



Perceptrones

- Banfi, Malena
- Fleischer, Lucas
- Perez Rivera, Mateo
- Szejer, Ian



1. Ejercicio 1

Perceptrón simple escalón



2. Ejercicio 2

Perceptron simple y no lineal



3. Ejercicio 3

Perceptron multicapa



Ejercicio 1

Consigna

Implementar el algoritmo de perceptrón simple con función de activación escalón y utilizar el mismo para aprender la función lógica “AND” y “OR exclusivo”.

¿Qué puede decir acerca de los problemas que puede resolver el perceptrón simple escalón en relación a los problemas planteados en la consigna?



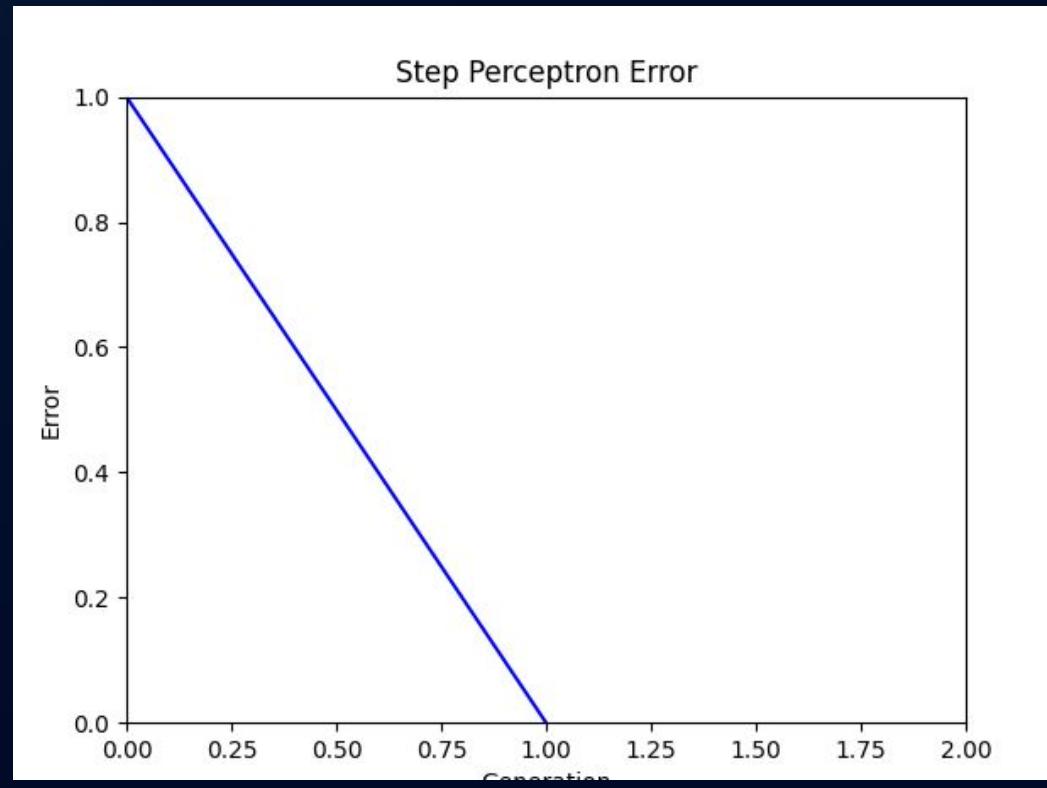
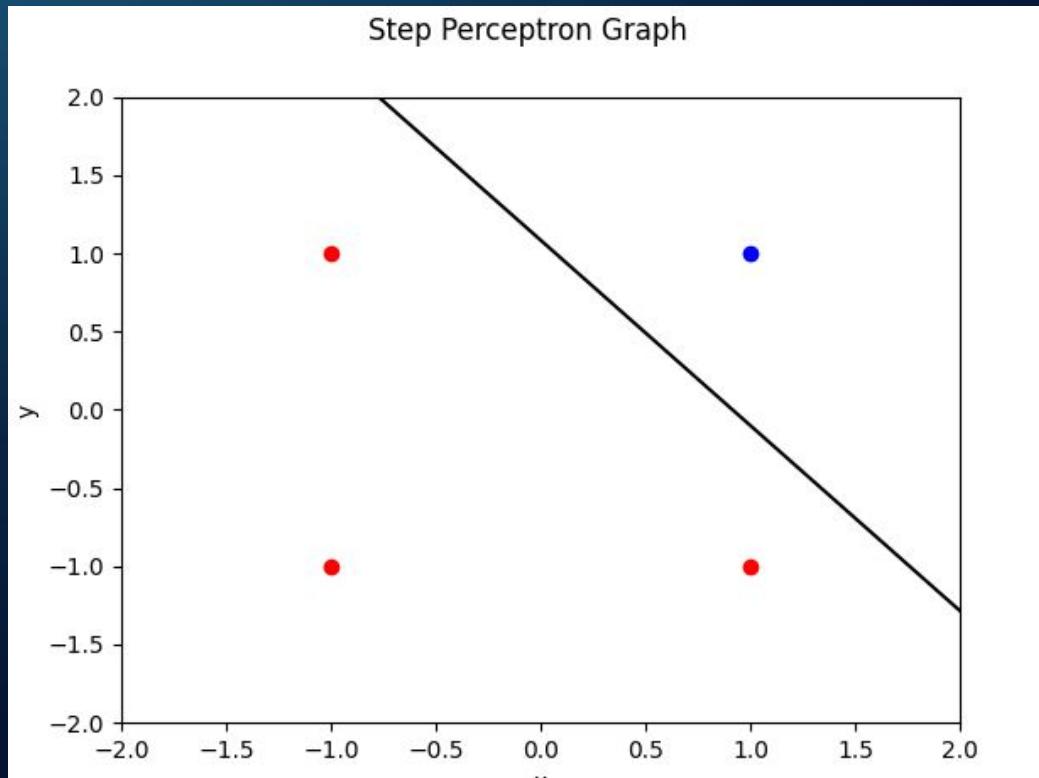
»» Perceptrón simple con función de activación escalón

Los perceptrones simples son un tipo de algoritmo de aprendizaje automático supervisado utilizado para la clasificación binaria de entradas en dos clases distintas.

Funciona mediante la asignación de pesos a las características de entrada de un patrón y la comparación de la suma ponderada de estas características con un umbral. Si la suma ponderada supera el umbral, el perceptrón asigna la clase positiva al patrón. Si la suma ponderada es menor o igual al umbral, se asigna la clase negativa al patrón.



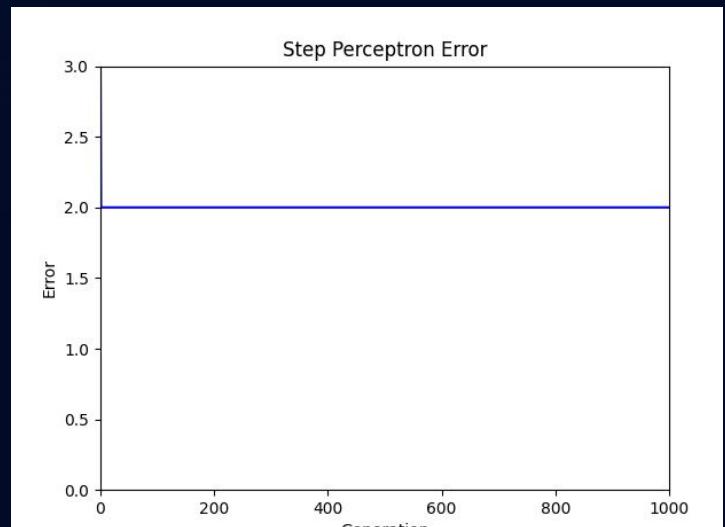
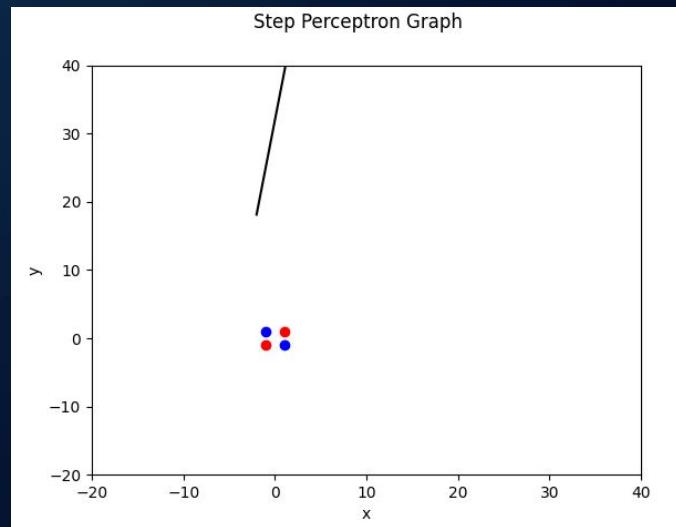
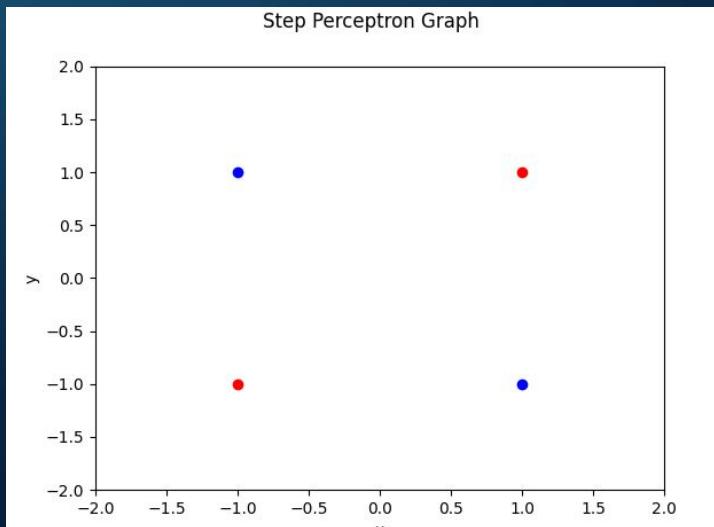
Resultados: AND



```
{  
  "operation": "and",  
  "learning_rate": 0.1,  
  "epochs": 1000  
}
```

Error Min = 0
Min weight = [0.501, 0.547, 0.462]

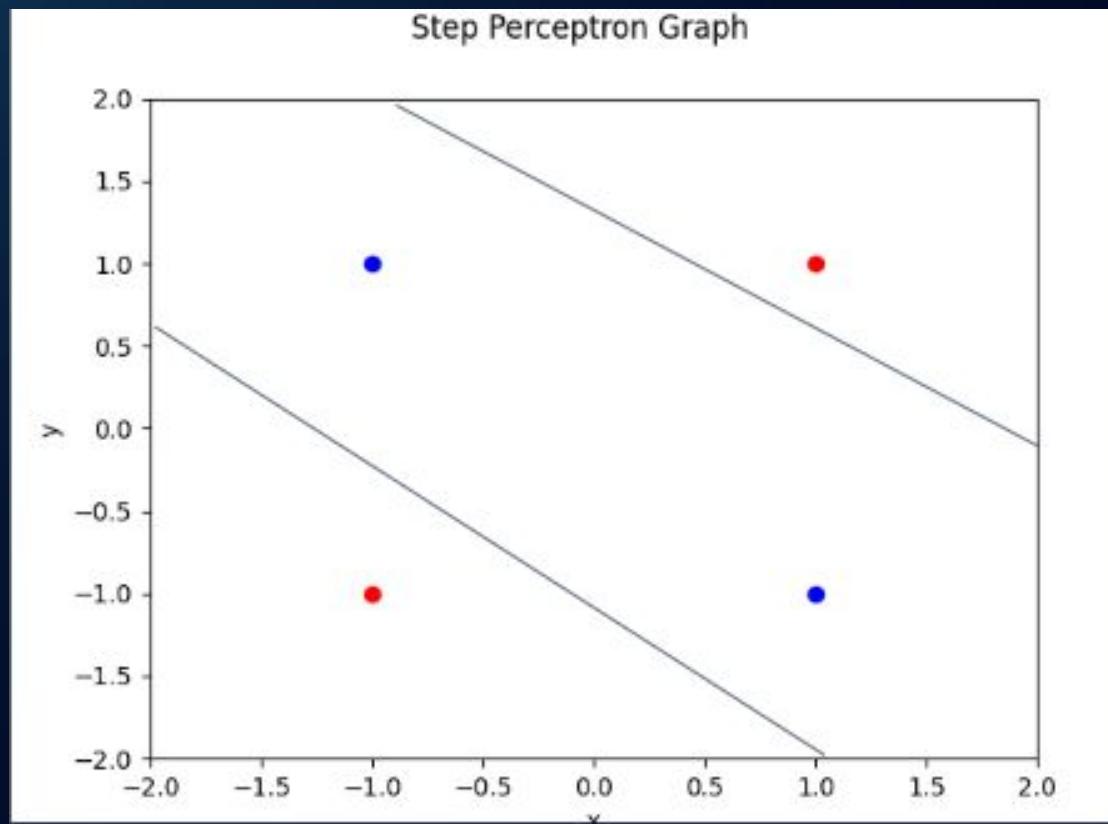
Resultados: XOR



```
{  
  "operation": "xor",  
  "learning_rate": 0.1,  
  "epochs": 1000  
}
```

Obtenemos este resultado debido a que no es un problema linealmente separable.

