# Perceptron simple y multicapa

**Redes neuronales artificiales**

# Introducción

01

## IMPLEMENTADO

En Python

02

## CONFIGURACIÓN

en config.yaml

03

## GRÁFICOS

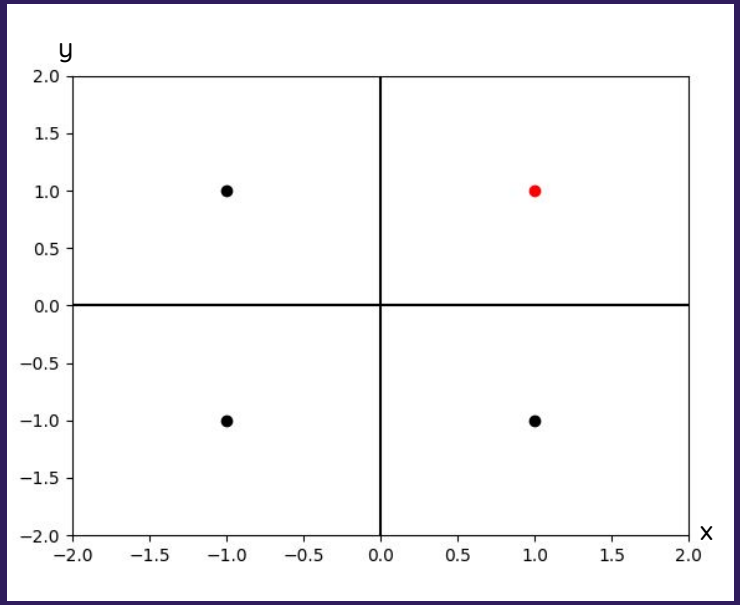
utilizando matplotlib

# 01 Perceptron simple

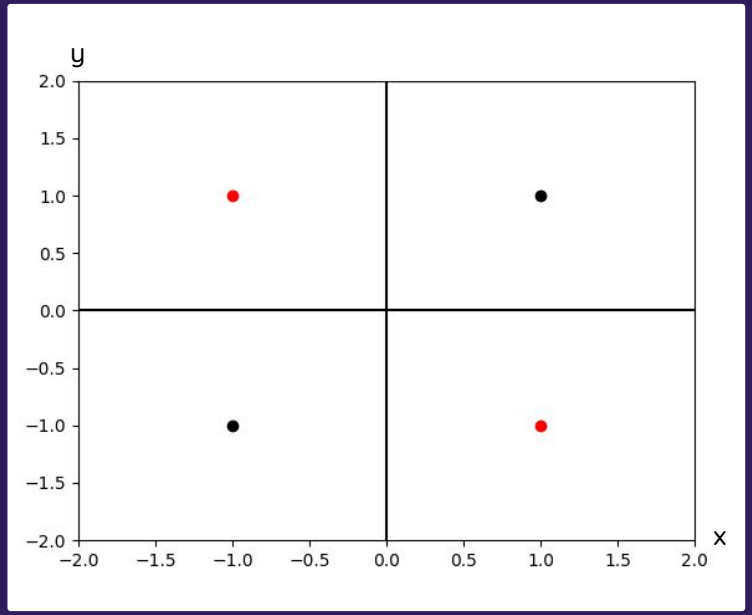
# Ejercicio 1

- Modelo McCulloch y Pitts primer acercamiento a nuestro modelo de perceptrón.
- Función de activación: signo.
- Modelo Rosenblatt asumiendo la conjetura de Hebb.
- Valores de entrada: puntos en r2.
- Valores de salida: 1 y -1

