# TP3: Perceptrón Simple y Multicapa


**Grupo 2**

Tomás Álvarez Escalante (60127)

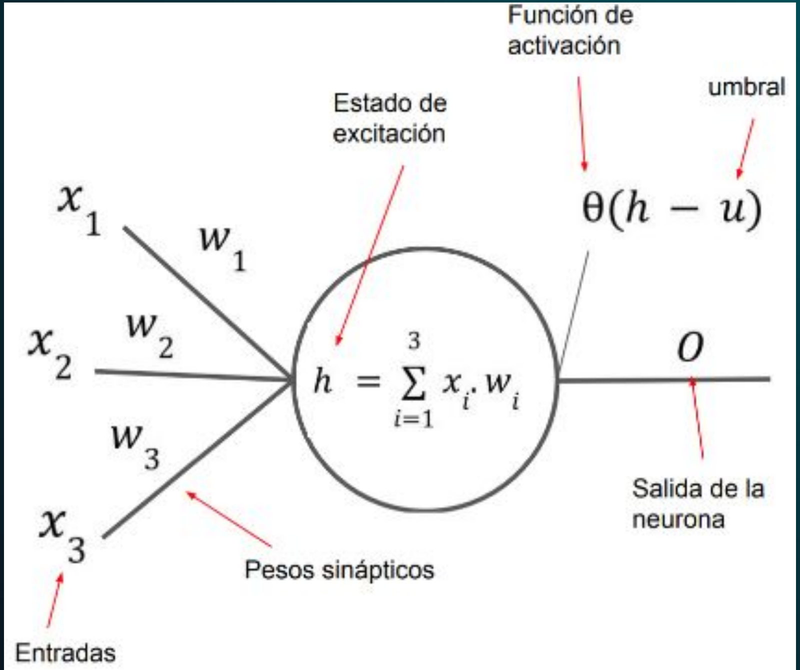
Alejo Francisco Caeiro (60692)

Lucas Agustín Ferreiro (61595)

Román Gómez Kiss (61003)

# TABLA DE CONTENIDOS

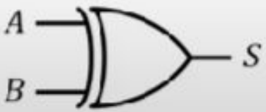
# Perceptrón Simple



# Perceptrón Simple Escalón

Implementar el algoritmo de perceptrón simple con función de activación escalón y utilizarlo para aprender los problemas lógicos AND y XOR


| A | B | S |
|---|---|---|
| 0 | 0 | 0 |
| 0 | 1 | 0 |
| 1 | 0 | 0 |
| 1 | 1 | 1 |




| A | B | S |
|---|---|---|
| 0 | 0 | 0 |
| 0 | 1 | 1 |
| 1 | 0 | 1 |
| 1 | 1 | 0 |

# Perceptrón Simple Escalón

Se utiliza únicamente para resolver problemas linealmente separables, es decir, los datos pertenecen excluyentemente a solo una de dos clases.

$$clase = \mathrm{O} = \theta\left(\sum_{i=0}^{n} x_i \cdot w_i\right) \qquad \theta(x) = \begin{cases} 1 & x \geq 0 \\ -1 & \text{en otro caso} \end{cases}$$

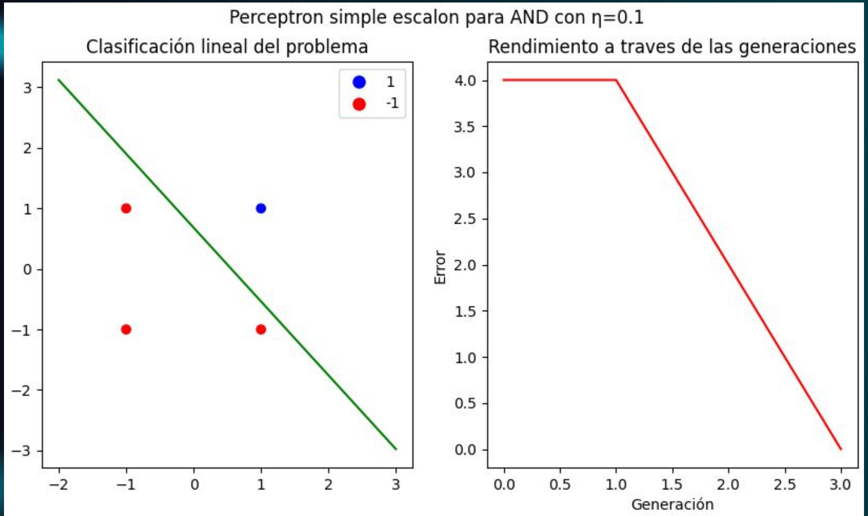
# ¿CÓMO APRENDE EL PERCEPTRÓN?

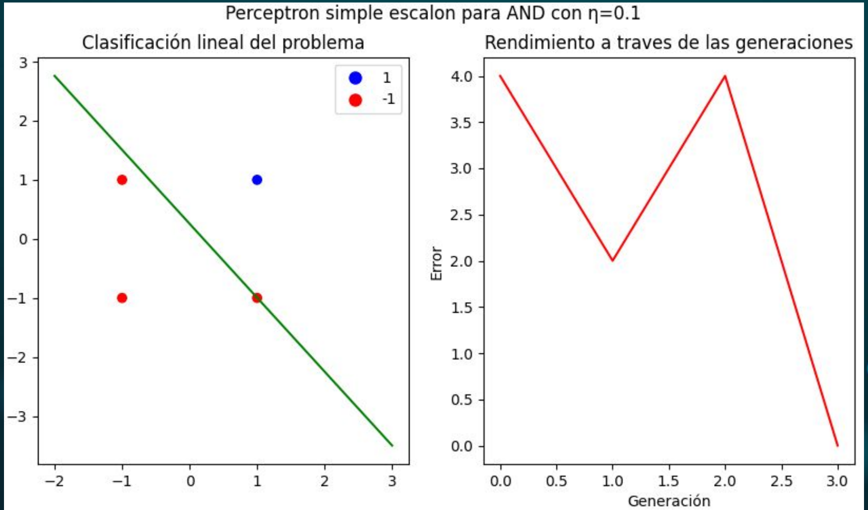
Mediante la actualización de los pesos sinápticos en un proceso iterativo usando el método incremental, ya que es más rápido y eficiente en términos de memoria.

$$w^{nuevo} = w^{anterior} + \Delta w$$

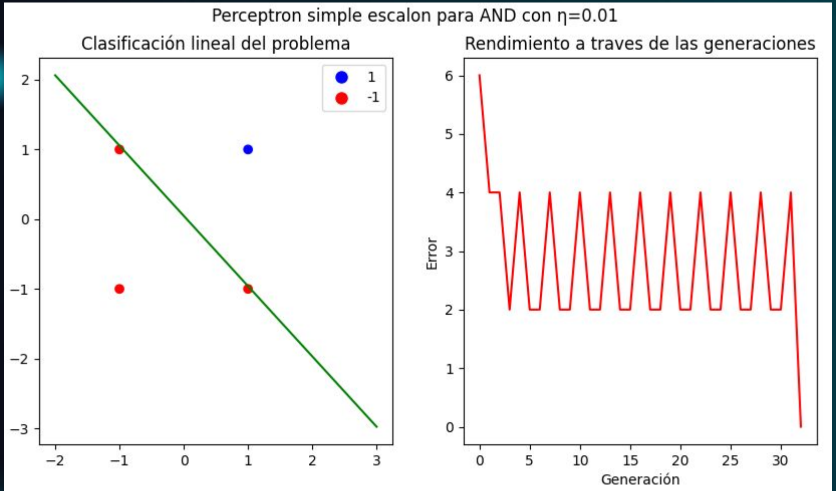
$$\Delta w = \eta(\zeta^{\mu} - O^{\mu})x^{\mu}$$

# AND
# η=0.1



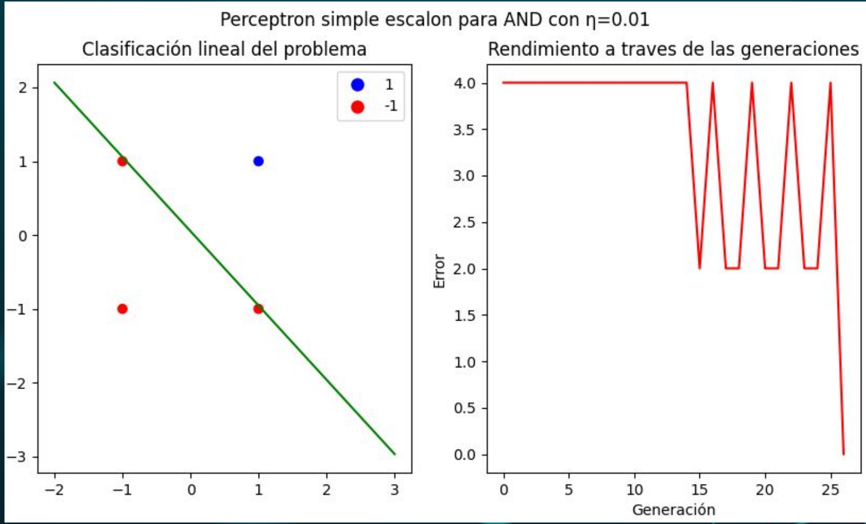




```
Finished training in 4 epochs
Perceptron: [-0.10316961  0.63578899  0.54031869]
Predicted: -1 AND 1 = -1. Expected: -1
Predicted: 1 AND -1 = -1. Expected: -1
Predicted: -1 AND -1 = -1. Expected: -1
Predicted: 1 AND 1 = 1. Expected: 1
```

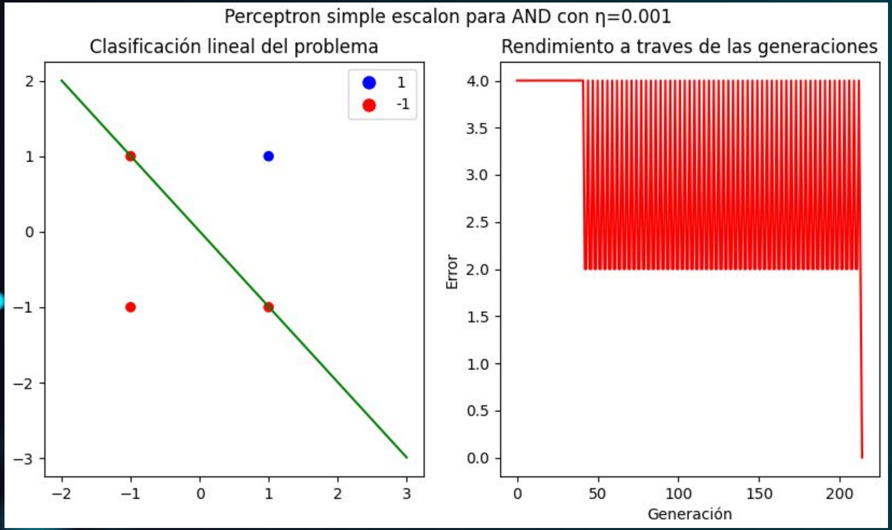



# AND
# $\eta$=0.01

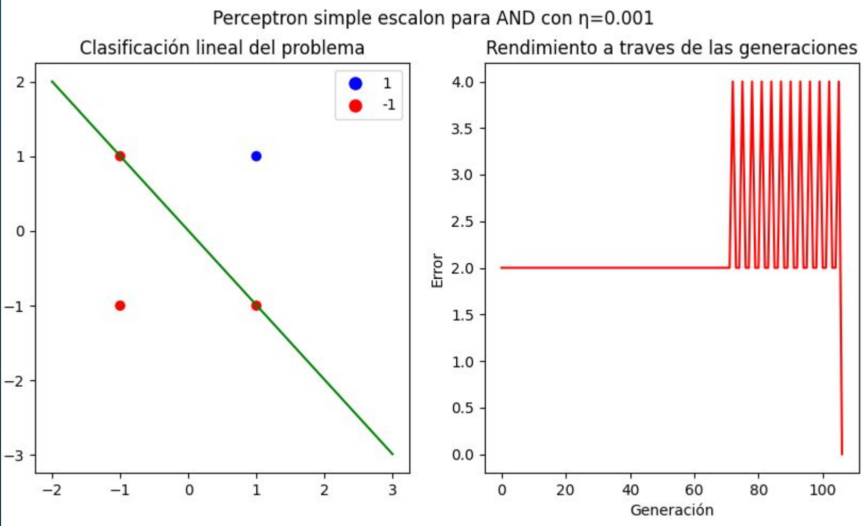
```
Finished training in 25 epochs
Perceptron: [-0.02732716  0.41984504  0.43870708]
Predicted: -1 AND 1 = -1. Expected: -1
Predicted: 1 AND -1 = -1. Expected: -1
Predicted: -1 AND -1 = -1. Expected: -1
Predicted: 1 AND 1 = 1. Expected: 1
```