Resultado esperado



»»» Problemas que puede resolver el perceptrón simple escalón en relación a los problemas planteados en la consigna



El perceptrón simple con función de activación escalón puede resolver problemas de clasificación binaria linealmente separables, es decir, aquellos en los que se pueden separar las dos clases con una línea recta o un hiperplano.

En el problema de la función lógica "AND", los datos se pueden separar linealmente en dos clases, lo que significa que el perceptrón simple escalón es capaz de encontrar un hiperplano que separa las dos clases correctamente.

Para aprender la función lógica "OR exclusivo", también conocida como XOR, el perceptrón simple con función de activación escalón no es suficiente ya que esta función no es linealmente separable. Esto significa que no puede ser representada por un único hiperplano. En este caso, se necesitaría un modelo más complejo, como una red neuronal multicapa con una función de activación no lineal, para resolver el problema.



»» Conclusion



AND

Es un problema linealmente separable, lo que permite una correcta resolución con este perceptrón.

XOR

NO es un problema linealmente separable, ya que requiere de dos rectas que separen las dos categorías (no existe un hiperplano).

Ejercicio 2

Consigna

Implementar el algoritmo de perceptrón simple lineal y perceptrón simple no lineal. Utilizar ambos para aprender a clasificar los datos en el archivo "TP2-ej2-conjunto.csv".



¿Qué es un perceptrón lineal?

Un perceptrón lineal es un modelo de red neuronal artificial que se utiliza para resolver problemas de estimación de resultados con múltiples entradas y una única salida, representables por un hiperplano. El modelo consta de una única neurona de salida que recibe varias entradas y realiza una suma ponderada de esas entradas. Luego, se aplica una función de activación en la suma ponderada para producir la salida del modelo.



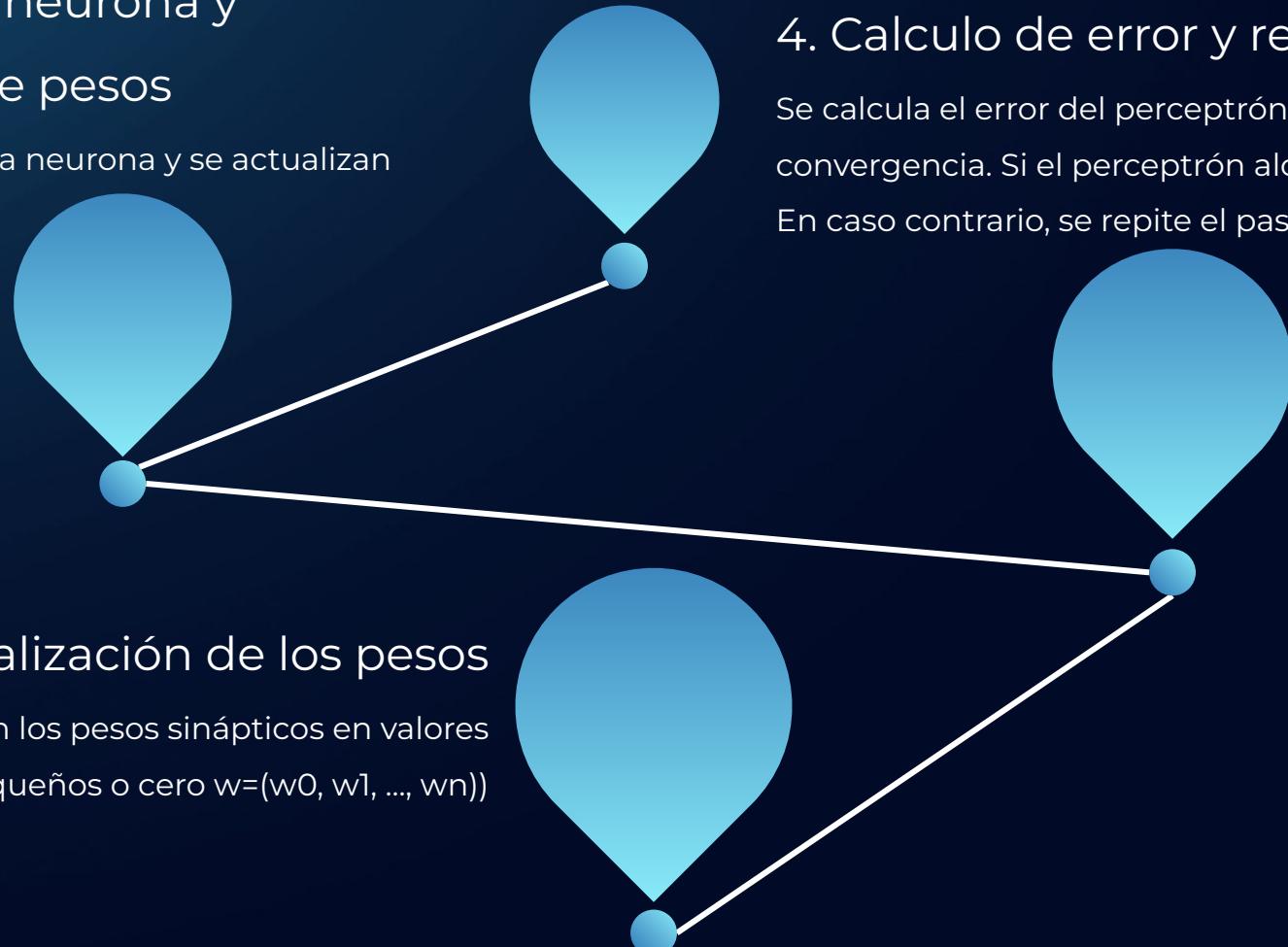
¿Qué es un perceptrón no lineal?

Un perceptrón no lineal es una extensión del perceptrón lineal que utiliza una función de activación no lineal en lugar de una función de activación lineal, por lo que puede resolver problemas tanto lineales como no lineales. A diferencia del perceptrón lineal, el perceptrón no lineal puede resolver problemas de donde los resultados no toman forma lineal, es decir, no son ubicables en un hiperplano.

»»» Perceptrón Simple Lineal: Algoritmo

3. Cálculo de la neurona y actualización de pesos

Se calcula la salida de la neurona y se actualizan los pesos sinápticos



1. Inicialización de los pesos

Se inicializan los pesos sinápticos en valores aleatorios pequeños o cero $w=(w_0, w_1, \dots, w_n)$

2. Definición de la tasa de aprendizaje y épocas

Suele ser un valor similar a 0.1.

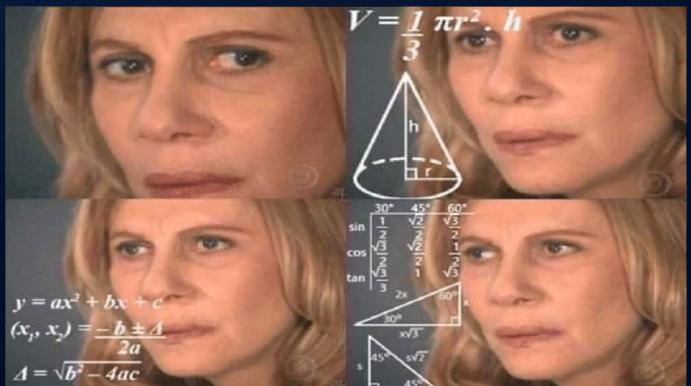
4. Calculo de error y repetición de pasos

Se calcula el error del perceptrón para verificar si se alcanzó convergencia. Si el perceptrón alcanzó convergencia, finalizar. En caso contrario, se repite el paso anterior junto con este

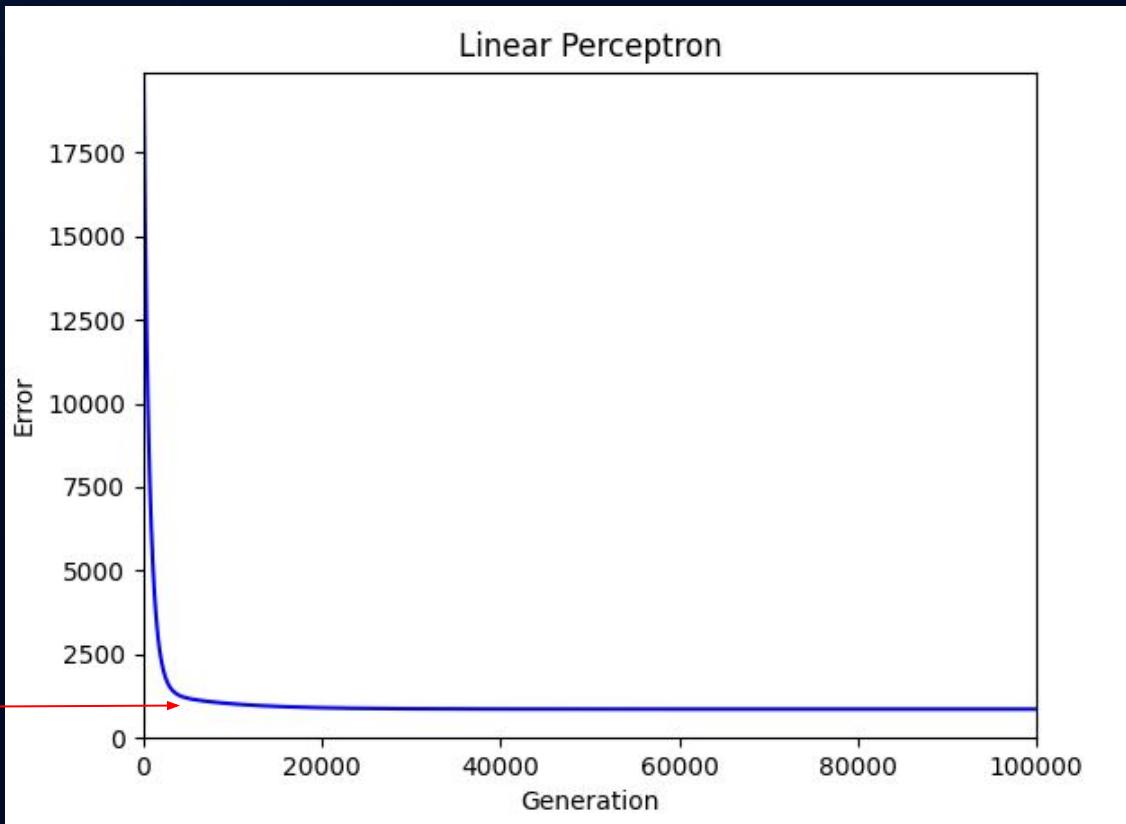
>>> Capacidad del perceptrón lineal para aprender la función de entrada

