Ejercicio 2

Perceptrón Lineal y No Lineal

Parámetros Utilizados

Función Theta	Lineal	Tanh	Logística
Tasa de Aprendizaje	0.001		
Máximo de Épocas	1000	2500	
Error Aceptable	8	1	
Épsilon de pesos	0.001	0.00001	
Beta	-	1	
Act. Pesos	batch		

Condición de corte

- Pesos iniciales aleatorios en el rango (-1, 1)

No Lineal: Reescalamiento de datos

Función Theta	Lineal	Tanh	Logística
Rango de Imagen	(-∞, +∞)	(-1, 1)	(0, 1)
Rango del dataset	[0.320, 88.184] *(tomando el mínimo y máximo)*		

Los perceptrones no lineales trabajan en rangos distintos al de nuestro dataset!
⇒ Debemos reescalar los datos a un rango que entiendan.

Para entrenar: Transformamos linealmente los datos de [0.320, 88.184] → Imagen(θ)

Al predecir: Transformamos linealmente la salida de Imagen(θ) → [0.320, 88.184]

¿Cómo comparamos los errores entre Thetas?

Previamente, estábamos calculando el error en base a la salida de θ, y luego “normalizando” este error a un porcentaje del error máximo posible para poder comparar.

Cuando uno usa un perceptrón no-lineal, va a querer reescalar las predicciones que haga al rango original de los datos, y por ende preferimos cambiar nuestra metodología y, al comparar distintos perceptrones, calcular el error en base a esto:

⇒ E de una predicción individual: $E_{individual} = (reescalar(O^{\mu}) - reescalar(\zeta^{\mu}))^2 / 2$

⇒ E de varias predicciones: $E_{dataset} = promedio(E_{individual})$

Comparación de Errores Por Época para cada método

θ	η	e_{max}	E_{ok}	β	$\varepsilon_{weights}$
Lineal	10^{-3}	1000	8	-	10^{-3}
No-lineal		2500	1	1	10^{-5}

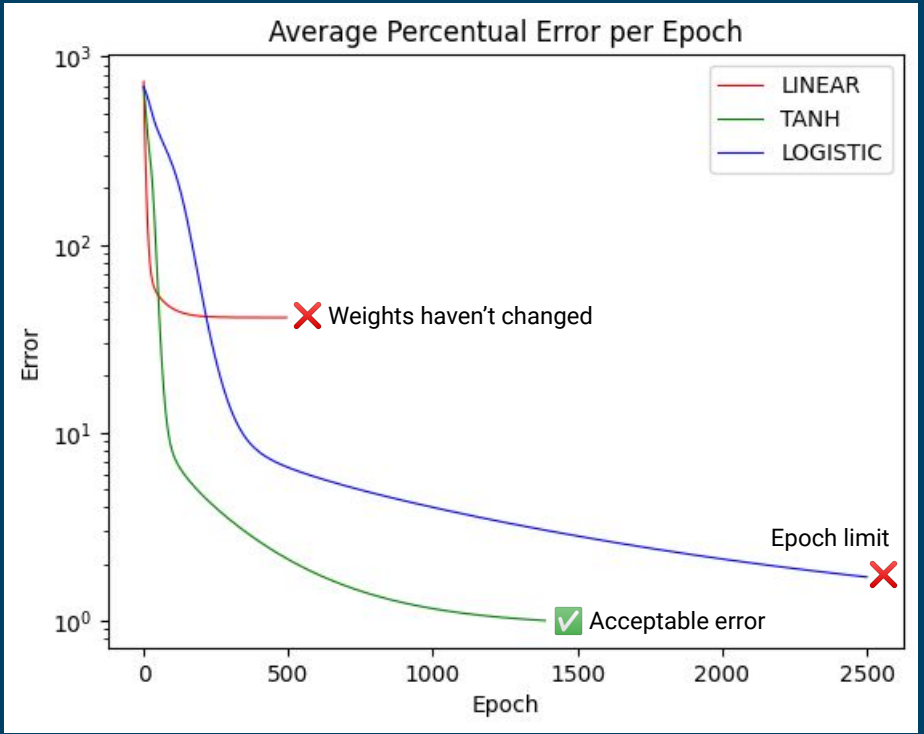
Cada perceptrón termina de entrenar por una razón distinta.

Tanh es indiscutiblemente el ganador acá; converge más rápido y consigue el menor error.

Lineal es incapaz de aprender este dataset. Su error es dos órdenes de magnitud peor.

Y si dejamos que terminen de converger?

$w_{inicial}$ = (-0.294, 0.292, 0.920, 0.730)



Comparación de Errores Por Época para cada método

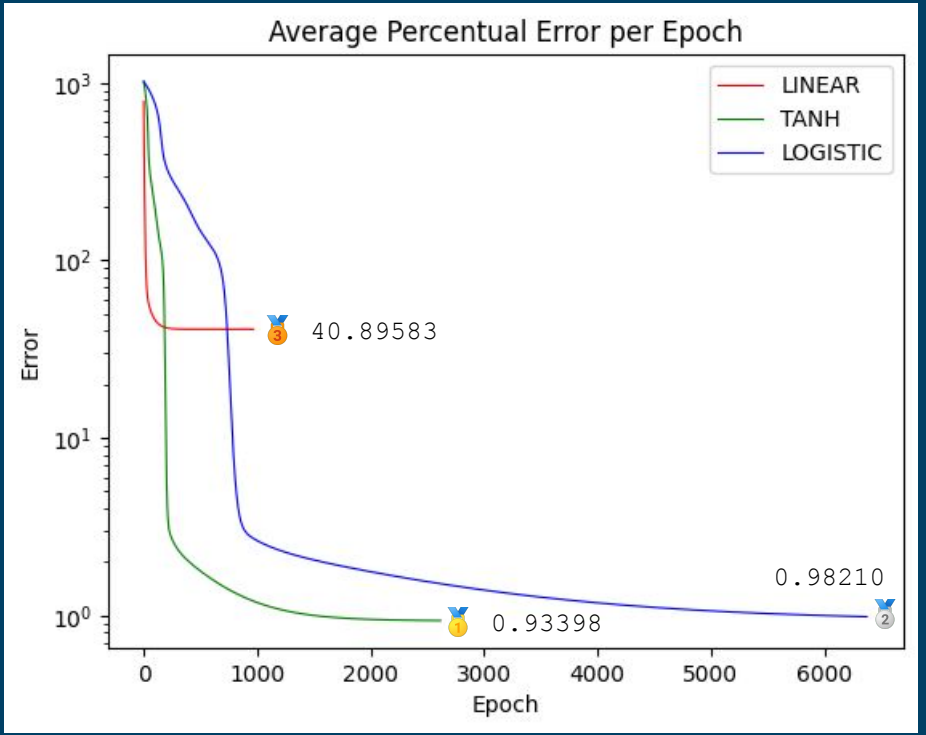
θ	η	e_{max}	E_{ok}	β	$\varepsilon_{weights}$
Lineal	10^{-3}	-	0	-	10^{-5}
No-lineal				1	

Queremos concluir sobre si la forma de los datos se parece más a Tanh o Logistic.

Sacamos el límite de épocas y de error aceptable, y encontramos que a largo plazo Logistic se acerca a Tanh, pero Tanh sigue manteniendo una ventaja.

⇒ Tanh es lo más apropiado para este dataset

$w_{inicial}$ = (-0.940, -0.765, 0.512, 0.958)



Conclusiones hasta ahora

“No se puede tomar sopa con un tenedor”

- ➔ El perceptrón lineal es incapaz de aprender este dataset.
- ➔ Theta=Tanh provee la mejor aproximación para este problema

Pero además…

- ➔ No hay un tipo de perceptrón que sea universalmente superior.
- ➔ Cada perceptrón es apto para distintos tipos de problemas.
- ➔ Es crítico utilizar un perceptrón apropiado a cada dataset

Capacidad de **Aprendizaje** y **Generalización** para distintas proporciones de datasets

Experimento 2.B

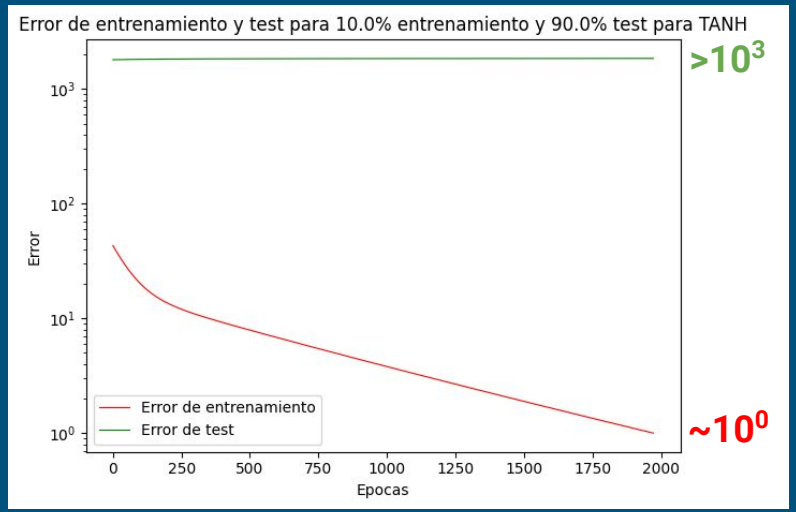
- Se tomó el dataset inicial “TP3-ej2-conjunto.csv”
- Se dividió al dataset en dos conjuntos con elementos al azar de la siguiente manera:
  - **Dataset de Entrenamiento** (X% del dataset inicial)
  - **Dataset de Test** ((100 - X)% del dataset inicial)

- El objetivo del experimento es **determinar cuál es la mejor proporción de datos para este problema en particular para maximizar la capacidad de Generalización**, comparando:
  - Error de Entrenamiento en función de Épocas
  - Error de Test en función de Épocas

Los pesos iniciales comenzarán todos siempre en 0 para minimizar el ruido.

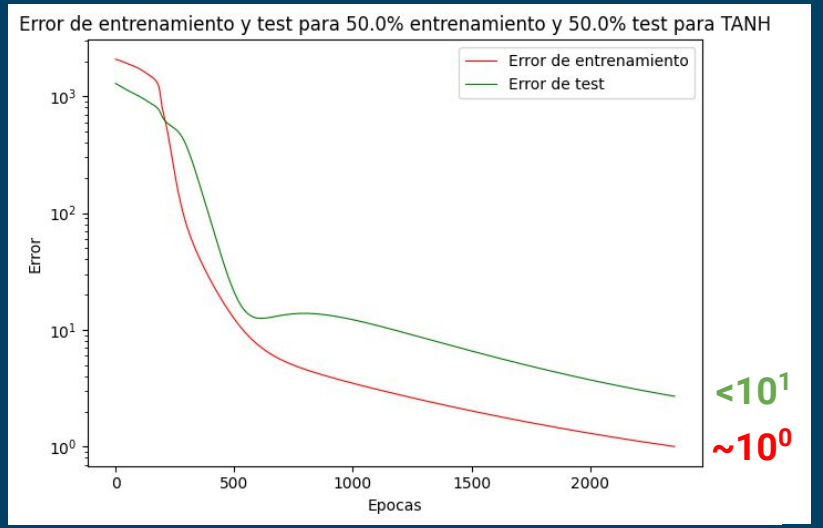
Tanh

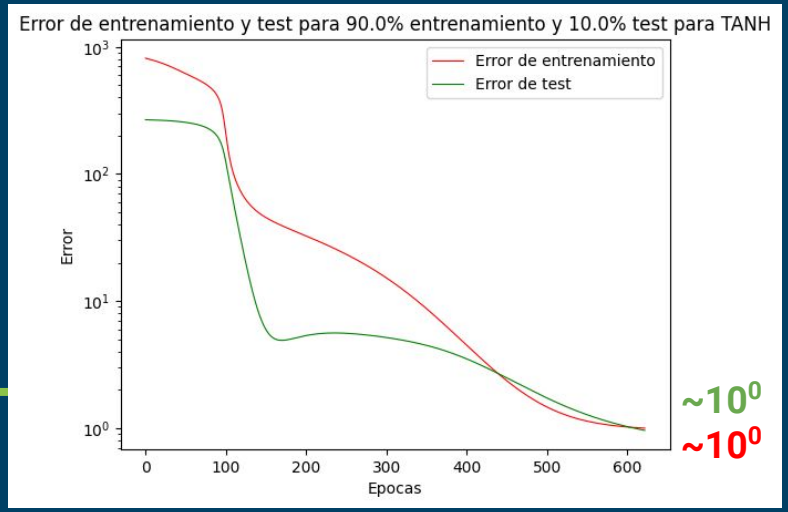
- $\beta = 1$
- $\eta = 10^{-3}$
- $e_{max} = 2500$
- Batch



Observamos que mejora con 90% para entrenamiento, y un 10% para el test.