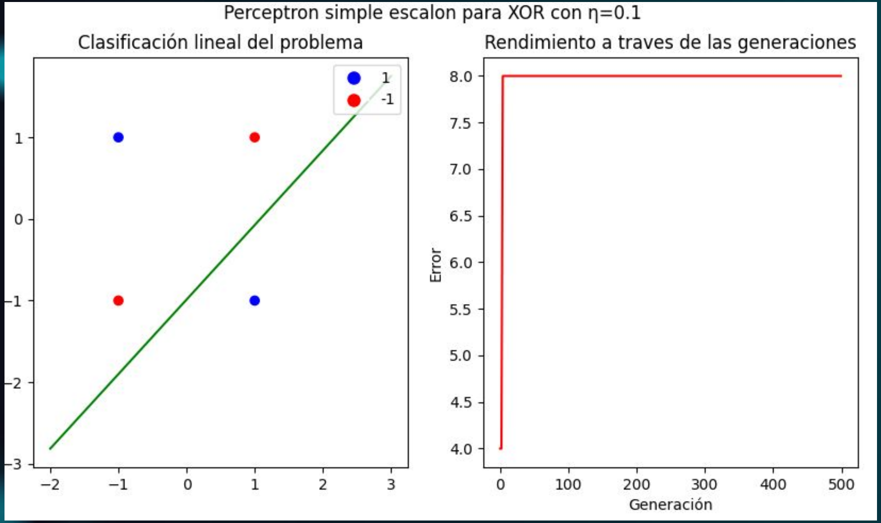


# AND
# η=0.001



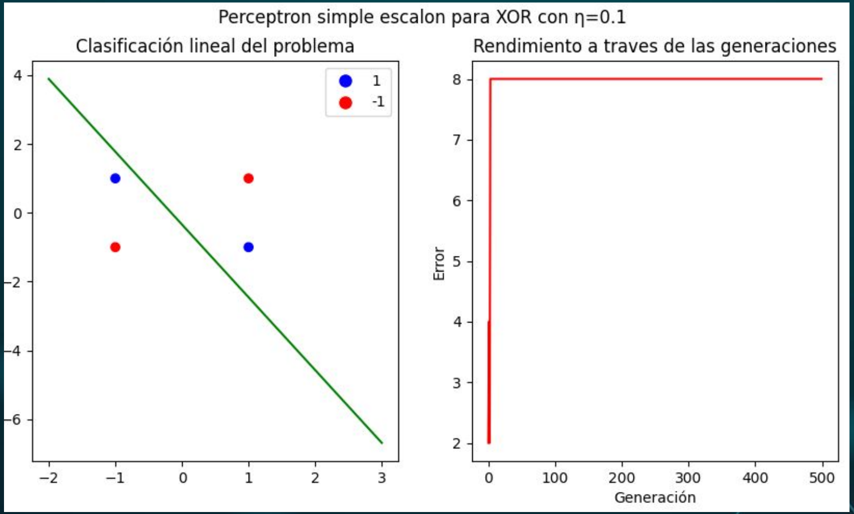

```
Finished training in 237 epochs
Perceptron: [-0.00443504  0.57355219  0.57434892]
Predicted: -1 AND 1 = -1. Expected: -1
Predicted: 1 AND -1 = -1. Expected: -1
Predicted: -1 AND -1 = -1. Expected: -1
Predicted: 1 AND 1 = 1. Expected: 1
```

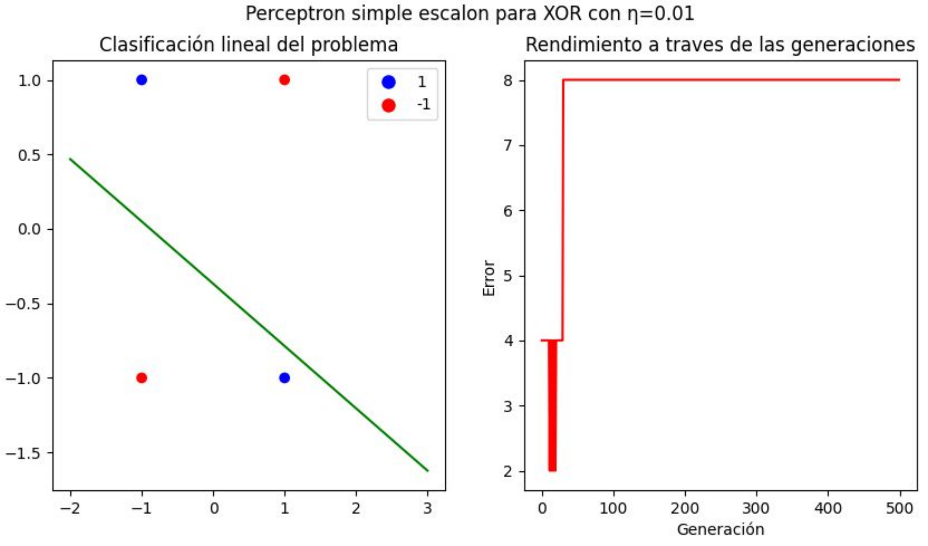





# XOR
# η=0.1

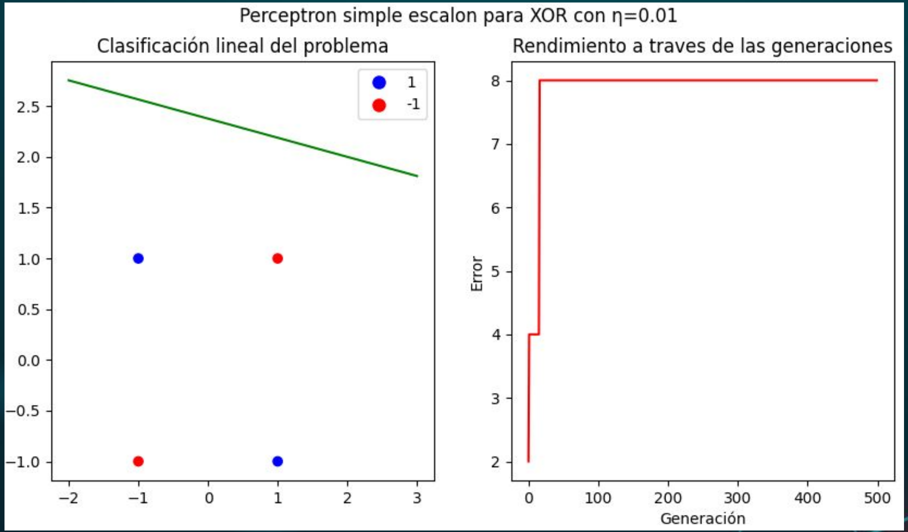
```
Finished training in 500 epochs
Perceptron: [-0.26332466  0.0930597  -0.20535885]
Predicted: -1 XOR 1 = -1. Expected: 1
Predicted: 1 XOR -1 = 1. Expected: 1
Predicted: -1 XOR -1 = -1. Expected: -1
Predicted: 1 XOR 1 = -1. Expected: -1
```

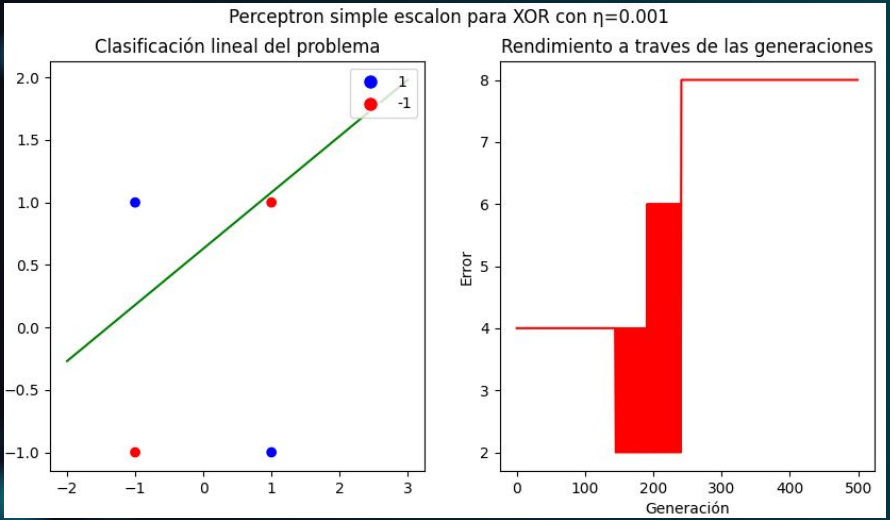





# XOR
# η=0.01

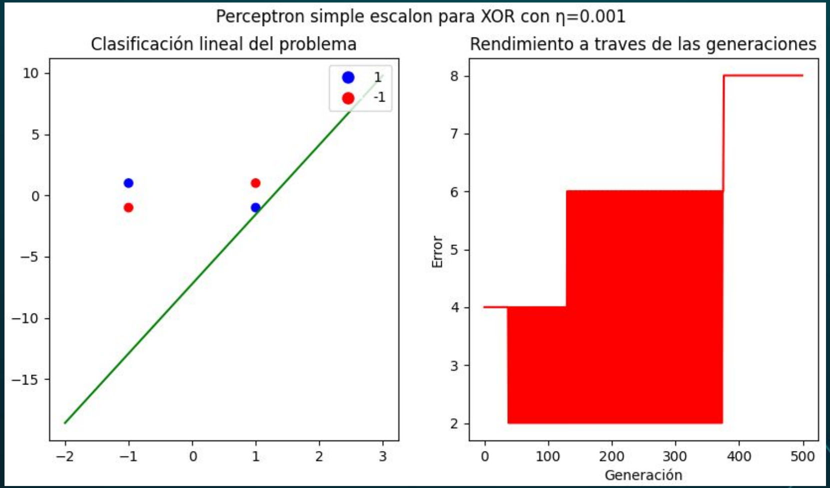
```
Finished training in 500 epochs
Perceptron: [-0.02034904  0.02147807 -0.01129699]
Predicted: -1 XOR 1 = -1. Expected: 1
Predicted: 1 XOR -1 = 1. Expected: 1
Predicted: -1 XOR -1 = -1. Expected: -1
Predicted: 1 XOR 1 = -1. Expected: -1
```






# XOR
# η=0.001

```
Finished training in 500 epochs
Perceptron: [-0.00235153  0.00184429 -0.00032526]
Predicted: -1 XOR 1 = -1. Expected: 1
Predicted: 1 XOR -1 = -1. Expected: 1
Predicted: -1 XOR -1 = -1. Expected: -1
Predicted: 1 XOR 1 = -1. Expected: -1
```



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

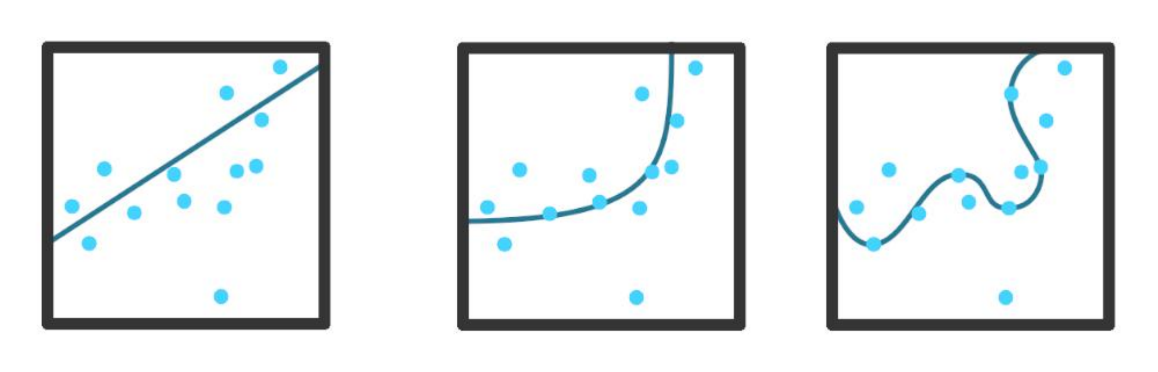
Solo puede resolver problemas linealmente separables

✓ AND
✗ XOR

A menor valor de tasa de aprendizaje ($\eta$) se tarda más en converger, es decir, se tarda más épocas.

# Perceptrón Simple Lineal y No Lineal

Evaluar la capacidad de cada uno de los perceptrones para aprender una función cuyas muestras se encuentran disponibles en un archivo.



# ¿CUÁL ES LA IDEA?

Ahora vamos a ver dos modelos de redes neuronales capaces de aprender a clasificar conjuntos de datos que no necesariamente son linealmente separables.

Este planteamiento es una tarea de estimación, lo que implica la búsqueda de una función lineal o no lineal que se acerque lo más posible a los datos suministrados.

La diferencia con el perceptrón lineal escalón es como se calcula $\Delta w$, pues ahora se van ajustando los pesos usando el **método del gradiente descendente**

$$O = \theta\left(\sum_{i=0}^{n} x_i \cdot w_i\right)$$

$$\Delta w = \eta(\zeta^{\mu} - O^{\mu})\theta'(h)x^{\mu}$$

## Perceptrón lineal

$$\theta(h) = h$$
$$\theta'(h) = 1$$

## Perceptrón no lineal

Tenemos dos formas (usando funciones sigmoideas):

$$\theta(x) = tanh(\beta x)$$

$$\theta(x) = \frac{1}{1 + e^{-2\beta x}}$$

$$\theta'(h) = \beta(1 - \theta^2(h))$$

$$\theta'(h) = 2\beta\theta(h)(1 - \theta(h))$$

# CONSIDERACIONES PARA LOS PERCEPTRONES NO LINEALES

Es importante tener en cuenta que las imágenes de las funciones sigmoideas están acotadas.

- (-1;1) para tanh
- (0;1) para logística

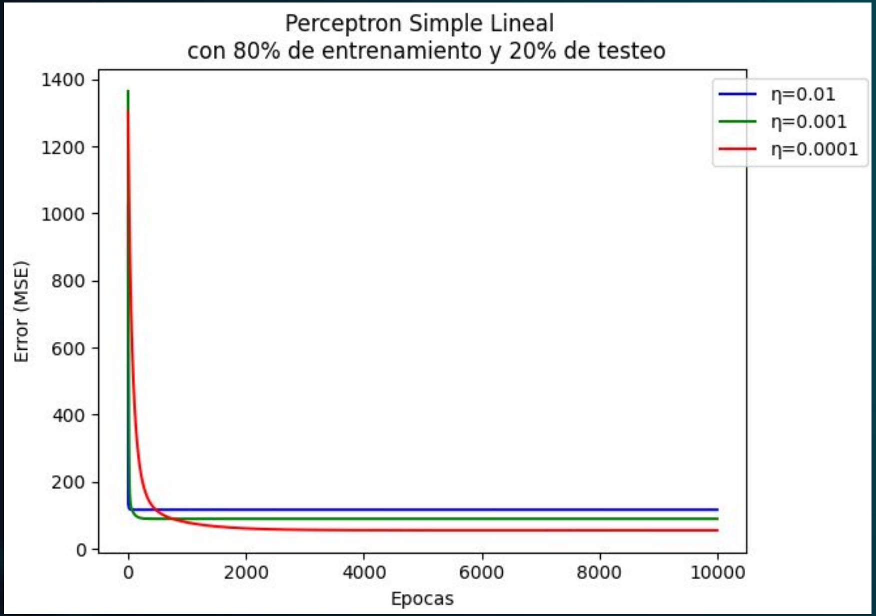
Por lo tanto, como las salidas esperadas de nuestros perceptrones se encuentran en todos los reales, tendremos que normalizar el valor de θ.