Como se puede ver en el gráfico, el error no bajaba de 858.95..., aun modificando las épocas (de 10.000 a 500.000, el learning rate, de 0.01 a 0.00001, variando el test size pero nada lo solucionaba.

```
TP3 > Exercise2 > {} config.json > ...
{
  "perceptron": "LINEAR",
  "learning_rate": 0.00001,
  "epochs": 100000,
  "beta": 0.8,
  "min_error": 0.02,
  "test_size": 22
}
```



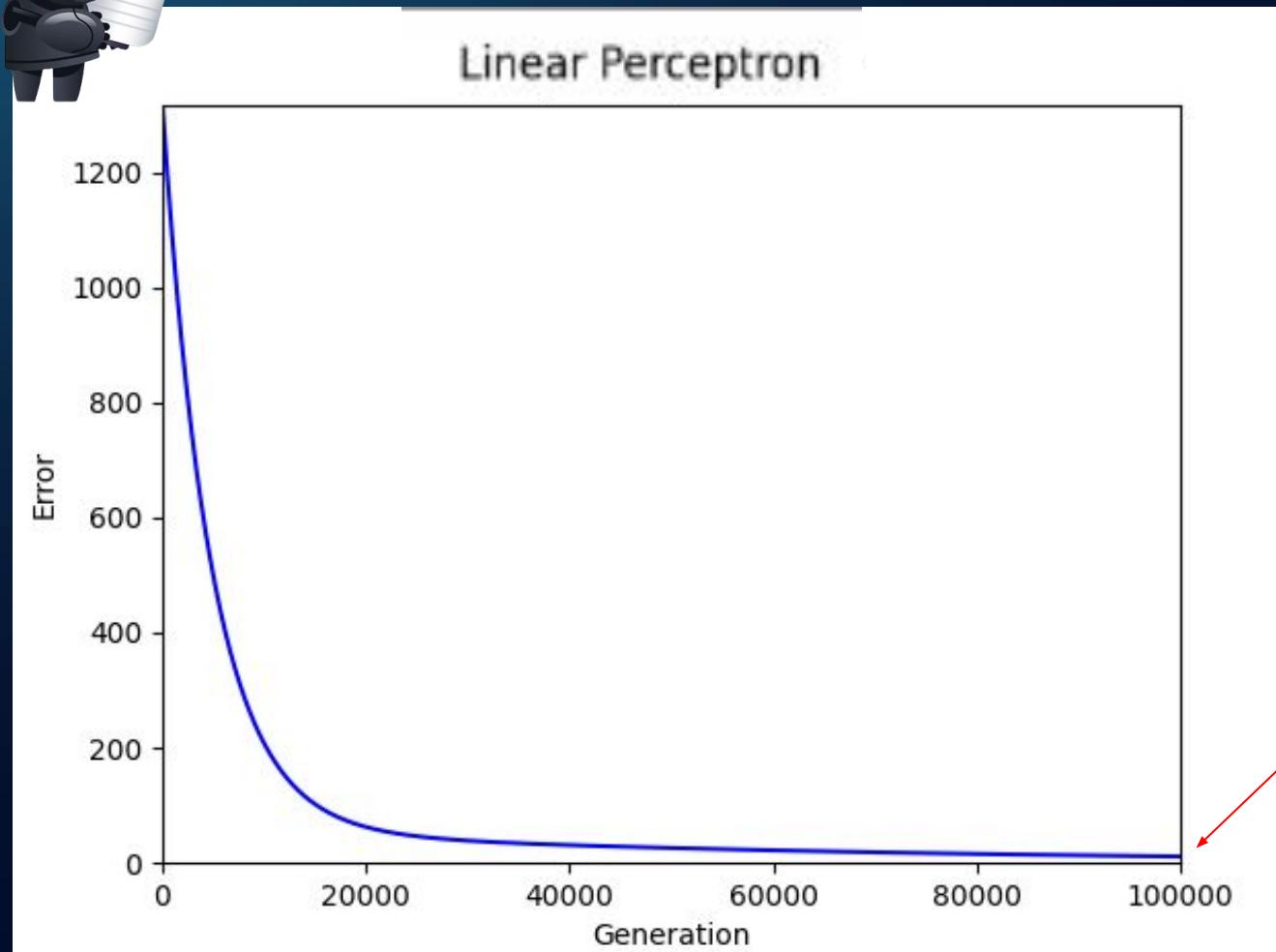
858.95..



The evaluate results are: [45.37705703449294, -14.945713431918023, 32.19926684313122, 34.133427339405046, 27.515081203852233, 13.750123066781816]
The expected results are: [40.131, 0.995, 24.974, 21.417, 18.243, 6.914]



Al preguntar por slack, se nos dio la idea de consultar un conjuntito de datos que se correspondan con la salida de la recta $y=2x$. Gracias a esto obtuvimos las conclusiones:



```
x1,y
1.000,20.000
2.000,40.000
1.500,30.000
0.000,0.000
4.000,80.000
8.000,160.000
-2.000,-40.000
```

```
{
  "perceptron": "LINEAR",
  "learning_rate": 0.0001,
  "epochs": 100000,
  "beta": 0.8,
  "min_error": 0.02,
  "test_size": 4
}
```

The evaluate results are: [79.67716445725746, 159.17580426263694, -39.570795250811756]
The expected results are: [80.0, 160.0, -40.0]



Conclusiones del perceptrón lineal

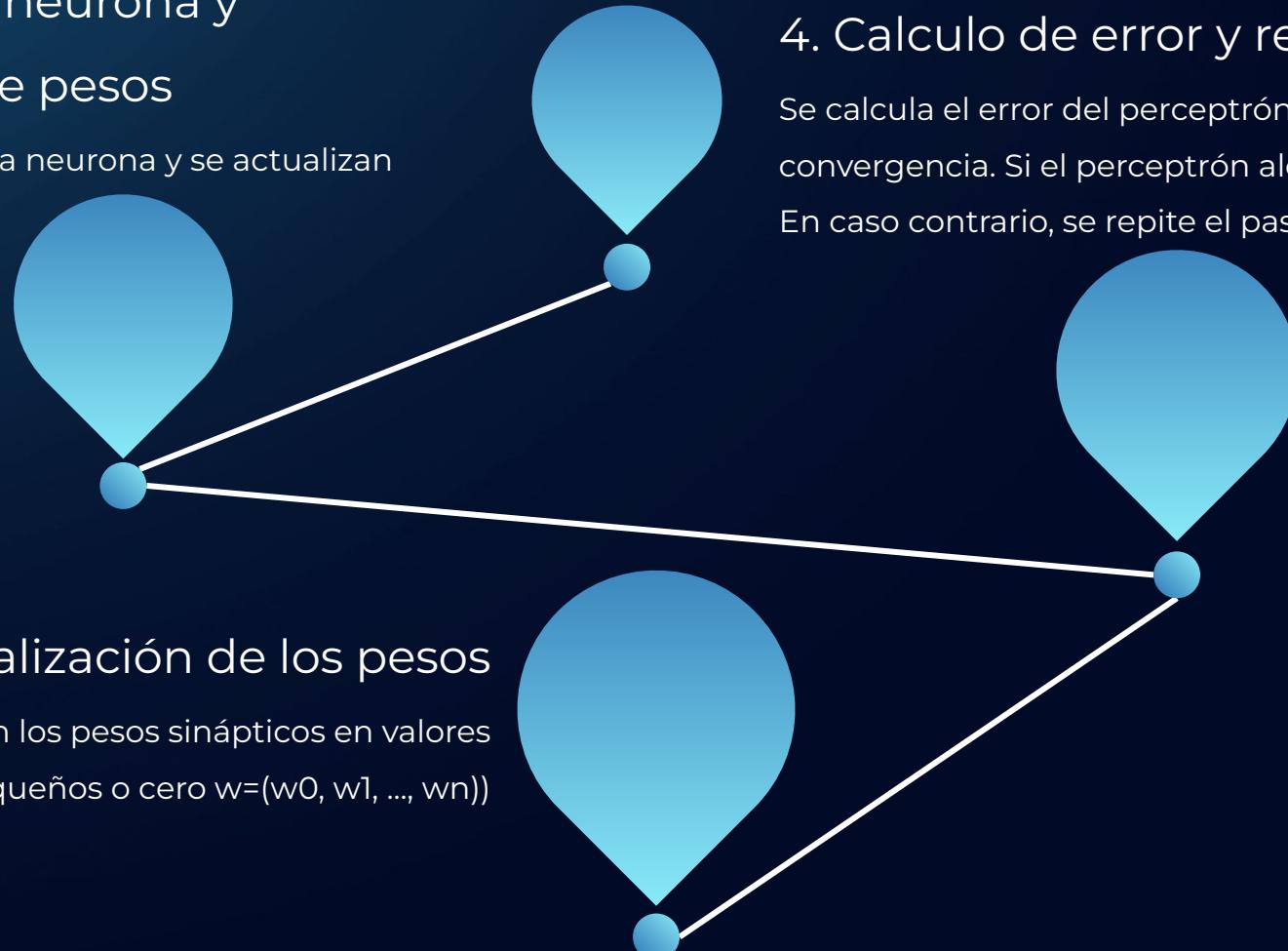
El perceptrón lineal es un modelo simple y limitado que solo puede resolver problemas de clasificación linealmente separables, por eso no puede resolver los datos de entrada del csv provisto.

Al probarlo con una función lineal, el mismo puede aprenderla de manera eficiente con poco error.

»» Perceptrón Simple No Lineal: Algoritmo

3. Cálculo de la neurona y actualización de pesos

Se calcula la salida de la neurona y se actualizan los pesos sinápticos



1. Inicialización de los pesos

Se inicializan los pesos sinápticos en valores aleatorios pequeños o cero $w=(w_0, w_1, \dots, w_n)$

2. Definición de la tasa de aprendizaje y épocas

Suele ser un valor similar a 0.1.

4. Calculo de error y repetición de pasos

Se calcula el error del perceptrón para verificar si se alcanzó convergencia. Si el perceptrón alcanzó convergencia, finalizar. En caso contrario, se repite el paso anterior junto con este

»»» Problema de la estandarización

Tenemos salidas esperadas en el mundo de los reales, sin embargo algunas de nuestras funciones de activación trabajan en un rango acotado y es importante el valor de la imagen de las funciones.



¿Cómo solucionamos ese problema?

Usamos una función llamada escalate la cual transforma los valores del output en valores pertenecientes al -1 a 1 para de esta forma poder tener una buena comparación con la salida de las funciones como TanH.

