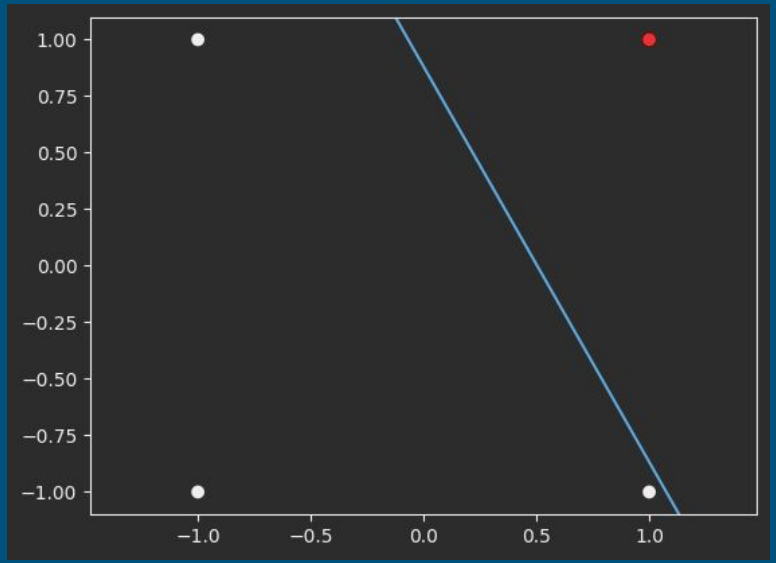
SIA - TP3

Perceptrones

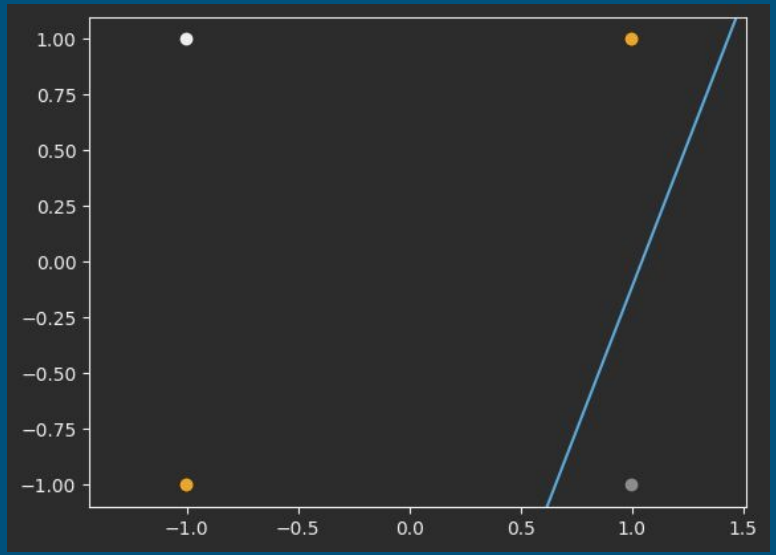
Ejercicio 1

Perceptrón Escalón

Dataset AND y XOR



AND



XOR

Conclusiones y aprendizajes

- AND es linealmente separable
- XOR no es linealmente separable
- XOR requiere al menos dos líneas rectas para separar las dos clases de puntos. Por lo tanto, un perceptrón escalón no es suficiente para resolver este problema.

Ejercicio 2

Perceptrón Lineal y No Lineal

Parámetros Utilizados

parámetro / theta	Lineal	Tanh	Log
Tasa de Aprendizaje	0.001		
Max Épocas	2000	4000	
Error Aceptado	8	0.0005	0.0004
Act. Pesos	batch		
Beta	No Aplica		
Rango de Error	(0, 3782)	(0, 1)	

- Pesos iniciales Aleatorios

¿Cómo comparamos los errores entre thetas?

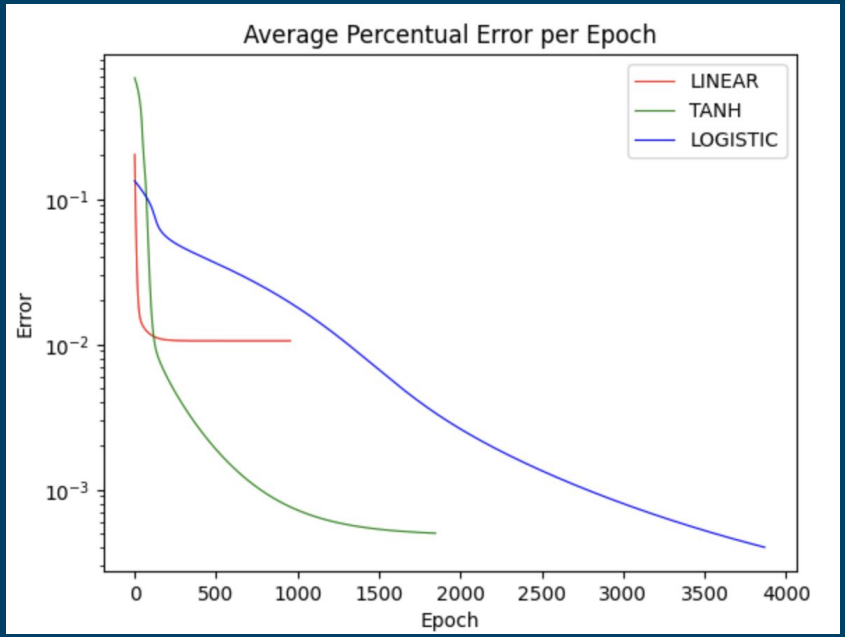
Cómo estamos usando el ECM cómo función de cálculo de error, y además, cada theta tiene un rango de valor de salida distinto, decidimos mapear el error promedio a un porcentaje de error.

Este porcentaje de error está definido dentro de un rango, que tomamos cómo:

(0, Vmax^2 * 0.5)

Siendo este Vmax, el valor máximo de la función en el rango.

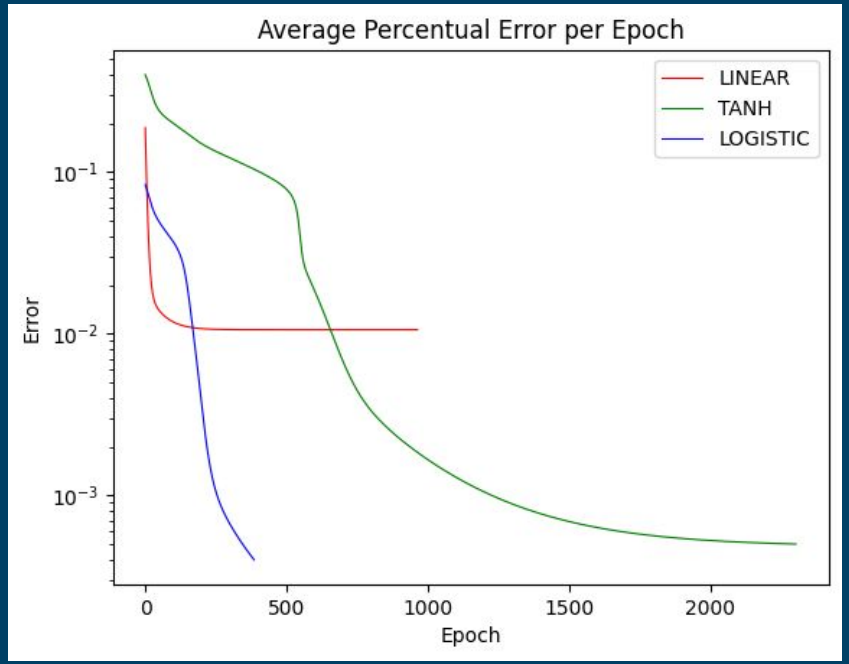
Comparación de Errores Por Época para cada método