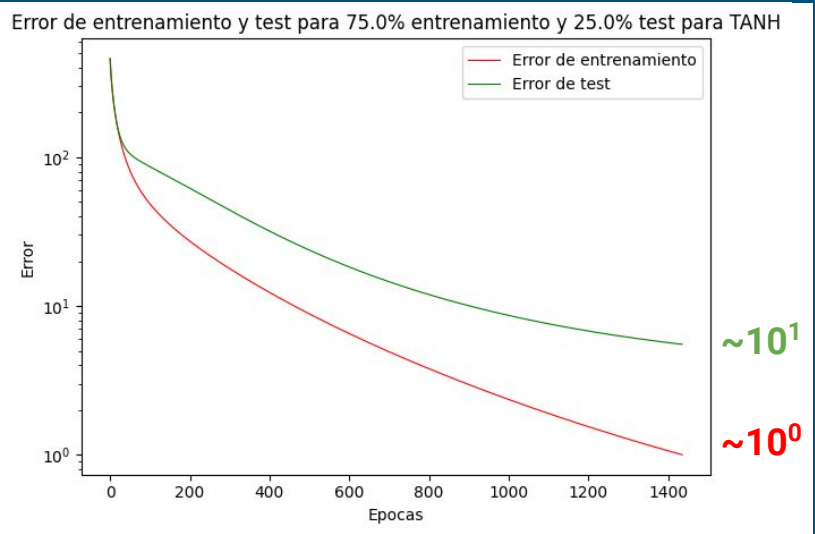
⇒ Continuamos Analizando



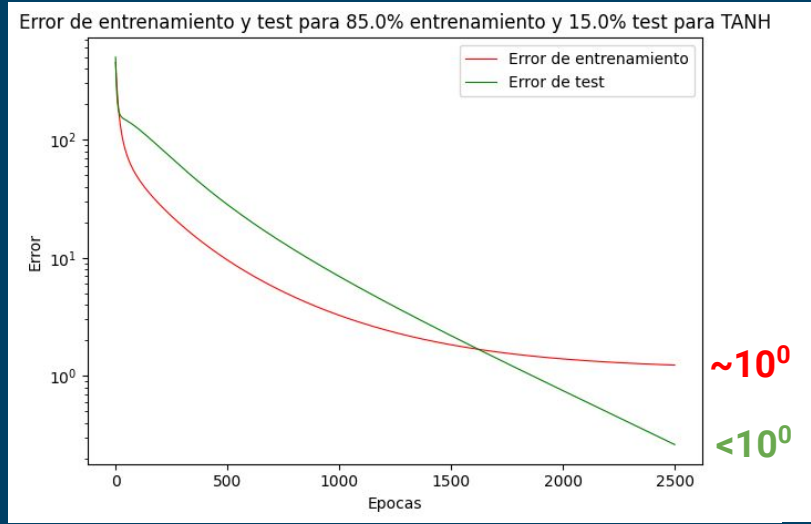


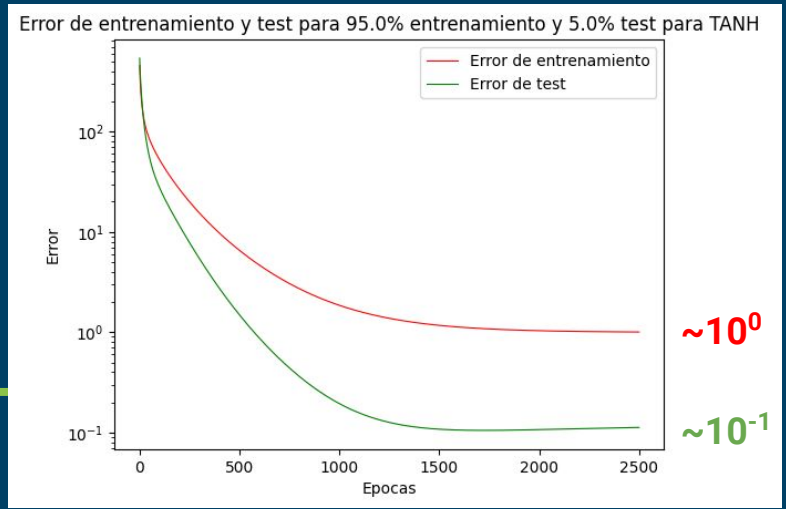
Tanh

- $\beta = 1$
- $\eta = 10^{-3}$
- $e_{max} = 2500$
- Batch



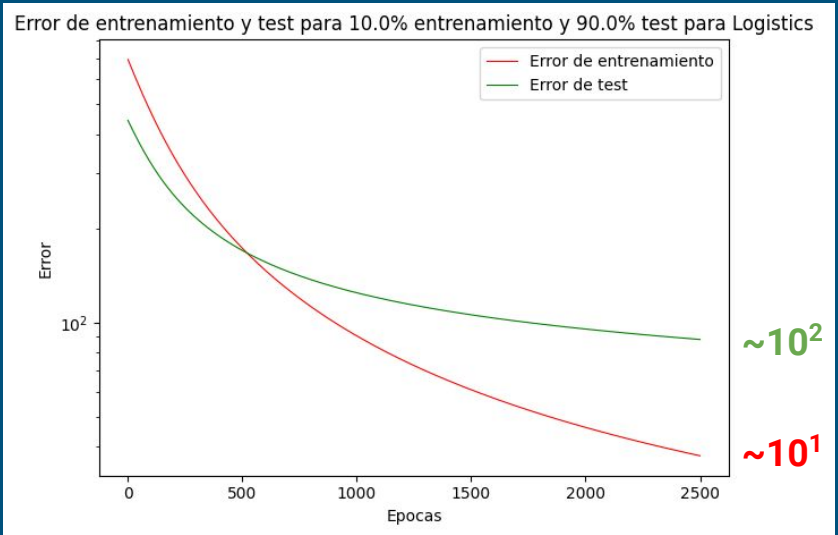
⇒ Observamos los mejores resultados con 95% para entrenamiento y 5% para pruebas.





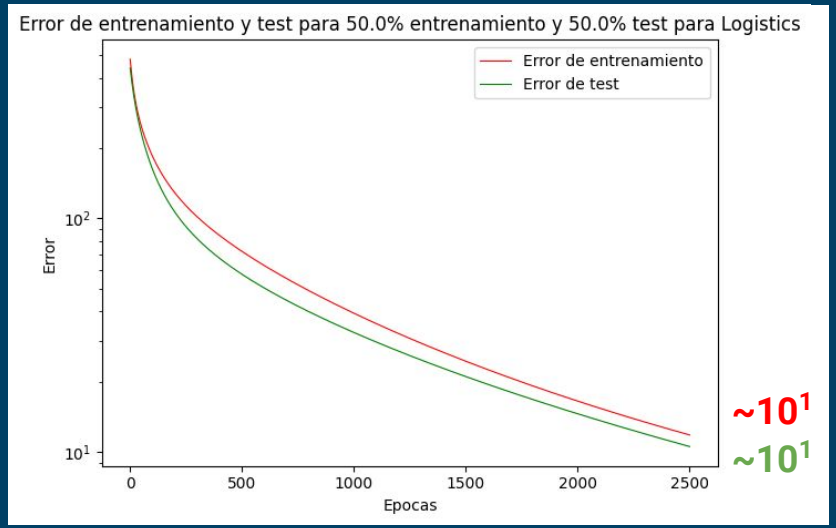
Logistic

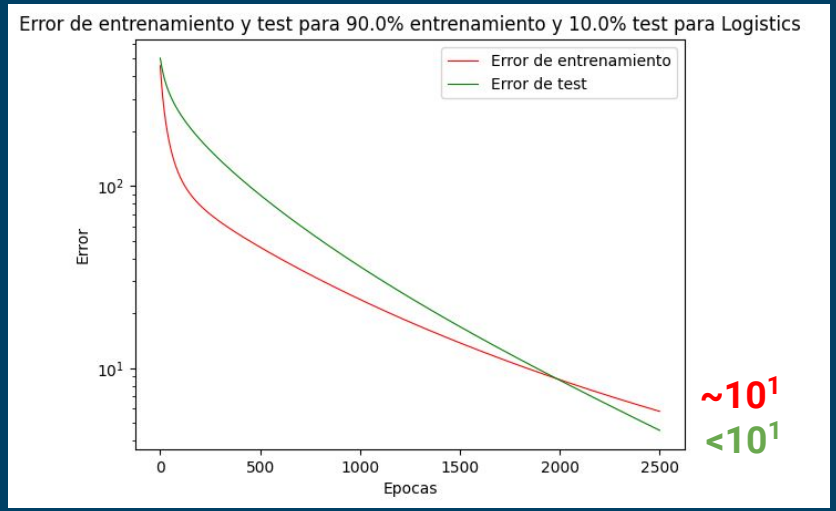
- $\beta = 1$
- $\eta = 10^{-3}$
- $e_{max} = 2500$
- Batch



Observamos que mejora con 90% para entrenamiento, y un 10% para el test.