# Problema a analizar

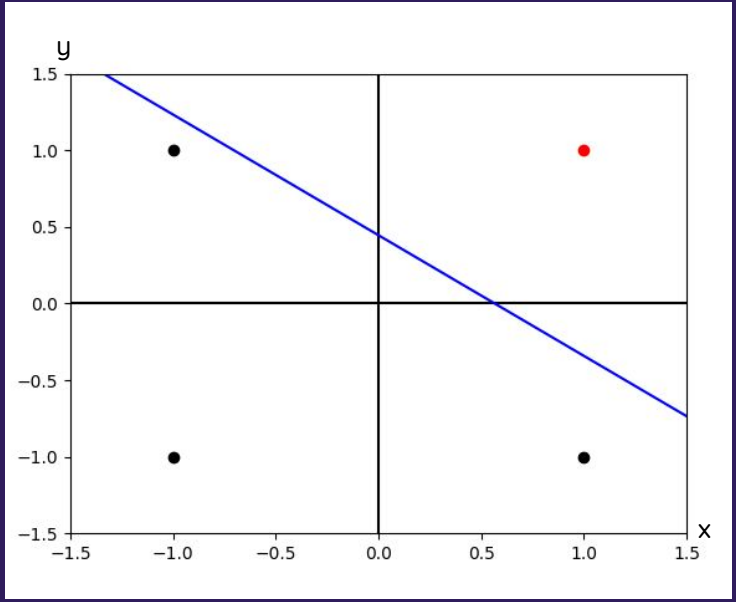


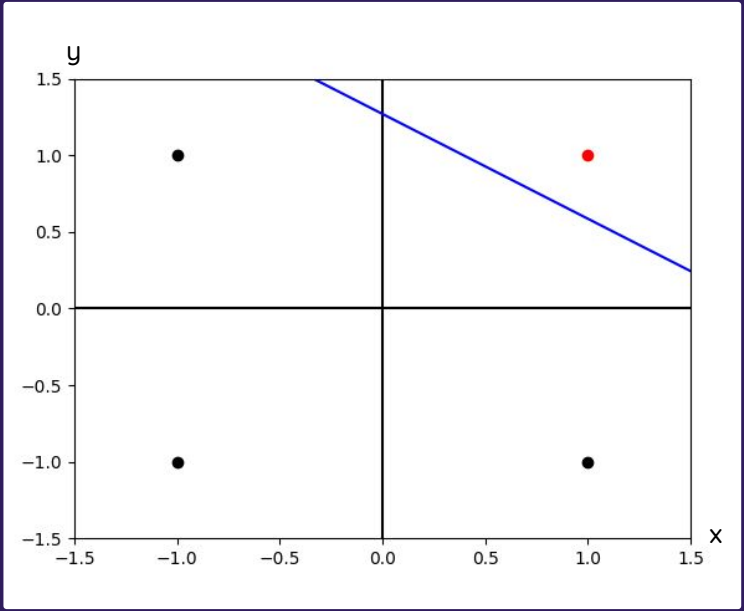

AND



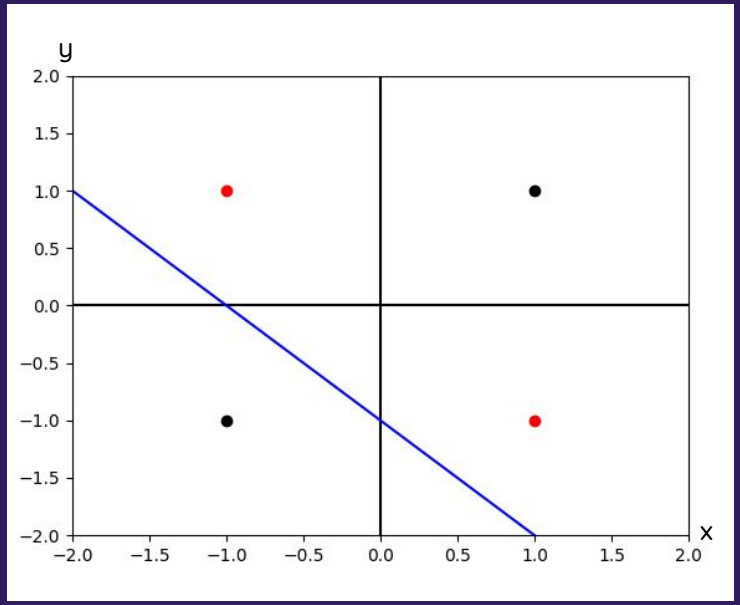

XOR

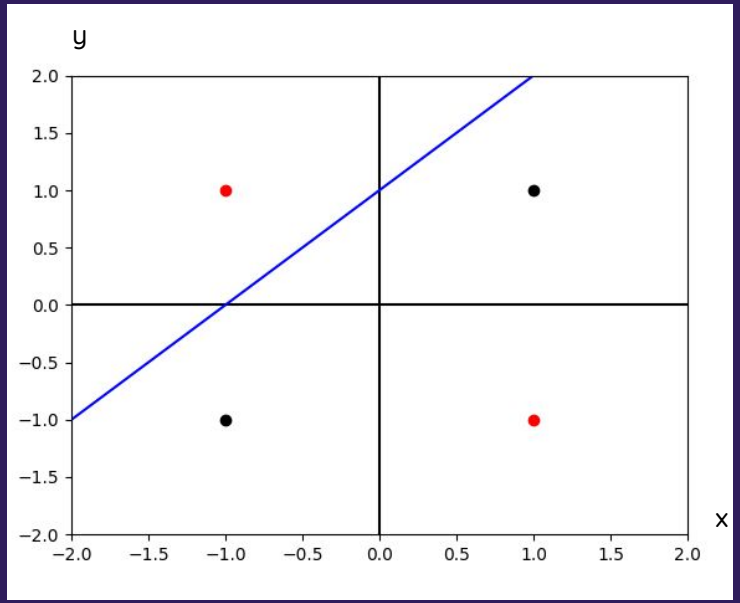
# AND


y
1.5
1.0
0.5
0.0
−0.5
−1.0
−1.5
−1.5
−1.0
−0.5
0.0
0.5
1.0
1.5
x


y
1.5
1.0
0.5
0.0
−0.5
−1.0
−1.5
−1.5
−1.0
−0.5
0.0
0.5
1.0
1.5
x

# XOR


y
x
2.0
1.5
1.0
0.5
0.0
−0.5
−1.0
−1.5
−2.0
−2.0
−1.5
−1.0
−0.5
0.0
0.5
1.0
1.5
2.0