Los datos se normalizaron utilizando el método ***Min-Max Feature Scaling***:

$$X' = \frac{X - X_{min}}{X_{max} - X_{min}}(b - a) + a$$

```
x1,x2,x3,y
1.200,-0.800,0.000,21.755
1.200,0.000,-0.800,7.176
1.200,-0.800,1.000,43.045
0.000,1.200,-0.800,2.875
7.900,1.000,0.000,26.503
0.400,0.000,2.700,68.568
0.000,0.400,2.700,61.301
-1.300,3.230,3.000,23.183
0.400,2.700,0.000,2.820
0.400,2.700,2.000,17.654
-1.300,0.000,3.230,72.512
0.000,-1.300,3.230,88.184
7.900,1.000,-2.000,4.653
1.800,0.000,1.600,49.000
0.000,-2.000,2.000,76.852
-0.500,0.600,0.000,7.871
0.000,1.800,1.600,18.543
-2.000,2.000,0.000,2.660
-0.500,0.600,2.500,51.000
7.900,0.000,1.000,64.107
-1.300,3.230,0.000,1.480
0.000,7.900,1.000,0.320
-2.000,0.000,2.000,40.131
-2.000,2.000,-1.000,0.995
0.000,-0.500,0.600,24.974
1.800,1.600,1.300,21.417
-0.500,0.000,0.600,18.243
1.800,1.600,0.000,6.914
```

28 muestras, con 3 valores de entrada y 1 de salida

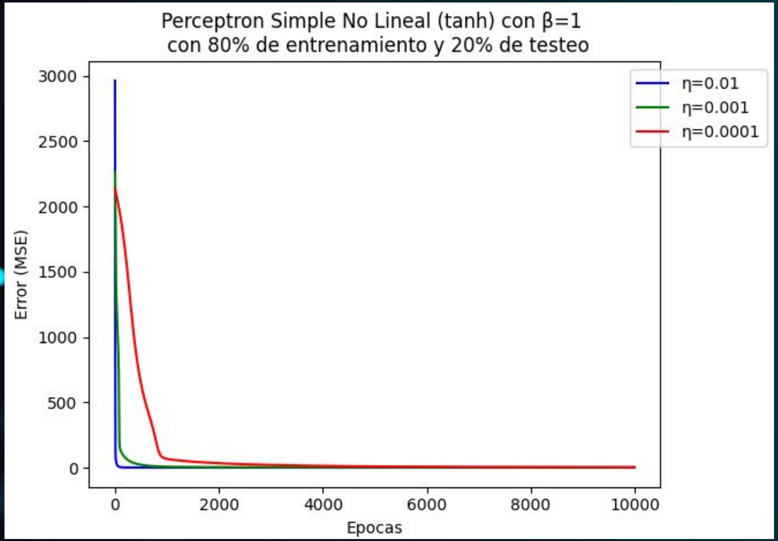


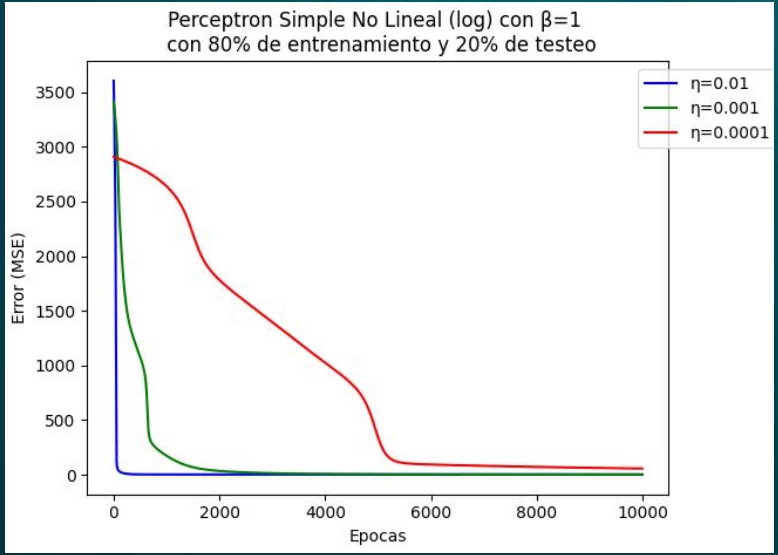

Podemos notar que el perceptrón simple lineal no es muy preciso.

Si bien da un valor “aproximado”, puede ocurrir que no sepa ajustar algunos datos.

$\eta = 0.001$
Épocas = 10000
Porcentaje de entrenamiento = 80%

```
Finished Training with LINEAR
Predicted: 2.9663584280520006. Expected: 4.653
Predicted: 31.360270688200306. Expected: 26.503
Predicted: 28.429852358580185. Expected: 18.543
Predicted: 57.03612643973123. Expected: 61.301
Predicted: -36.68596150676703. Expected: 0.32
Predicted: 35.57299006962017. Expected: 24.974
Train MSE: 64.92987740719872
 Test MSE: 270.6927686520575
Finished testing
Weights: [22.41566556  2.30669706 -9.27830167 14.19695613]. MSE_train: 64.92987740719872. MSE_test: 270.6927686520575
```

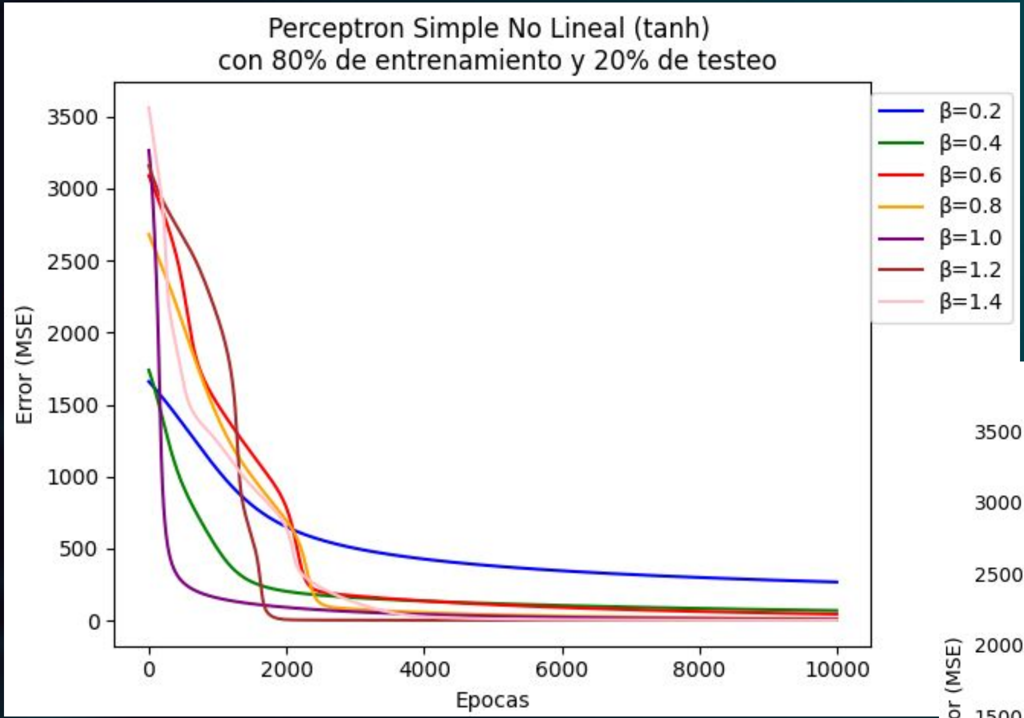


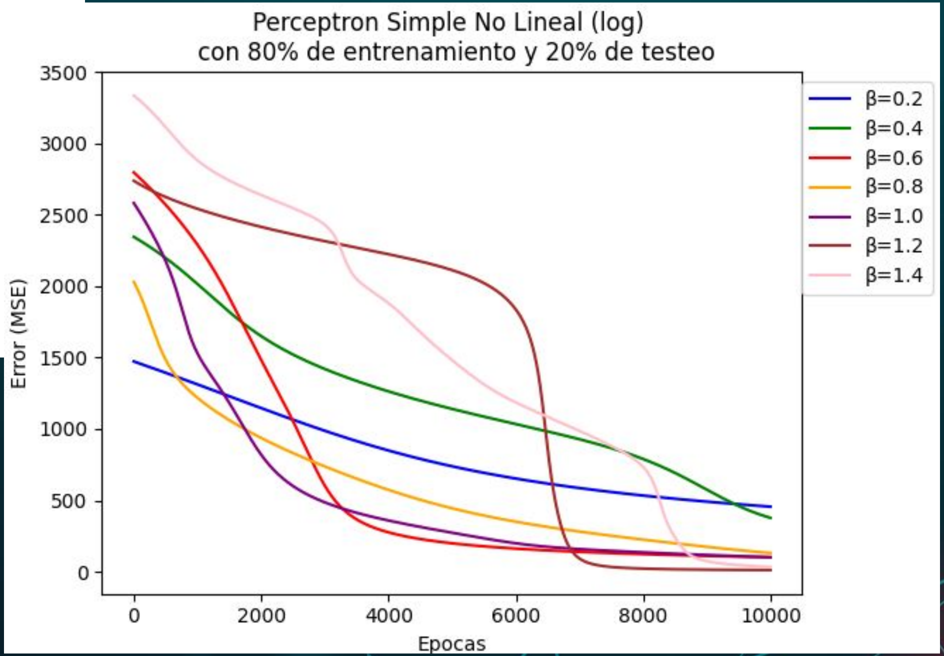

```
Finished Training with NON_LINEAR_LOG
Predicted: 3.9841794327716524. Expected: 4.653
Predicted: 45.800647418314796. Expected: 43.045
Predicted: 2.354520420104784. Expected: 2.82
Predicted: 51.078459493491486. Expected: 49.0
Predicted: 22.392178644604876. Expected: 23.183
Predicted: 42.594076539979554. Expected: 40.131
Train MSE: 1.9701305659856465
 Test MSE: 3.211620534757263
Finished testing
Weights: [-0.95113396  0.11090819 -0.35714093  0.56759701]. MSE_train: 1.9701305659856465. MSE_test: 3.211620534757263
```

```
Finished Training with NON_LINEAR_TANH
Predicted: 68.07939732929461. Expected: 68.568
Predicted: 71.28325063208051. Expected: 72.512
Predicted: 1.273001433588409. Expected: 1.48
Predicted: 26.372927461594916. Expected: 26.503
Predicted: 2.3900071733417514. Expected: 2.875
Predicted: 20.938225334776188. Expected: 21.417
Train MSE: 2.2886957810986983
 Test MSE: 0.3787946787827862
Finished testing
Weights: [-0.98676108  0.11417033 -0.34716075  0.5735556 ]. MSE_train: 2.2886957810986983. MSE_test: 0.3787946787827862
```

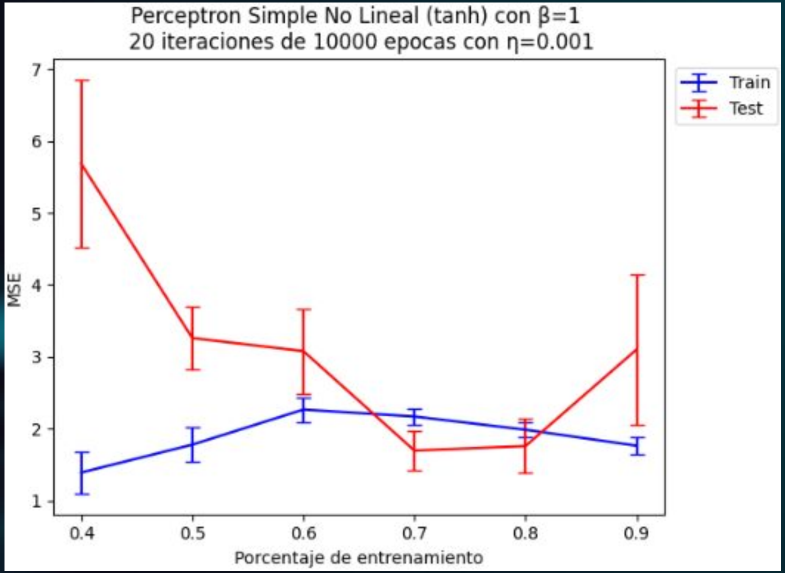
η = 0.001
Épocas = 10000
Porcentaje de entrenamiento = 80%
Beta = 1

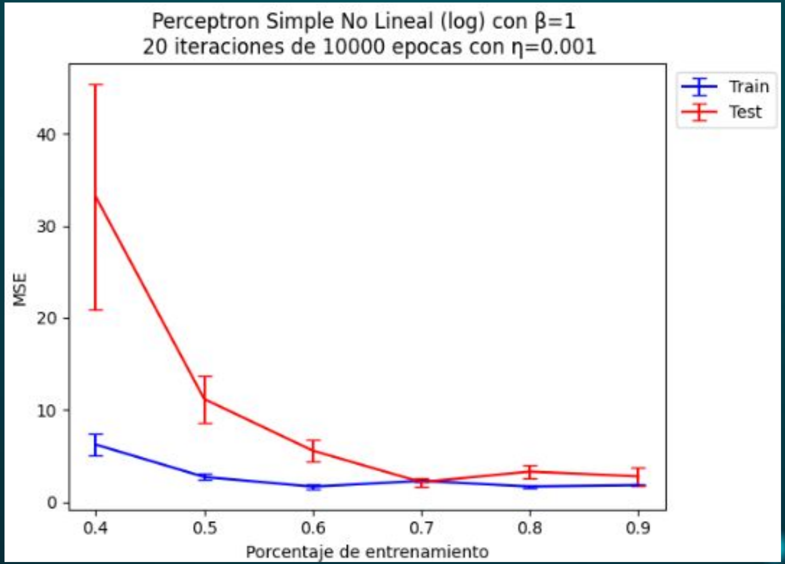

Perceptron Simple No Lineal (tanh)
con 80% de entrenamiento y 20% de testeo
Error (MSE)
Epocas
β=0.2
β=0.4
β=0.6
β=0.8
β=1.0
β=1.2
β=1.4
0
500
1000
1500
2000
2500
3000
3500
0
2000
4000
6000
8000
10000

η = 0.001


Perceptron Simple No Lineal (log)
con 80% de entrenamiento y 20% de testeo
Error (MSE)
Epocas
β=0.2
β=0.4
β=0.6
β=0.8
β=1.0
β=1.2
β=1.4
0
500
1000
1500
2000
2500
3000
3500
0
2000
4000
6000
8000
10000

# Evaluar la capacidad de generalización del perceptrón simple no lineal utilizando un subconjunto para entrenar y otro para testear.