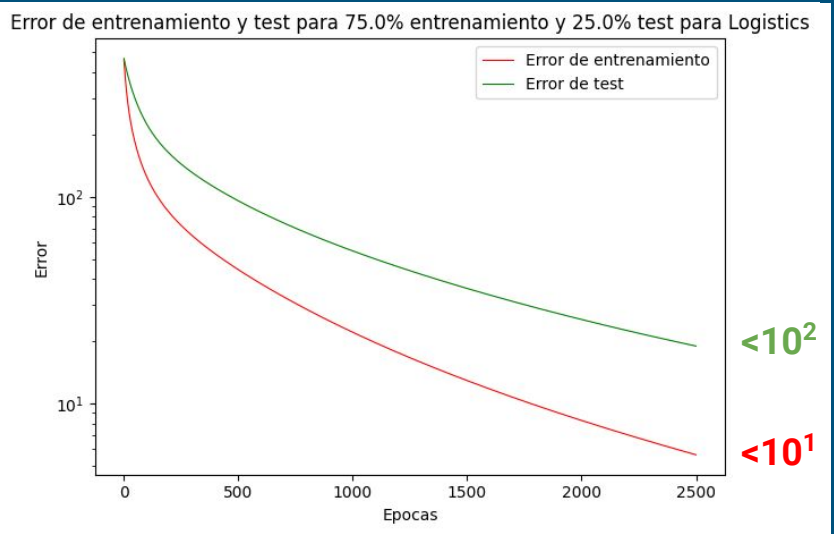
⇒ Continuamos Analizando



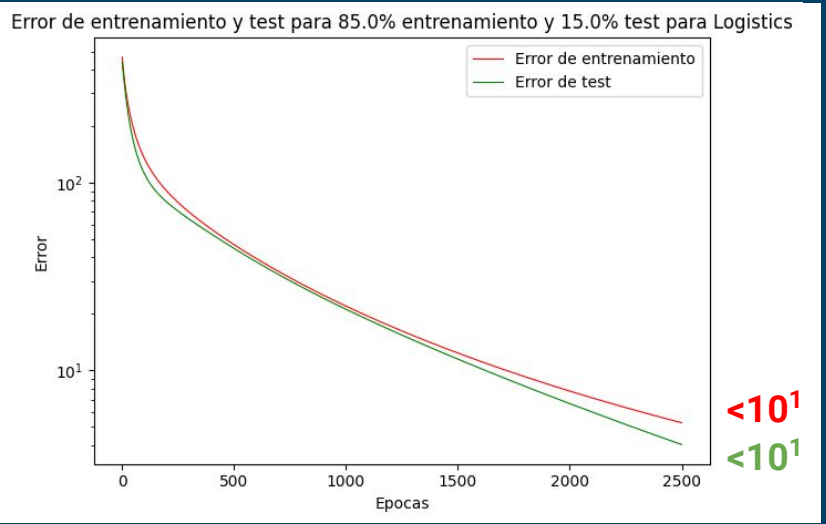


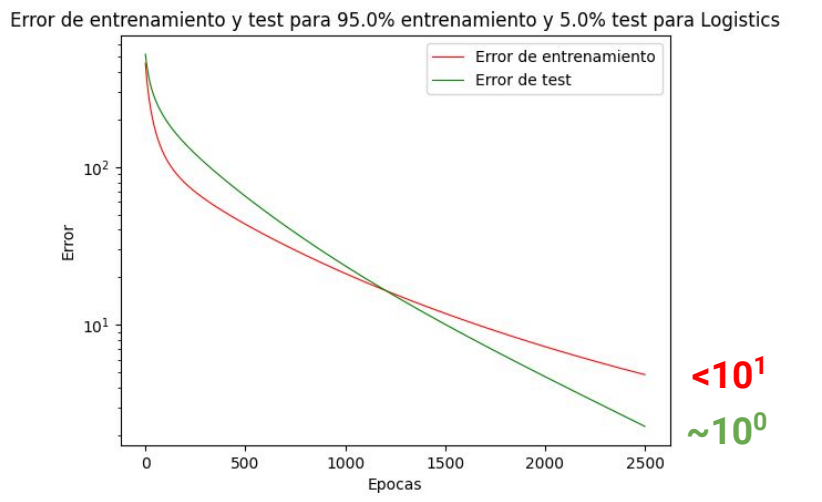
Logistic

- $\beta = 1$
- $\eta = 10^{-3}$
- $e_{max} = 2500$
- Batch



⇒ Observamos los mejores resultados con 95% para entrenamiento y 5% para pruebas.





Observaciones

- Pudimos observar que, para este problema en particular, mientras **mayor proporción de elementos en el dataset de Entrenamiento**, se obtuvieron **mejores resultados en Generalización**
- Cabe destacar que esto **NO es una regla general**, en un problema de diferentes características, un dataset de entrenamiento demasiado grande puede llevar al problema de **sobreajuste**

Experimento 2.C

- En este experimento, nuestro objetivo es determinar cuál es la mejor combinación de elementos del dataset que minimizan el error de test.
- En este experimento los pesos iniciales serán inicializados siempre en 0 para eliminar ruido
- Para llevar a cabo esto, vamos a utilizar la técnica de **Cross-Validation**
  1. Separamos el dataset en **k = 4 folds**, es decir distribuir los elementos del conjunto en 4 subconjuntos (7 elementos cada uno) de forma aleatoria
  2. Entrenamos el modelo con k-1 folds y utilizamos el restante para test (75% entrenamiento, 25% test)
  3. Repetimos el paso anterior hasta cubrir todas las combinaciones posibles
  4. Determinamos qué combinación obtuvo mejores resultados para generalización, es decir, que combinación obtuvo menor error de test

Folds

Fold 1			
x1	x2	x3	y
1.200	-800	1.000	43.045
0	-1.300	3.230	88.184
1.800	0	1.600	49.000
0	1.800	1.600	18.543
-2.000	2.000	0	2.660
-1.300	3.230	0	1.480
0	-0.500	0.600	24.974

Fold 2			
x1	x2	x3	y
1.200	0	-0.800	7.176
0	1.200	-0.800	2.875
0	-2.000	2.000	76.852
-500	0.600	2.500	51.000
7.900	0	1.000	64.107
1.800	1.600	1.300	21.417
1.800	1.600	0	6.914

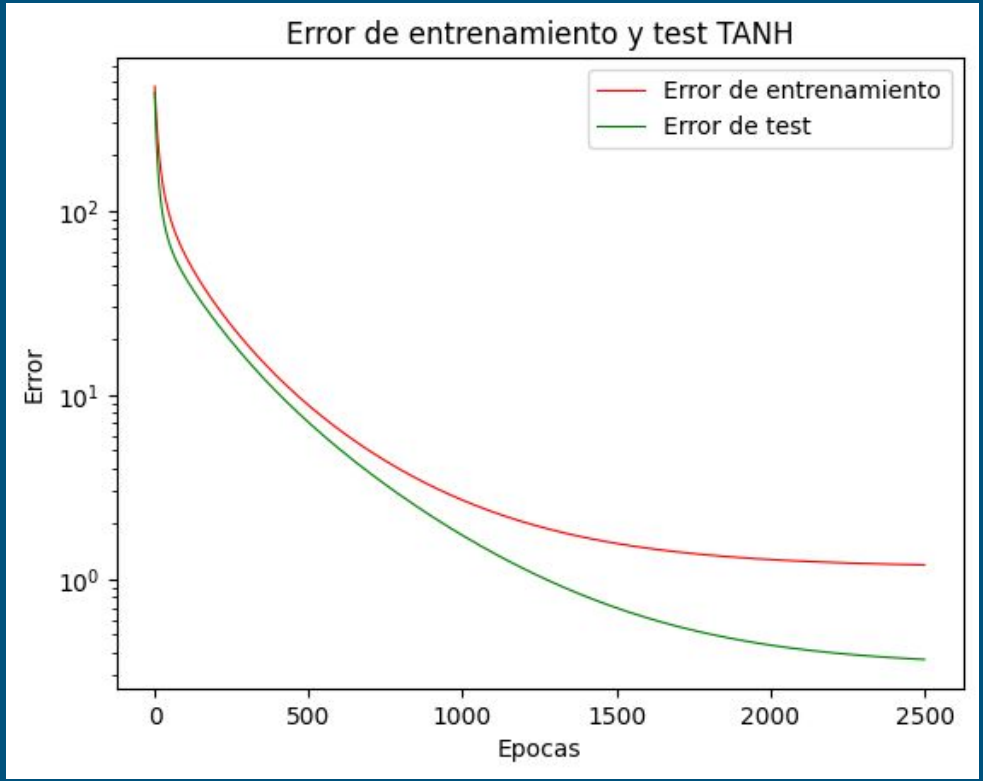
Fold 3			
x1	x2	x3	y
1.200	-0.800	0	21.755
7.900	1.000	0	26.503
0	0.400	2.700	61.301
-1.300	0	3.230	72.512
7.900	1.000	-2.000	4.653
-0.500	0.600	0	7.871
-2.000	2.000	-1.000	0.995

Fold 4			
x1	x2	x3	y
0.400	0	2.700	68.568
-1.300	3.230	3.000	23.183
0.400	2.700	0	2.820
0.400	2.700	2.000	17.654
0	7.900	1.000	0.320
-2.000	0	2.000	40.131
-500	0	0.600	18.243