En este caso, Log converge más lento que todas

Comparación de Errores Por Época para cada método

En este caso, Tanh converge más lento que todas



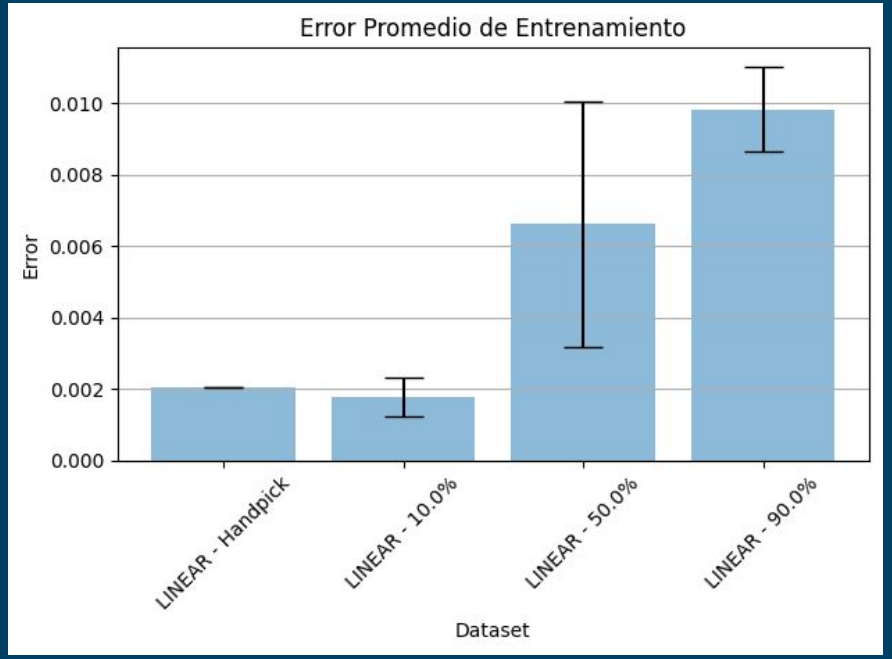
Métodos de Selección de Dataset

- Separación porcentual aleatoria
  - 10% Entrenamiento - 90% Test
  - 50% / 50%
  - 90% / 10%
- Elección Arbitraria - Handpick
  - Se tomaron los siguiente valores arbitrariamente
    - un valor por cada Xi = 0
    - un valor por Y es Máximo
    - un valor por cada Xi es Máximo
    - un valor por cada Xi Mínimo

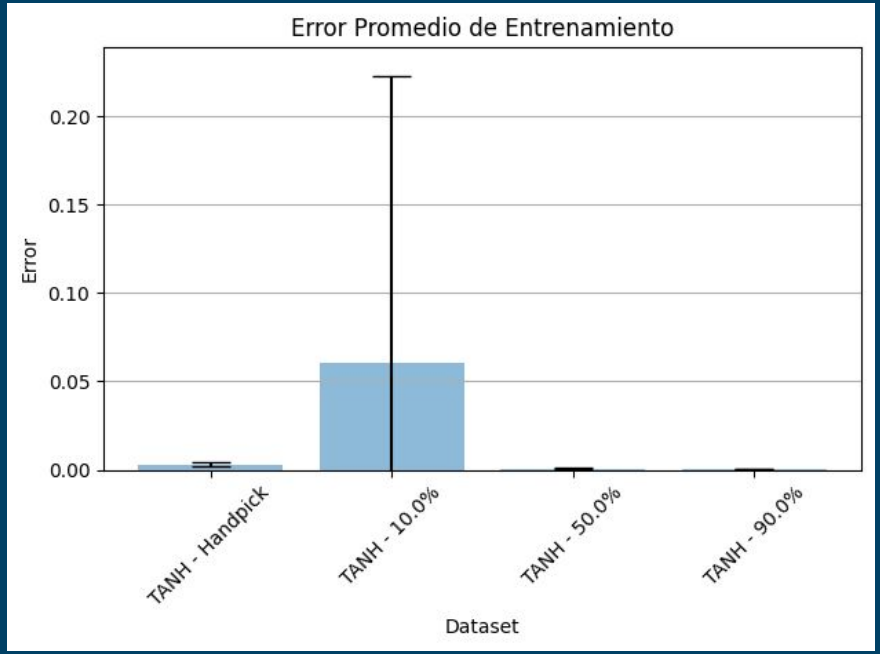
Para cada uno de estos métodos, se realizó la prueba 10 veces.

Capacidad de Aprendizaje

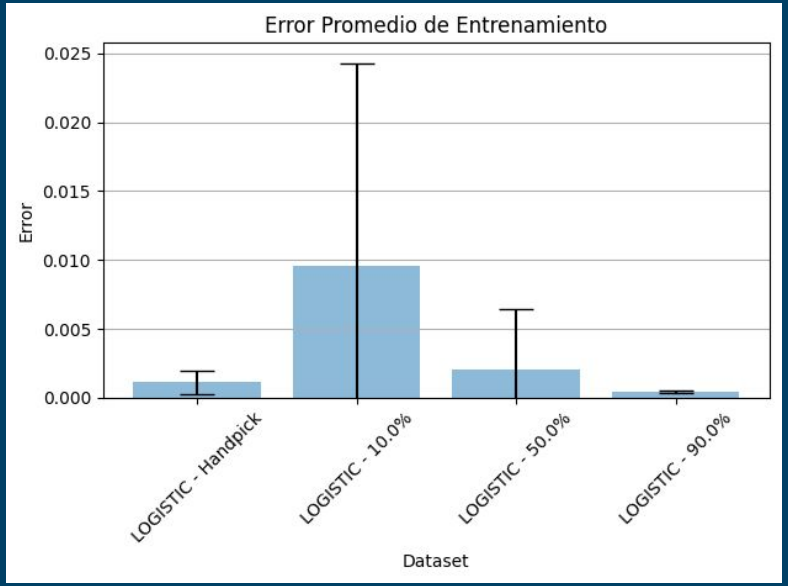
Lineal


Error Promedio de Entrenamiento
Error
0.010
0.008
0.006
0.004
0.002
0.000
LINEAR - Handpick
LINEAR - 10.0%
LINEAR - 50.0%
LINEAR - 90.0%
Dataset

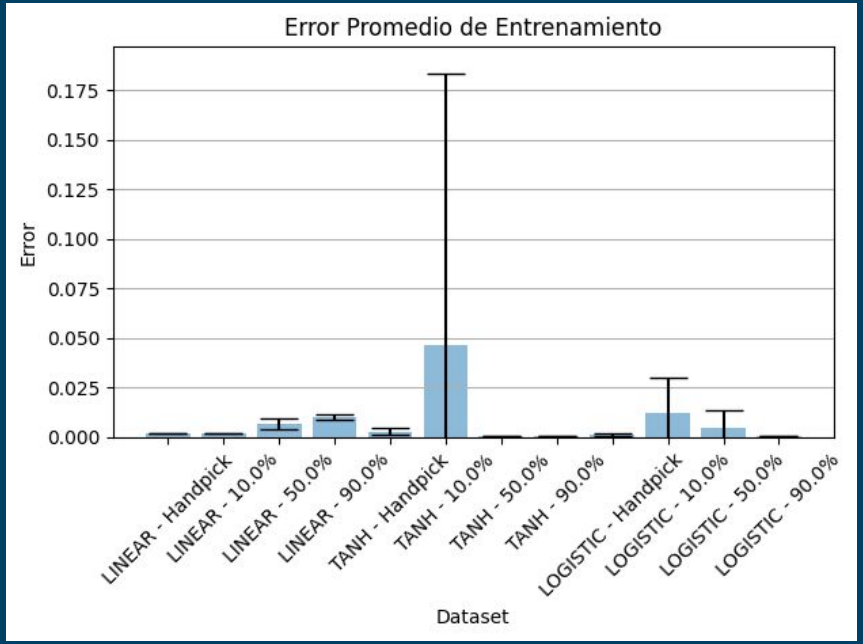
Tanh


Error Promedio de Entrenamiento
Error
0.20
0.15
0.10
0.05
0.00
TANH - Handpick
TANH - 10.0%
TANH - 50.0%
TANH - 90.0%
Dataset