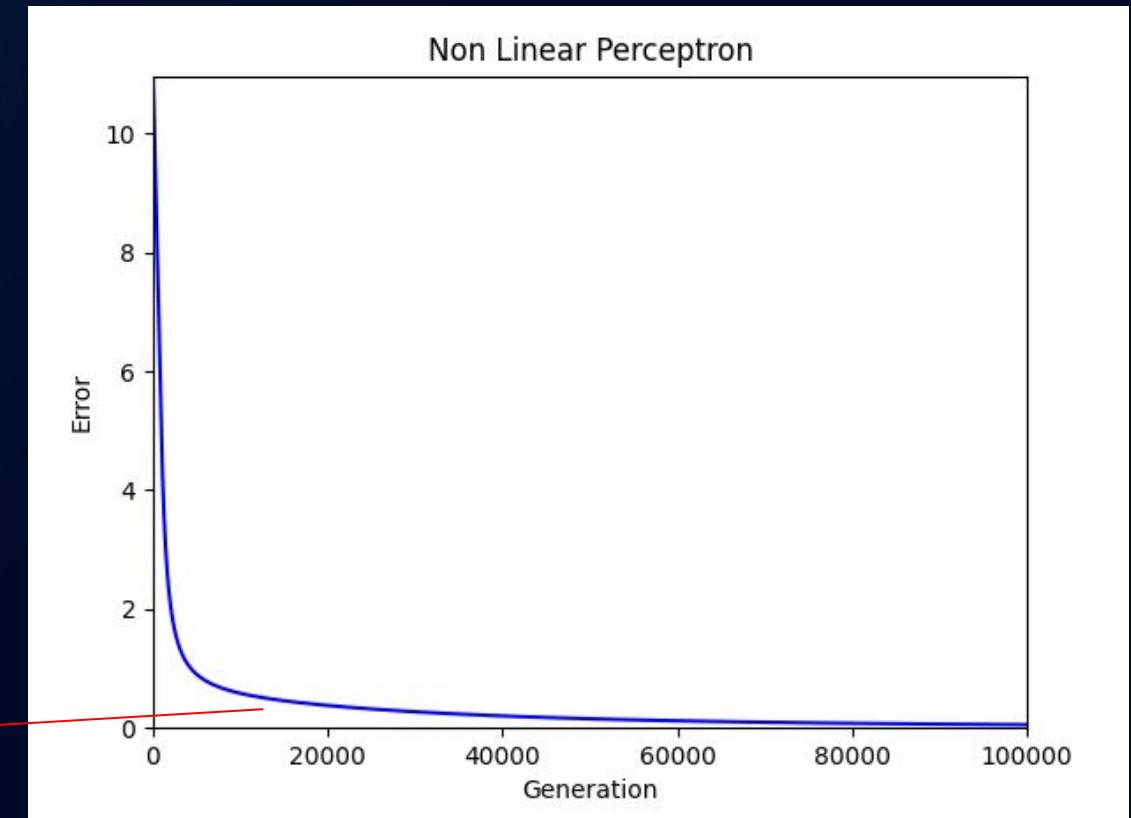


>>> Capacidad del perceptrón no lineal para aprender la función de entrada

```
{  
    "perceptron": "NO_LINEAR",  
    "learning_rate": 0.00001,  
    "epochs": 100000,  
    "beta": 0.8,  
    "min_error": 0.02,  
    "test_size": 22  
}
```

0,0527

0,0527



No linear Accuracy: 0.5

The evaluate results are: [0.06307331994737461, -0.9707811490620095, -0.2613925460624294, -0.49205088218729387, -0.4534380193742128, -0.8230285844874488]

The expected results are: [-0.09380406 -0.98463535 -0.43881453 -0.51978057 -0.59202859 -0.8499044]

»»» Perceptrón No Lineal: Capacidad de generalización



Capacidad de generalización

Es la capacidad de obtener buenos resultados con datos que no estén ni sean parte del conjunto que se va a usar para entrenar.



Factor de aprendizaje

Determina cuanto se modifican los pesos en cada paso.



División de datos de entrada

Se dividió los datos de entrada en 2 subconjuntos:

- Entrenamiento y prueba



Selección de resultados

Si el dato que dio como resultado el perceptrón, está dentro del rango definido, se toma como un logro, en caso contrario, no

»»» Medición del accuracy



Evaluación de la capacidad de generalización del perceptrón simple no lineal utilizando, de los datos provistos, un subconjunto para entrenar y otro para testear.



Resultados: Capacidad de Generalización

Accuracy

Accuracy = 0.666

```
{  
  "perceptron": "NO_LINEAR",  
  "learning_rate": 0.00001,  
  "epochs": 100000,  
  "beta": 0.8,  
  "min_error": 0.02,  
  "test_size": 25  
}
```

Accuracy

Accuracy = 0.045

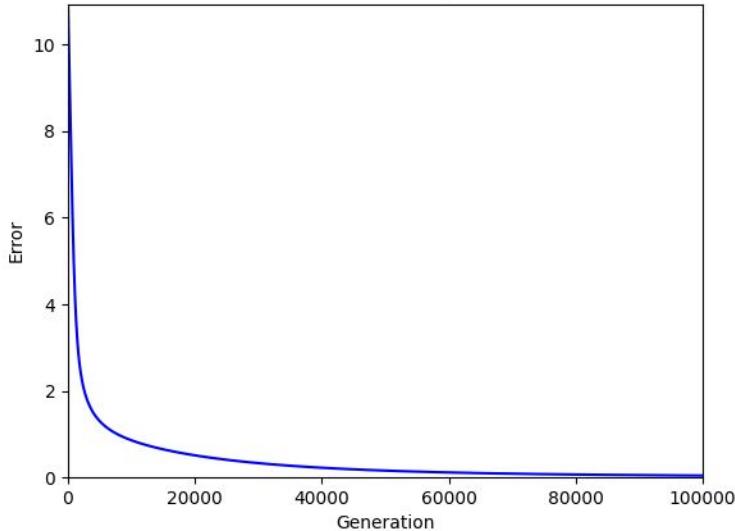
```
{  
  "perceptron": "NO_LINEAR",  
  "learning_rate": 0.00001,  
  "epochs": 100000,  
  "beta": 0.8,  
  "min_error": 0.02,  
  "test_size": 6  
}
```

Accuracy

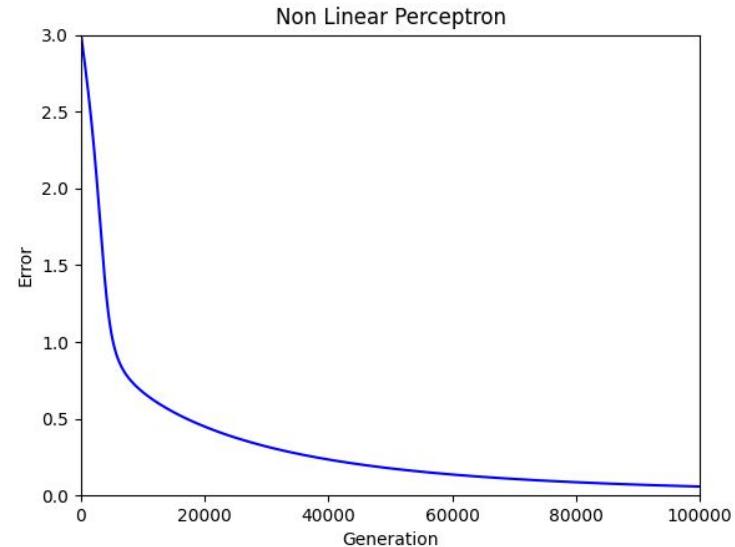
Accuracy = 0.231

```
{  
  "perceptron": "NO_LINEAR",  
  "learning_rate": 0.00001,  
  "epochs": 100000,  
  "beta": 0.8,  
  "min_error": 0.02,  
  "test_size": 15  
}
```

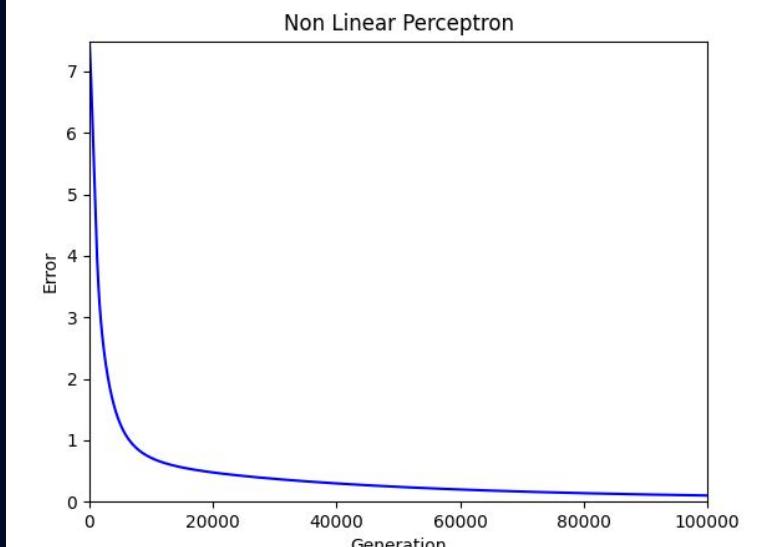
Non Linear Perceptron



Non Linear Perceptron



Non Linear Perceptron

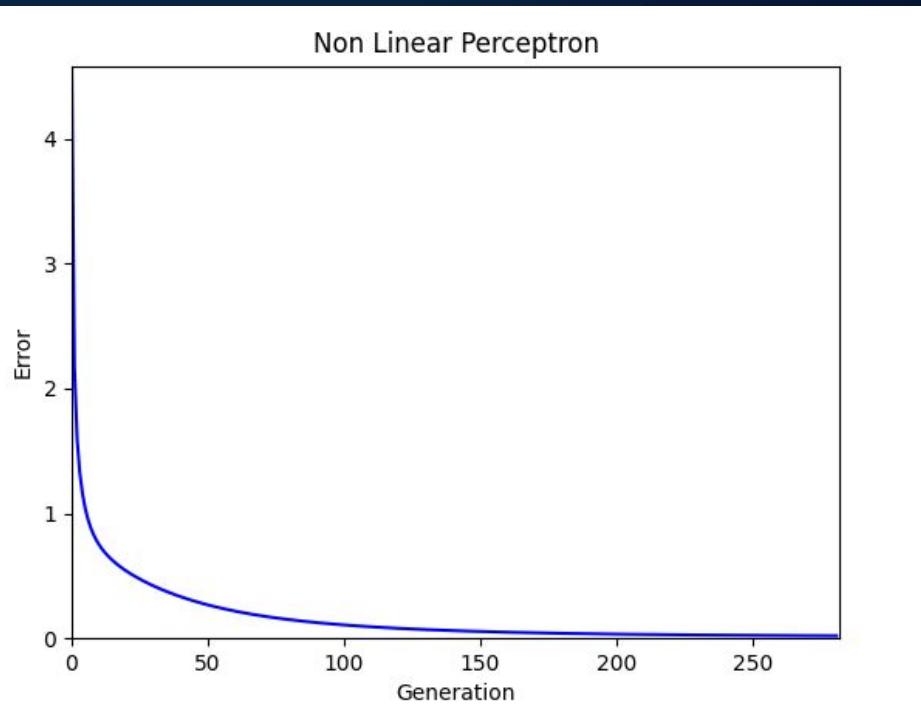


»» Conclusion: Capacidad de Generalización

Accuracy

Accuracy = 0.615

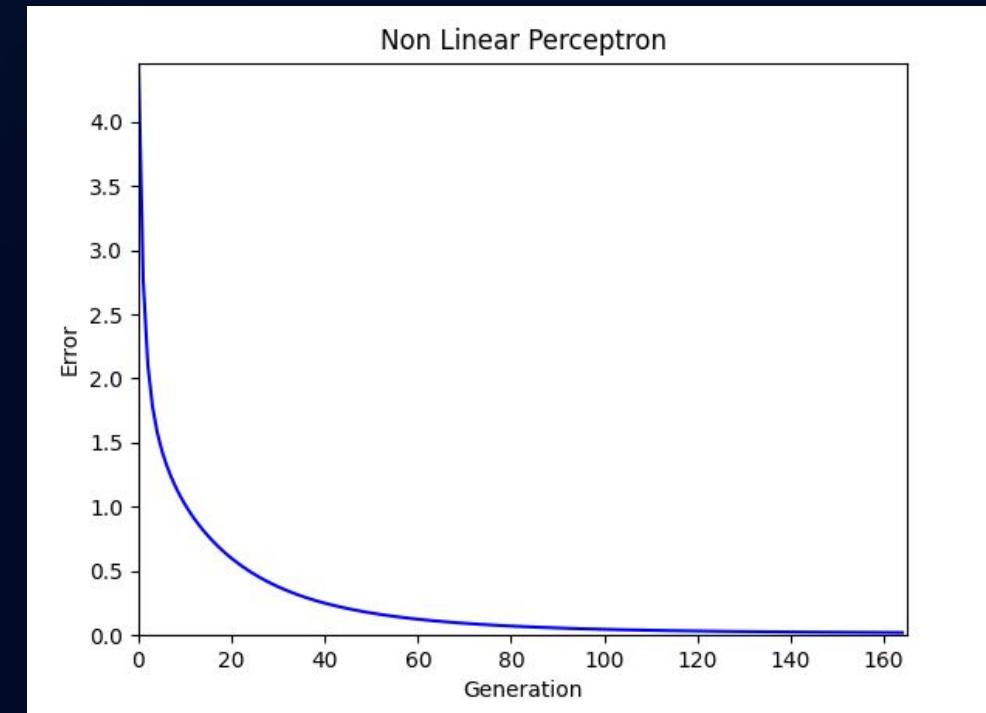
```
{  
  "perceptron": "NO_LINEAR",  
  "learning_rate": 0.01,  
  "epochs": 50000,  
  "beta": 0.8,  
  "min_error": 0.02,  
  "test_size": 15  
}
```