y
x
2.0
1.5
1.0
0.5
0.0
−0.5
−1.0
−1.5
−2.0
−2.0
−1.5
−1.0
−0.5
0.0
0.5
1.0
1.5
2.0

# Conclusiones

- Gráficos realizados con n=0.1

AND

- Linealmente separable.

XOR

- No es linealmente separable.

# 02 Perceptron lineal y no lineal

# Ejercicio 2


## Perceptron lineal

Función de activación: identidad.

## Perceptron no lineal

Función de activación: tangente hiperbólica.

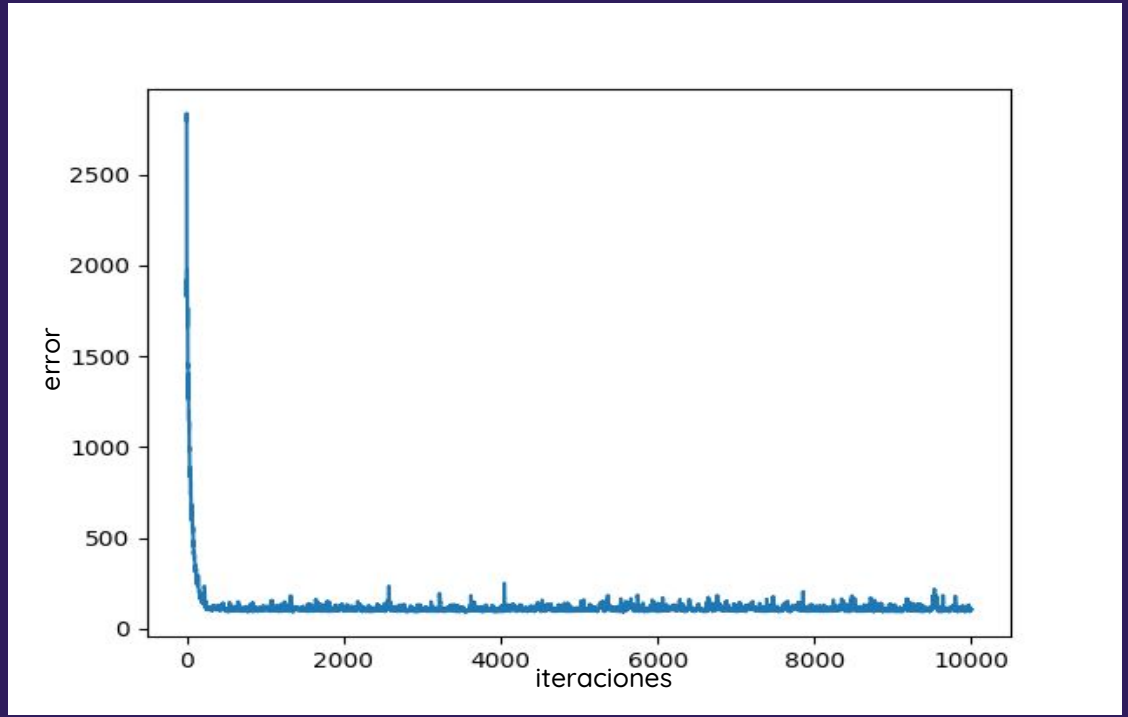
# Ejercicio 2

- Conjunto de entrenamiento 200 valores de entrada
- Valor de entrada: tuplas de tres valores reales
- Valor de salida: un número real