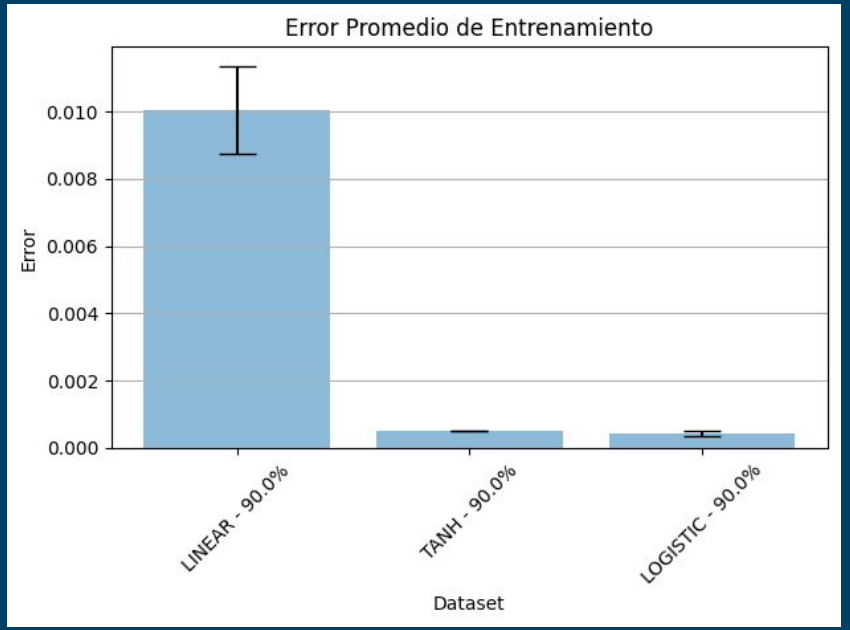
Log


Error Promedio de Entrenamiento
0.025
0.020
0.015
0.010
0.005
0.000
Error
LOGISTIC - Handpick
LOGISTIC - 10.0%
LOGISTIC - 50.0%
LOGISTIC - 90.0%
Dataset

Comparamos todas

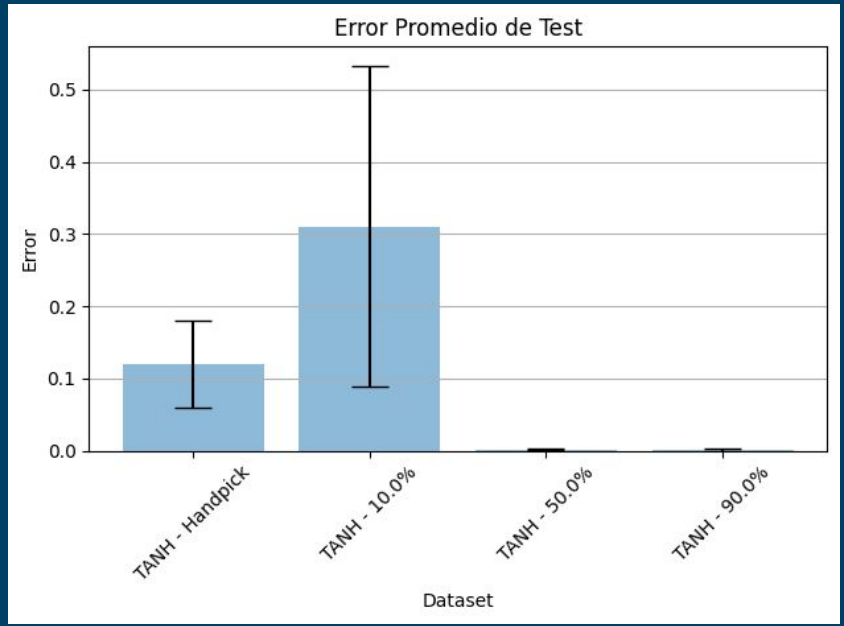


Comparamos la mejor de cada una

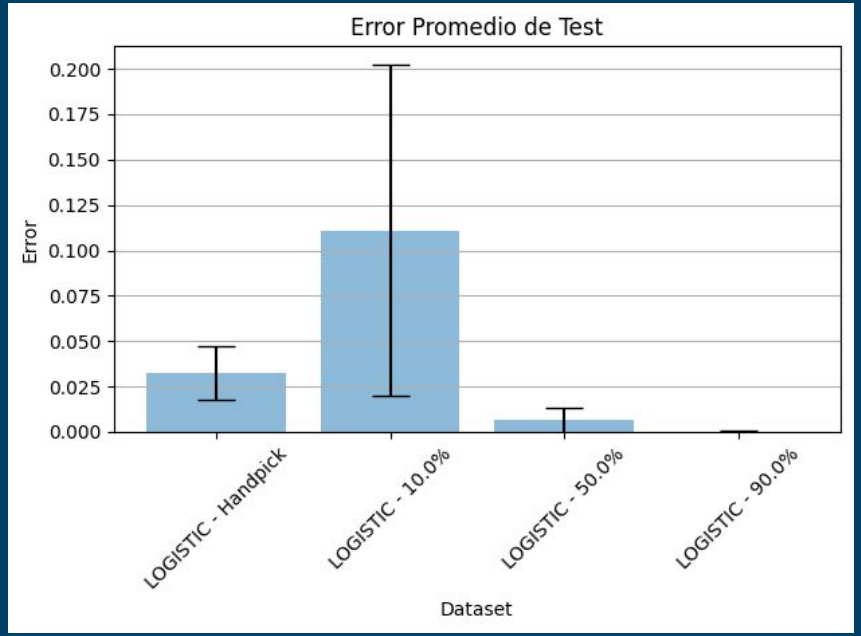


Capacidad de Generalización (no lineales)

Tanh

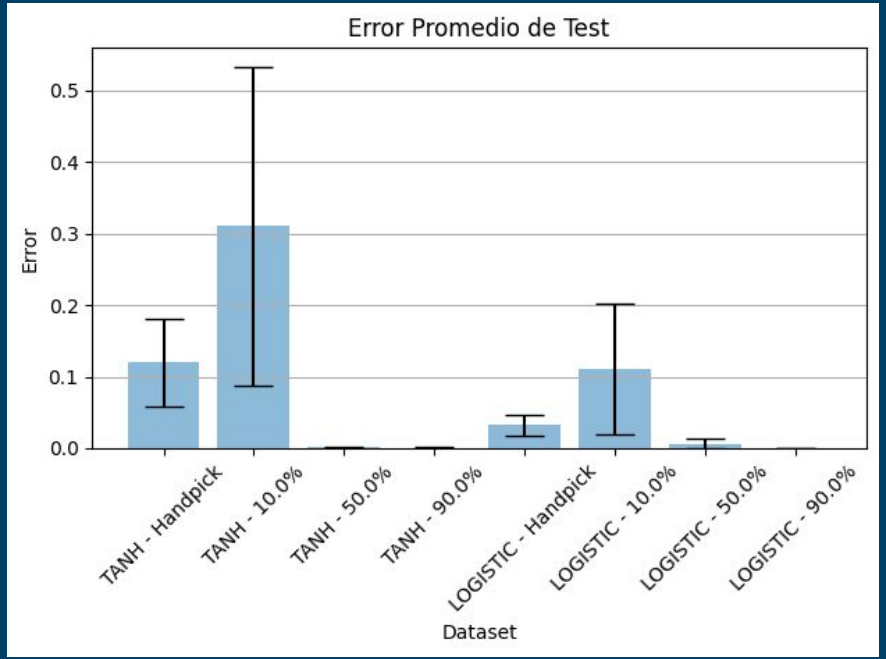

Error Promedio de Test
Error
0.0
0.1
0.2
0.3
0.4
0.5
TANH - Handpick
TANH - 10.0%
TANH - 50.0%
TANH - 90.0%
Dataset