Accuracy

Linear Accuracy = 1.0

```
{  
  "perceptron": "NO_LINEAR",  
  "learning_rate": 0.01,  
  "epochs": 50000,  
  "beta": 0.8,  
  "min_error": 0.02,  
  "test_size": 25  
}
```



»»» ¿Cómo elegirían el mejor conjunto de entrenamiento? y ¿Qué efecto tiene la elección en la capacidad de generalización del perceptrón?

Claramente, el mejor conjunto de entrenamiento, será el que menos error nos de y más accuracy obtengamos.

Para obtener el mismo no hay una fórmula lineal para elegir los datos ya que los mismos al variar se afectan entre sí.



»»» ¿Cómo elegirían el mejor conjunto de entrenamiento? y ¿Qué efecto tiene la elección en la capacidad de generalización del perceptrón?

Lo que sí pudimos ver es que:

- A mayor conjunto de entrenamiento, más aprende y por lo tanto tendrá mayor accuracy al testear los resultados.
- Una buena regla es separar los conjuntos de training y test en un porcentaje de 80% y 20% respectivamente
- Dar conjuntos variados de training
- Si el learning rate es demasiado alto, el algoritmo de entrenamiento puede saltar demasiado en el espacio de parámetros y pasar por alto el mínimo global óptimo. Esto puede llevar a que no converja, se quede oscilando y tenga un rendimiento pobre en los datos de prueba.
- Bajar mucho el learning rate puede llevar a que el algoritmo de entrenamiento se quede atascado en un mínimo local y no pueda alcanzar el mínimo global óptimo.
- Si se ejecutan demasiadas épocas, el modelo puede sobreajustarse a los datos de entrenamiento y tener un rendimiento pobre en los datos de prueba.





Conclusiones del perceptrón no lineal

El perceptrón no lineal tiene la capacidad de resolver problemas no lineales, lo que lo hace adecuado para una amplia gama de aplicaciones en las que los datos pueden no estar linealmente separados. Por esto es que el mismo puede aprender la función de entrada sin problema a diferencia del perceptrón lineal.

Ejercicio 3

Consigna

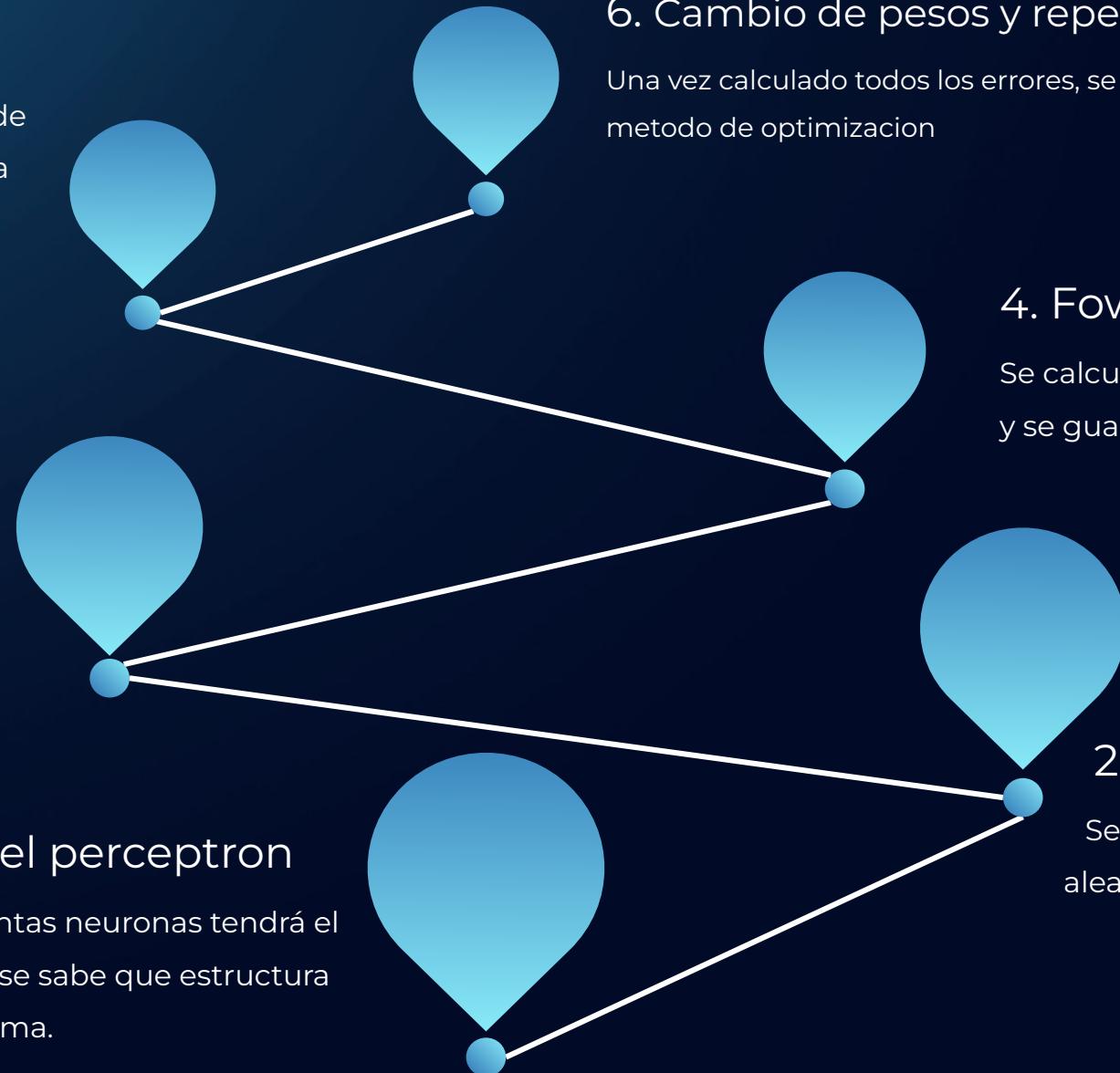
Implementar el algoritmo de perceptrón multicapa y utilizarlo para aprender.



»»» Perceptrón Multicapa: Algoritmo

5. Back propagation

Se calcula el error del perceptrón y de todas las neuronas, arrancando en la capa de salida con los resultados esperados y en todas las capas intermedias usando la derivada del error con la regla de la cadena



3. Definición de la tasa de aprendizaje y épocas

Suele ser un valor similar a 0.001.

1. Decidir la estructura del perceptron

Se debe elegir cuantas capas y cuantas neuronas tendrá el perceptrón, esto es difícil ya que no se sabe que estructura es la correcta para resolver el problema.

4. Foward propagation

Se calculan los resultados en todas las neuronas y se guardan para poder recalcular los pesos

2. Inicialización de los pesos

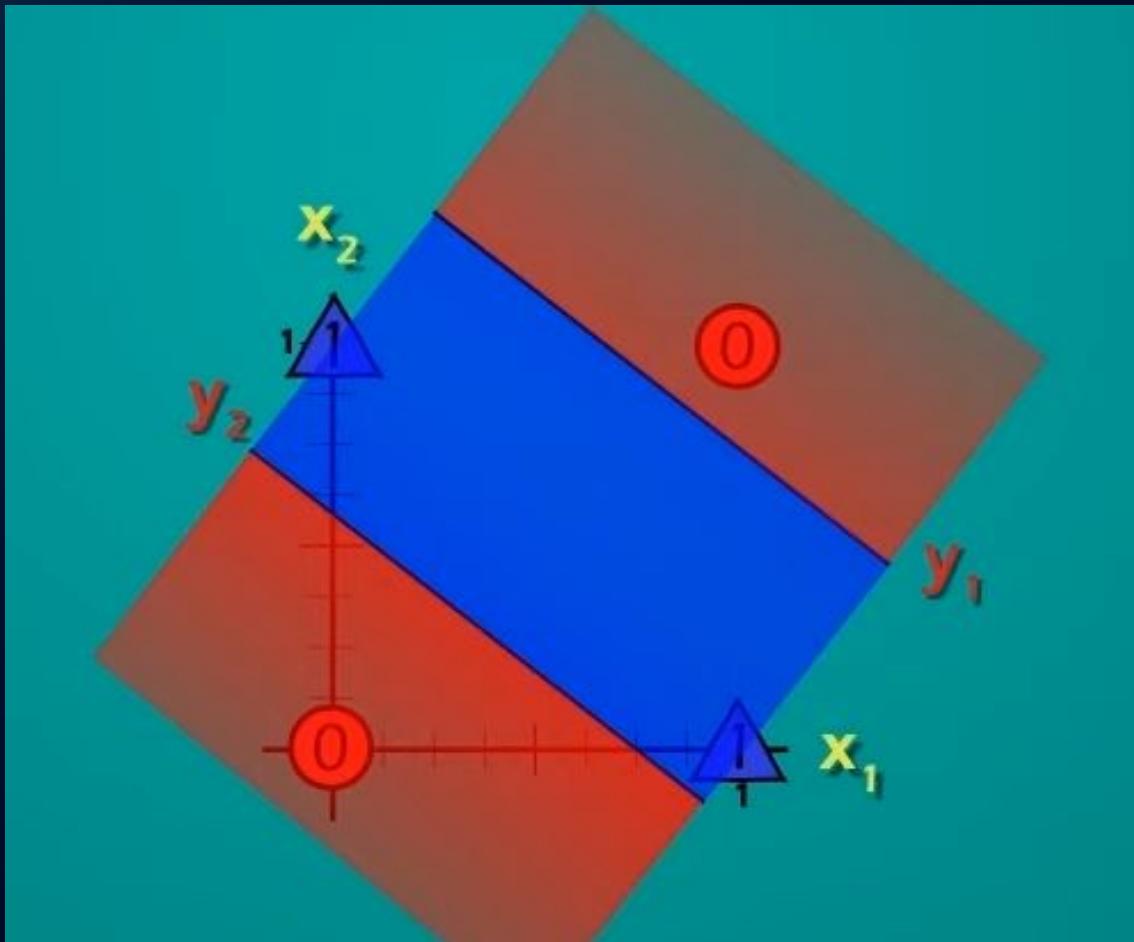
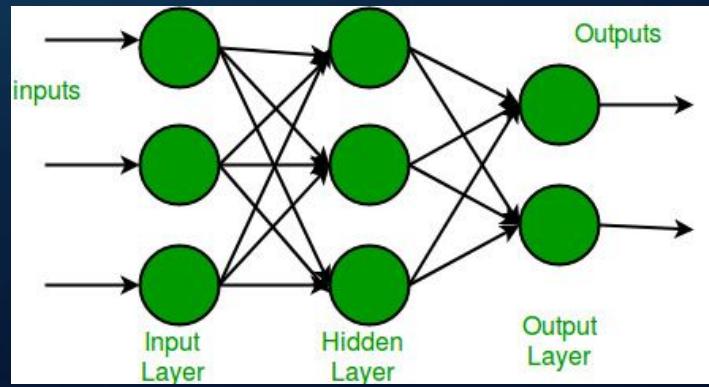
Se inicializan los pesos sinápticos en valores aleatorios pequeños o cero $w=(w_0, w_1, \dots, w_n)$