Combinación 1-2-3

- $\beta = 1$
- $\eta = 10^{-3}$
- $e_{max} = 2500$
- Batch

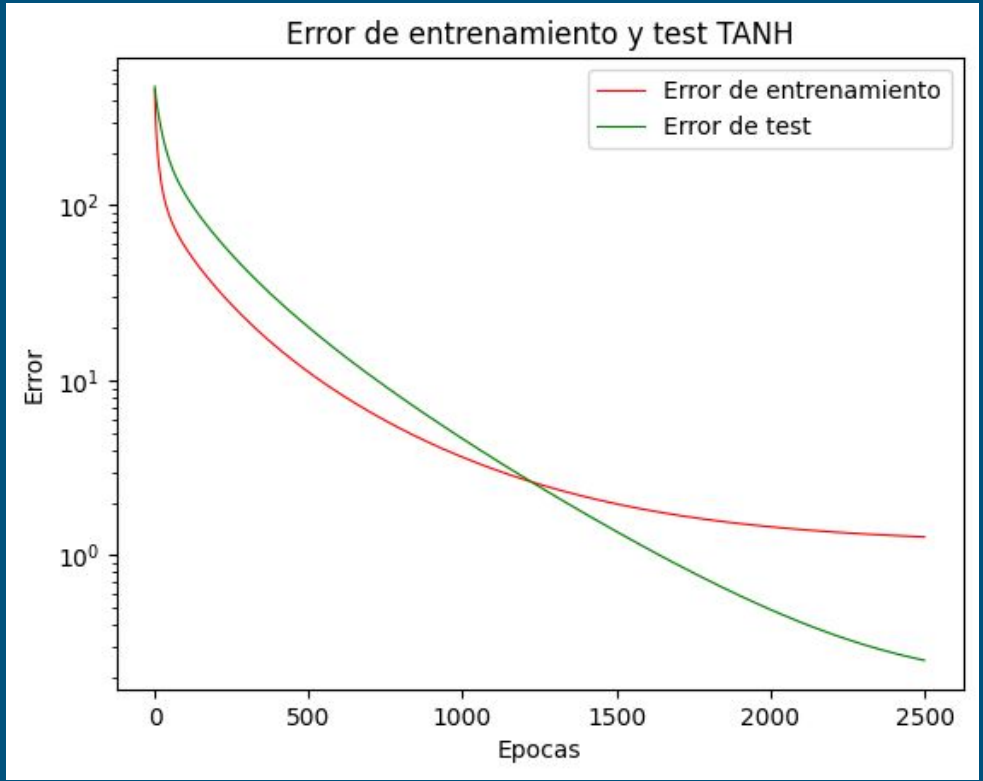
Error de Test: 0.3667771461297337



Combinación 1-2-4

- $\beta = 1$
- $\eta = 10^{-3}$
- $e_{max} = 2500$
- Batch

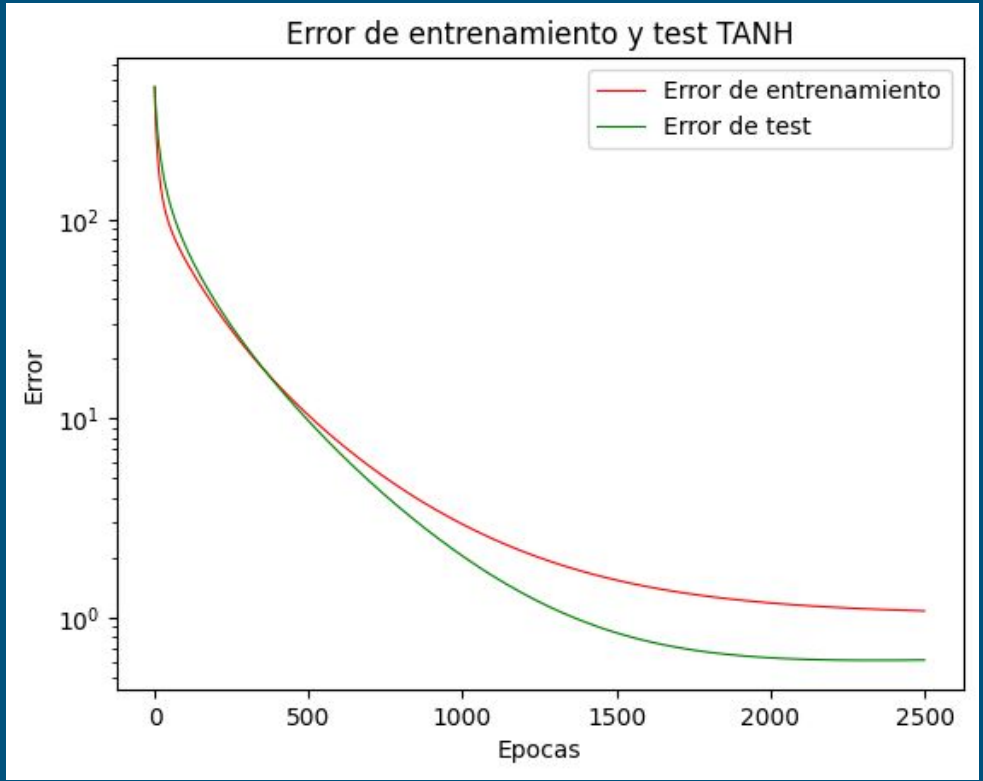
Error de Test: 0.2511152074416489



Combinación 1-3-4

- $\beta = 1$
- $\eta = 10^{-3}$
- $e_{max} = 2500$
- Batch

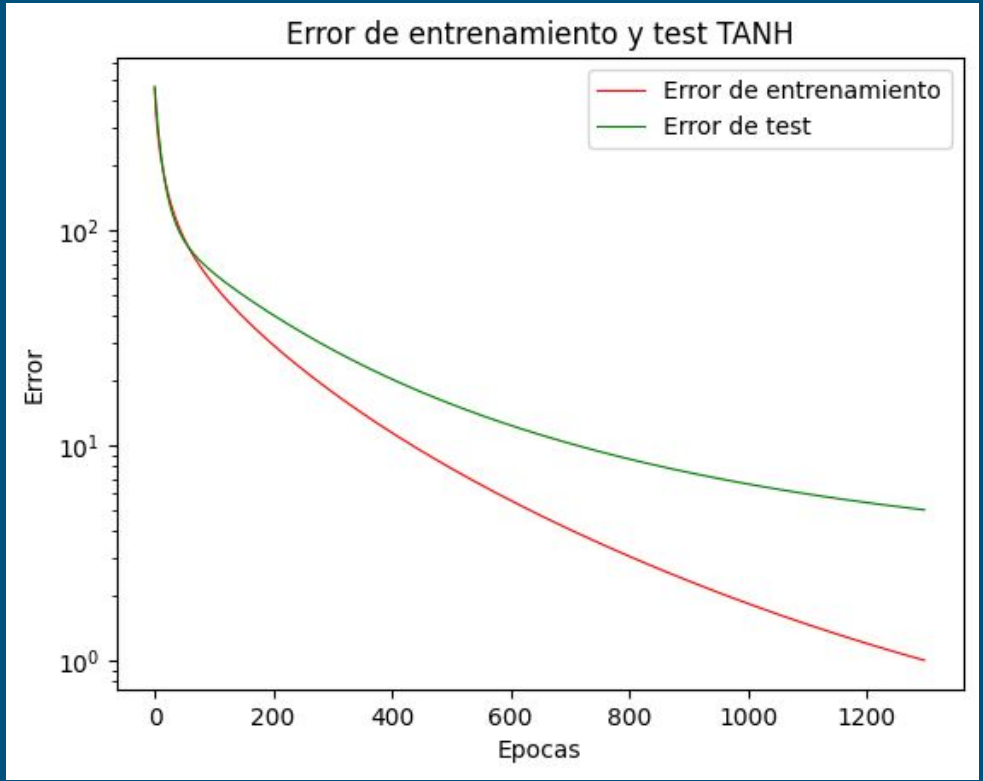
Error de Test: 0.6108202576427333



Combinación 2-3-4

- $\beta = 1$
- $\eta = 10^{-3}$
- $e_{max} = 2500$
- Batch

Error de Test: 5.008956151119848



Resumen de Resultados

Combinación	Error de Test
1-2-3	0.3667771461297337
1-2-4	**0.2511152074416489**
1-3-4	0.6108202576427333
2-3-4	5.008956151119848

Observamos que al quitar el Fold 1 del dataset de entrenamiento llegamos a un resultado de Generalización muy malo comparado con con las combinación que sí lo incluyen.

También observamos que la combinación del Fold 1 y 2 genera buenos resultados.

Observación!

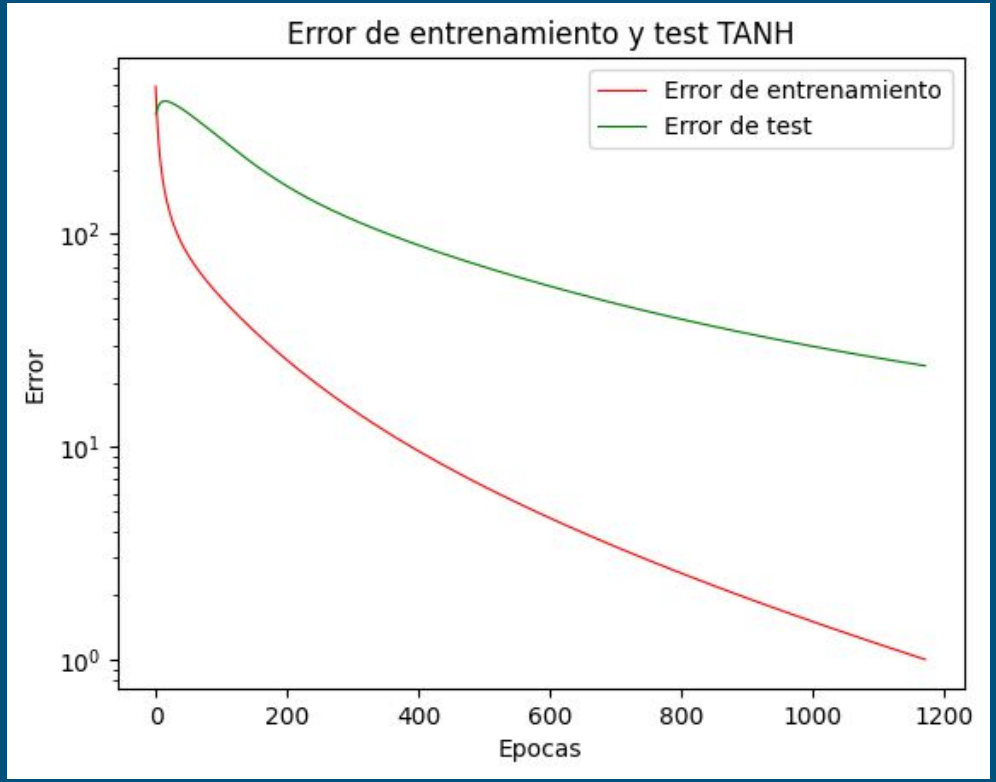
- Con los resultados anteriores, nos propusimos encontrar alguna característica en común entre el fold 1 y 2 que podría llegar a tener relación con los resultados obtenidos.
- Notamos que **ambos folds son los que suman mayor valor de salida.** Por lo tanto, nos propusimos crear dos nuevos experimentos, uno en donde el conjunto de entrenamiento cuente con los **elementos con mayor valor de salida**, y otro, donde el conjunto cuente con los **elementos de menor valor de salida**

Elementos de menor valor de salida

- $\beta = 1$
- $\eta = 10^{-3}$
- $e_{max} = 2500$
- Batch

Error de Test: 23.973516857410903

Hasta ahora, el peor error que se obtuvo



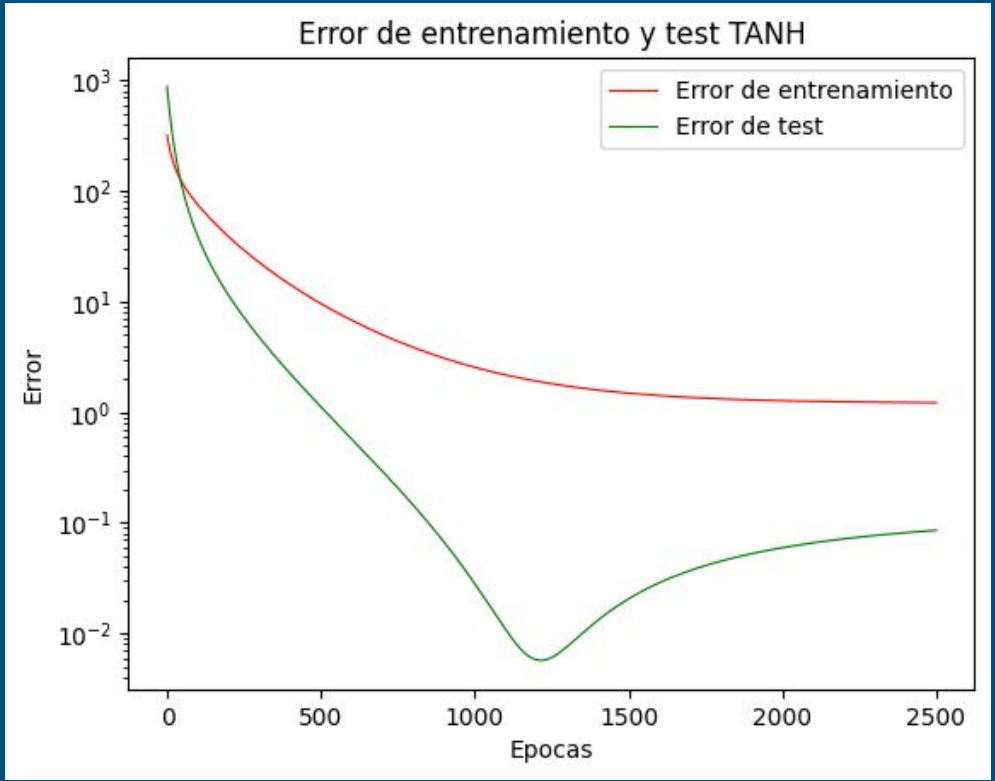
Elementos de mayor valor de salida

- $\beta = 1$
- $\eta = 10^{-3}$
- $e_{max} = 2500$
- Batch

Error de Test: 0.085668291727609 55

Se observa sobreajuste entre las épocas 1000 y 1500.