# Perceptron lineal

## error

n: 0.01



# Perceptrón Lineal

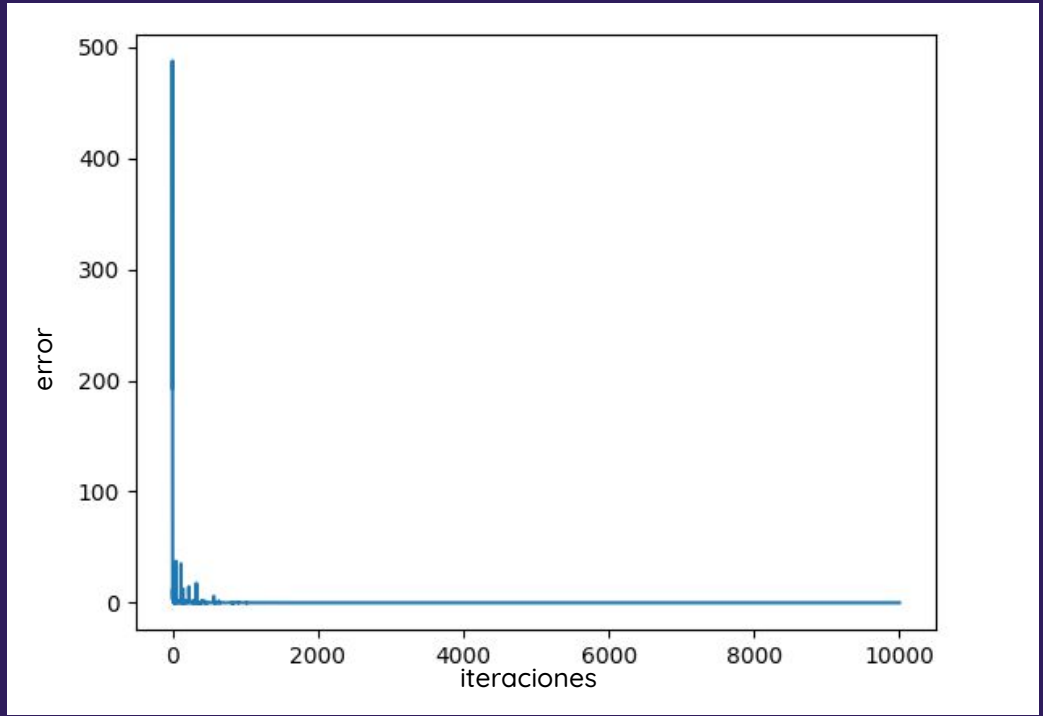
- No se ajusta a los valores del problema
- Probamos con entradas y salidas que se ajusten a:

  $Y(x) = 2 * x$
- Esta función corresponde a una transformación lineal
  - Sean $u,v \in \mathbb{R}$, $F(u+v) = 2*(u+v) = 2*u + 2*v = F(u) + F(v)$
  - Sean $u,v \in \mathbb{R}$, $F(u*v) = 2*k*u = k*2*u = k*F(u)$
  - $F(0) = 2*0 = 0$
- Ahora el perceptrón devuelve lo siguiente:

# Perceptron lineal

## error

n: 0.01



# Conclusiones

- El perceptrón lineal logra aproximar los valores para una transformación lineal
- Los valores dados para el ejercicio no son linealmente separables

# Perceptrón No Lineal

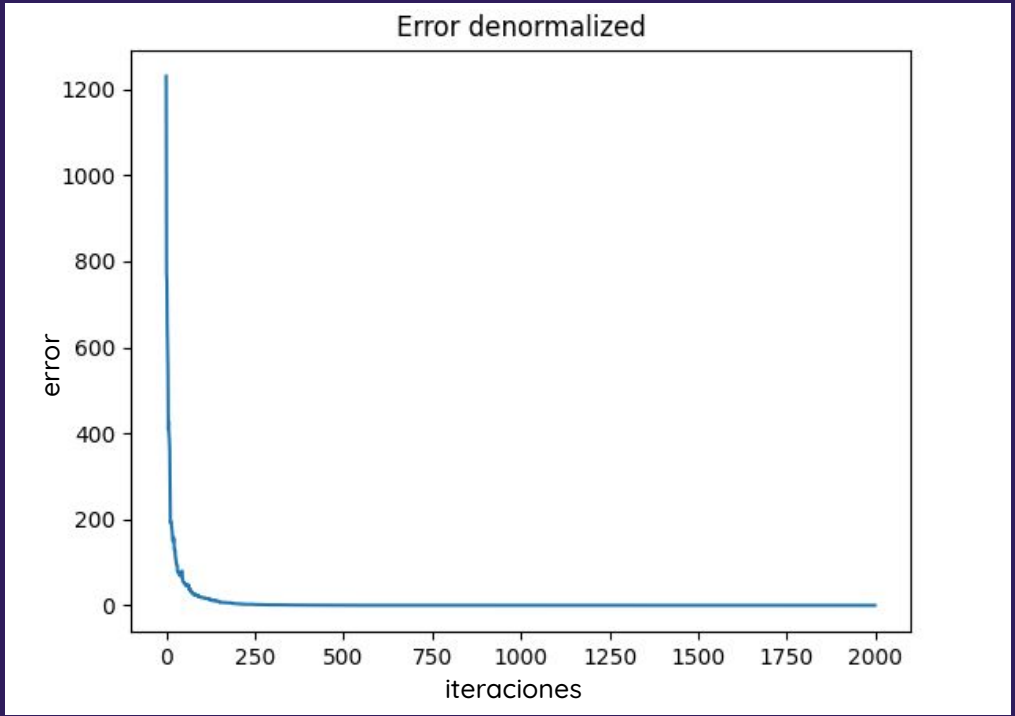
- Se normalizaron los valores de salida entre [-1, 1] porque los valores del conjunto de entrenamiento están fuera de ese intervalo.
- Para normalizar se utilizó la función:

$$2 * ( Y - \min(Y) ) / ( \max(Y) - \min(Y) ) - 1$$

# Perceptron no lineal

## error

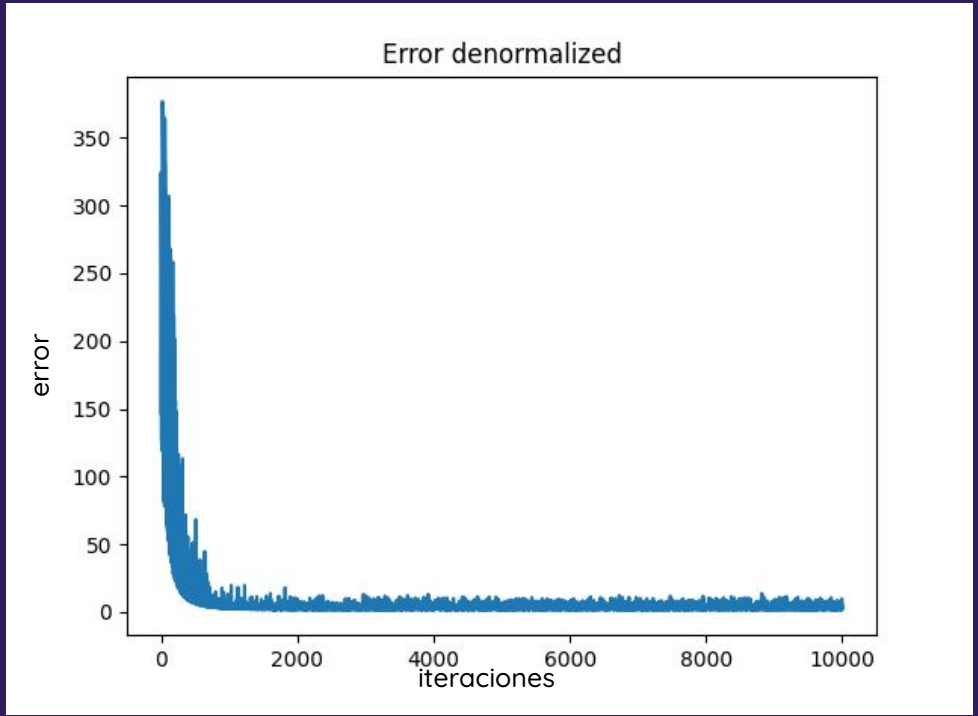
n: 0.1
β: 0.05



# Perceptron no lineal

## transformación lineal error

n: 0.1
β: 0.4



# Conclusiones

- El perceptrón no lineal también logra aproximar correctamente los valores para una transformación lineal.
- Los valores dados para el ejercicio no son linealmente separables

# Capacidad de generalización

- K = 10
- 90 - 10
- El grupo 9 fue el mejor para los testeos
- El perceptrón no lineal tiene una buena capacidad de generalización

| Group | Training error | Testing error |
|---|---|---|
| 1 | 0.48950 | 0.41296 |
| 2 | 0.50130 | 0.35865 |
| 3 | 0.51153 | 0.29826 |
| 4 | 0.48810 | 0.44662 |
| 5 | 0.49027 | 0.44147 |
| 6 | 0.51203 | 0.28725 |
| 7 | 0.48115 | 0.49968 |
| 8 | 0.50395 | 0.33064 |
| 9 | 0.51268 | 0.28114 |
| 10 | 0.50675 | 0.33015 |

# 03 Perceptron multicapa

# Ejercicio 3

- Perceptrones como los que venimos utilizando.
  Se dividen en distintas capas.
- Existen capas ocultas.
- Los pesos se actualizan de forma incremental.