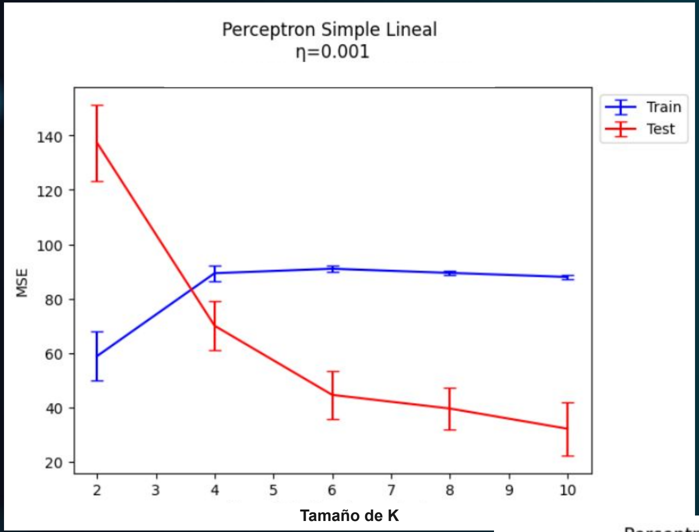

- Los mejores porcentajes para entrenar y testear son 70%-30% u 80%-20% respectivamente.
- Tanto con porcentajes muy bajos como muy altos para el conjunto de entrenamiento, se produce una pérdida de capacidad de generalización (overfitting) del perceptrón.

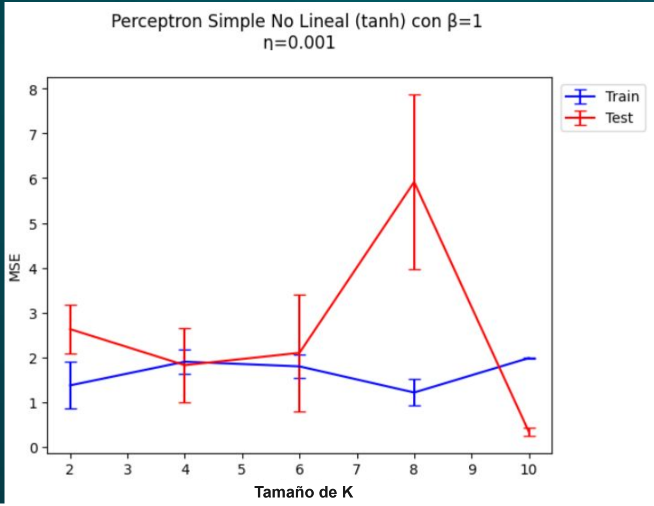
# CROSS K-VALIDATION

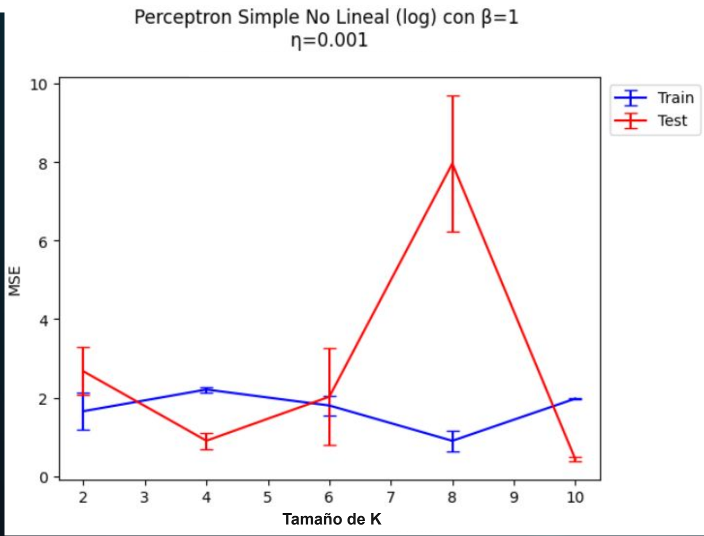
También se contempló el uso del método validación K-cruzada para testear la capacidad de generalización.

En el mismo dividimos al conjunto de datos en K subconjuntos de igual tamaño, y luego entrenamos K veces el modelo utilizando uno de los subconjuntos para testeo y los demás para entrenamiento.

De esta forma se maximizan el conjunto de testeo y entrenamiento, y al finalizar utilizamos un promedio ponderado del MSE para seleccionar los valores del perceptrón más aptos para resolver el problema



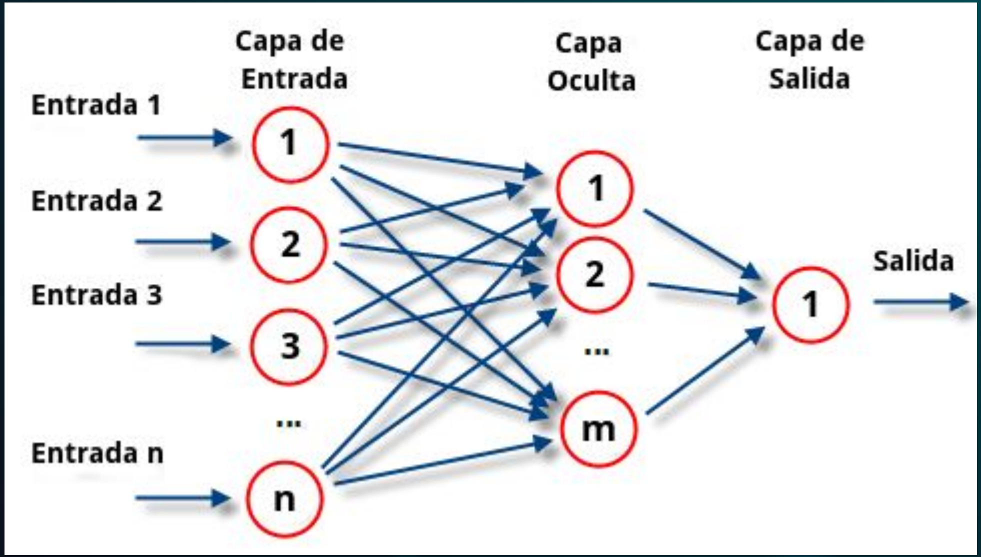




Podemos notar que los mejores valores de K a utilizar son 4 o 6

# Perceptrón multicapa

Utilizar el algoritmo para aprender sobre la función XOR, discriminar si un número es par y determinar qué dígito se corresponde con la entrada a la red



# Salidas de las neuronas

## Forward Propagation

### Primera capa intermedia

$$V_j^1 = \theta\left(\sum_{k=0} x_k^{\mu} * w_{jk}^1\right)$$

### Capas intermedias

$$V_j^m = \theta\left(\sum_{k=1} V_k^{m-1} * w_{jk}^m\right)$$

### Capa de Salida

$$O_i = \theta\left(\sum_{j=0} V_j^{M-1} * W_{ik}\right)$$

# Actualización de pesos

Para este perceptrón se utiliza el algoritmo de backpropagation. Durante la fase de retropropagación del error se utiliza el delta para calcular cómo se deben ajustar los pesos de las conexiones en la red neuronal.

$$\delta_i^M = (\zeta_i - O_i)\theta'(h_i^M)$$
$$\Delta W_{ij} = \eta \delta_i^M V_j^{M-1}$$

$$\delta_j^m = (\textstyle\sum_i \delta_I^{m+1} w_{ij}^{m+1})\theta'(h_j^m)$$
$$\Delta w_{jk}^m = \eta \delta_j^m V_k^{m-1}$$

# Métodos de optimización

Los métodos de optimización se utilizan para mejorar el rendimiento de los algoritmos de aprendizaje, incluyendo el perceptrón. De esta forma ayudan a converger más rápido y no caer en mínimos locales y/o quedarse oscilando en torno al mínimo e incluso divergir.

## Momentum

Se encarga de acelerar el proceso de aprendizaje, conservando el momento del movimiento anterior que hicimos en los pesos y saber si empezamos a oscilar. De esta manera, el modelo puede moverse más suavemente hacia el mínimo global de la función, en lugar de oscilar alrededor de él.

ALPHA = 0.8

## Adam

Es una combinación del método de optimización RMSProp y Momentum muy usado en la práctica.
Su implementación es un poco compleja y fue vista detalladamente en las clases.

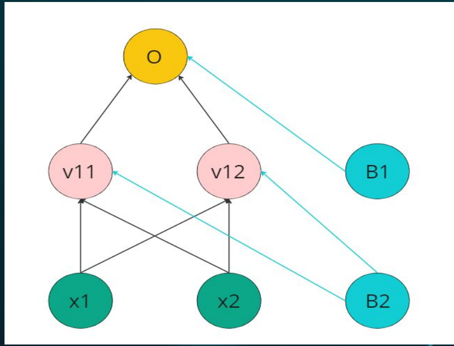
BETA1 = 0.9
BETA2 = 0.999
EPSILON = 1e-8

# Operación XOR

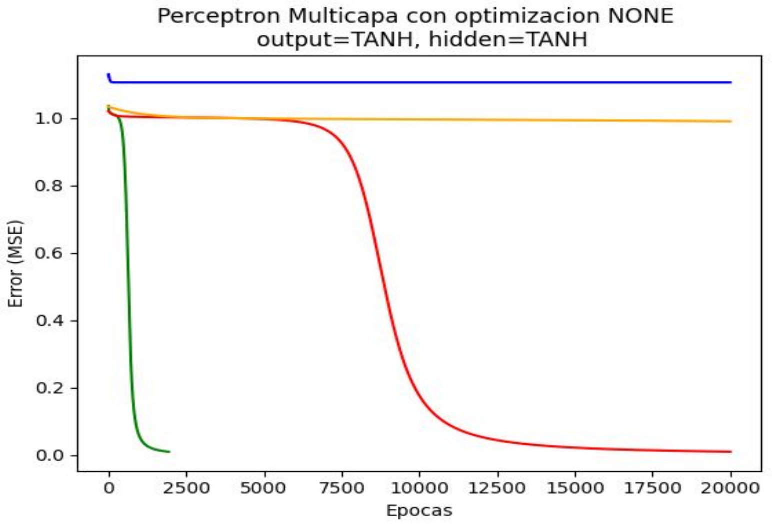
- Recordemos que el problema no era linealmente separable.
- Ahora con un perceptrón multicapa, si podremos resolverlo.
- Entrenamos y testeamos con todo el conjunto de datos (pues nuestro dataset es pequeño)
- Arquitectura planteada:
  - Capa de entrada de 2 nodos + 1 bias
  - Capa oculta de 4 respectivamente + 1 bias
  - Capa de salida con 1 nodo

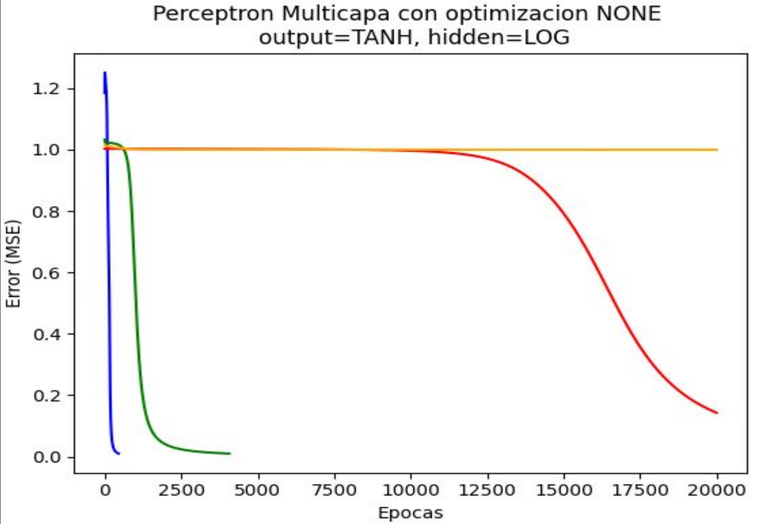
**XOR**

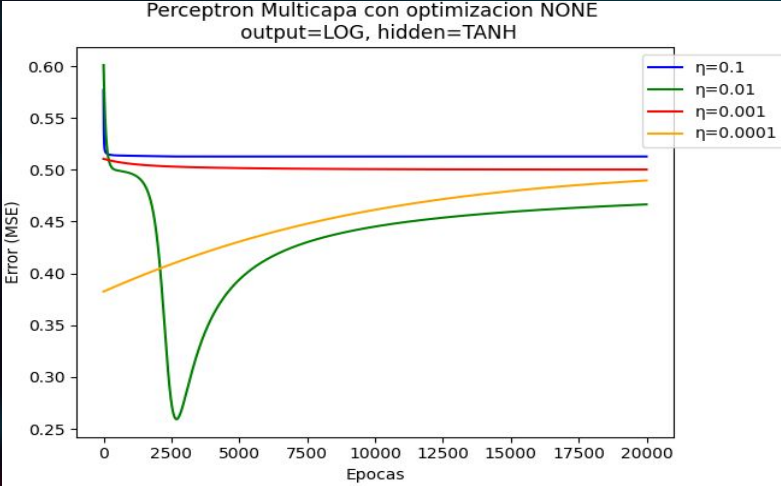
| A | B | Expected |
|---|---|---|
| -1 | 1 | 1 |
| 1 | -1 | 1 |
| 1 | 1 | -1 |
| -1 | -1 | -1 |