Los resultados en generalización son muy buenos, podemos observar que el error de test es menor al error de entrenamiento durante la mayor parte de las épocas



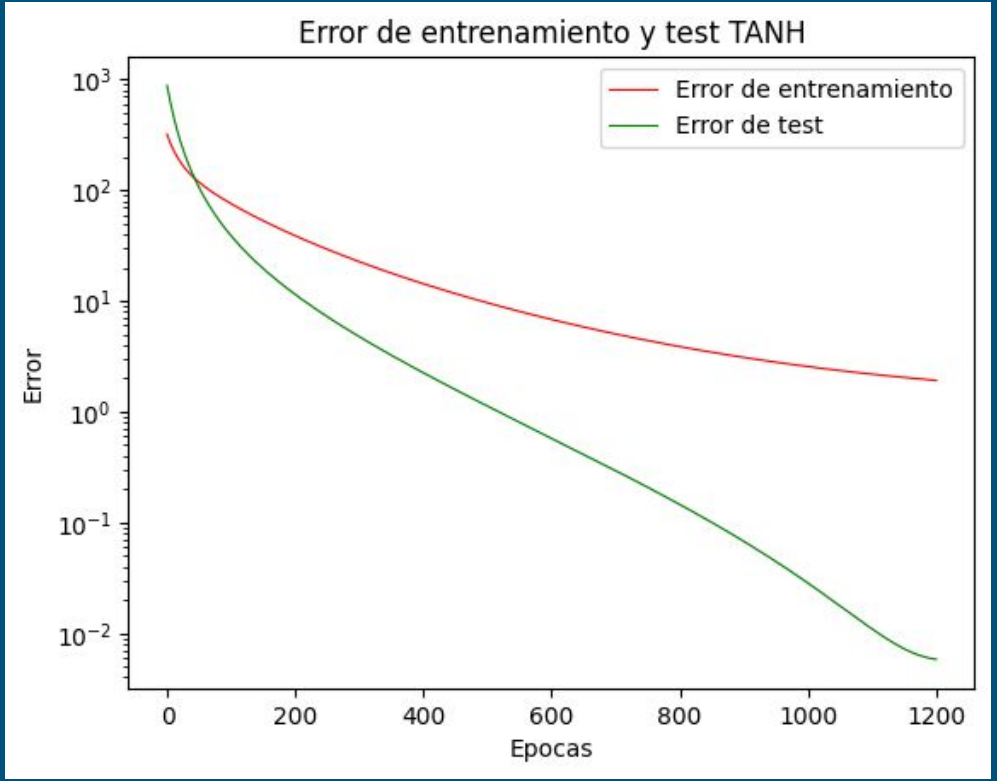
Elementos de mayor valor de salida

- $\beta = 1$
- $\eta = 10^{-3}$
- $e_{max} = 1200$
- Batch

Modificamos la condición de corte por épocas a 1200, para obtener el mejor modelo entrenado que observamos hasta el momento

Error de Test:
0.0057915422645573944 🥳

Mejor modelo para capacidad de Generalización 🥇



Conclusiones y aprendizajes

- El método de selección de elementos del dataset para aprendizaje y testing tiene una gran influencia en el entrenamiento del perceptrón. Un buen dataset, diverso y representativo, resultará en mejores resultados.
- Este problema particular se beneficia de utilizar altos porcentajes de los datos para entrenar.
- Sin embargo, siempre hay que tener en cuenta que un exceso de datos de training pueden causar un *sobreajuste*, donde el perceptrón se enfoca en reconocer los datos de entrenamiento particulares, perdiendo capacidad de generalización.

Ejercicio 3

Perceptrón Multicapa

Implementación

La clase `MultilayerPerceptron` recibe en su constructor una lista (capas) de listas (capa) de `Perceptron` y un optimizador que puede ser Momentum o Gradiente Descendente. El perceptrón sabe ajustarse y realizar el feedforward y el back propagation

```
MultilayerPerceptron(perceptrons, GradientDescent())
```

Implementación

Luego hay dos funciones:

```
def train_multilayer_perceptron(multilayer_perceptron: MultilayerPerceptron, dataset: list[np.ndarray[float]],
                                dataset_outputs: list[list[float]], config: TrainerConfig) -> MultilayerTrainerResult:
```

Para entrenar un perceptrón multicapa. Recibe el perceptrón, el input, el expected output y la configuración. Retorna un resultado que consta de la cantidad de épocas, el historial de pesos, el historial de errores y la razón de finalización.

```
def evaluate_multilayer_perceptron(multilayer_perceptron: MultilayerPerceptron, dataset: list[list[int]], dataset_outputs: list[list[int]],
                                   print_output: bool) -> dict[str, float | ndarray | list[list[int]]]:
```

Para testear al perceptrón luego de entrenarlo. Recibe el perceptrón, el input, el expected output y un booleano para imprimir o no el output. Retorna el error, el expected output (in, out) y el output del perceptrón.

Implementación

Existen archivos de configuración que permiten declarar los parámetros que utilizará el perceptrón multicapa.

(Cada perceptrón podría tener una función diferente pero por default utiliza la del archivo de configuración).

Ejercicio 3.A

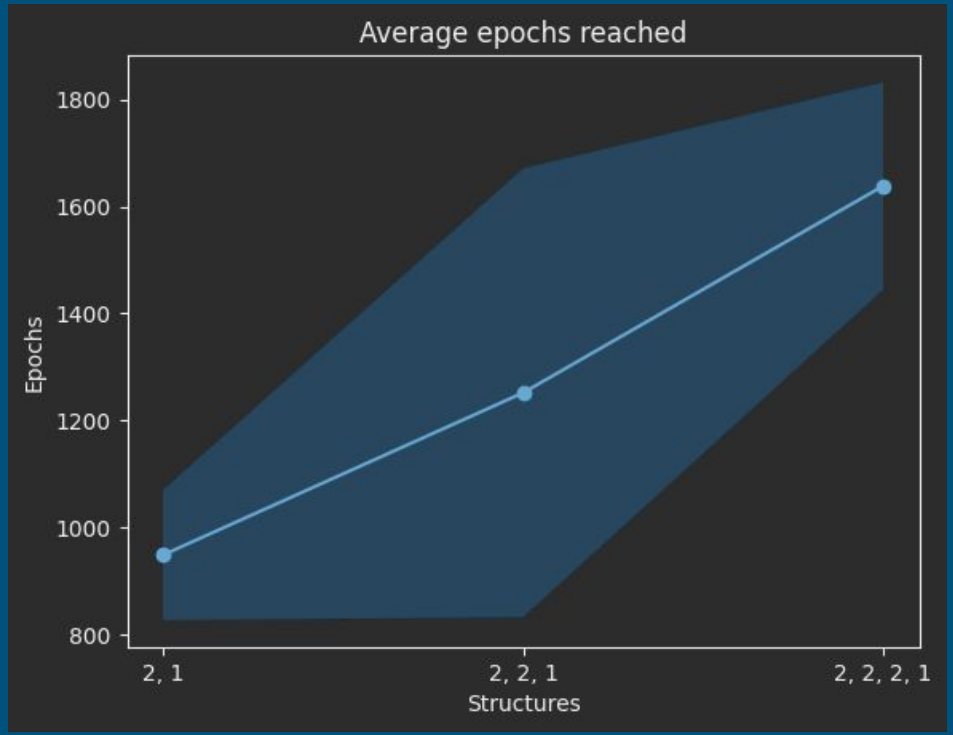
- Se utilizó la función de activación tanh
- Actualización de pesos batch
- Se utilizó una capa [2, 1] con método de gradiente descendente.

- [2, 1]
- Batch
- Tanh β = 1
- w ∈ [-0.4, 0.4]

Input	Expected	Actual
[1, 1]	-1	-0.92590
[1, -1]	1	0.946881
[-1, 1]	1	0.946817
[-1, -1]	-1	-0.92978
Error		0.008
Epochs		843

Variando la estructura

- Podemos ver que mientras mayor sea el número de capas y neuronas en el modelo, mayor será el número de épocas necesarias para que el modelo converja.



Conclusiones y aprendizajes

- Se puede resolver el XOR ya que el perceptrón multicapa permite resolver problemas complejos que no son linealmente separables.
- En general, el número de capas y neuronas en la red neuronal tiene un impacto en la velocidad de convergencia.
- Agregar más capas y neuronas no necesariamente conduce a una mejora en la misma. En algunos casos, como este, puede incluso ser perjudicial debido al aumento en la complejidad de la red y la posibilidad de sobreajuste.

Ejercicio 3.B

Definiciones y problemática

Caracterización del problema

- El input de la red es un mapa de bits de 7x5 vectorizado o aplanado en un arreglo de 35 posiciones.
- Tenemos 10 muestras disponibles, vamos a utilizar una parte para testeo y otra para aprendizaje.

A tener en cuenta

- Output: 1 (par) y -1 (impar). Definimos el rango [0,1] como par y [-1, 0) como impar.
- Las condiciones de corte disponibles para utilizar son las mismas que las mencionadas en el ejercicio 2.
- La función de error con respecto a la salida esperada utilizada es:

$$E(O) = \frac{1}{2}\sum_{i}(\zeta^{\mu} - O_{i}^{\mu})^{2}$$

Estructuras utilizadas

Capa de salida y de entrada

- Capa de salida: 1 neurona, representando el output anteriormente nombrado
- Capa de entrada: 35 neuronas, una por cada “pixel” en la imagen, que se corresponde con el input

Capas intermedias (approach)

- Empezamos con 1 capa intermedia con un número moderado de neuronas de 10 a 20. Luego, agregaremos una capa intermedia más para hacer una prueba con una estructura más compleja.
- Recordemos que agregar demasiadas capas o neuronas puede llevar a un sobreajuste (overfitting) de la red neuronal, donde esta se ajusta demasiado bien a los datos de entrenamiento y no generaliza bien nuevos inputs.

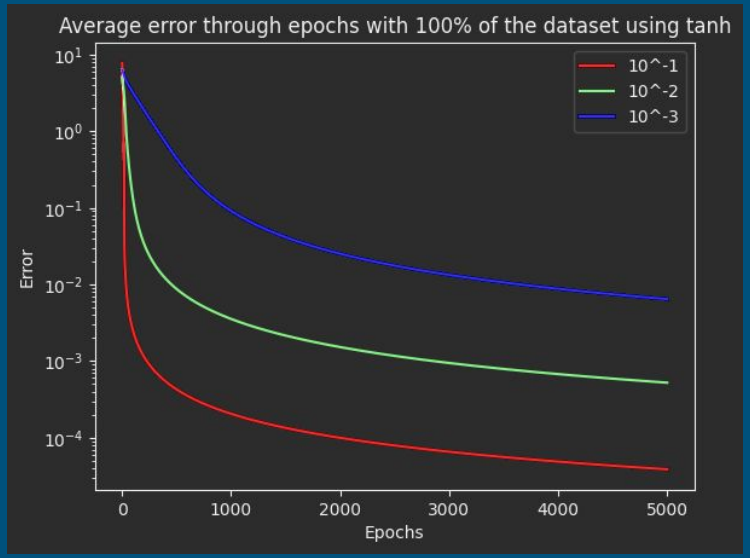
Elección de Tasa de Aprendizaje

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

- ➔ Convergencia en cantidad de épocas razonable.
- ➔ Minimizando oscilaciones en el error.
- ➔ Vamos a utilizar como función de activación tanh en todos los perceptrones.