Log

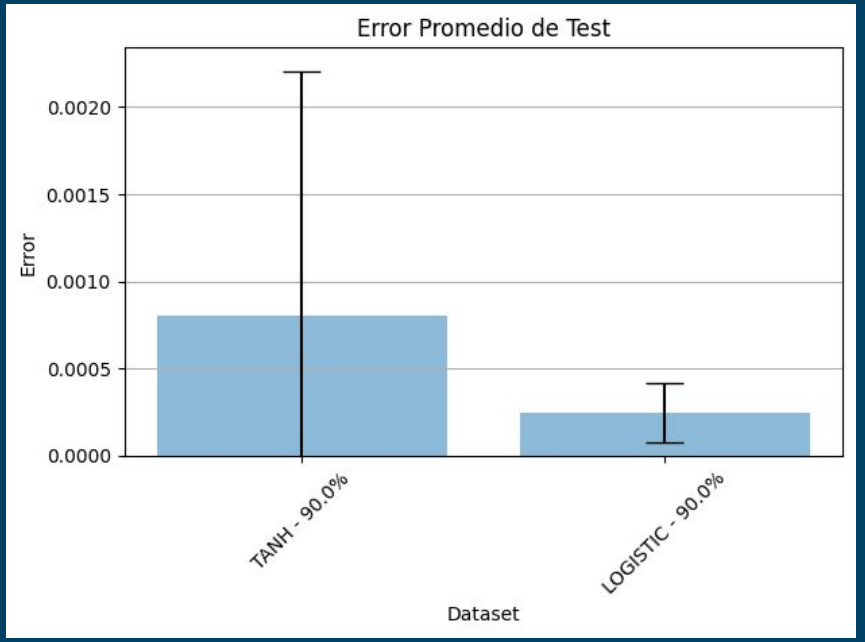

Error Promedio de Test
Error
0.200
0.175
0.150
0.125
0.100
0.075
0.050
0.025
0.000
LOGISTIC - Handpick
LOGISTIC - 10.0%
LOGISTIC - 50.0%
LOGISTIC - 90.0%
Dataset

Comparamos todas

Log > Tanh



Comparamos la mejor de cada una



Conclusiones y aprendizajes

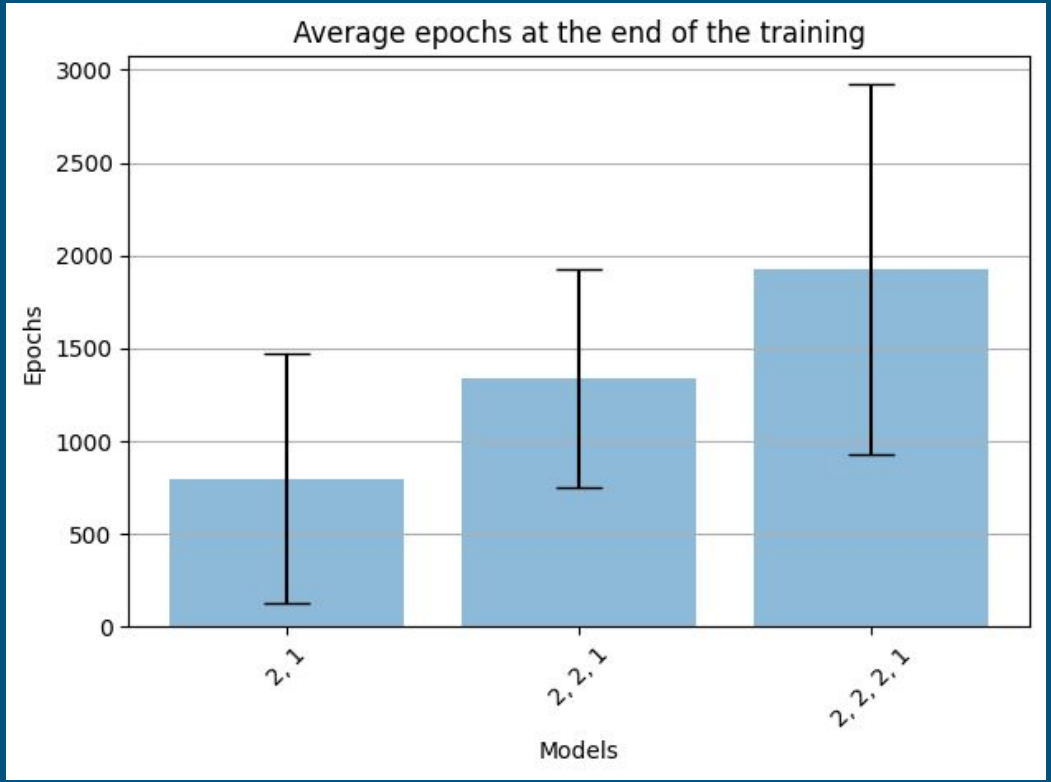
- El método de selección de dataset para aprendizaje y testing tiene una gran influencia en cómo se entrena el perceptrón. Un buen dataset, diverso y representativo, es garantía de mejores resultados.
- Tanto para aprendizaje y generalización, mientras mayor sea la cantidad de datos utilizados a la hora de entrenar, mejores son los resultados.
- La función theta logistic aproxima mejor la solución del problema, cabe destacar que esto puede variar según las características del mismo.
- El perceptrón lineal no llega a aprender lo suficiente con un *eta* pequeño en pocas épocas y cuanto más sea el eta, más rápido se puede llegar a estancar.

Ejercicio 3

Perceptrón Multicapa

Ejercicio 3.A

[2, 1] - [2, 2, 1] - [2, 2, 2, 1]


Average epochs at the end of the training
3000
2500
2000
1500
1000
500
0
Epochs
2, 1
2, 2, 1
2, 2, 2, 1
Models

Conclusiones y aprendizajes

- Se puede resolver el XOR ya que el perceptrón multicapa permite funciones no lineales (además que al tener múltiples capas puede aprender representaciones jerárquicas de los datos)

```
RESULTS AFTER EPOCH 464 (weights [-1.49968628  0.21641105  2.80975208  2.50718888  0.58049203 -0.58184857])
[Data 0, Neuron Output 0] ✅ expected: -1 got: -0.9235676892160725 data: [1 1]
[Data 1, Neuron Output 0] ✅ expected: 1 got: 0.9386993996996635 data: [ 1 -1]
[Data 2, Neuron Output 0] ✅ expected: 1 got: 0.9338647154977358 data: [-1  1]
[Data 3, Neuron Output 0] ✅ expected: -1 got: -0.9227576333956027 data: [-1 -1]
```