6. Cambio de pesos y repetición de pasos

Una vez calculado todos los errores, se cambien los pesos con algún metodo de optimizacion

	PERCEPTRÓN SIMPLE	PERCEPTRÓN MULTICAPA
Proceso para obtener la salida de la neurona (“predecir”)	$O = \theta\left(\sum_{i=0}^n x_i \cdot w_i\right)$	$V_j^m = \theta\left(\sum_{k=1}^{m-1} V_k^{m-1} \cdot w_{mk}\right)$ $m = 1 \dots M \quad (O_i = V_i^M, \quad x_i = V_i^0)$
Función de error con respecto a la salida esperada	$E(O) = \frac{1}{2} \sum_{\mu=0}^{p-1} (\zeta^\mu - O^\mu)^2$	$E(O) = \frac{1}{2} \sum_{\mu=0}^{p-1} (\zeta^\mu - O^\mu)^2$
Proceso de “aprendizaje” para ajustar los pesos	$w^{nuevo} = w^{anterior} + \Delta w$ $\Delta w = -\eta \frac{\partial E}{\partial w}$	$\Delta w_{ij} = \eta \delta_i V_j$ $\delta_i = (\zeta_i - O_i) \theta'(h_i)$ $\Delta w_{jk}^m = \eta \delta_j^m V_k^{m-1}$ $\delta_j^m = (\sum_i \delta_i^{m+1} w_{ij}^{m+1}) \theta'(h_j^m)$

»» Imagen visual de un Perceptrón Multicapa

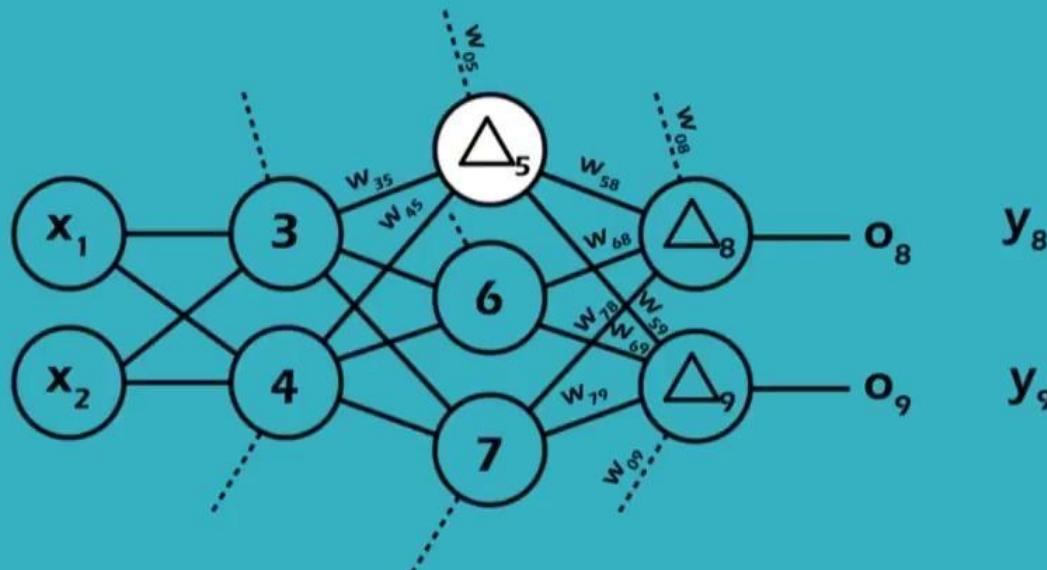


»»» Funcionamiento del Perceptrón Multicapa

$$w_{ij} \leftarrow w_{ij} + \alpha \Delta_j x_i$$

CAPA DE SALIDA $\Delta_j = o_j(1 - o_j)(y_j - o_j)$

CAPAS OCULTAS $\Delta_5 = o_5(1 - o_5) \sum_{k=1}^{n \text{ neurons}} w_{jk} \Delta_k$

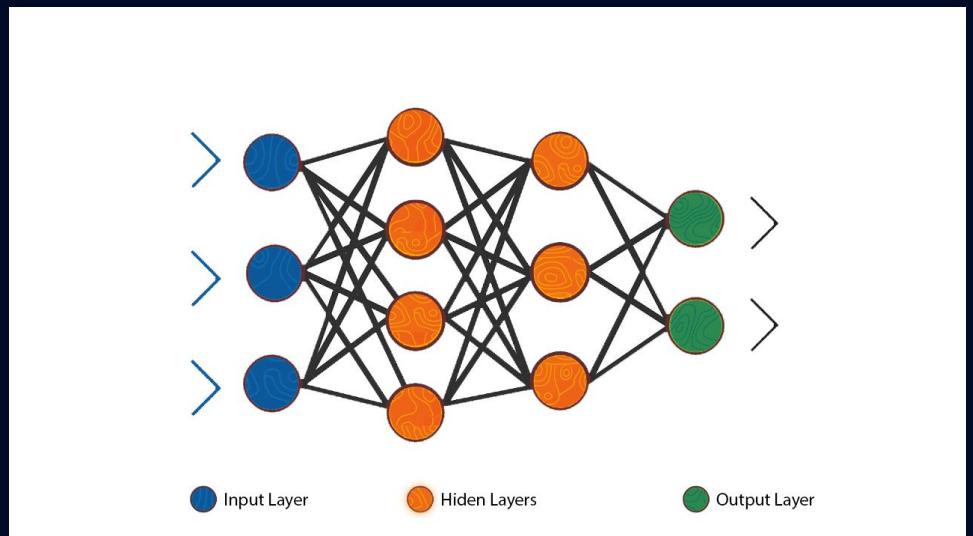


»»» Forward Propagation

Es el algoritmo que me permite calcular la salida de la neurona.

Forward propagation es el proceso en el que un modelo de red neuronal utiliza los parámetros del modelo y las entradas para calcular y producir una predicción de salida. Durante la fase de forward propagation, las entradas se pasan a través de las capas de la red neuronal, una por una, mientras se aplican los pesos y el bias a las entradas en cada capa. El proceso se repite hasta que se alcanza la capa de salida de la red neuronal, que proporciona una predicción para la entrada dada.

En términos más técnicos, forward propagation se realiza mediante la aplicación de una función de activación a los resultados de la multiplicación de la entrada de la red neuronal y los pesos, a lo que se le agrega el bias. El resultado se utiliza como entrada para la siguiente capa, y el proceso se repite para todas las capas hasta que se obtiene la salida final de la red neuronal.

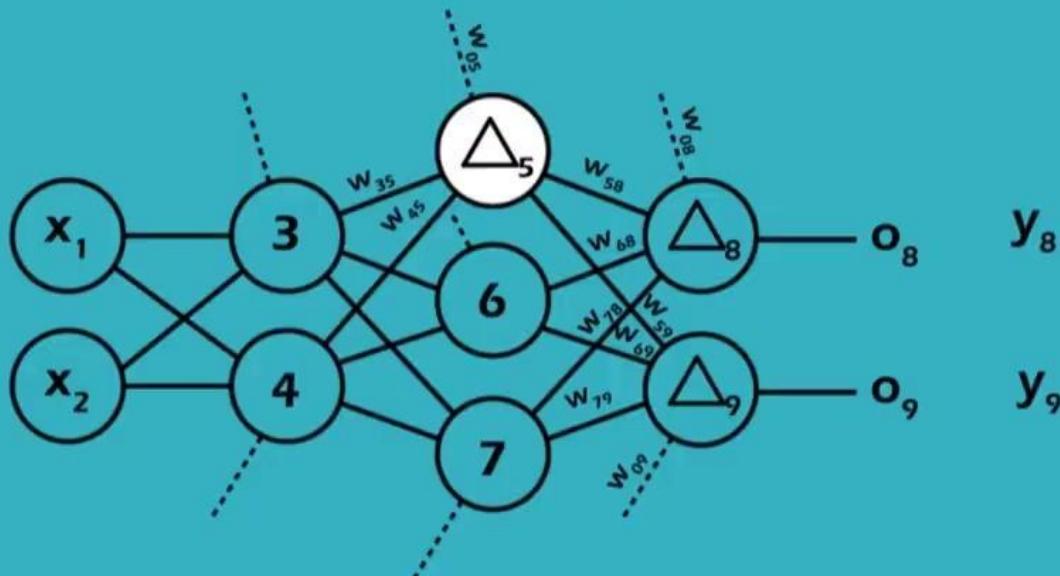


»»» Back Propagation

$$w_{ij} \leftarrow w_{ij} + \alpha \Delta_j x_i$$

CAPA DE SALIDA $\Delta_j = o_j(1 - o_j)(y_j - o_j)$

CAPAS OCULTAS $\Delta_5 = o_5(1 - o_5) \sum_{k=1}^{n \text{ neurons}} w_{jk} \Delta_k$



Ejercicio 3 A

Consigna

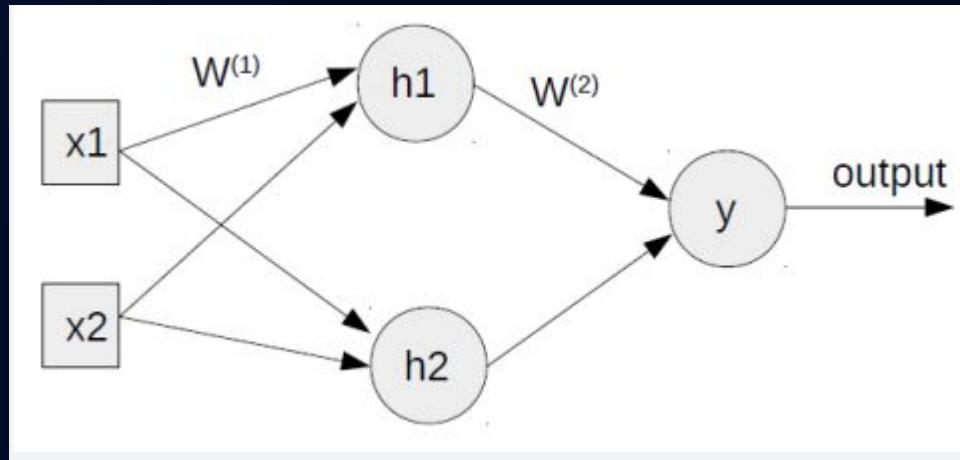
Implementar el algoritmo de perceptrón multicapa y utilizarlo para aprender la Función lógica “O exclusivo” presentada en el Ejercicio 1 (mismos datos y misma salida esperada).