# Ejercicio 3.1

Función lógica ‘O exclusivo’ con entradas:

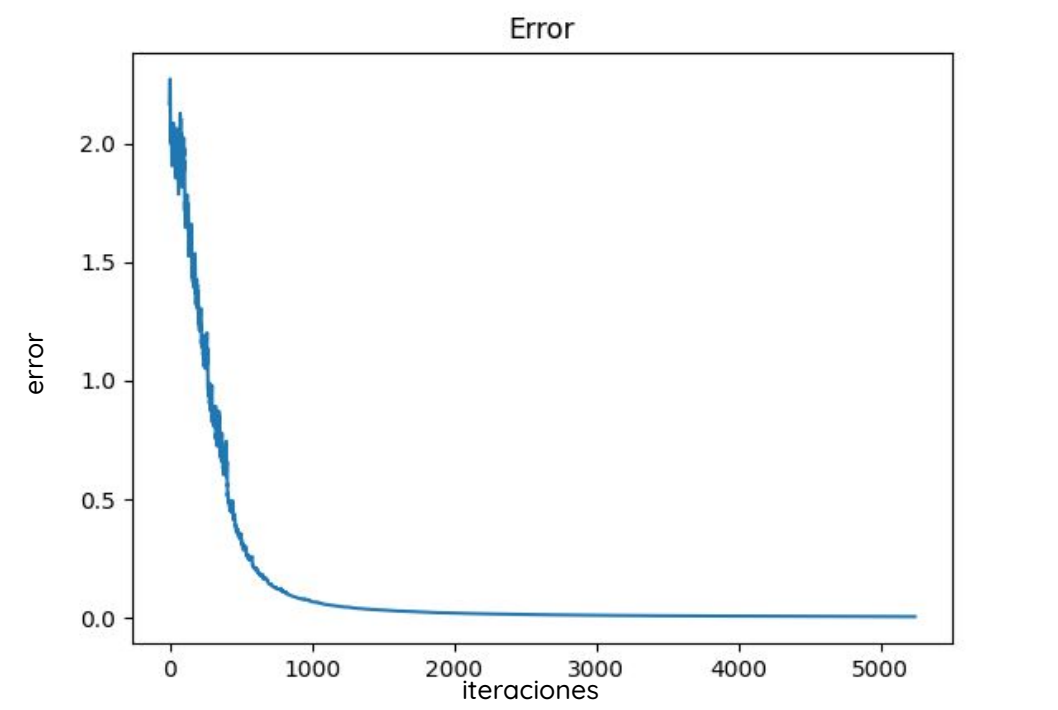
$$x = \{\{-1, 1\}, \{1, -1\}, \{-1, -1\}, \{1, 1\}\},$$

y salida esperada:

$$y = \{1, 1, -1, -1\}.$$

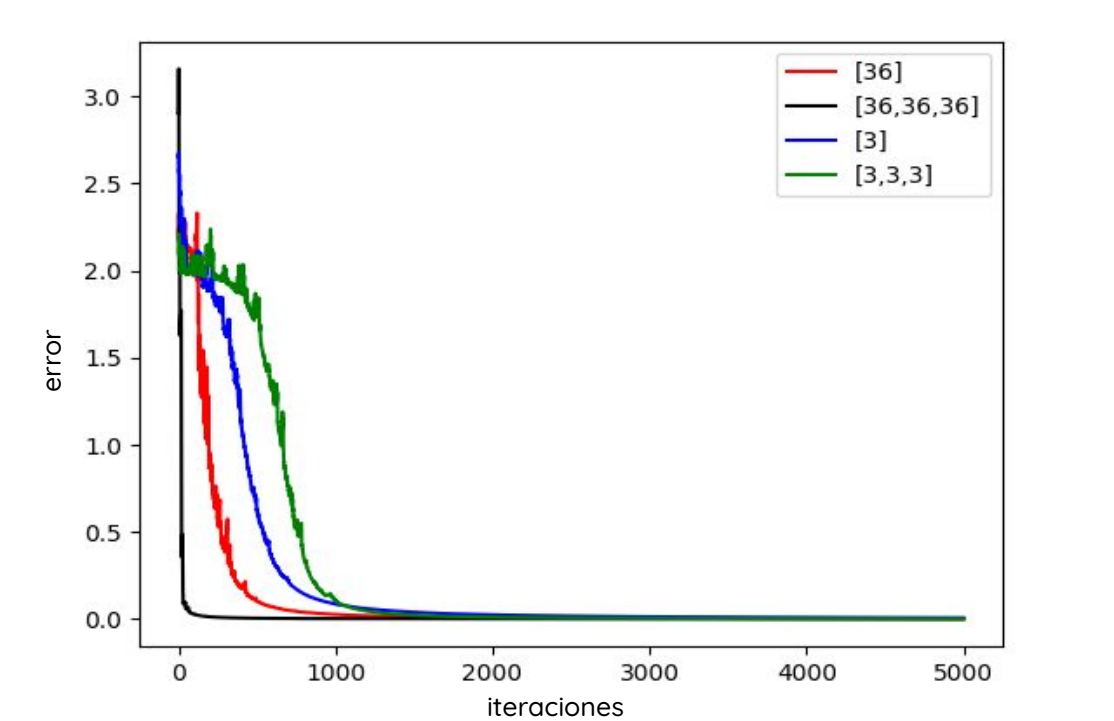
# Ejercicio 3.1
## error

n: 0.01



# Ejercicio 3.1
## error

n: 0.1



# Conclusiones

- Perceptrón multicapa resuelve el problema del XOR.
- Es mejor aumentar la cantidad de nodos en cada capa que aumentar la cantidad de capas.