Ejercicio 3.B

Output

```
[Data 0, Neuron Output 0] ❌ expected: 1 got: 0.9292615603690724 data: [0, 1, 1, 1, 0, 1, 0, 0, 0, 1, 1, 0, 0, 1, 1, 1, 0, 1, 0, 1, 1, 1, 0, 0, 1, 1, 0, 0, 0, 1, 0, 1, 1, 1, 0]
[Data 1, Neuron Output 0] ❌ expected: -1 got: -0.987441216258414 data: [0, 0, 1, 0, 0, 0, 1, 1, 0, 0, 0, 0, 1, 0, 0, 0, 0, 1, 0, 0, 0, 0, 1, 0, 0, 0, 0, 1, 0, 0, 0, 1, 1, 1, 0]
[Data 2, Neuron Output 0] ❌ expected: 1 got: 0.8621120338676861 data: [0, 1, 1, 1, 0, 1, 0, 0, 0, 1, 0, 0, 0, 0, 1, 0, 0, 0, 1, 0, 0, 0, 1, 0, 0, 0, 1, 0, 0, 0, 1, 1, 1, 1, 1]
[Data 3, Neuron Output 0] ❌ expected: -1 got: -0.9064458029618984 data: [0, 1, 1, 1, 0, 1, 0, 0, 0, 1, 0, 0, 0, 0, 1, 0, 0, 1, 1, 0, 0, 0, 0, 0, 1, 1, 0, 0, 0, 1, 0, 1, 1, 1, 0
[Data 4, Neuron Output 0] ❌ expected: 1 got: 0.9489525374363017 data: [0, 0, 0, 1, 0, 0, 0, 1, 1, 0, 0, 1, 0, 1, 0, 1, 0, 0, 1, 0, 1, 1, 1, 1, 1, 0, 0, 0, 1, 0, 0, 0, 0, 1, 0]
[Data 5, Neuron Output 0] ❌ expected: -1 got: -0.9728782443551849 data: [1, 1, 1, 1, 1, 1, 0, 0, 0, 0, 1, 1, 1, 1, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 1, 1, 0, 0, 0, 1, 0, 1, 1, 1, 0
[Data 6, Neuron Output 0] ❌ expected: 1 got: 0.9396145251403647 data: [0, 0, 1, 1, 0, 0, 1, 0, 0, 0, 1, 0, 0, 0, 0, 1, 1, 1, 1, 0, 1, 0, 0, 0, 1, 1, 0, 0, 0, 1, 0, 1, 1, 1, 0]
[Data 7, Neuron Output 0] ❌ expected: -1 got: -0.9872019401938853 data: [1, 1, 1, 1, 1, 0, 0, 0, 0, 1, 0, 0, 0, 1, 0, 0, 0, 1, 0, 0, 0, 1, 0, 0, 0, 0, 1, 0, 0, 0, 0, 1, 0, 0, 0
[Data 0, Neuron Output 0] ✅ expected: 1 got: 0.936633199014711 data: [0, 1, 1, 1, 0, 1, 0, 0, 0, 1, 1, 0, 0, 1, 1, 1, 0, 1, 0, 1, 1, 1, 0, 0, 1, 1, 0, 0, 0, 1, 0, 1, 1, 1, 0]
[Data 1, Neuron Output 0] ✅ expected: -1 got: -0.9883190298228692 data: [0, 0, 1, 0, 0, 0, 1, 1, 0, 0, 0, 0, 1, 0, 0, 0, 0, 1, 0, 0, 0, 0, 1, 0, 0, 0, 0, 1, 0, 0, 0, 1, 1, 1, 0
[Data 2, Neuron Output 0] ✅ expected: 1 got: 0.872989208577505 data: [0, 1, 1, 1, 0, 1, 0, 0, 0, 1, 0, 0, 0, 0, 1, 0, 0, 0, 1, 0, 0, 0, 1, 0, 0, 0, 1, 0, 0, 0, 1, 1, 1, 1, 1]
[Data 3, Neuron Output 0] ✅ expected: -1 got: -0.9103501697313592 data: [0, 1, 1, 1, 0, 1, 0, 0, 0, 1, 0, 0, 0, 0, 1, 0, 0, 1, 1, 0, 0, 0, 0, 0, 1, 1, 0, 0, 0, 1, 0, 1, 1, 1, 0
[Data 4, Neuron Output 0] ✅ expected: 1 got: 0.9513665240017575 data: [0, 0, 0, 1, 0, 0, 0, 1, 1, 0, 0, 1, 0, 1, 0, 1, 0, 0, 1, 0, 1, 1, 1, 1, 1, 0, 0, 0, 1, 0, 0, 0, 0, 1, 0]
[Data 5, Neuron Output 0] ✅ expected: -1 got: -0.9746304883852132 data: [1, 1, 1, 1, 1, 1, 0, 0, 0, 0, 1, 1, 1, 1, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 1, 1, 0, 0, 0, 1, 0, 1, 1, 1, 0
[Data 6, Neuron Output 0] ✅ expected: 1 got: 0.943235559932482 data: [0, 0, 1, 1, 0, 0, 1, 0, 0, 0, 1, 0, 0, 0, 0, 1, 1, 1, 1, 0, 1, 0, 0, 0, 1, 1, 0, 0, 0, 1, 0, 1, 1, 1, 0]
[Data 7, Neuron Output 0] ✅ expected: -1 got: -0.9880035941151759 data: [1, 1, 1, 1, 1, 0, 0, 0, 0, 1, 0, 0, 0, 1, 0, 0, 0, 1, 0, 0, 0, 1, 0, 0, 0, 0, 1, 0, 0, 0, 0, 1, 0, 0, 0

Epoch: 20, End Reason: EndReason.ACCEPTABLE_ERROR_REACHED, Error: 0.0173

-------Evaluating after training-------

[Data 1] error: 0.017583144249004225 ✅ expected: [1] got: [0.81247323] data: [0, 1, 1, 1, 0, 1, 0, 0, 0, 1, 1, 0, 0, 0, 1, 0, 1, 1, 1, 0, 1, 0, 0, 0, 1, 1, 0, 0, 0, 1, 0, 1, 1,
[Data 2] error: 0.027729002003289687 ❌ expected: [-1] got: [-0.76450477] data: [0, 1, 1, 1, 0, 1, 0, 0, 0, 1, 1, 0, 0, 0, 1, 0, 1, 1, 1, 1, 1, 0, 0, 0, 0, 1, 0, 0, 0, 1, 0, 0, 1,

Multilayer perceptron after training for 20 epochs has an average error of 0.022656073126146956 ❌
```

Elección de Tasa de Aprendizaje

Para cada uno de los metodos de optimizacion, buscamos cuál es una tasa de aprendizaje razonable teniendo en cuenta:

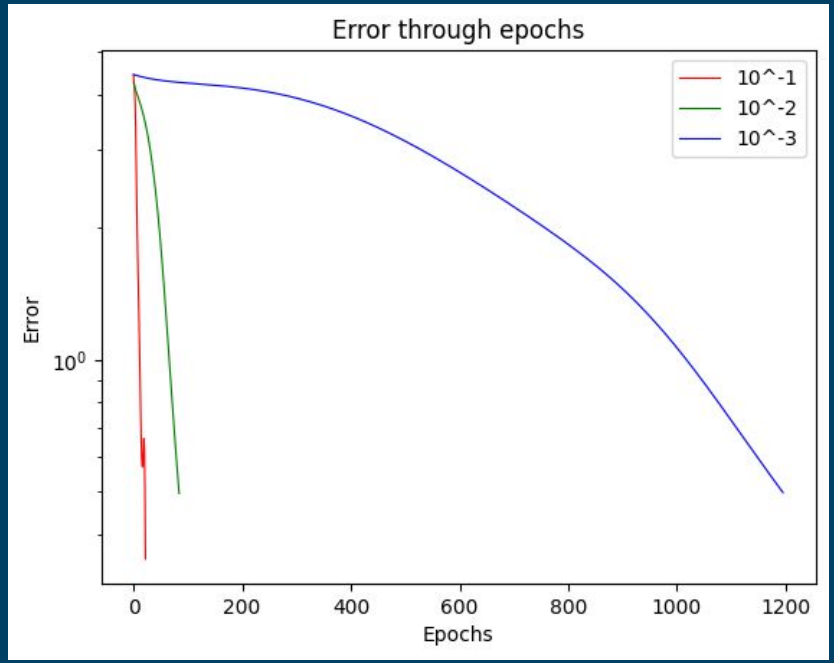
- Convergencia en cantidad de épocas razonable.
- Minimizando oscilaciones en el error.

