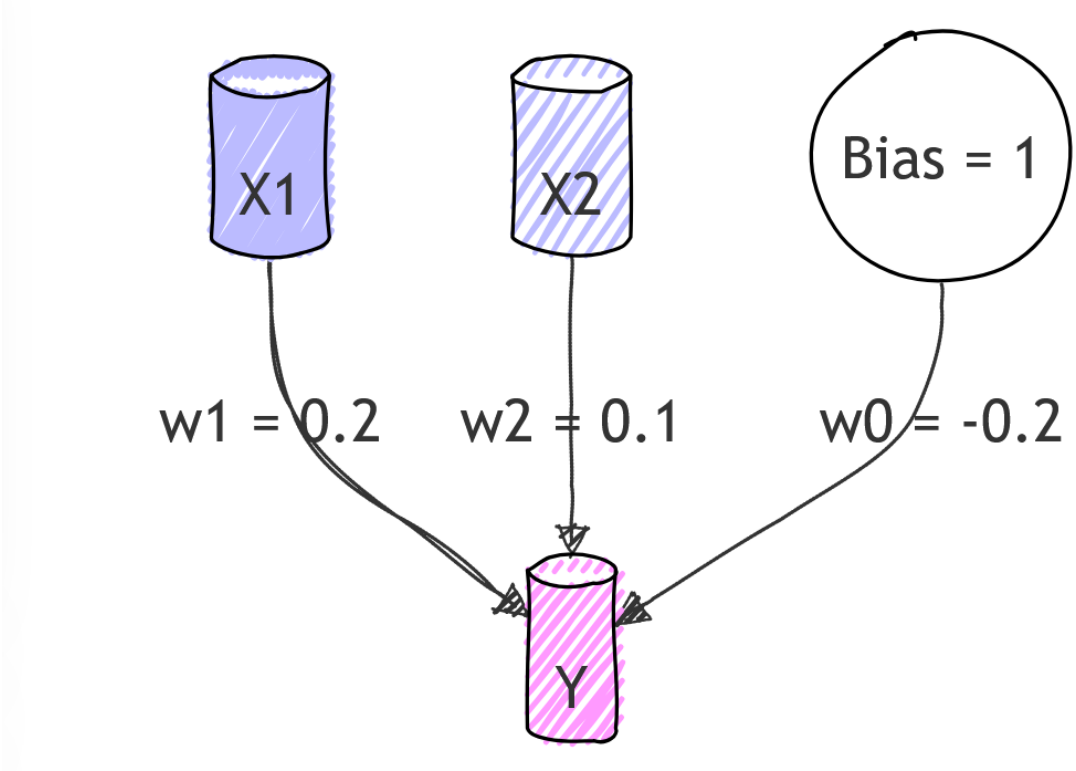
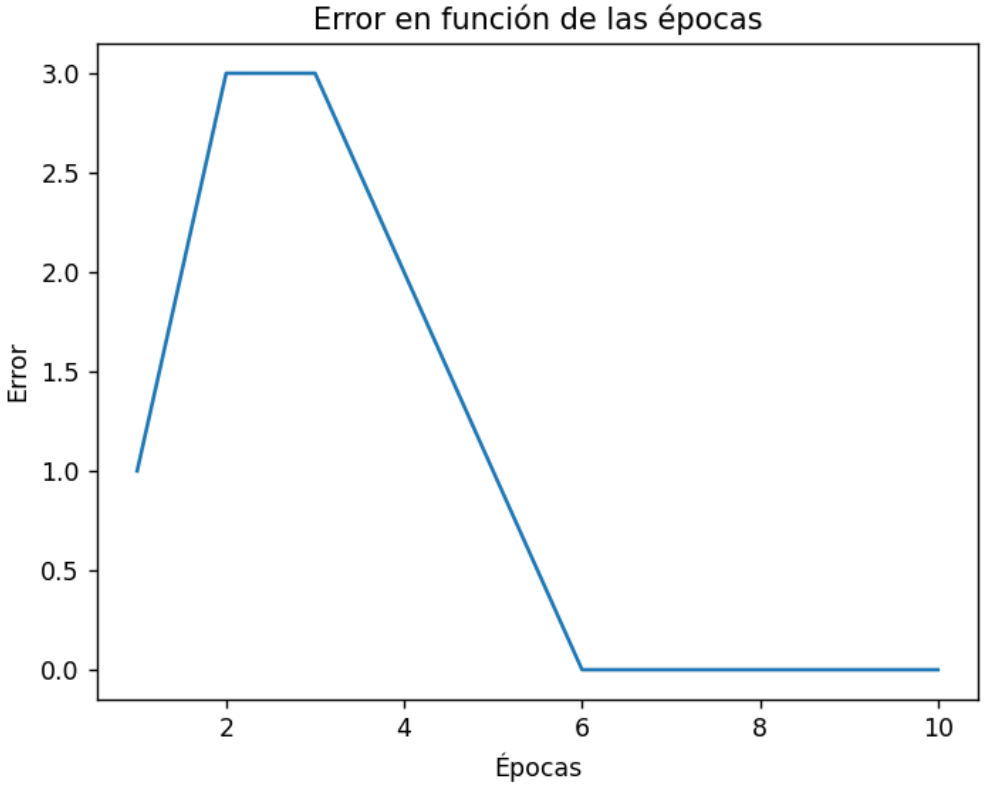