>>> Estructura

- Planteamos 2 nodos iniciales
- Planteamos 1 capa intermedia:
 - TanhLayer que tiene 2 nodo y recibe 2 datos
- Plantemos 1 nodo final Tanh
- Learning rate: 0.01
- Epochs: 20000
- Error_min: 0.02
- Los betas en 0

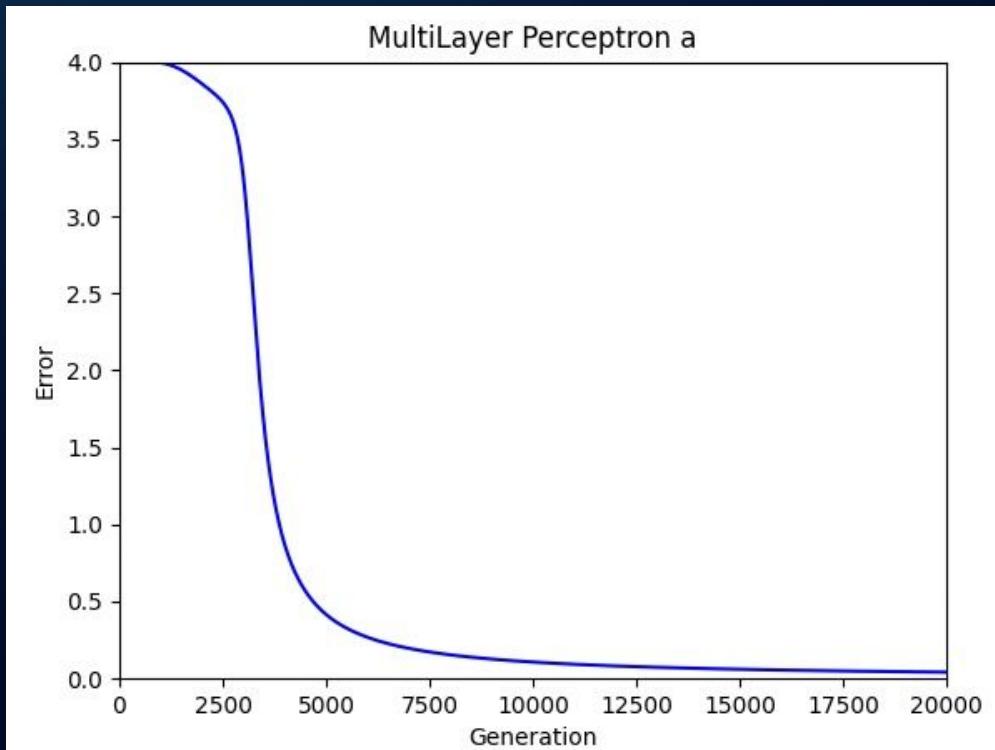
```
input=0output=
0.0069
expected output= [-1]
input=1output=
0.0093
expected output= [-1]
input=2output=
0.0095
expected output= [1]
input=3output=
0.0068
expected output= [1]
error entrenamiento=4.000000295054443
```



Resultados: Curva de errores

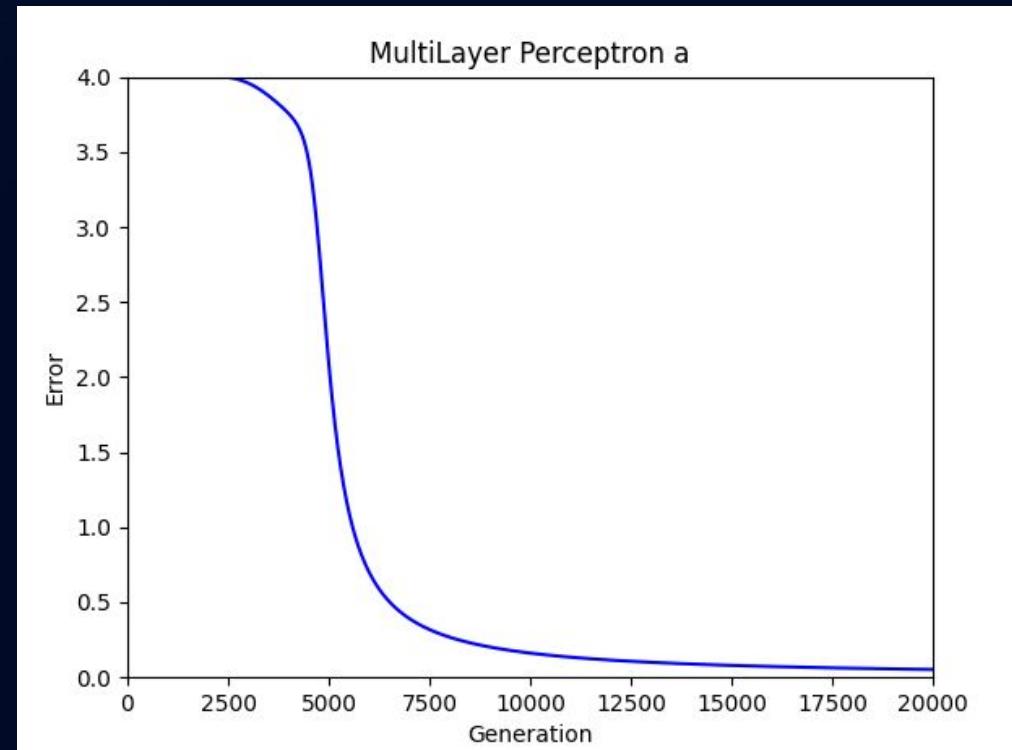
Con MOMENTUM

```
    "learning_rate": 0.01,  
    "epochs": 20000,  
    "beta1": 0,  
    "beta2": 0,  
    "min_error": 0.02,  
    "optimisation_method": "momentum",  
    "momentum": 0.2  
}
```



sin MOMENTUM

```
    "learning_rate": 0.01,  
    "epochs": 20000,  
    "beta1": 0,  
    "beta2": 0,  
    "min_error": 0.02,  
    "optimisation_method": "NONE",  
    "momentum": 0.2  
}
```



Ejercicio 3 B

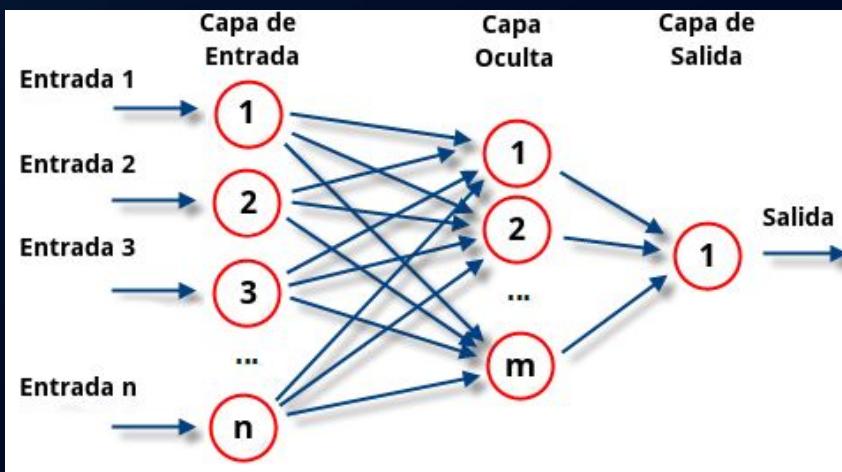
Consigna

Implementar el algoritmo de perceptrón multicapa y utilizarlo para aprender a discriminar si un número es “par”, con entradas dadas por el conjunto de números decimales del 0 al 9 en el archivo “TP2-ej3-digitos.txt”.



»»» Estructura

- Planteamos 35 nodos iniciales
- Planteamos 1 capa intermedia:
 - Sigmoidal que tiene 16 neuronas y recibe 35 datos
- Plantemos 1 nodo final ReLu
- Learning rate: 0.01
- Epochs: 15000
- Error_min: 0.02
- Los betas en 0



```
input=0output= 1.0000
expected output= [1]
input=1output= 0.0000
expected output= [0]
input=2output= 1.0000
expected output= [1]
input=3output= 0.0000
expected output= [0]
input=4output= 1.0000
expected output= [1]
input=5output= 0.0000
expected output= [0]
input=6output= 1.0000
expected output= [1]
input=7output= 0.0000
expected output= [0]
error entrenamiento=1.9243208693175973e-08
TEST
input=0output= 0.5338
expected output= [1]
input=1output= 0.1671
expected output= [0]
error test=0.6333162052722916
```

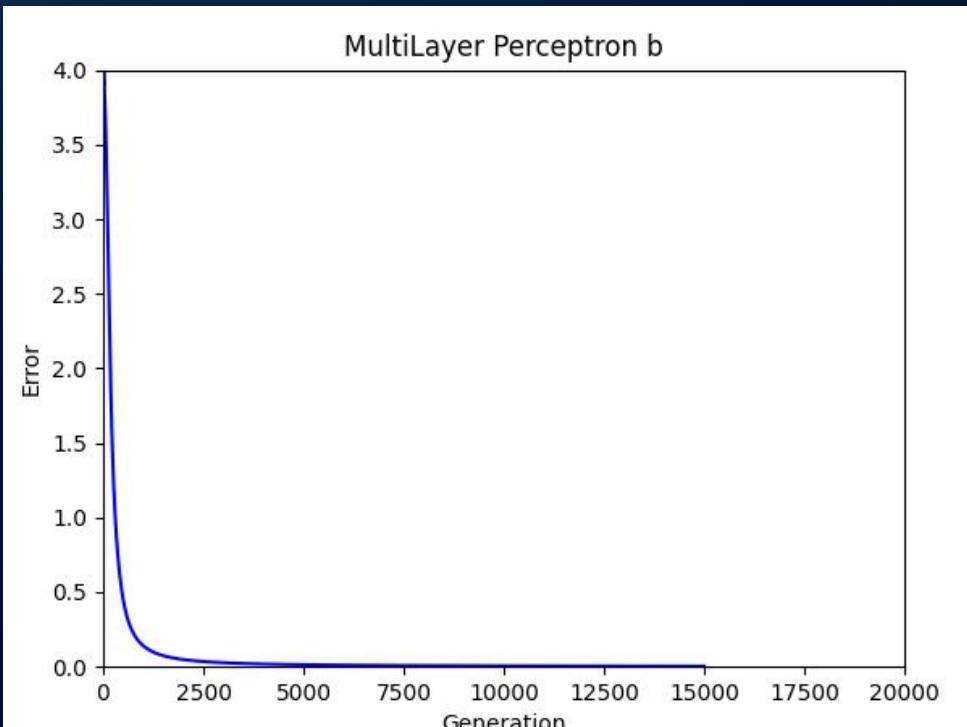
Resultados: Curva de errores

Con MOMENTUM

Accuracy

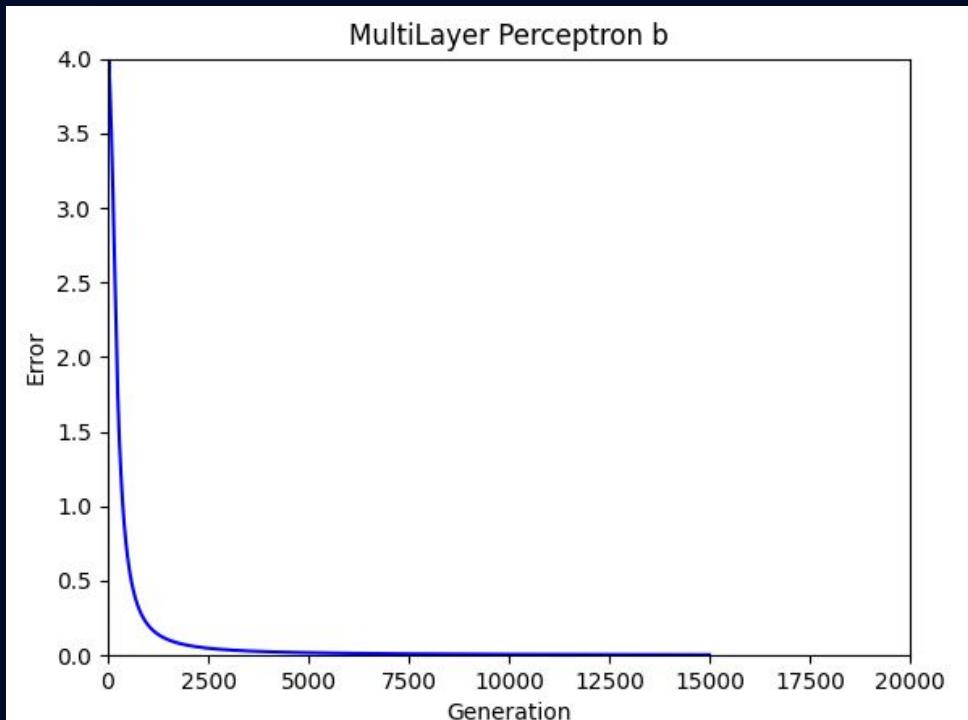
Accuracy = 0.5

```
{  
    "learning_rate": 0.01,  
    "epochs": 15000,  
    "beta1": 0,  
    "beta2": 0,  
    "min_error": 0.02,  
    "optimisation_method": "momentum",  
    "momentum": 0.2  
}
```



sin MOMENTUM

```
{  
    "learning_rate": 0.01,  
    "epochs": 15000,  
    "beta1": 0,  
    "beta2": 0,  
    "min_error": 0.02,  
    "optimisation_method": "NONE",  
    "momentum": 0.2  
}
```