Nuestro error objetivo va a ser: 10^-4

Gradiente Descendente

- Una tasa mayor a 10^-2 nos da una convergencia muy lenta comparado con valores inferiores
- 10^-1 nos da una convergencia muy rápida y un buen valor de error.

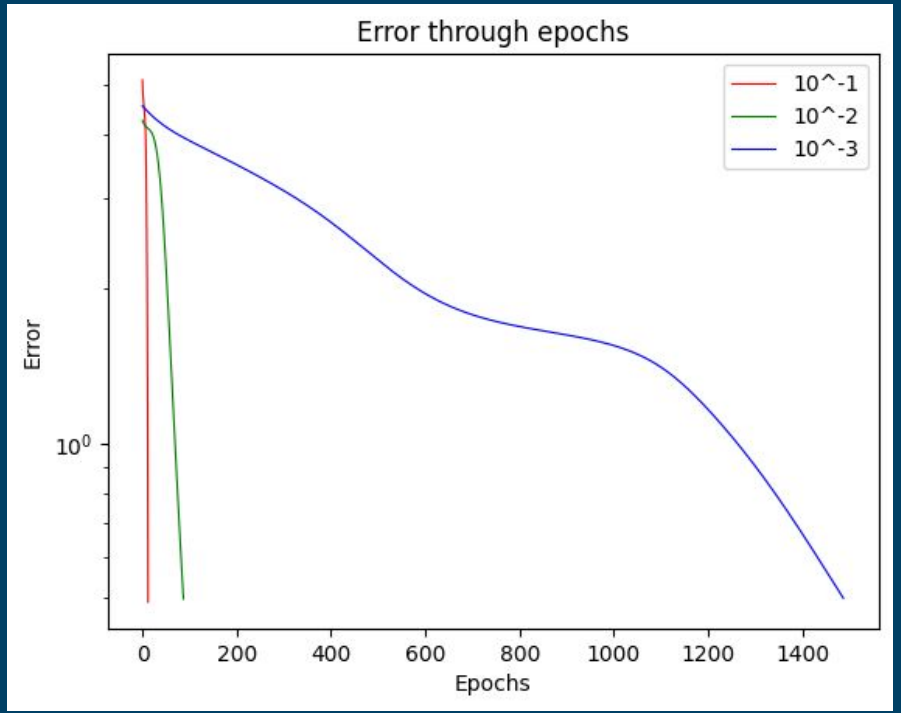
=> Tomamos 10^-1



Momentum

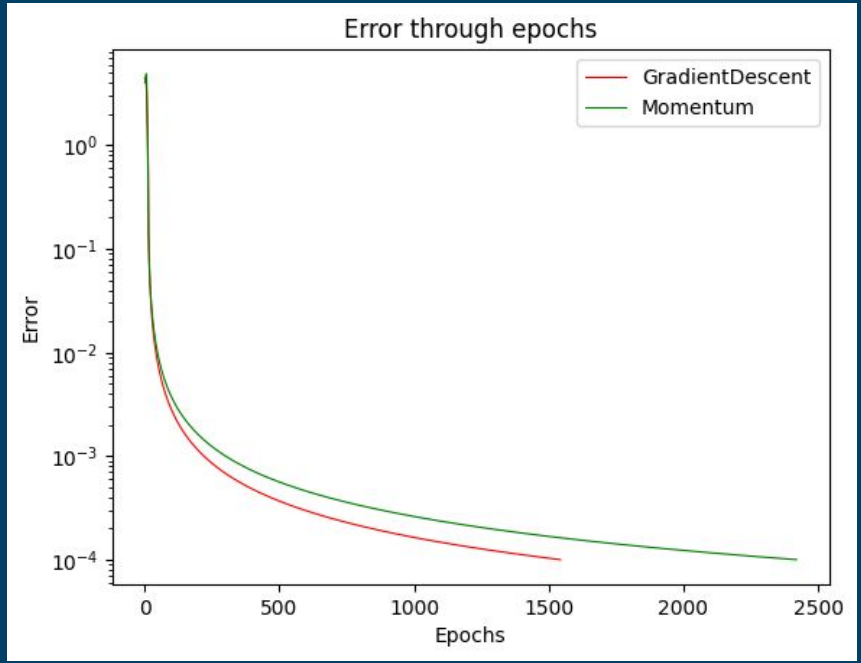
- Se observa lo mismo que en el gradiente descendente

=> Tomamos 10^-1



Aprendizaje: Comparación de Métodos

Observamos la Convergencia



Generalización: Comparación de Métodos

Cantidad de corridas: 15

Modificamos el error aceptable a 10^-1 para poder realizar muchas corridas

Conclusiones y aprendizajes

- La paridad de los números no permite ser generalizada con el formato de mapa de dígitos.
- Más épocas permiten llegar a un menor error.

Ejercicio 3.C

Elección de Tasa de Aprendizaje

Para cada uno de los metodos de optimizacion, buscamos cuál es una tasa de aprendizaje razonable teniendo en cuenta:

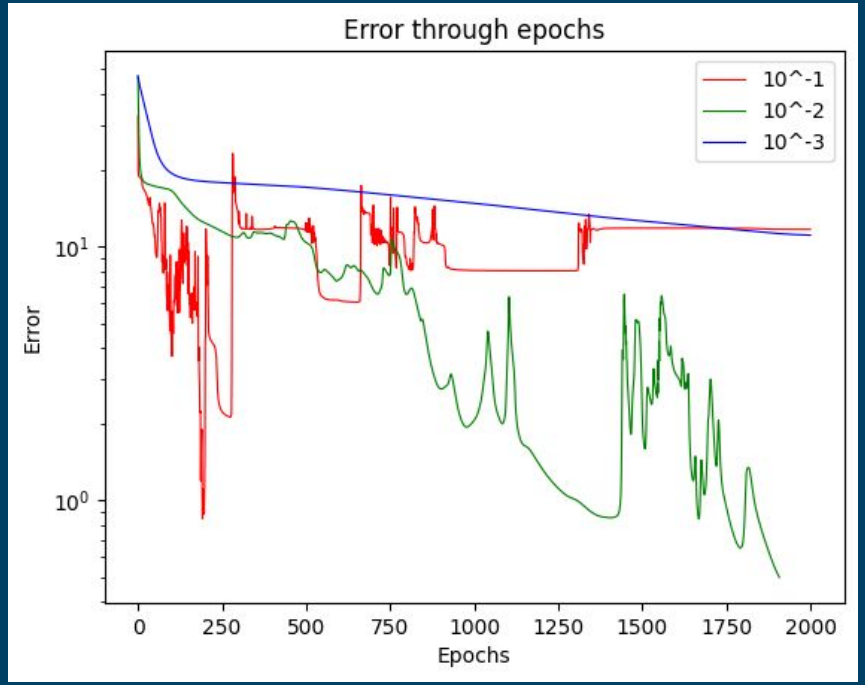
- Convergencia en cantidad de épocas razonable.
- Minimizando oscilaciones en el error.