Figura 1: Red Neuronal para la puerta lógica AND

Ahora modificaremos el código para mostrar el error a lo largo de las épocas, obtenemos la siguiente gráfica:



Con esta gráfica se potencia nuestra respuesta de que con 5 épocas es suficiente para que el perceptrón aprenda lo que es una puerta and.

Ahora vamos a realizar lo mismo pero para las distintas puertas lógicas:

- **OR**
- **XOR - Exclusiva**
- **NOT**
- **NAND**
- **NOR**

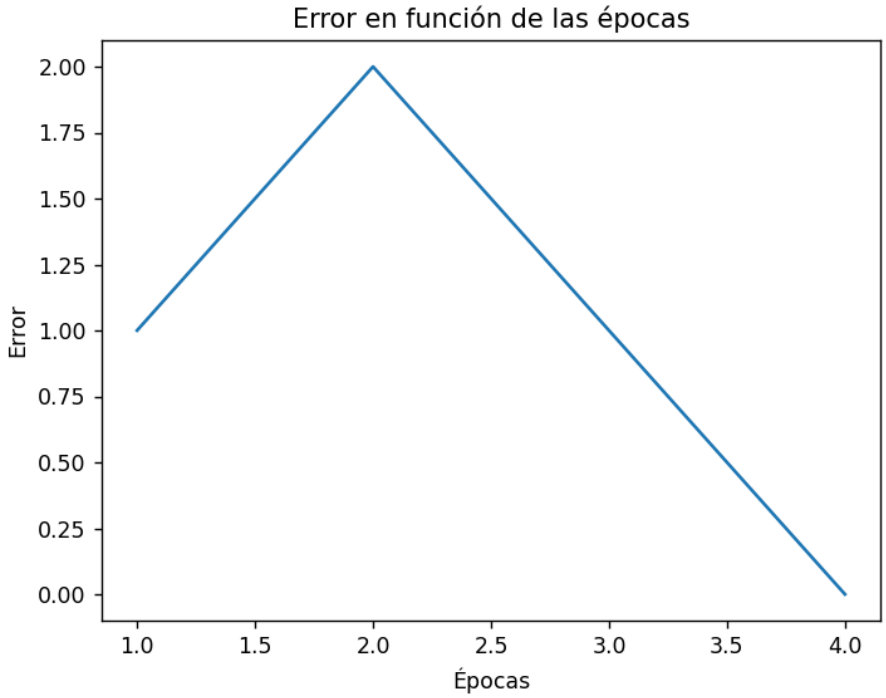
2.1.1. Puerta OR

Solo hemos tenido que modificar las entradas y sus etiquetas, obtenemos este resultado al ejecutar el nuevo código:

```
Entrada: [0 0], Predicción: 0
Entrada: [0 1], Predicción: 1
Entrada: [1 0], Predicción: 1
Entrada: [1 1], Predicción: 1
Pesos finales:  [0.  0.1 0.1]
```