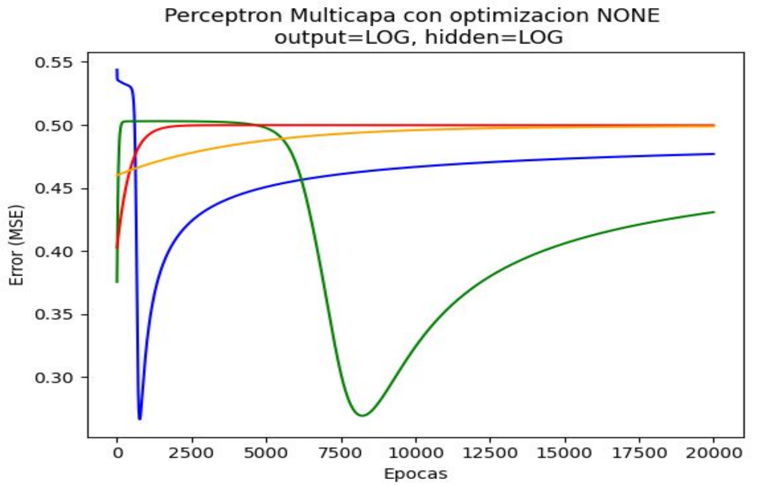


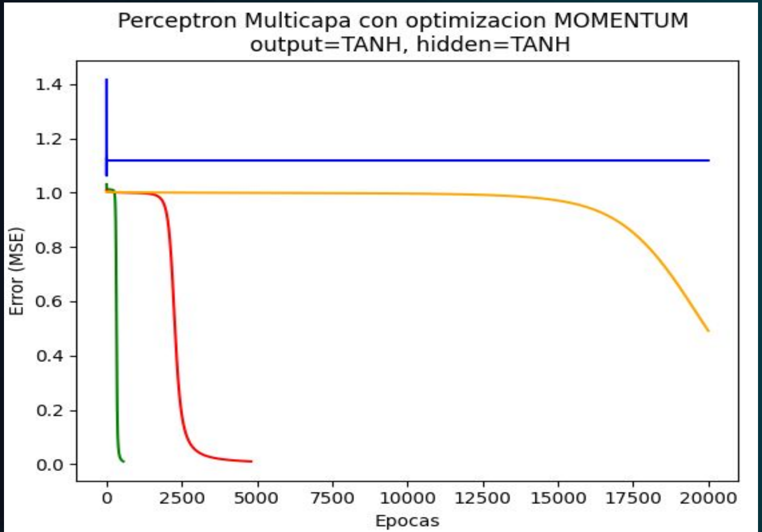
# Variando la tasa de aprendizaje con distintos métodos de activación y optimización

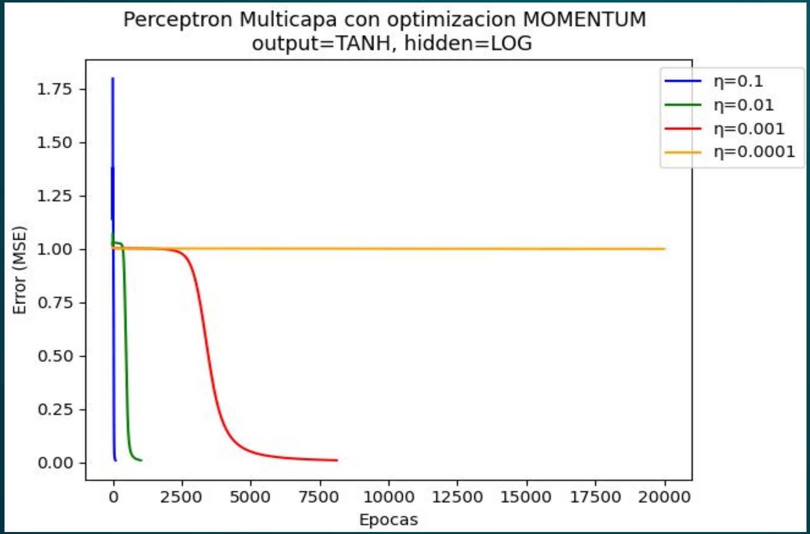

Perceptron Multicapa con optimizacion NONE
output=TANH, hidden=TANH
Error (MSE)
Epocas
0.0 0.2 0.4 0.6 0.8 1.0
0 2500 5000 7500 10000 12500 15000 17500 20000

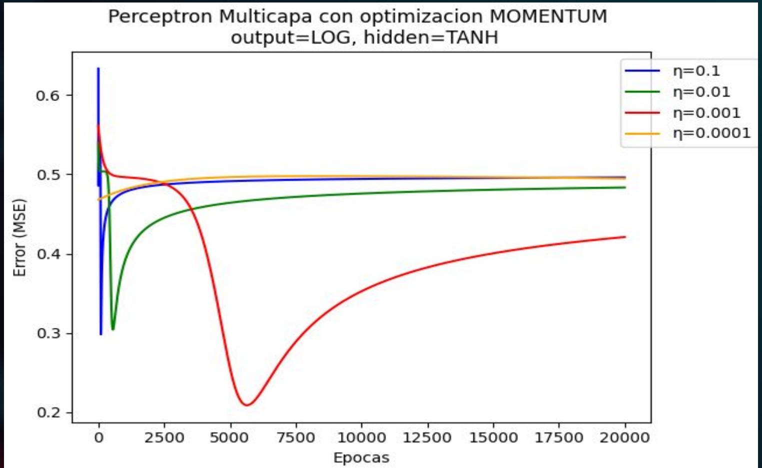

Perceptron Multicapa con optimizacion NONE
output=TANH, hidden=LOG
Error (MSE)
Epocas
0.0 0.2 0.4 0.6 0.8 1.0 1.2
0 2500 5000 7500 10000 12500 15000 17500 20000

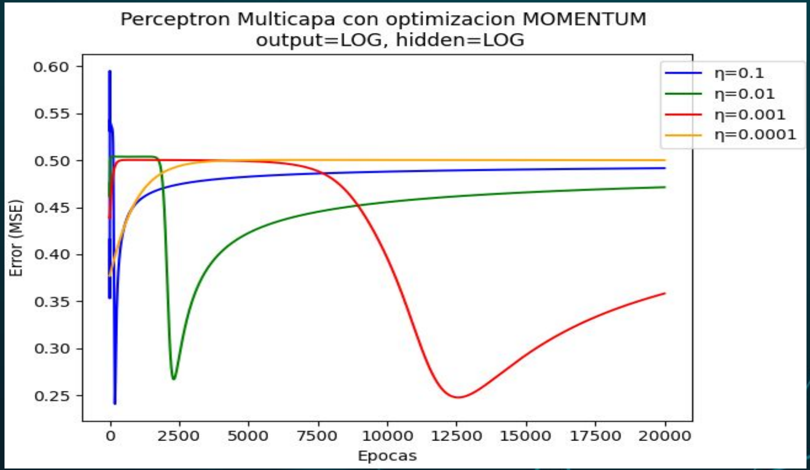

Perceptron Multicapa con optimizacion NONE
output=LOG, hidden=TANH
η=0.1
η=0.01
η=0.001
η=0.0001
Error (MSE)
Epocas
0.25 0.30 0.35 0.40 0.45 0.50 0.55 0.60
0 2500 5000 7500 10000 12500 15000 17500 20000

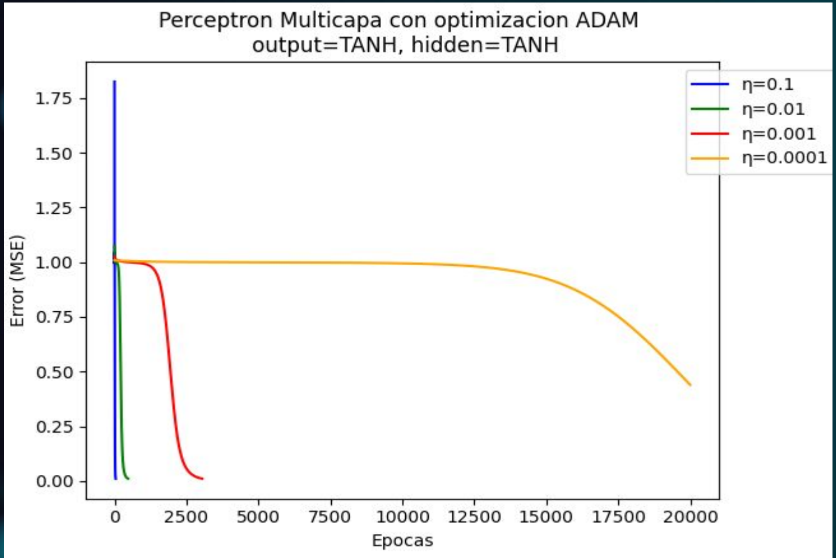

Perceptron Multicapa con optimizacion NONE
output=LOG, hidden=LOG
Error (MSE)
Epocas
0.30 0.35 0.40 0.45 0.50 0.55
0 2500 5000 7500 10000 12500 15000 17500 20000

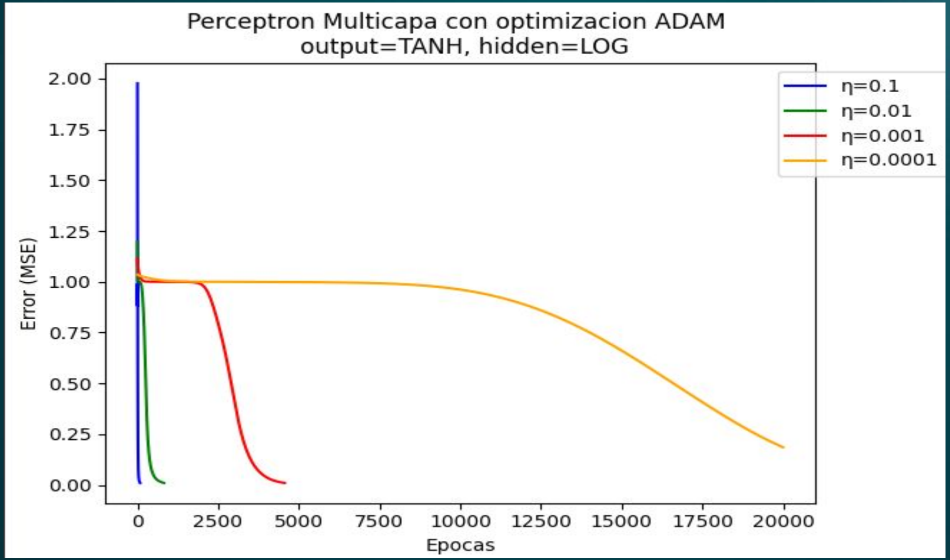

Perceptron Multicapa con optimizacion MOMENTUM
output=TANH, hidden=TANH
Error (MSE)
Epocas
1.4
1.2
1.0
0.8
0.6
0.4
0.2
0.0
0
2500
5000
7500
10000
12500
15000
17500
20000

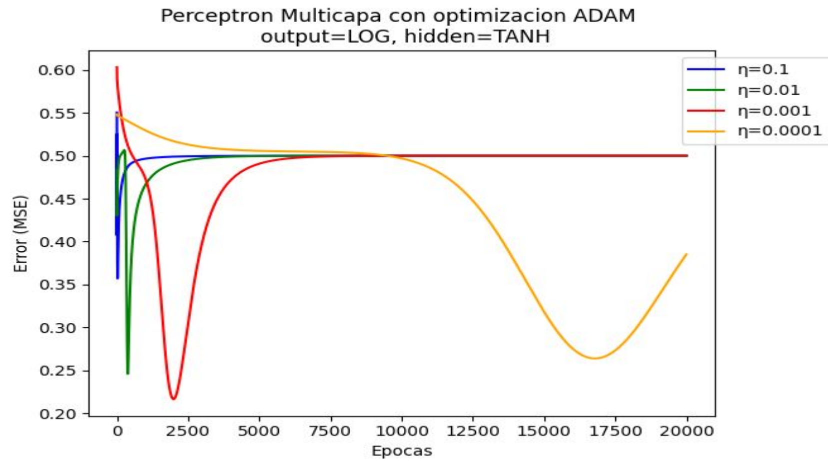

Perceptron Multicapa con optimizacion MOMENTUM
output=TANH, hidden=LOG
η=0.1
η=0.01
η=0.001
η=0.0001
Error (MSE)
Epocas
1.75
1.50
1.25
1.00
0.75
0.50
0.25
0.00
0
2500
5000
7500
10000
12500
15000
17500
20000

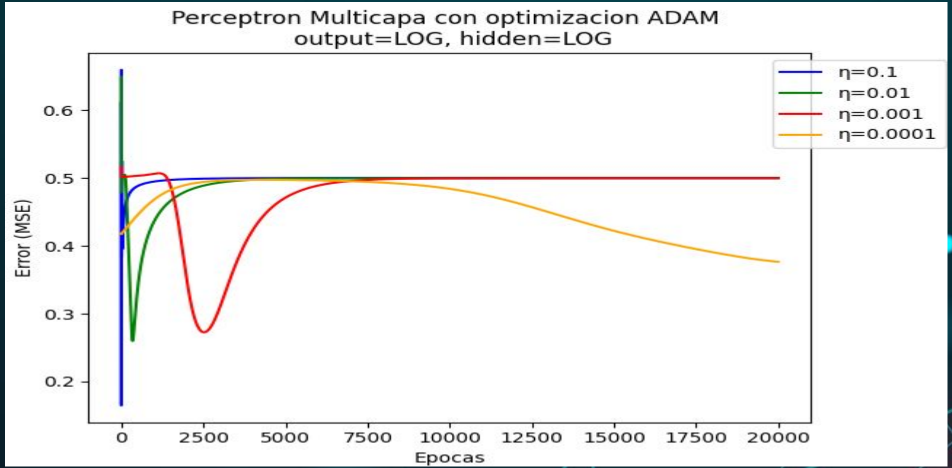

Perceptron Multicapa con optimizacion MOMENTUM
output=LOG, hidden=TANH
η=0.1
η=0.01
η=0.001
η=0.0001
Error (MSE)
Epocas
0.6
0.5
0.4
0.3
0.2
0
2500
5000
7500
10000
12500
15000
17500
20000