Nuestro error objetivo va a ser: 0.5

Gradiente Descendente

- A partir de **10^-3** se ve una convergencia sin saltos en error
- 10^-3 converge demasiado lento para ser elegido
- **10^-2** presenta saltos pero ofrece una convergencia rápida con error bajo

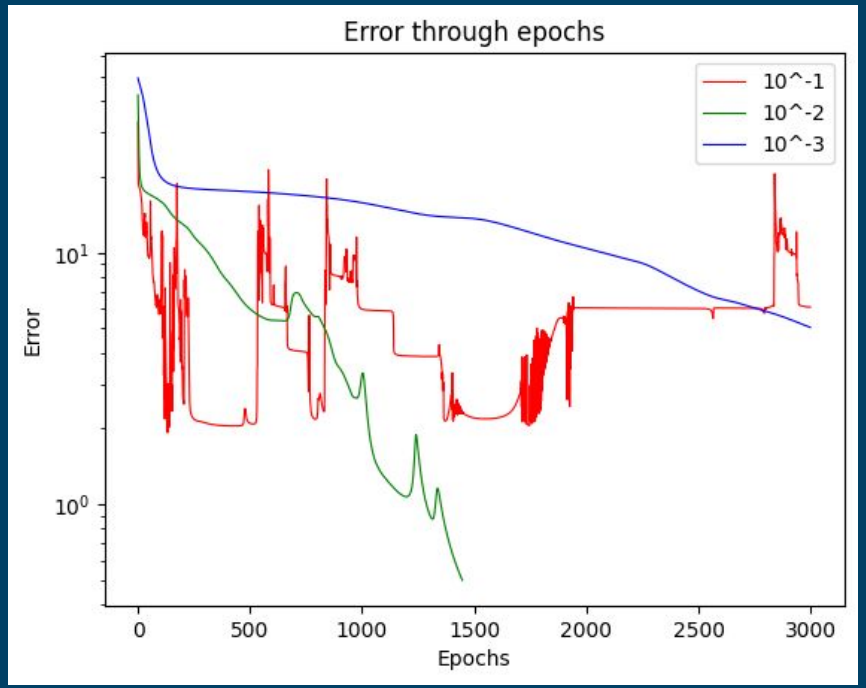
=> Tomamos 10^-2



Momentum

- De nuevo con 10^-2 se observa rápida convergencia con menos error
- 10^-1 se estanca en lo que parece ser un mínimo local

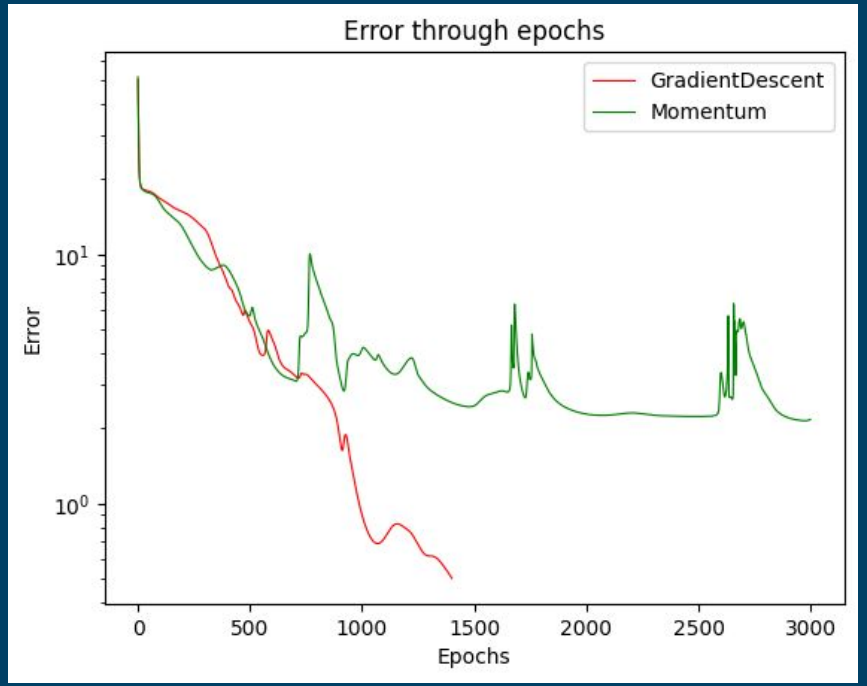
=> Tomamos 10^-2



Aprendizaje: Comparación de Métodos

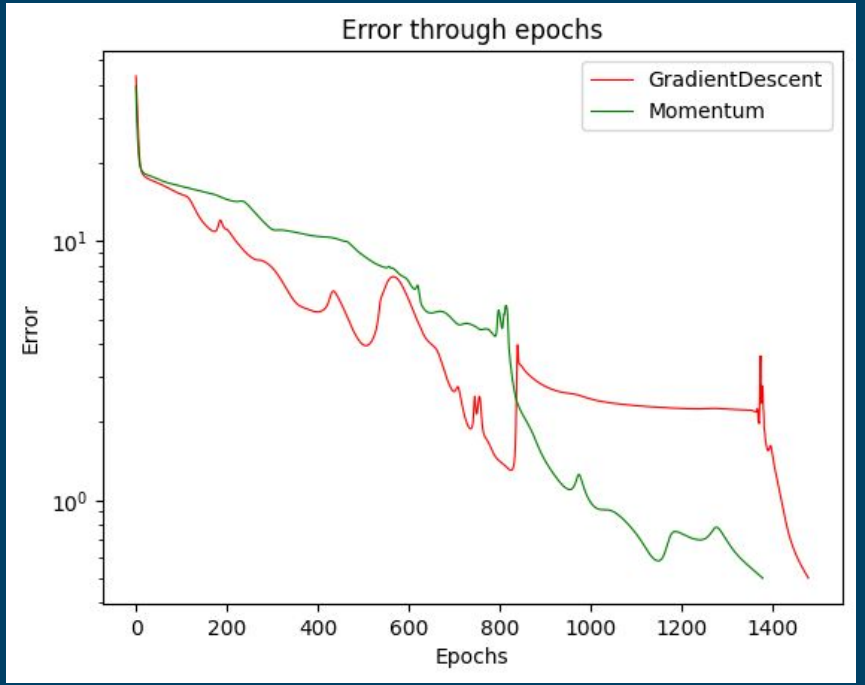
¡No siempre convergen!