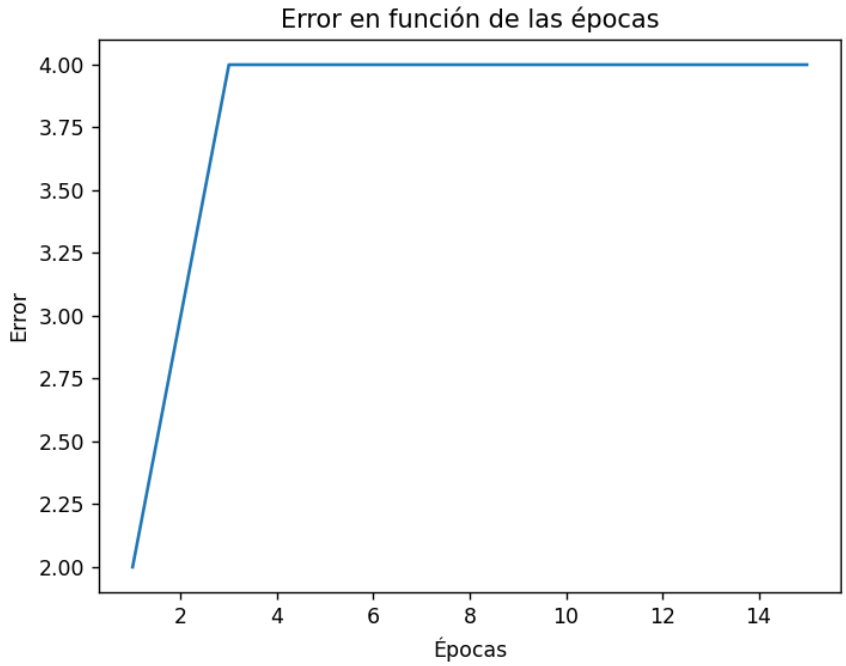
Figura 2: Puerta OR predicciones y pesos finales

En la imagen anterior se puede ver que lo acierta correctamente, hay que destacar que en este caso con 4 épocas ya se obtiene un erro de 0. Esto lo podemos apreciar en la siguiente gráfica:



2.1.2. Puerta XOR

Al igual que antes, tan solo hemos tenido que modificar las entradas y etiquetas, hemos obtenido la siguiente gráfica del error:



Como se aprecia en la gráfica anterior vemos que el perceptrón nunca aprende correctamente a clasificar a una puerta XOR. Además, si nos fijamos en las predicciones que hace el perceptrón obtenemos lo siguiente:

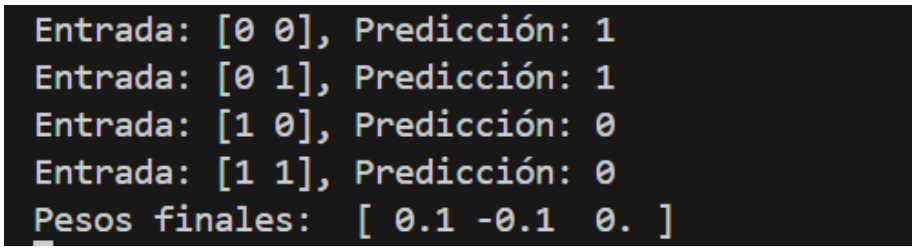


Figura 3: Puerta XOR predicciones y pesos finales

2.1.3. Puerta NOT

Obtenemos los siguientes pesos:

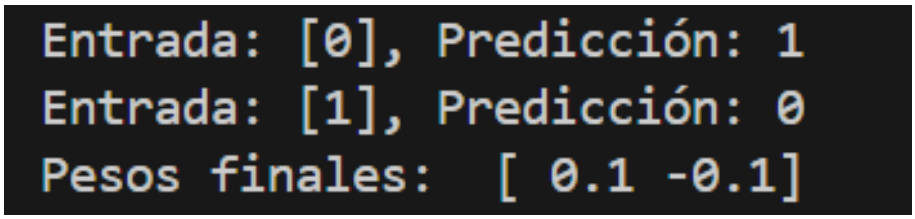
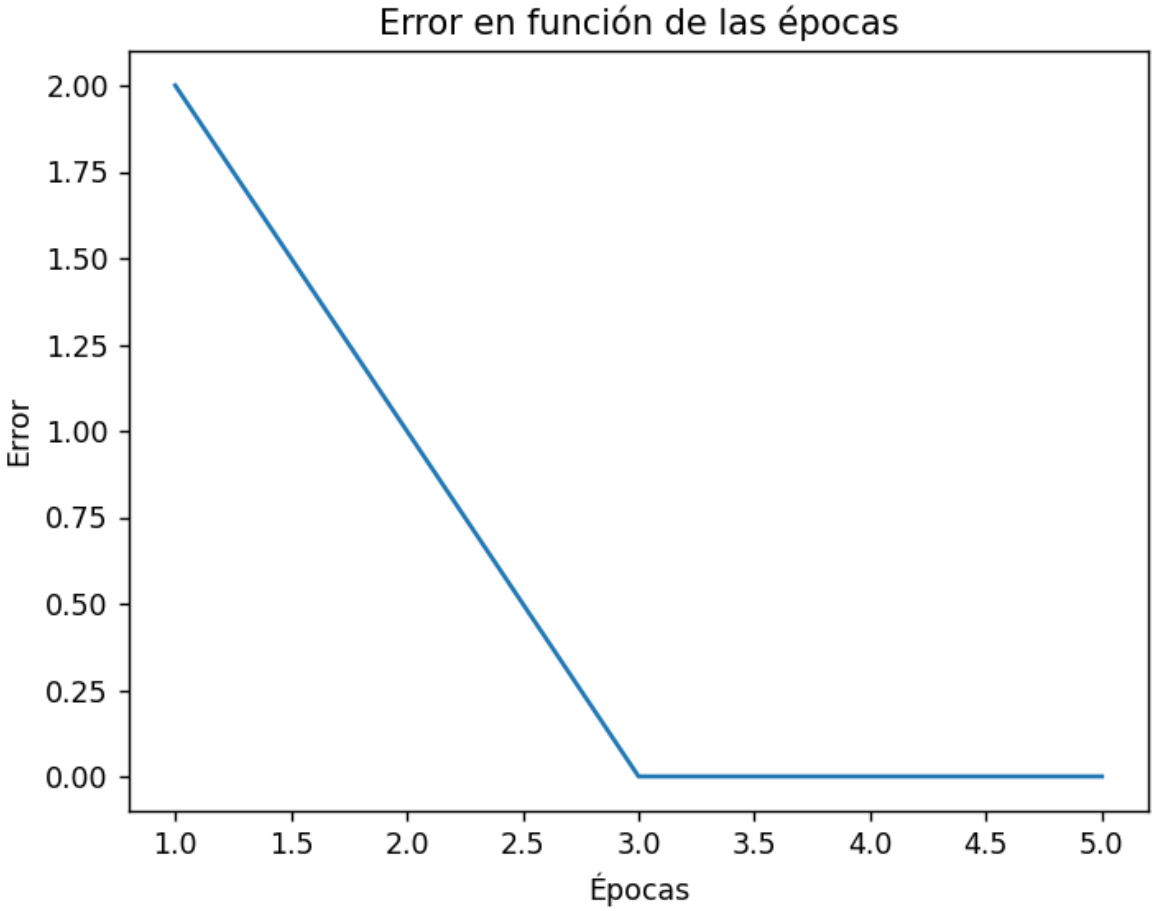


Figura 4: Puerta NOT predicciones y pesos finales

En la imagen anterior se aprecia que el perceptrón predice correctamente para las entradas de prueba. Además, obtenemos la siguiente gráfica con el distinto error a lo largo de las distintas épocas:



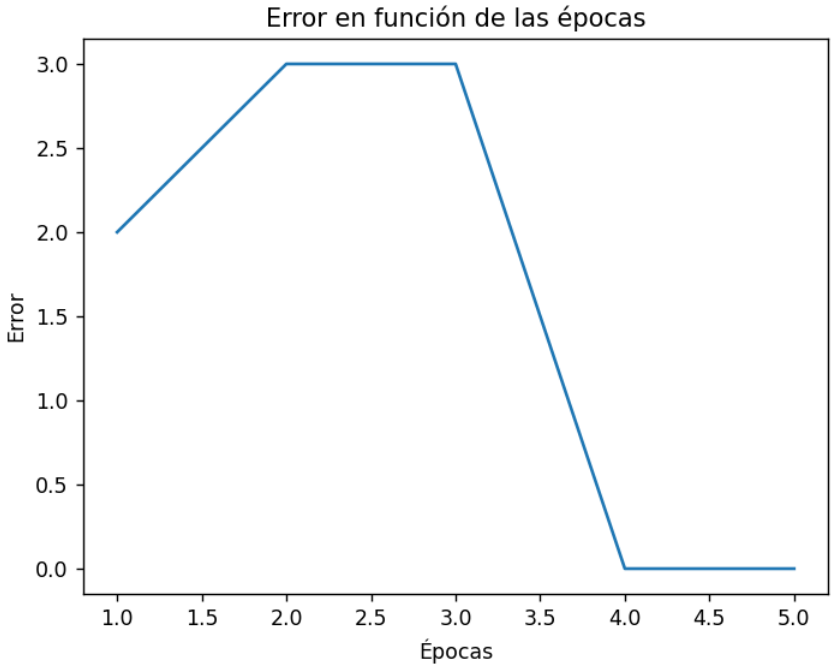
2.1.4. Puerta NAND

Obtenemos los siguientes pesos:

```
Entrada: [0 0], Predicción: 1
Entrada: [0 1], Predicción: 1
Entrada: [1 0], Predicción: 1
Entrada: [1 1], Predicción: 0
Pesos finales:  [ 0.2 -0.2 -0.1]
```

Figura 5: Puerta NAND predicciones y pesos finales

En la imagen anterior se aprecia que el perceptrón predice correctamente para las entradas de prueba. Además, obtenemos la siguiente gráfica con el distinto error a lo largo de las distintas épocas:



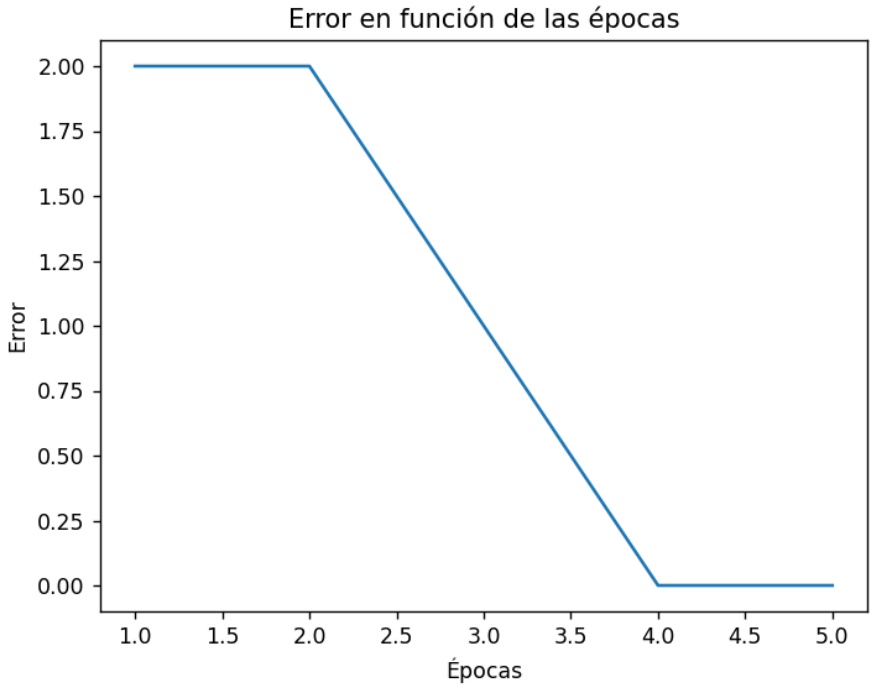
2.1.5. Puerta NOR

Obtenemos los siguientes pesos:

```
Entrada: [0 0], Predicción: 1
Entrada: [0 1], Predicción: 0
Entrada: [1 0], Predicción: 0
Entrada: [1 1], Predicción: 0
Pesos finales:  [ 0.1 -0.1 -0.1]
```

Figura 6: Puerta NOR predicciones y pesos finales

En la imagen anterior se aprecia que el perceptrón predice correctamente para las entradas de prueba. Además, obtenemos la siguiente gráfica con el distinto error a lo largo de las distintas épocas:



2.1.6. Conclusiones

El perceptrón ha sido capaz de aprender para todas las puertas lógicas excepto una, la **puerta XOR**, yo creo que se debe a que esta es la unión de dos puertas AND, esto se traduce en que no es una función linealmente separable. Por ello la solución sería añadir una capa oculta, es decir, crear un perceptrón multicapa (MLP).