Perceptron Multicapa con optimizacion MOMENTUM
output=LOG, hidden=LOG
η=0.1
η=0.01
η=0.001
η=0.0001
Error (MSE)
Epocas
0.60
0.55
0.50
0.45
0.40
0.35
0.30
0.25
0
2500
5000
7500
10000
12500
15000
17500
20000


Perceptron Multicapa con optimizacion ADAM
output=TANH, hidden=TANH
η=0.1
η=0.01
η=0.001
η=0.0001
Error (MSE)
0.00
0.25
0.50
0.75
1.00
1.25
1.50
1.75
0
2500
5000
7500
10000
12500
15000
17500
20000
Epocas


Perceptron Multicapa con optimizacion ADAM
output=TANH, hidden=LOG
η=0.1
η=0.01
η=0.001
η=0.0001
Error (MSE)
0.00
0.25
0.50
0.75
1.00
1.25
1.50
1.75
2.00
0
2500
5000
7500
10000
12500
15000
17500
20000
Epocas


Perceptron Multicapa con optimizacion ADAM
output=LOG, hidden=TANH
η=0.1
η=0.01
η=0.001
η=0.0001
Error (MSE)
0.20
0.25
0.30
0.35
0.40
0.45
0.50
0.55
0.60
0
2500
5000
7500
10000
12500
15000
17500
20000
Epocas


Perceptron Multicapa con optimizacion ADAM
output=LOG, hidden=LOG
η=0.1
η=0.01
η=0.001
η=0.0001
Error (MSE)
0.2
0.3
0.4
0.5
0.6
0
2500
5000
7500
10000
12500
15000
17500
20000
Epocas

- Tal como dijimos anteriormente, a diferencia del perceptrón simple escalón, el perceptrón multicapa es capaz de resolver el problema del XOR

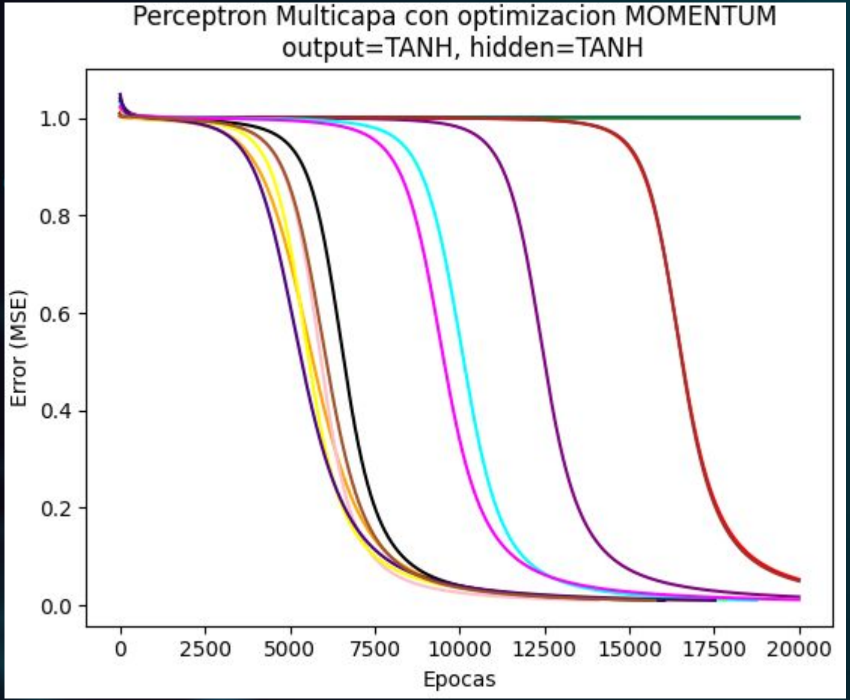
- En los casos que se utiliza la función de activación sigmoidea logística se puede ver que a partir de cierto punto el error comienza a aumentar. Esto se debe a que, debido a la forma de la función, puede ocurrir que los gradientes desaparezcan o "exploten" durante el entrenamiento.

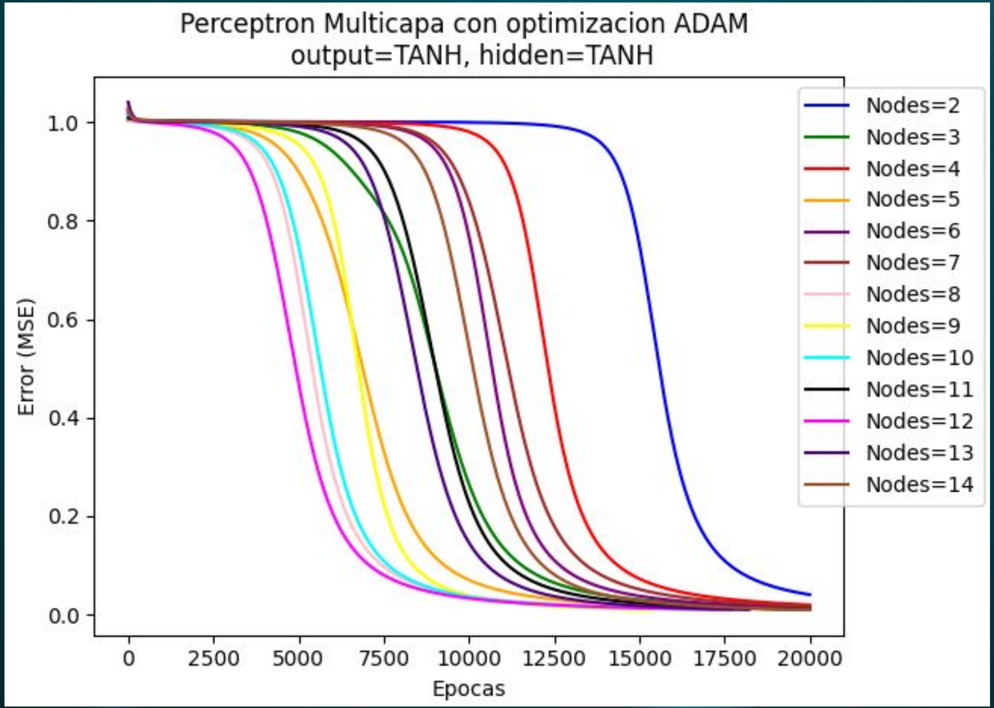
- Podemos notar también que el uso de un método de optimización mejora en la convergencia de aprendizaje del perceptrón, siendo Adam el mejor método.

- Los mejores valores de tasa de aprendizaje son 0.01 o 0.001

# Variando la cantidad de nodos de la capa oculta

- tanh como activación tanto de capa de salida como capas ocultas
- ADAM como optimización
- $\eta = 0.001$

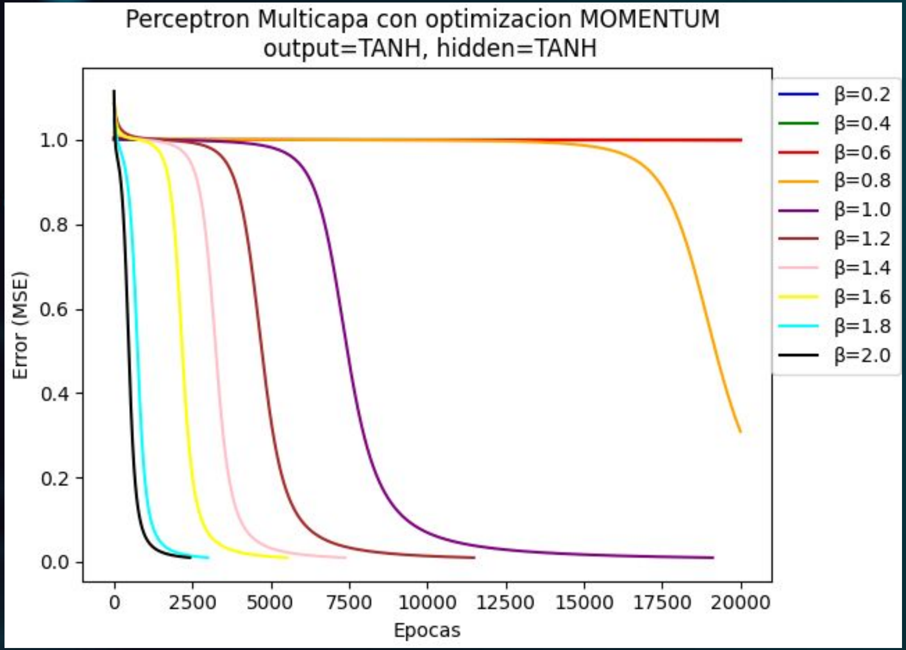


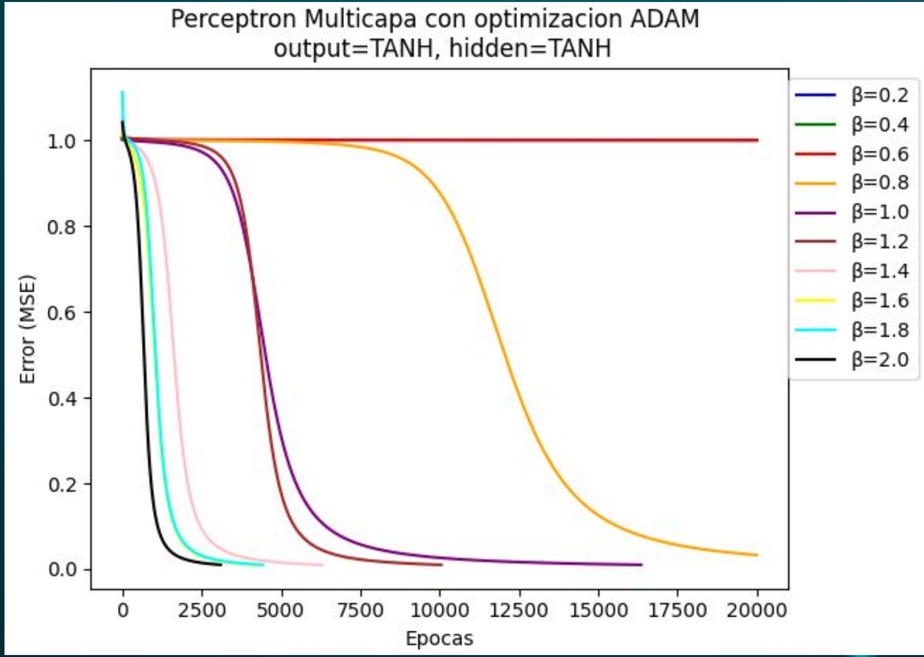




Para este problema podemos notar que a medida que se aumenta la cantidad de nodos de la única capa oculta del perceptrón, se logra encontrar la solución más rápido.

# Variando el valor de beta de la función de activación

**η = 0.001**




Podemos notar que a medida que se aumenta el valor de beta de la función de activación, se logra encontrar la solución más rápido.

# Paridad de dígitos

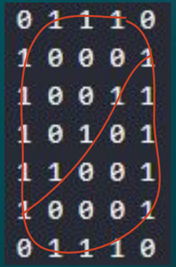
Queremos que dada una entrada de números representados por “imágenes” de 7x5 píxeles, entrenar a nuestro perceptrón multicapa para que prediga la paridad de cada número. Es decir, nuestro dataset será el siguiente:

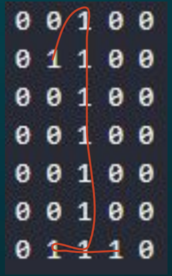
- Input: mapa de bits de 7x5
  - Se “aplana” dicho mapa para así obtener un total de 10 muestras, representados por un vector de 35 xi
- Output: 1 si el número es par, -1 si es impar

La arquitectura a utilizar será:

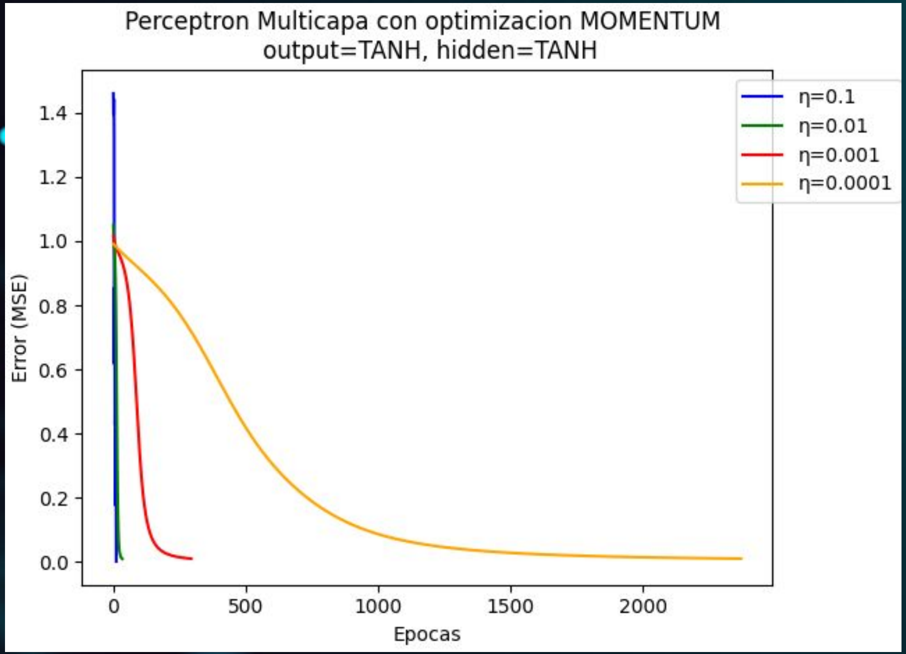
- Capa de entrada de 35 nodos + 1 bias
- 2 capas oculta de 16 y 10 nodos respectivamente + 1 bias en cada capa
- Capa de salida con 1 nodo