# Ejercicio 3.2

Discriminar si un número es par, con entradas dadas por el conjunto de números decimales del 0 al 9 representados por imágenes de 5 x 7 pixeles.

# Capacidad de generalización

- K = 5
- 5 grupos de dos quedan dos números para testear
- El perceptrón no lineal tiene una buena capacidad de generalización

| Dataset | Valor esperado | Predicción |
|---|---|---|
| 2 | 1 | 0.96248 |
| 8 | 1 | 0.94881 |
| 5 | -1 | -0.98365 |
| 3 | -1 | -0.93966 |
| 1 | -1 | -0.97592 |
| 7 | -1 | -0.97385 |
| 4 | 1 | 0.97773 |
| 6 | 1 | 0.98265 |
| Testing | | |
| 0 | 1 | 0.45382 |
| 9 | -1 | 0.75934 |

# Predicciones

## Entrenando solo pares

| Testing set | Valor esperado | Predicción |
|---|---|---|
| 1 | -1 | 0.80916 |
| 3 | -1 | 0.99612 |
| 5 | -1 | 0.99442 |
| 7 | -1 | 0.98294 |
| 9 | -1 | 0.98753 |

## Entrenando solo impares

| Testing set | Valor esperado | Predicción |
|---|---|---|
| 0 | 1 | -0.98150 |
| 2 | 1 | -0.87985 |
| 4 | 1 | -0.91791 |
| 6 | 1 | -0.97525 |
| 8 | 1 | -0.96021 |

# Conclusiones

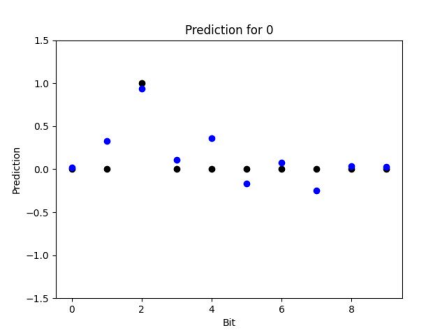
- Clasifica correctamente los elementos del conjunto de entrenamiento en par e impar.
- Los valores de beta grandes siempre llegan a la solución. En cambio, los valores de beta pequeños solo llegan con una tasa de aprendizaje grande/pequeña.

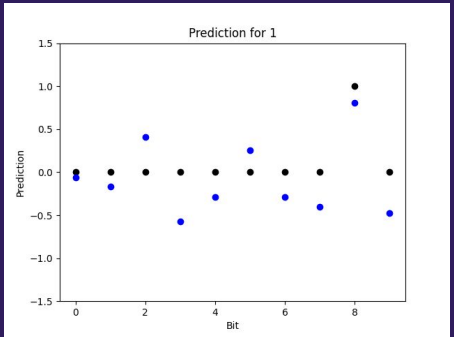
# Ejercicio 3.3

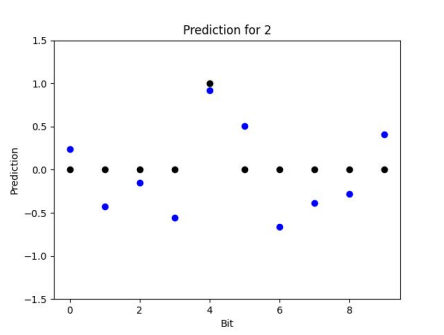
Construir un perceptrón multicapa con 10 unidades de salidas de modo que cada salida represente a un dígito. Las entradas dadas son números del 0 al 9 representados por imágenes de 5x7 pixeles.