2.2. Actividad 2

En esta segunda actividad vamos a experimentar con el dataset Iris. Con el código proporcionado obtenemos una precisión del **88 %**, esta precisión me parece aceptable para lo sencillo que es un perceptrón. En la siguiente imagen obtenemos los pesos de este perceptrón:

```
Pesos finales: [[-0.14641831  0.2444286  -0.29530878 -0.26000665]
 [ 0.05856732 -0.01941723  0.25339183 -0.15177791]
 [ 0.04995448  0.13135182  0.5926189   0.45932789]]
```

Figura 7: Pesos del perceptrón de SKLearn

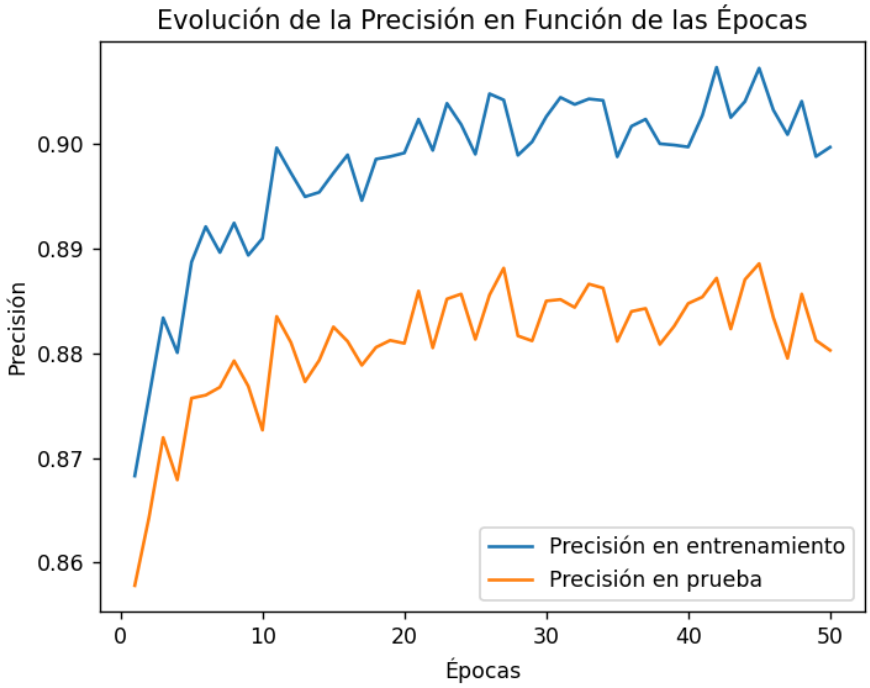
Ahora vamos a ingresar valores nuestros y ver que nos indica el perceptrón, hemos hecho una prueba y obtenemos lo siguiente:

```
Introduce la longitud del sépalo (cm): 12
Introduce el ancho del sépalo (cm): 5
Introduce la longitud del pétalo (cm): 7
Introduce el ancho del pétalo (cm): 9
El tipo de flor predicho es: virginica
```

Figura 8: Predicción del perceptrón para valores inventados para el dataset Iris

2.3. Actividad 3

Finalmente vamos a ver la capacidad del perceptrón para aprender dígitos manuscritos, para ellos usaremos el mnist_784. Con el código original obtenemos una **precisión del 87 %** , el perceptrón usa **10 pesos**, uno para cada número. Además, vamos a mostrar el aprendizaje a lo largo de las distintas épocas:



3. Conclusiones

A través de estos experimentos con el perceptrón, hemos comprobado tanto su utilidad como sus limitaciones en tareas de clasificación. Observamos que el perceptrón es eficaz para aprender funciones lógicas linealmente separables como AND, OR, NAND, y NOR, pero no es capaz de aprender funciones no linealmente separables como XOR. Esto demuestra la necesidad de redes con capas adicionales para resolver problemas más complejos y no lineales, como los que se encuentran en el mundo real.

Además, al aplicar el perceptrón al dataset *Iris* y al conjunto de datos de dígitos *MNIST*, pudimos ver que, aunque logra un rendimiento razonable, su precisión es limitada en comparación con modelos más complejos. Por ejemplo, en *Iris* conseguimos un 88 % de

precisión, lo cual es aceptable para un modelo tan simple, y en *MNIST* alcanzamos un 87 % con un solo peso para cada dígito.

En general, estos ejercicios ilustran que el perceptrón es una herramienta valiosa para entender los conceptos básicos del aprendizaje supervisado y para aprender a clasificar datos linealmente separables. Sin embargo, cuando enfrentamos problemas más complejos, se vuelve necesario utilizar modelos de redes neuronales multicapa para mejorar la precisión y el rendimiento.