Ejercicio 3 C

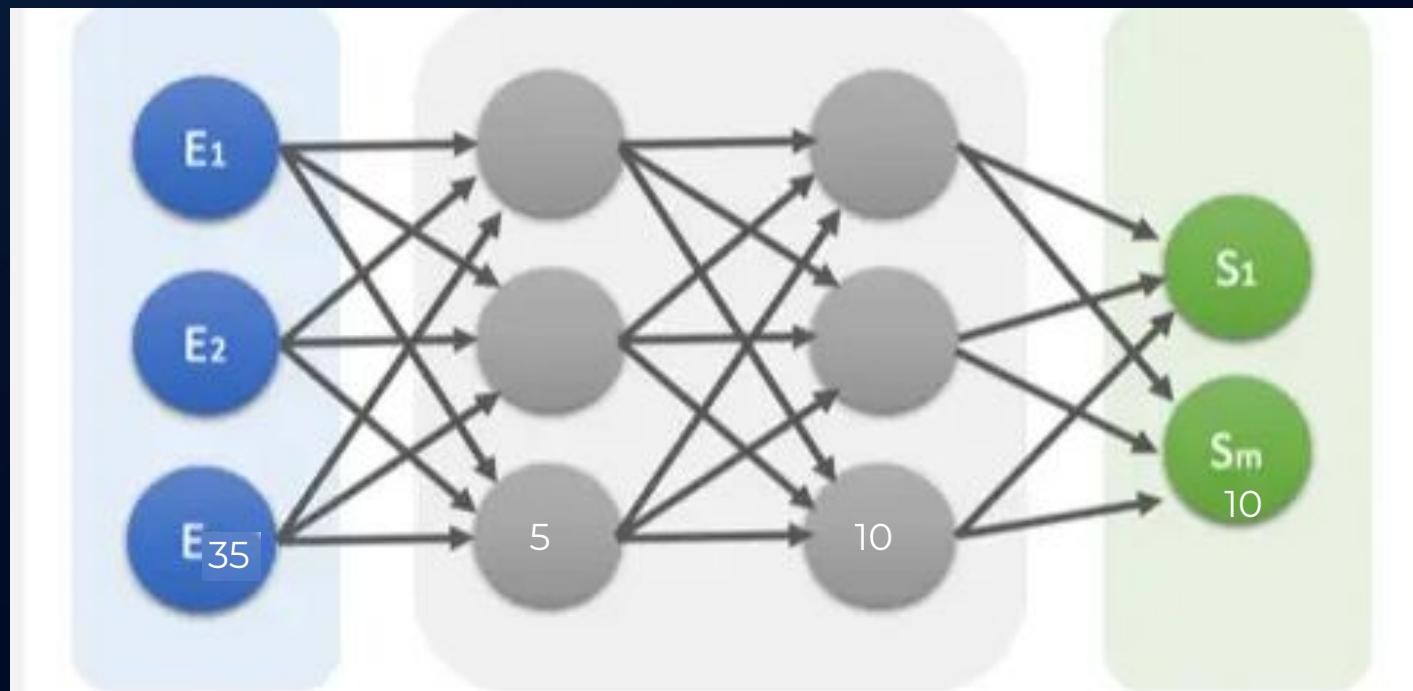
Consigna

Implementar el algoritmo de perceptrón multicapa y utilizarlo para aprender a determinar qué dígito se corresponde con la entrada a la red. Por ejemplo, si alimentamos al perceptrón multicapa con una imagen del dígito “7”, la salida esperada será “7” (la salida puede tomar valores entre 0 y 9). Nótese que se utiliza el mismo perceptrón multicapa, con una salida de 10 neuronas. Una vez que la red haya aprendido, utilizar patrones correspondientes a los dígitos del conjunto de datos, con sus píxeles afectados por ruido. Evaluar los resultados.



»»» Estructura

- Planteamos 35 nodos iniciales
- Planteamos 2 capas intermedia:
 - Sigmoidal que tiene 5 neuronas y recibe 35 datos
 - Sigmoidal que tiene 10 neuronas y reciba 5 datos
- Plantemos 10 nodos finales ReLu
- Learning rate: 0.01
- Epochs: 15000
- Error_min: 0.02
- Los betas en 0



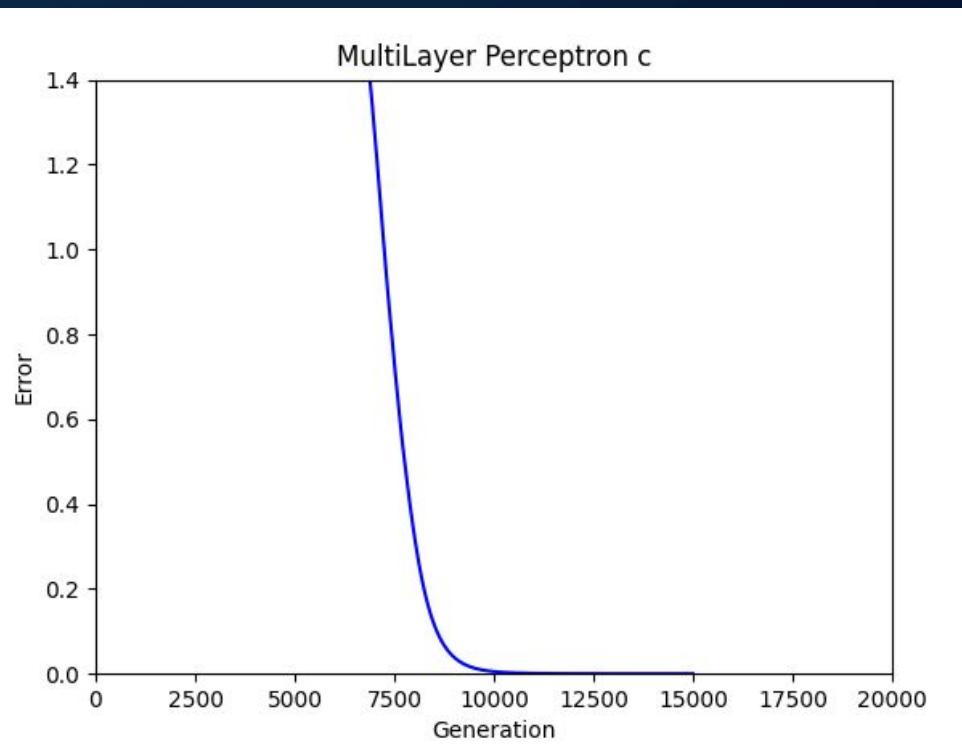
Resultados: Curva de errores

Con MOMENTUM

Accuracy

Accuracy = 0.0

```
    "learning_rate": 0.01,  
    "epochs": 20000,  
    "beta1": 0,  
    "beta2": 0,  
    "min_error": 0.02,  
    "optimisation_method": "momentum",  
    "momentum": 0.2  
}
```

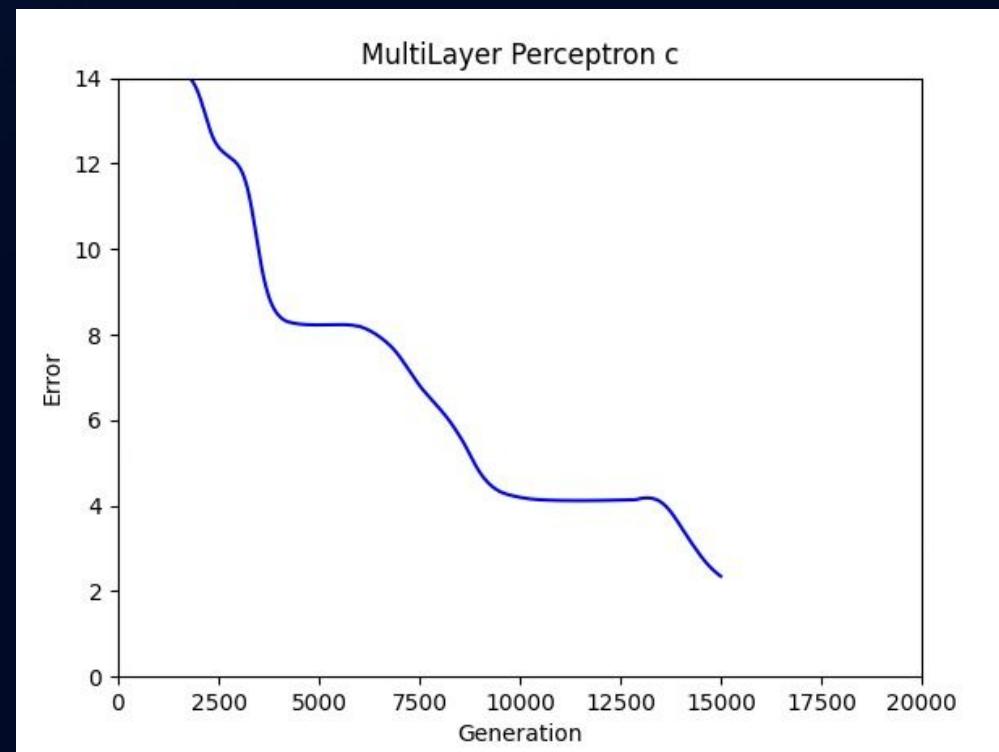


sin MOMENTUM

Accuracy

Accuracy = 0.0

```
    "learning_rate": 0.01,  
    "epochs": 30000,  
    "beta1": 0,  
    "beta2": 0,  
    "min_error": 0.02,  
    "optimisation_method": "NONE",  
    "momentum": 0.2  
}
```



»»» Conclusiones



Item A

A diferencia del ejercicio 1, el perceptrón multicapa puede resolver el problema XOR porque tiene múltiples capas ocultas que le permiten aprender representaciones no lineales de los datos, lo que no es posible con un perceptrón simple.



Item B

Se puede reconocer perfectamente si un número es par o no sobre el conjunto de entrenamiento, y tiende a intentar reconocer sobre el conjunto de test(tira valor 0.5 de que el número 0 sea par). Esto puede ser porque reconoce cierto patrón sobre los números pares como podría ser que como los números pares suelen ser más cerrados está viendo ese patrón o algún otro que no sabemos.



Item C

Se puede reconocer perfectamente al conjunto de entrenamiento pero falla con el test de training, esto es porque casi siempre es imposible que pueda diferenciar y poner en la categoría correcta al número (que el número 9 vaya a la categoría 9 por ejemplo) ya que como ningún elemento en el conjunto entraba en la categoría 9, no pudo entrenar esos caminos neuronales.



Fin.