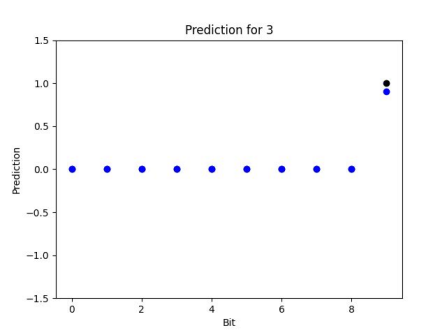
# Ejercicio 3.3

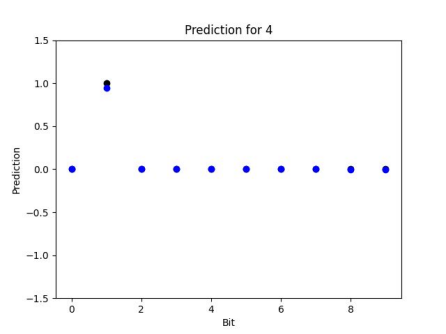
## Se agrega ruido una vez con p = 0.02

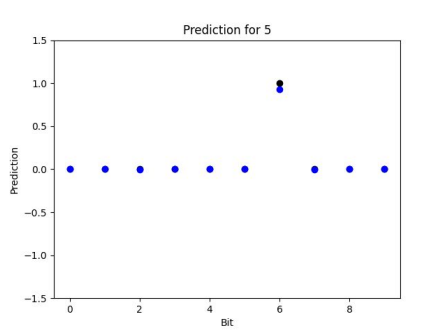


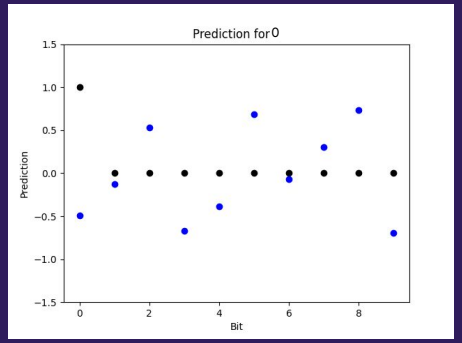



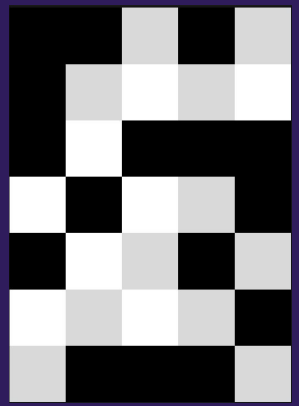



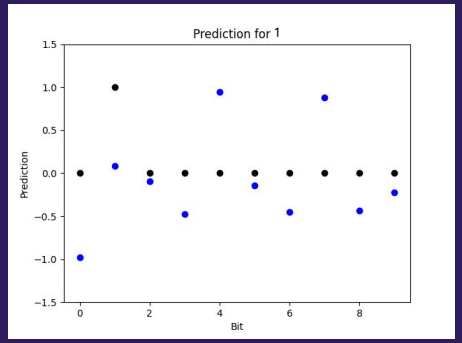



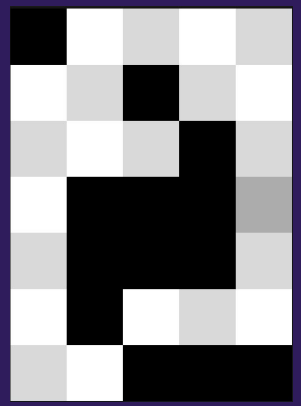
# Ejercicio 3.3

## Se agrega ruido 35 veces con p = 0.02



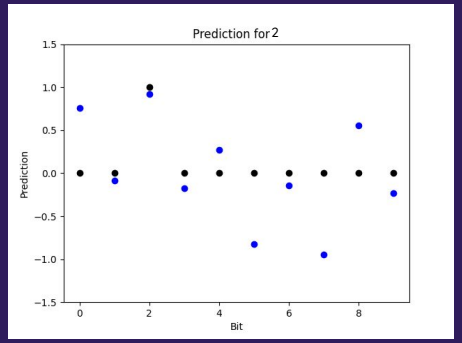



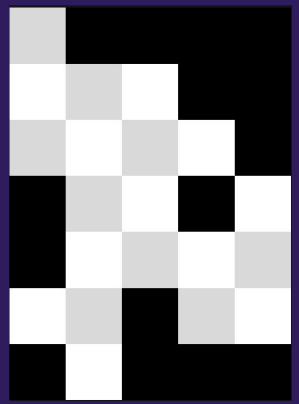


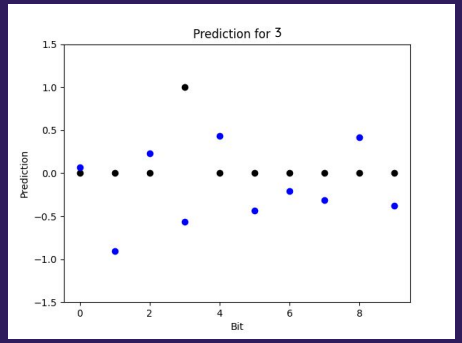



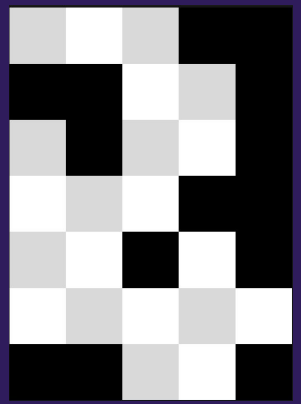
# Ejercicio 3.3

## Se agrega ruido 35 veces con p = 0.02





# Conclusiones

- Con muy poco ruido en la primer iteración no altera mucho el resultado.
- A mayor ruido, mayor error.