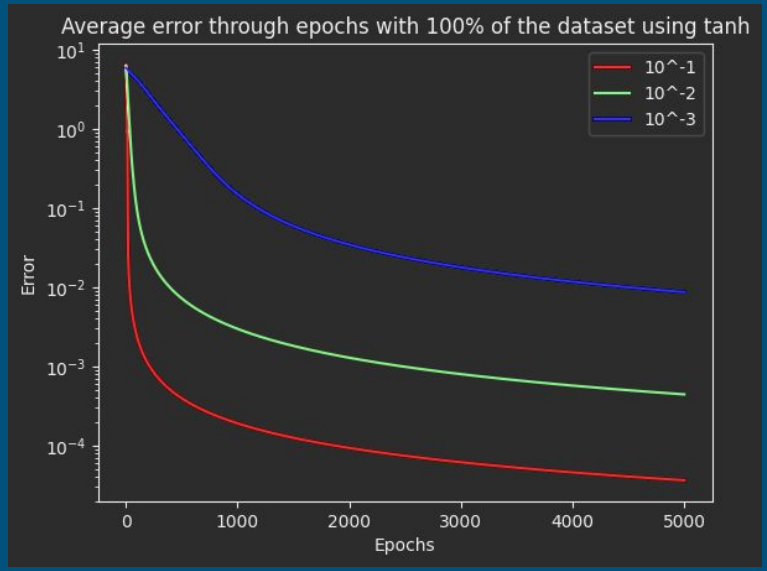
Nuestro error objetivo va a ser: 10^{-4}

Momentum



⇒ Una tasa mayor a 10^{-2} nos da una convergencia muy lenta comparado con valores inferiores. 10^{-1} nos da una convergencia muy rápida.

Gradiente Descendente



Análogo al caso anterior.
⇒ Tomamos 10^{-1}

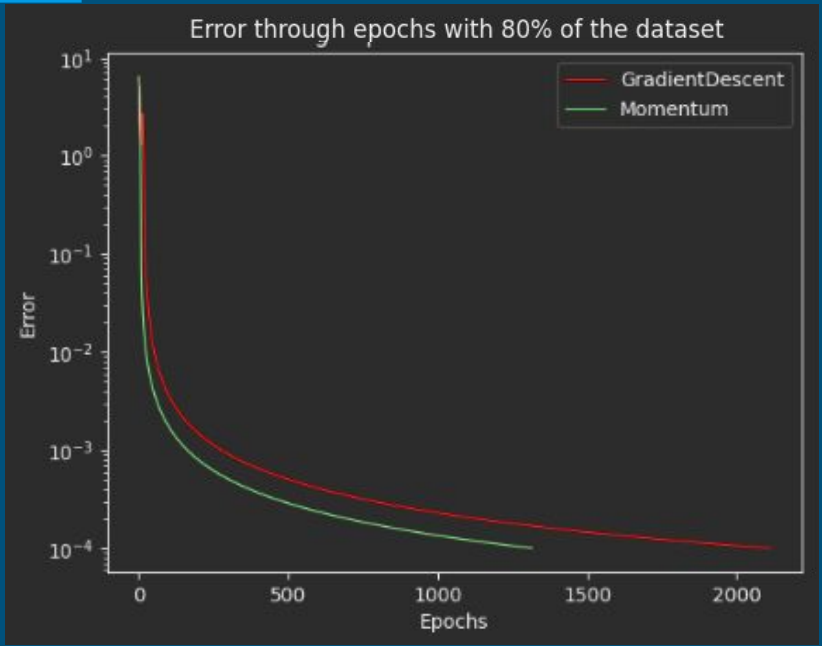
- [35, 15, 1]
- Incremental
- $\beta = 1$
- $w \in [-0.4, 0.4]$

Dataset para aprendizaje y testeo

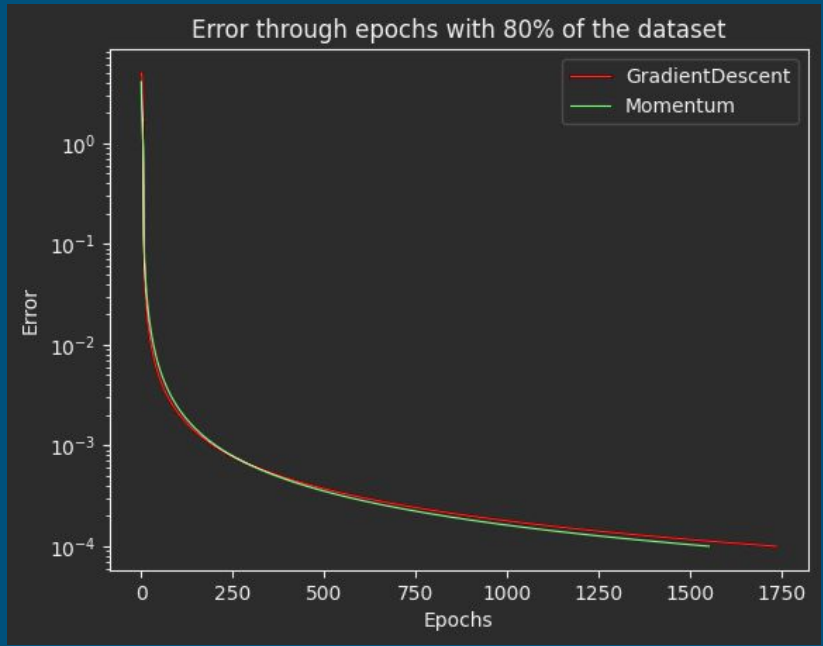
- Para la estructura mencionada [35, 15, 1] variamos el porcentaje del dataset utilizado para training y para testing:
  - 80% para training y 20% para testing
  - 50% para ambos
  - 20% para training y 80% para testing.
- Por último, tomaremos una estructura [35, 15, 15, 1] y haremos un análisis similar para ver si llegamos a resultados parecidos.
- Se utilizará el η definido en la filmina anterior. El resto de los parámetros permanecen idénticos.

Aprendizaje y comparación de métodos (I)

Error con el 80% del dataset



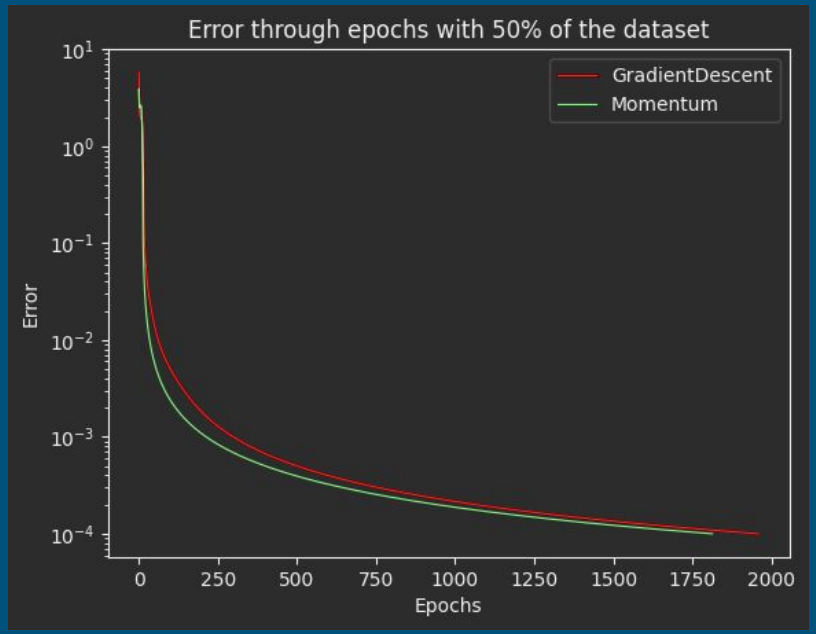
[35, 15, 1]



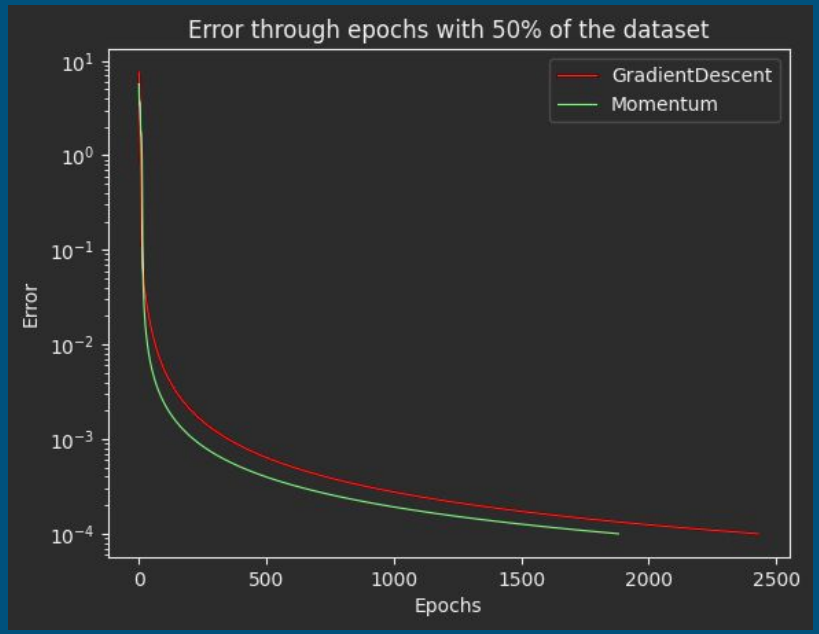
[35, 15, 15, 1]

Aprendizaje y comparación de métodos (II)

Error con el 50% del dataset



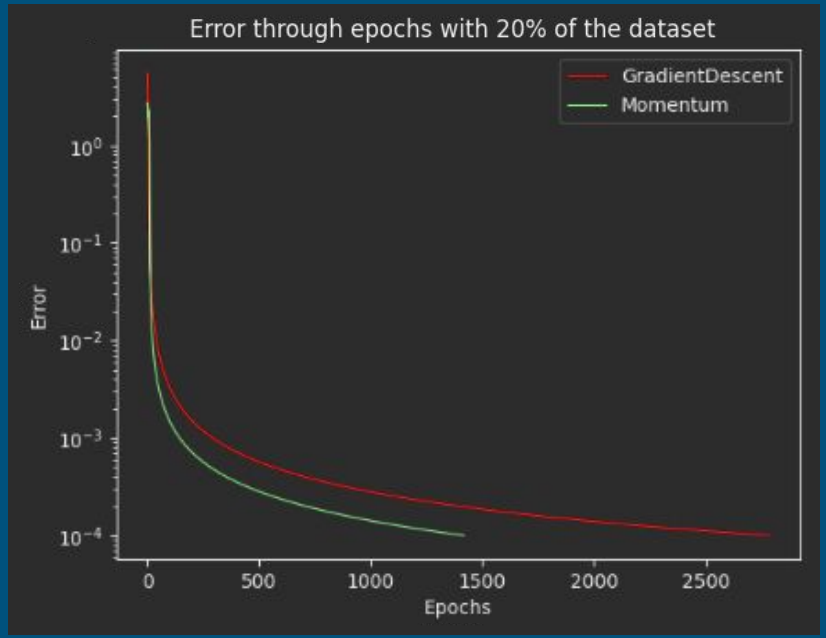
[35, 15, 1]



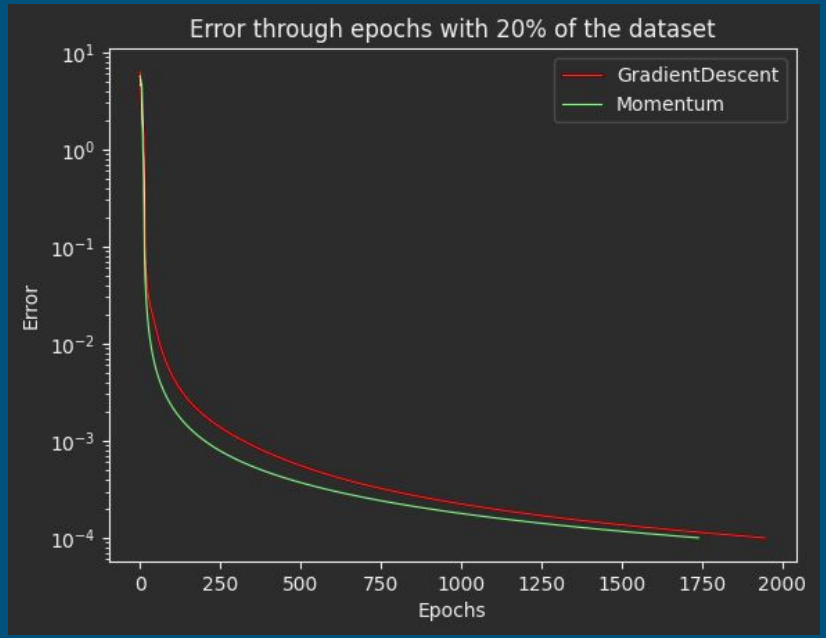
[35, 15, 15, 1]