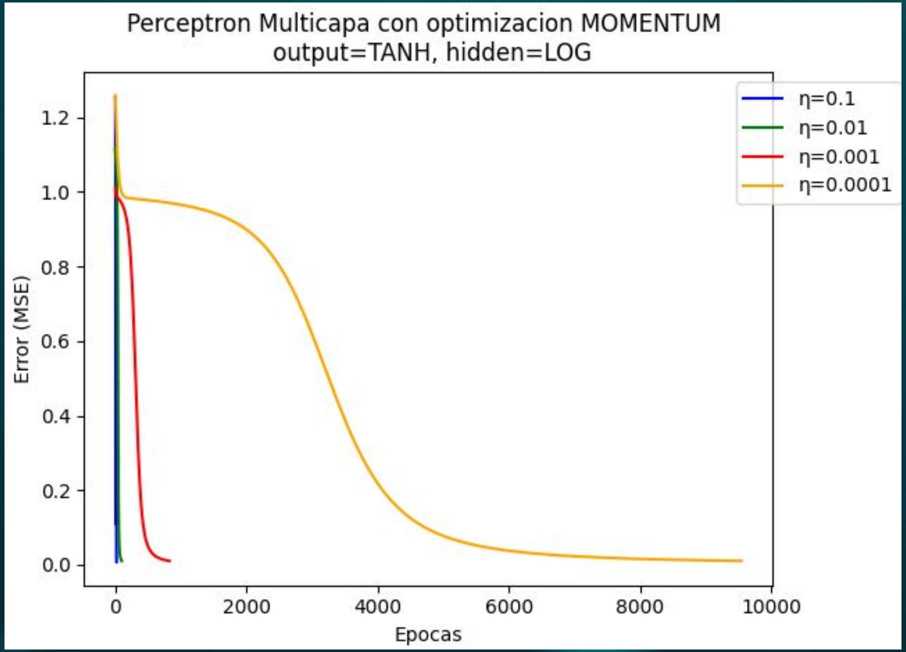
***__Regla de los ⅔:__*** cantidad de nodos de capas ocultas para un problema simple ~ ⅔ del tamaño del input + el tamaño de salida

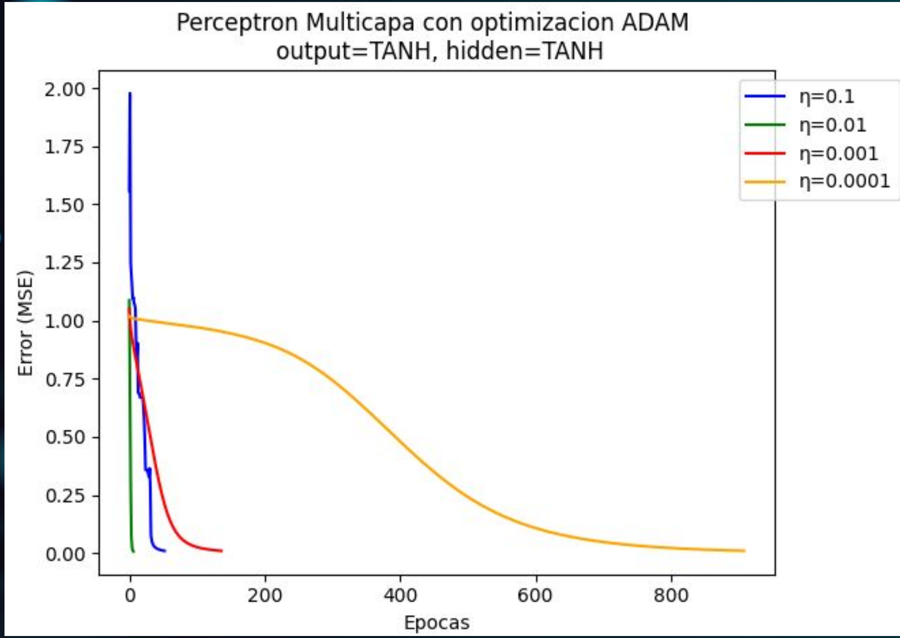


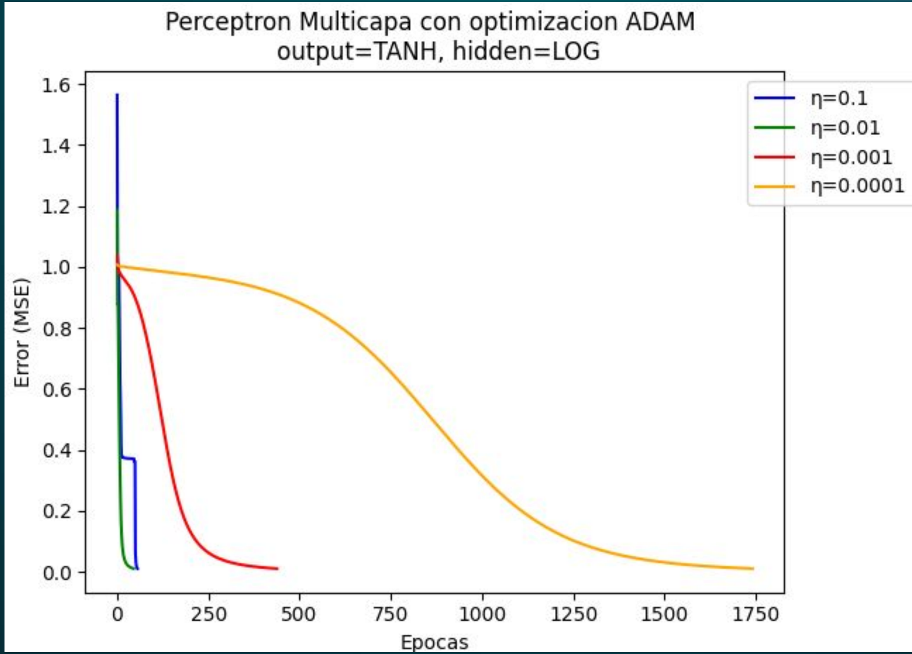



# Variando la tasa de aprendizaje con distintos métodos de activación y optimización


Perceptron Multicapa con optimizacion MOMENTUM
output=TANH, hidden=TANH
η=0.1
η=0.01
η=0.001
η=0.0001
Error (MSE)
1.4
1.2
1.0
0.8
0.6
0.4
0.2
0.0
0
500
1000
1500
2000
Epocas


Perceptron Multicapa con optimizacion MOMENTUM
output=TANH, hidden=LOG
η=0.1
η=0.01
η=0.001
η=0.0001
Error (MSE)
1.2
1.0
0.8
0.6
0.4
0.2
0.0
0
2000
4000
6000
8000
10000
Epocas



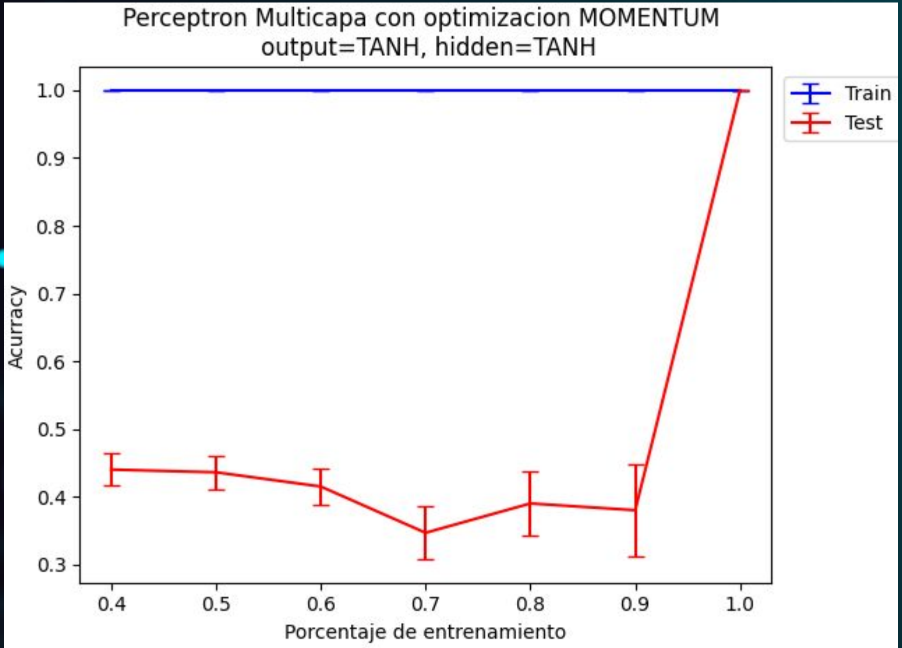



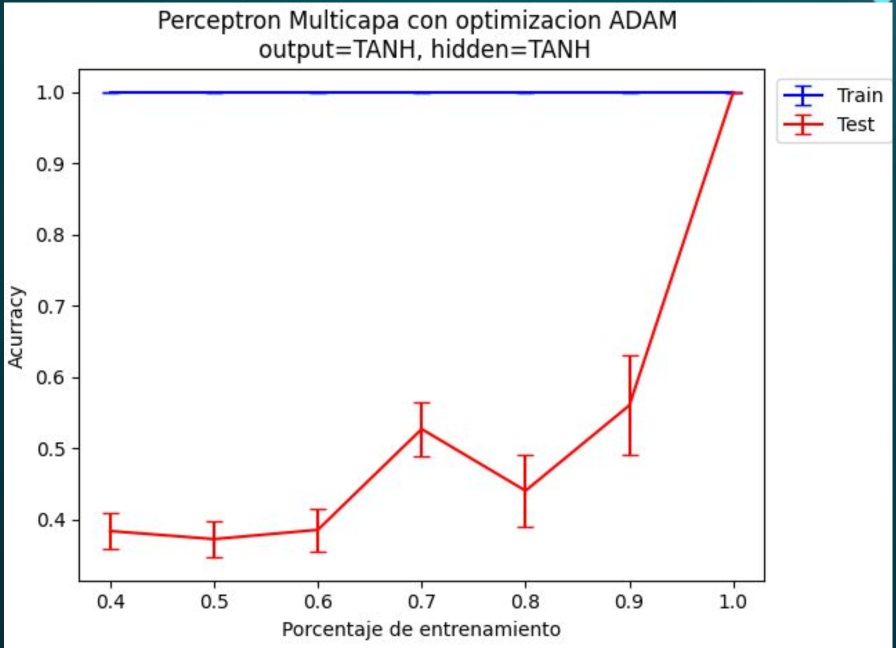

- Podemos observar que el uso de la función tanh para ambas activaciones tiene mejor performance que combinándola con la función logística
- También observamos que el método de optimización Adam converge más rápido a la solución que Momentum

# Capacidad de generalización

## Variando porcentajes de entrenamiento

η = 0.01 Épocas = 50000 Iteraciones = 50 Beta = 1




- Si se entrena con el 100% de los datos, también se testea con el 100% de los datos → accuracy = 1
- Puede comprobarse que la red elegida logró acertar aproximadamente un 50% de las muestras testeadas. Creemos que esto tiene sentido, pues “adivinar” la paridad de un número es lo mismo que la probabilidad de “lanzar una moneda” (50-50), confirmando así que la red no está logrando generalizar y devuelve un valor al azar.
- Como las muestras son de 35 datos, cuando se testea un número que nunca entreno (y no conoce su “forma”) es prácticamente imposible que logre acertar la distribución de 1s y 0s en la imagen de 7x5 para conocer la paridad.