En este ejemplo, Momentum (alfa=0.9) probablemente esté atrapado en un bache



Aprendizaje: Comparación de Métodos

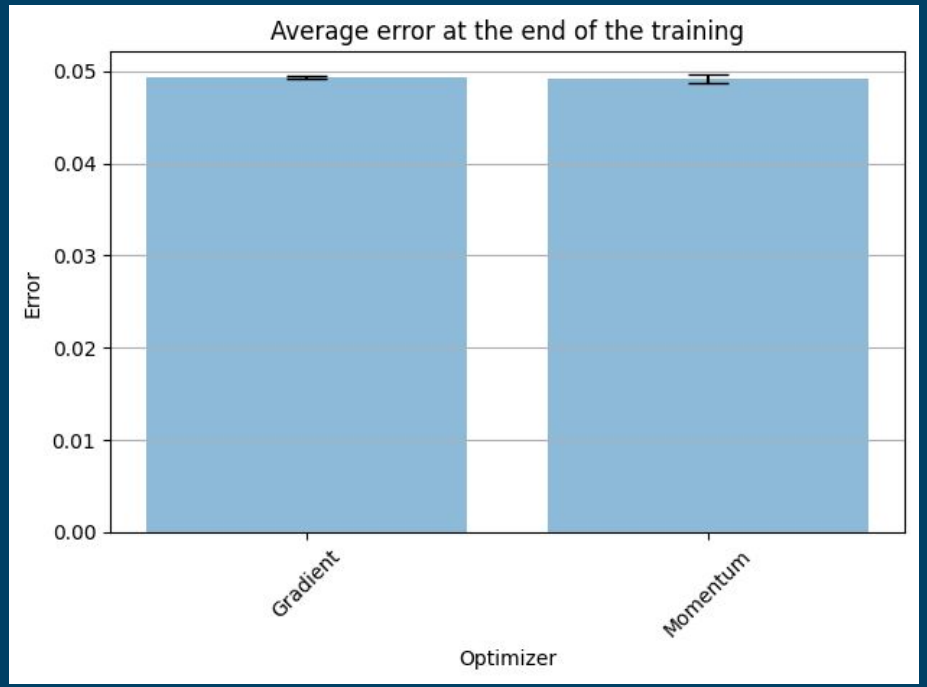
Momentum (alfa = 0.9) muestra mejores resultados en resultados de convergencia que Gradiente Descendente



Generalización: Comparación de Métodos

Cantidad de corridas: 5

Generaliza mejor que el ejercicio anterior ya que este si tiene una relación entre output y mapa de dígitos.



Evaluación con ruido

```
-------Evaluating after training-------

[Data 1] error: 0.06316125151740735 ✅ expected: [1, -1, -1, -1, -1, -1, -1, -1, -1, -1] got: [ 0.75773081 -0.99999996 -0.99960559 -0.84773501 -0.81917883 -0.99973223
 -0.89161824 -0.9993928  -0.99999513 -0.99985271] data: [0, 1, 1, 1, 0, 1, 0, 0, 0, 1, 1, 0, 0, 1, 1, 1, 0, 1, 0, 1, 1, 1, 0, 0, 1, 1, 0, 0, 0, 1, 0, 1, 1, 1, 0]
[Data 2] error: 0.04811190476089395 ✅ expected: [-1, 1, -1, -1, -1, -1, -1, -1, -1, -1] got: [-0.99775791  0.80281422 -0.99989861 -0.99999291 -0.99999905 -0.99998433
 -0.86983088 -0.99919346 -0.99564499 -0.79906985] data: [0, 0, 1, 0, 0, 0, 1, 1, 0, 0, 0, 0, 1, 0, 0, 0, 0, 1, 0, 0, 0, 0, 1, 0, 0, 0, 0, 1, 0, 0, 0, 1, 1, 1, 0]
[Data 3] error: 0.09098007503894184 ✅ expected: [-1, -1, 1, -1, -1, -1, -1, -1, -1, -1] got: [-0.9999998  -0.98671887  0.65644378 -0.99243569 -0.91400209 -0.99997707
 -0.99991489 -0.99658043 -0.79755612 -0.87628741] data: [0, 1, 1, 1, 0, 1, 0, 0, 0, 1, 0, 0, 0, 0, 1, 0, 0, 0, 1, 0, 0, 0, 1, 0, 0, 0, 1, 0, 0, 0, 1, 1, 1, 1, 1]
[Data 4] error: 0.0420669727713226 ✅ expected: [-1, -1, -1, 1, -1, -1, -1, -1, -1, -1] got: [-0.99997776 -0.99999183 -0.96730392  0.785426   -0.96815207 -0.99972599
 -0.91215107 -0.99999929 -0.8318004  -0.99997389] data: [0, 1, 1, 1, 0, 1, 0, 0, 0, 1, 0, 0, 0, 0, 1, 0, 0, 1, 1, 0, 0, 0, 0, 0, 1, 1, 0, 0, 0, 1, 0, 1, 1, 1, 0]
[Data 5] error: 0.08292958246417155 ✅ expected: [-1, -1, -1, -1, 1, -1, -1, -1, -1, -1] got: [-0.80356999 -0.99999047 -0.80877073 -0.98176185  0.74145592 -0.999986
 -0.99672768 -0.84769741 -0.99998644 -0.98207452] data: [0, 0, 0, 1, 0, 0, 0, 1, 1, 0, 0, 1, 0, 1, 0, 1, 0, 0, 1, 0, 1, 1, 1, 1, 1, 0, 0, 0, 1, 0, 0, 0, 0, 1, 0]
[Data 6] error: 0.032791394120961735 ✅ expected: [-1, -1, -1, -1, -1, 1, -1, -1, -1, -1] got: [-0.98099674 -0.99999912 -0.98175285 -0.83909079 -0.9983238   0.85149801
 -0.9988077  -0.99964274 -0.97124149 -0.87306427] data: [1, 1, 1, 1, 1, 1, 0, 0, 0, 0, 1, 1, 1, 1, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 1, 1, 0, 0, 0, 1, 0, 1, 1, 1, 0]
[Data 7] error: 0.03049598958007202 ✅ expected: [-1, -1, -1, -1, -1, -1, 1, -1, -1, -1] got: [-0.89886541 -0.84863341 -0.99241319 -0.99611995 -0.95872217 -0.99999988
  0.83880925 -0.99629771 -0.99994646 -0.99109568] data: [0, 0, 1, 1, 0, 0, 1, 0, 0, 0, 1, 0, 0, 0, 0, 1, 1, 1, 1, 0, 1, 0, 0, 0, 1, 1, 0, 0, 0, 1, 0, 1, 1, 1, 0]
[Data 8] error: 0.050111020433191954 ✅ expected: [-1, -1, -1, -1, -1, -1, -1, 1, -1, -1] got: [-0.89842101 -0.99999691 -0.99468858 -0.99999488 -0.83028723 -0.94139411
 -0.99999881  0.82985726 -0.99999905 -0.83061904] data: [1, 1, 1, 1, 1, 0, 0, 0, 0, 1, 0, 0, 0, 1, 0, 0, 0, 1, 0, 0, 0, 1, 0, 0, 0, 0, 1, 0, 0, 0, 0, 1, 0, 0, 0]
[Data 9] error: 0.041203867673660625 ✅ expected: [-1, -1, -1, -1, -1, -1, -1, -1, 1, -1] got: [-0.99999996 -0.87255857 -0.98951409 -0.87705729 -0.99996095 -0.99980272
 -0.99977504 -0.99999944  0.77437046 -0.99427535] data: [0, 1, 1, 1, 0, 1, 0, 0, 0, 1, 1, 0, 0, 0, 1, 0, 1, 1, 1, 0, 1, 0, 0, 0, 1, 1, 0, 0, 0, 1, 0, 1, 1, 1, 0]
[Data 10] error: 0.07370325669099785 ✅ expected: [-1, -1, -1, -1, -1, -1, -1, -1, -1, 1] got: [-0.99999997 -0.99778091 -0.72241016 -0.99999608 -0.99994765 -0.99974458
 -0.99999758 -0.99789525 -0.90992379  0.7505461 ] data: [0, 1, 1, 1, 0, 1, 0, 0, 0, 1, 1, 0, 0, 0, 1, 0, 1, 1, 1, 1, 0, 0, 0, 0, 1, 0, 0, 0, 1, 0, 0, 1, 1, 0, 0]

Multilayer perceptron after training for 1000 epochs has an average error of 0.05555553150516214 ✅
```

Preguntas?