Aprendizaje y comparación de métodos (III)

Error con el 20% del dataset



[35, 15, 1]

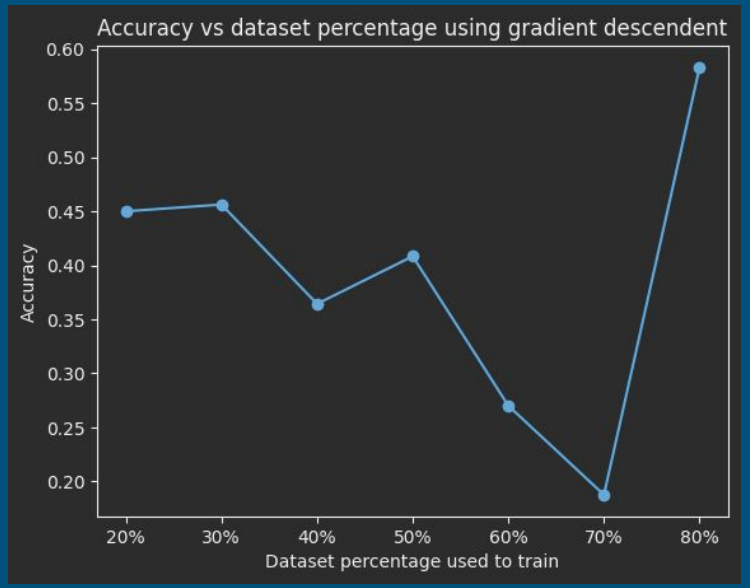


[35, 15, 15, 1]

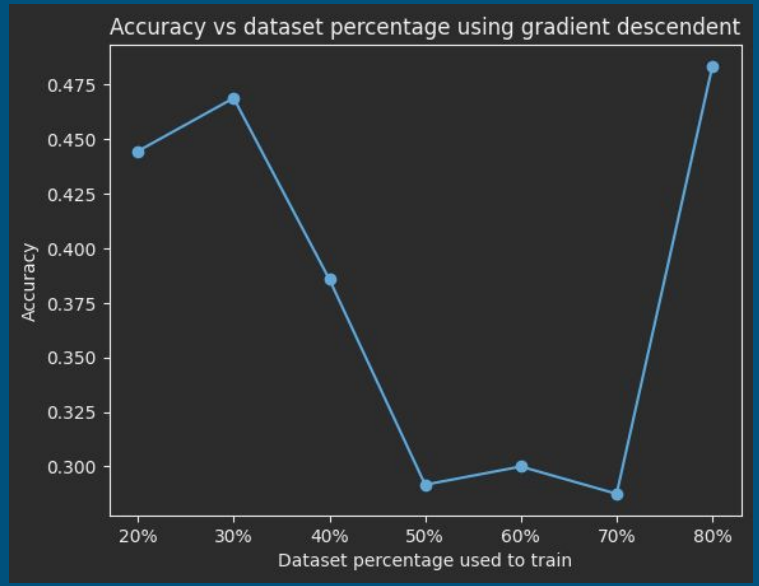
Aprendizaje: Comparación de Métodos

- ➔ Darle más porcentaje de dataset como input no indica una convergencia más rápida en épocas.

Generalización y comparación de métodos y estructuras (I)



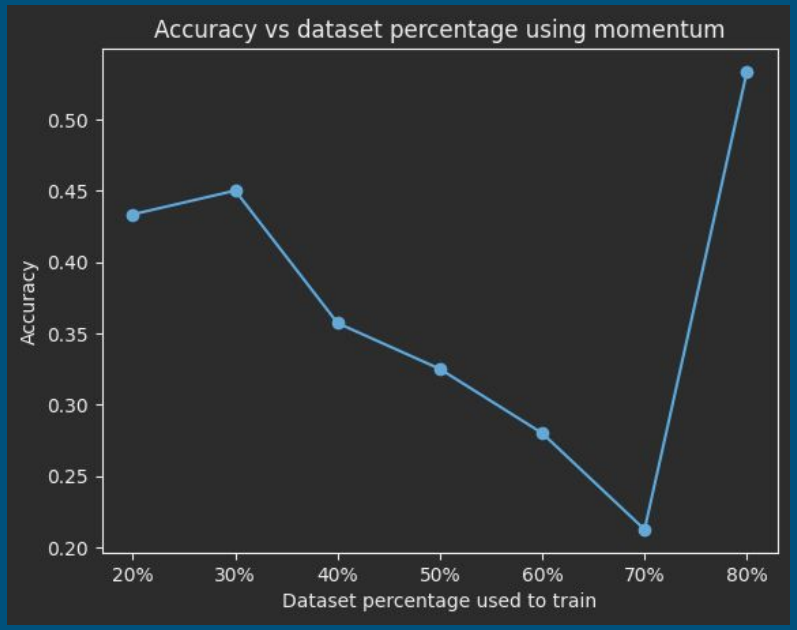
[35, 15, 1]



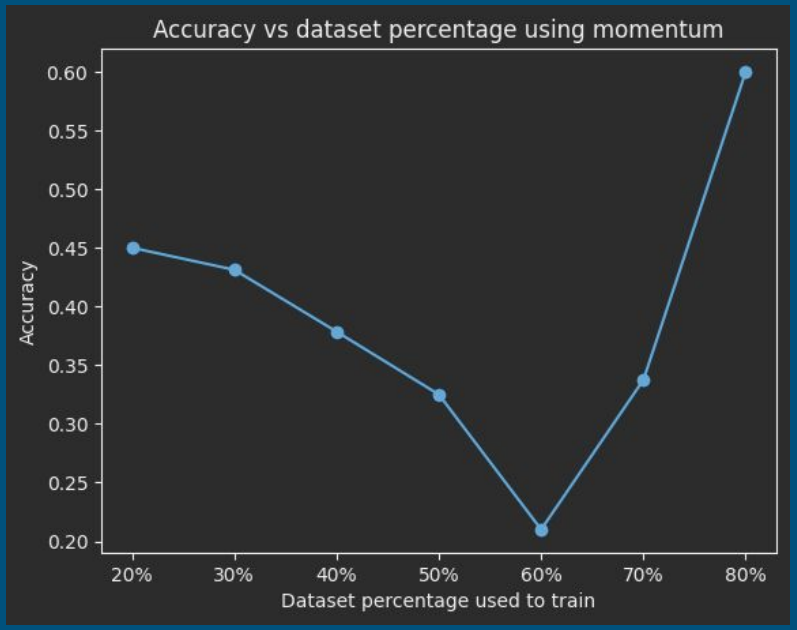
[35, 15, 15, 1]

num_runs = 30

Generalización y comparación de métodos y estructuras (II)



[35, 15, 1]



[35, 15, 15, 1]

num_runs = 30

¿Qué está pasando? (Conclusión)

¡El perceptrón está “adivinando”!

Adivinar la paridad de un número a partir de su mapa de dígitos es lo mismo que la probabilidad de lanzar una moneda. La red no está logrando generalizar correctamente.

Ejercicio 3.C

Comparación de Tasa de Aprendizaje (η)

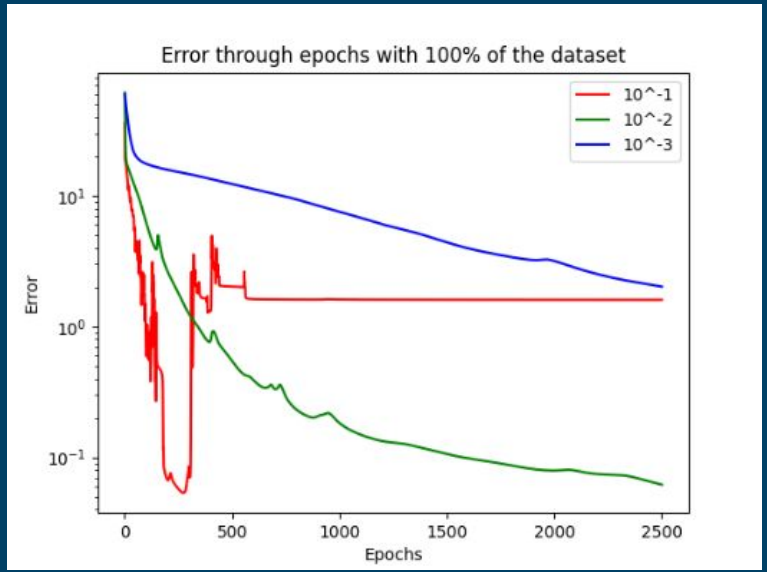
Para cada uno de los metodos de optimizacion, buscamos cuál es una tasa de aprendizaje razonable teniendo en cuenta:

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

- [35, 15, 10]
- Incremental
- Tanh β = 1
- error < 0.05
- epochs < 2500
- w ∈ [-0.4, 0.4]

Gradiente Descendente

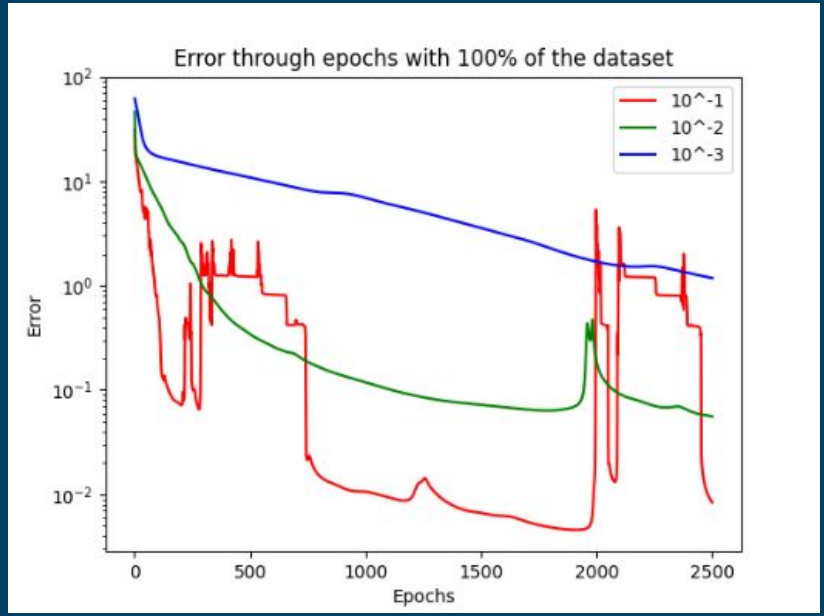
- ➔ $\eta = 10^{-3}$ presenta errores mayores a 1
- ➔ $\eta = 10^{-2}$ presenta errores razonables del orden de 10^{-1} en mayor cantidad de épocas.
- ➔ $\eta = 10^{-1}$ en 400 épocas llega a un error de orden menor que el $\eta = 10^{-2}$. Sin embargo…