# Distinguir un número

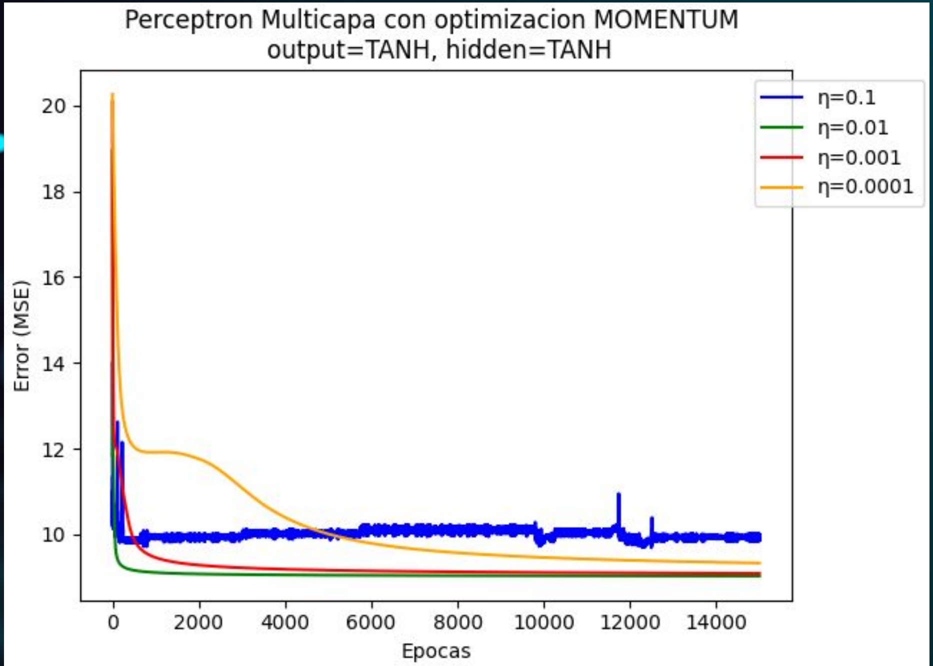
Nuestro dataset es análogo al del problema anterior, pero ahora la arquitectura propuesta será:

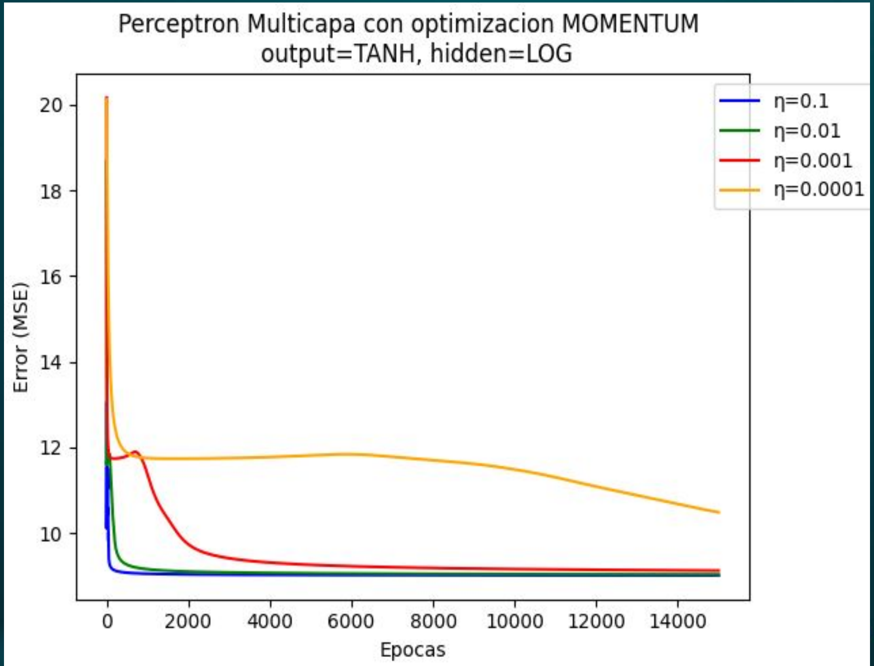
- Capa de entrada de 35 nodos + 1 bias
- 2 capas oculta de 25 y 15 nodos respectivamente + 1 bias en cada capa
- Capa de salida con 10 nodos

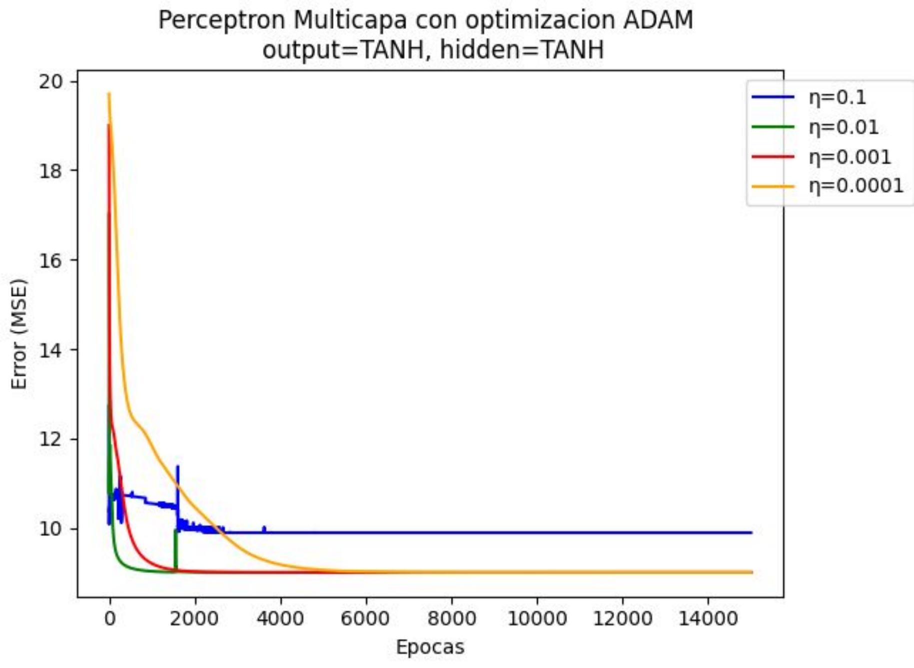
Se decidió agregar una capa oculta más para agregarle mayor poder computacional al perceptrón.

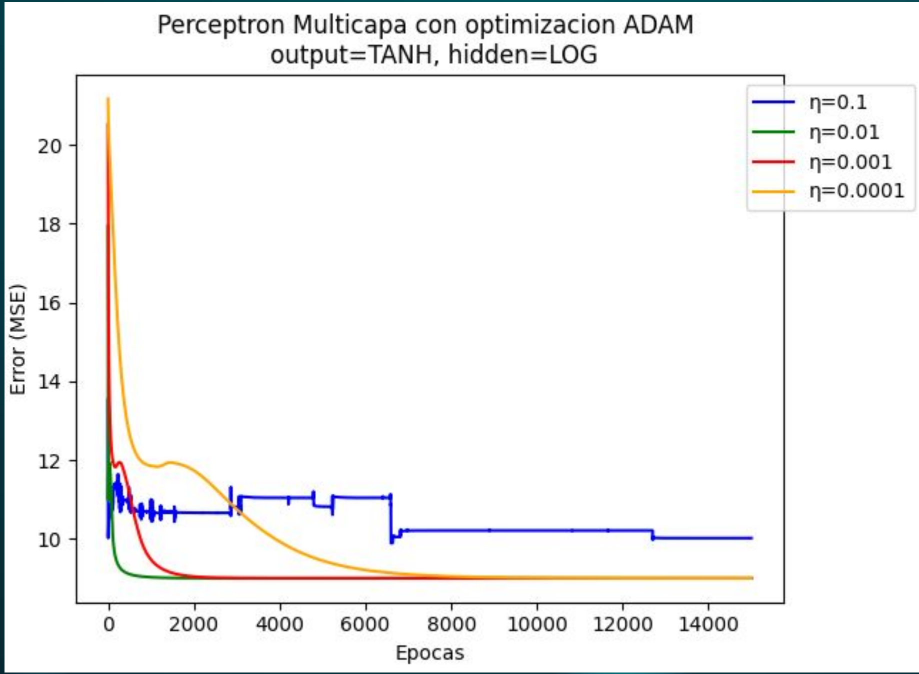
***Regla de los ⅔:*** cantidad de nodos de capas ocultas para un problema simple ~ ⅔ del tamaño del input + el tamaño de salida

# Variando la tasa de aprendizaje con distintos métodos de activación y optimización


Perceptron Multicapa con optimizacion MOMENTUM
output=TANH, hidden=TANH
η=0.1
η=0.01
η=0.001
η=0.0001
Error (MSE)
Epocas
20
18
16
14
12
10
0
2000
4000
6000
8000
10000
12000
14000

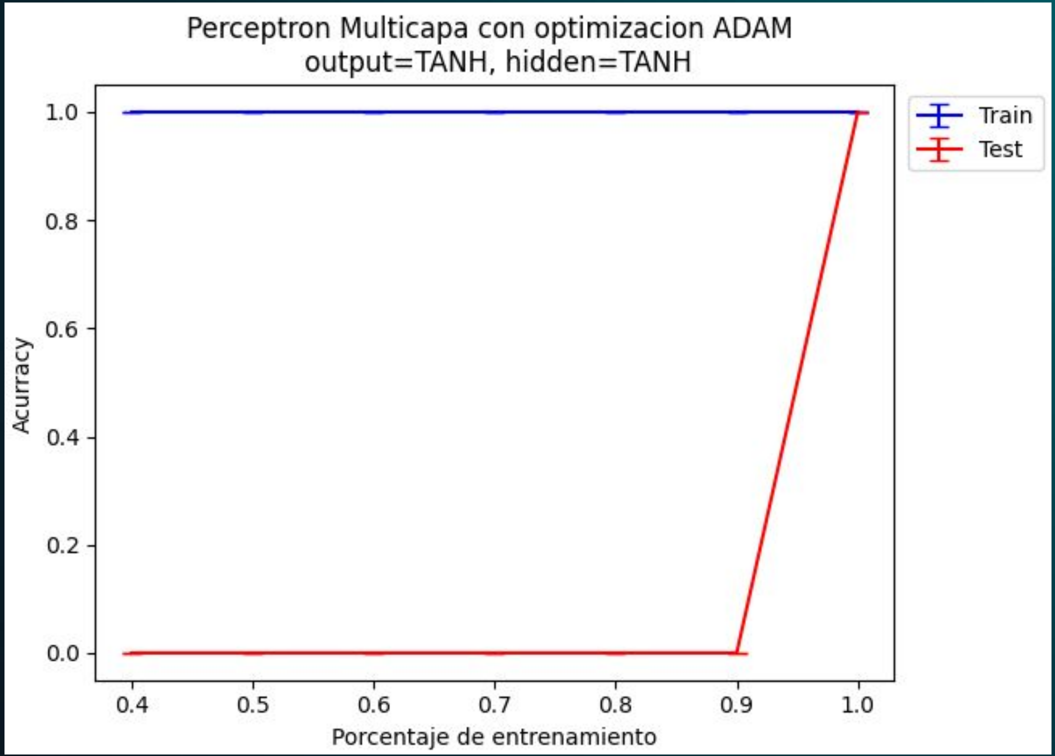

Perceptron Multicapa con optimizacion MOMENTUM
output=TANH, hidden=LOG
η=0.1
η=0.01
η=0.001
η=0.0001
Error (MSE)
Epocas
20
18
16
14
12
10
0
2000
4000
6000
8000
10000
12000
14000


Perceptron Multicapa con optimizacion ADAM
output=TANH, hidden=TANH
Error (MSE)
Epocas
η=0.1
η=0.01
η=0.001
η=0.0001
0
2000
4000
6000
8000
10000
12000
14000
10
12
14
16
18
20


Perceptron Multicapa con optimizacion ADAM
output=TANH, hidden=LOG
Error (MSE)
Epocas
η=0.1
η=0.01
η=0.001
η=0.0001
0
2000
4000
6000
8000
10000
12000
14000
10
12
14
16
18
20

# Capacidad de generalización

## Variando porcentajes de entrenamiento

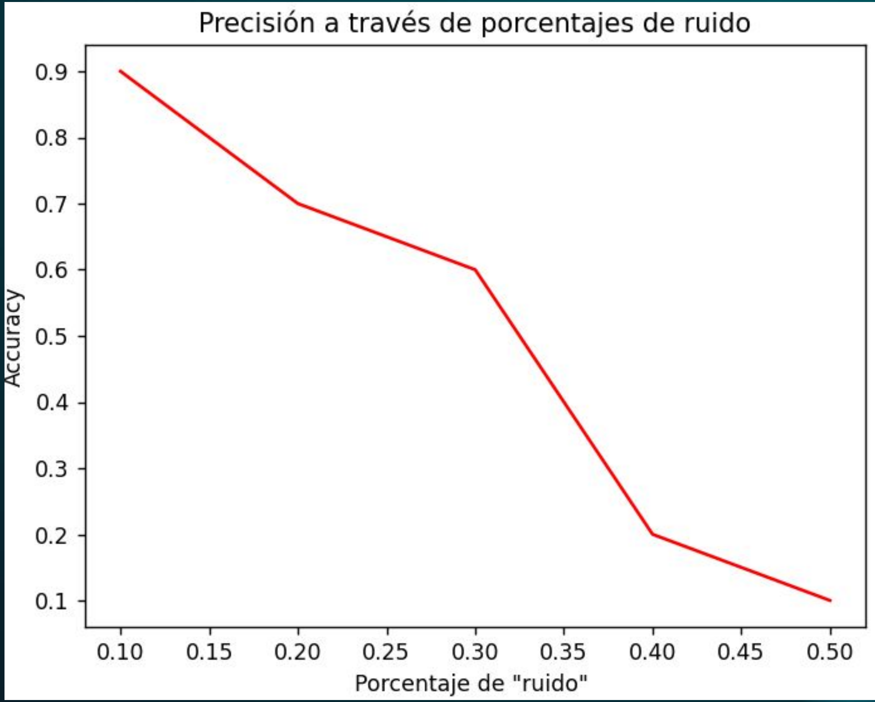



- Si se entrena con el 100% de los datos, también se testea con el 100% de los datos → accuracy = 1
- Como las muestras son de 35 datos, cuando se testea un número que nunca entreno (y no conoce su "forma") es prácticamente imposible que logre acertar la distribución de 1s y 0s correspondiente que matcheen con la imagen de 7x5 para saber de qué número se trata.

90%-10% (train-test)   η = 0.00   50000 epocas

```
Finished Training.
 MSE: 9.012157328601814
Expected 3.  Guess 8
Test Accuracy: 0.0
Test MSE = 12.976815796855877
---------------Mutacion=0.1--------
Expected 0.  Guess 0
Expected 1.  Guess 1
Expected 2.  Guess 2
Expected 3.  Guess 8
Expected 4.  Guess 4
Expected 5.  Guess 5
Expected 6.  Guess 6
Expected 7.  Guess 7
Expected 8.  Guess 8
Expected 9.  Guess 9
Accuracy = 0.9
---------------Mutacion=0.2--------
Expected 0.  Guess 0
Expected 1.  Guess 1
Expected 2.  Guess 2
Expected 3.  Guess 8
Expected 4.  Guess 4
Expected 5.  Guess 5
Expected 6.  Guess 6
Expected 7.  Guess 7
Expected 8.  Guess 9
Expected 9.  Guess 9
Accuracy = 0.8
---------------Mutacion=0.3--------
Expected 0.  Guess 0
Expected 1.  Guess 4
Expected 2.  Guess 2
Expected 3.  Guess 2
Expected 4.  Guess 4
Expected 5.  Guess 5
Expected 6.  Guess 6
Expected 7.  Guess 7
Expected 8.  Guess 4
Expected 9.  Guess 9
Accuracy = 0.7
```

**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**



# ¡Gracias!