Gradiente Descendente

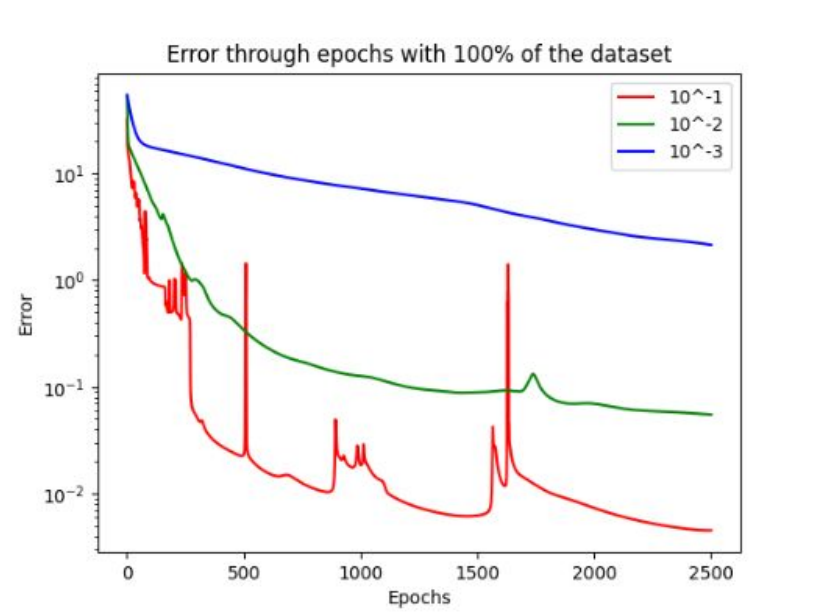
Sin embargo…

Puede que llegue a una región donde los gradientes son muy grandes y el paso de actualización es demasiado grande, lo que hace que el algoritmo salte entre diferentes áreas del espacio de parámetros, sin poder converger a un mínimo global.



Momentum

- $\eta = 10^{-3}$ presenta errores abismales
- $\eta = 10^{-2}$ presenta errores razonables del orden de 10^{-1}.
- Pero $\eta = 10^{-3}$ en la misma cantidad de épocas que 10^{-2} obtiene errores de un orden menor. Sin embargo, presenta *jittering*.



Sobre qué η elegir

Se puede elegir 10^{-2} y aumentar el límite de épocas ⇒ mayor tiempo en ejecución

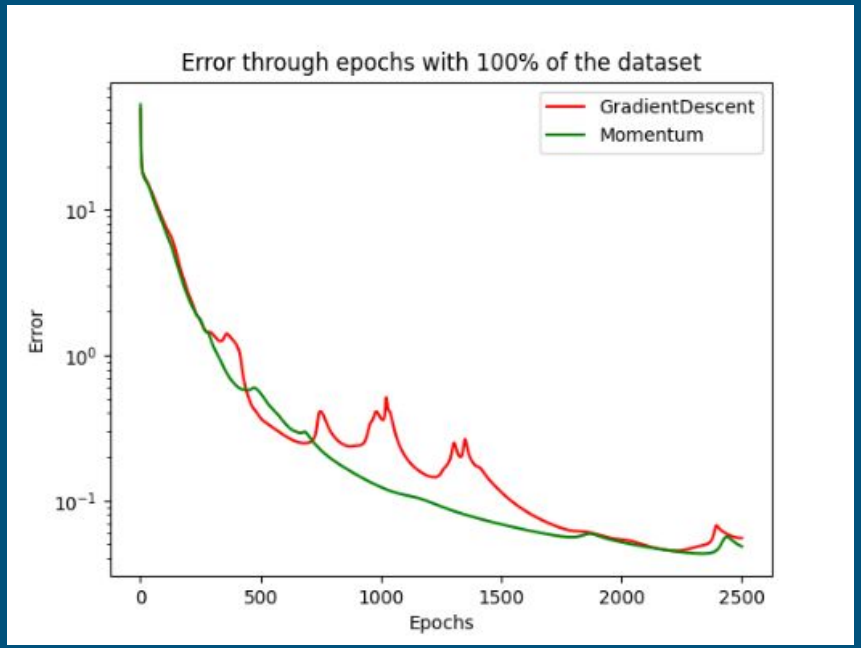
ó

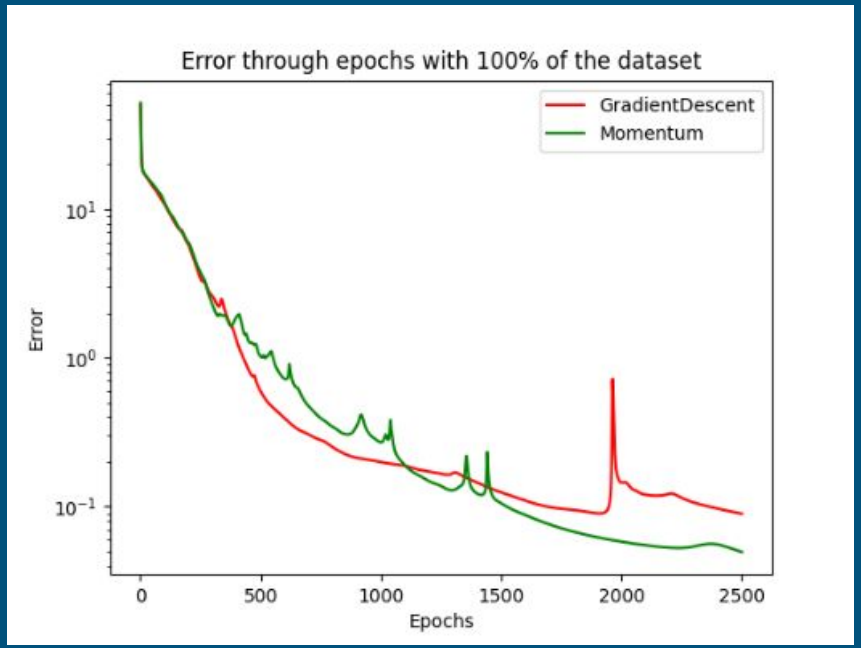
Se puede elegir 10^{-1} y tomar sólo los que converjan a un determinado error, sabiendo que puede haber jittering.

Aprendizaje y comparación de métodos (I)

Error con el 100% del dataset

- Incremental
- Tanh $\beta = 1$
- $\eta = 0.01$
- epochs < 2500
- $w \in [-0.4, 0.4]$





[35, 15, 10]

num_runs = 5

[35, 15, 15, 10]

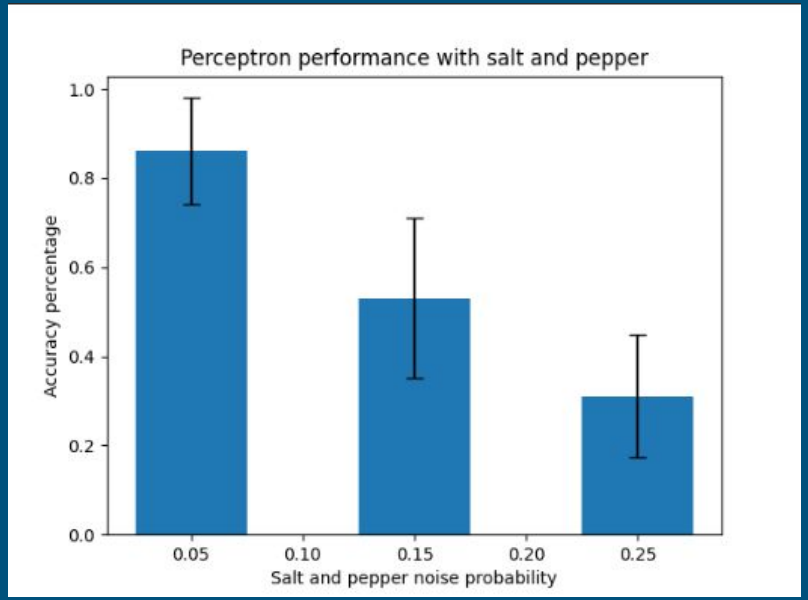
Aprendizaje: Comparación de Métodos

Momentum (α = 0.9) muestra mejores resultados de convergencia que Gradiente Descendente.

Generalización y Funciones de Ruido

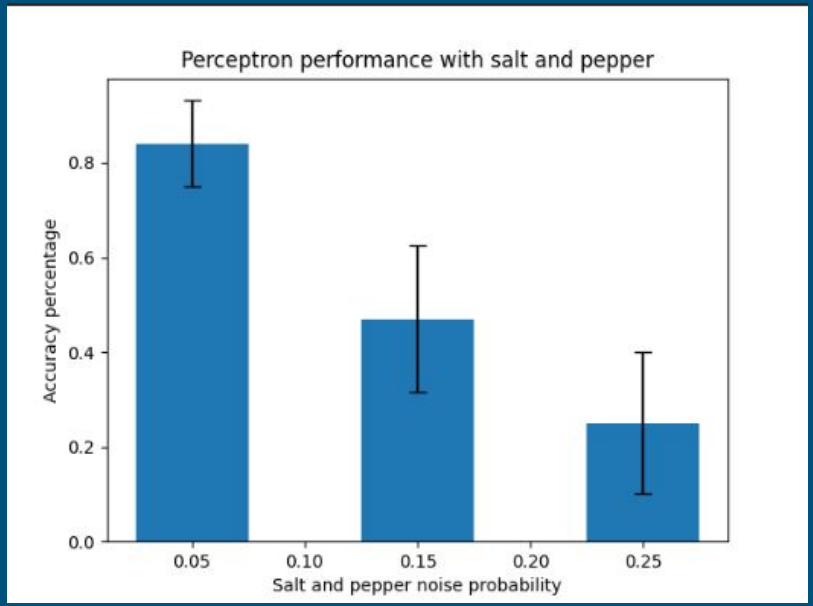
- Incremental
- Tanh β = 1
- η = 0.01
- e < 0.05
- epochs < 2500
- w $\in$ [-0.4, 0.4]

Generalización y comparación de estructuras (I)
Funciones de ruido (I)
Salt and Pepper



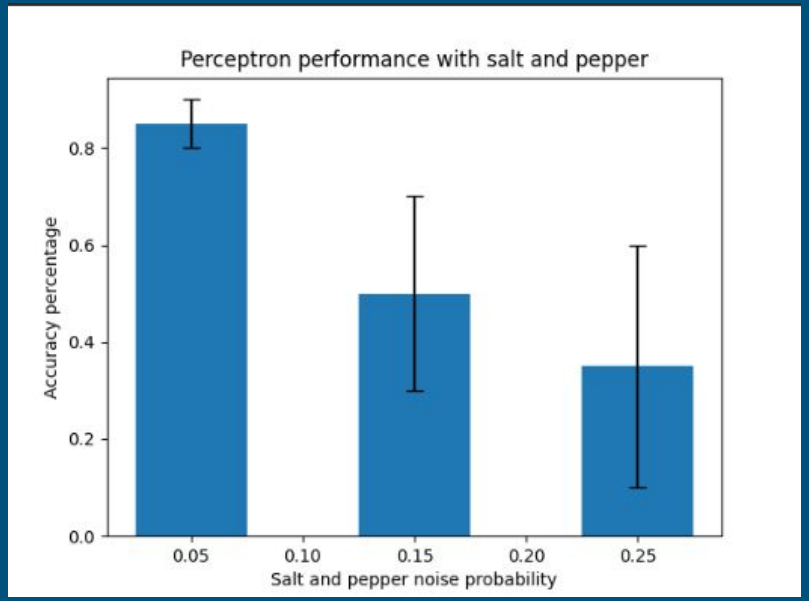
[35, 15, 10]

num_runs = 10



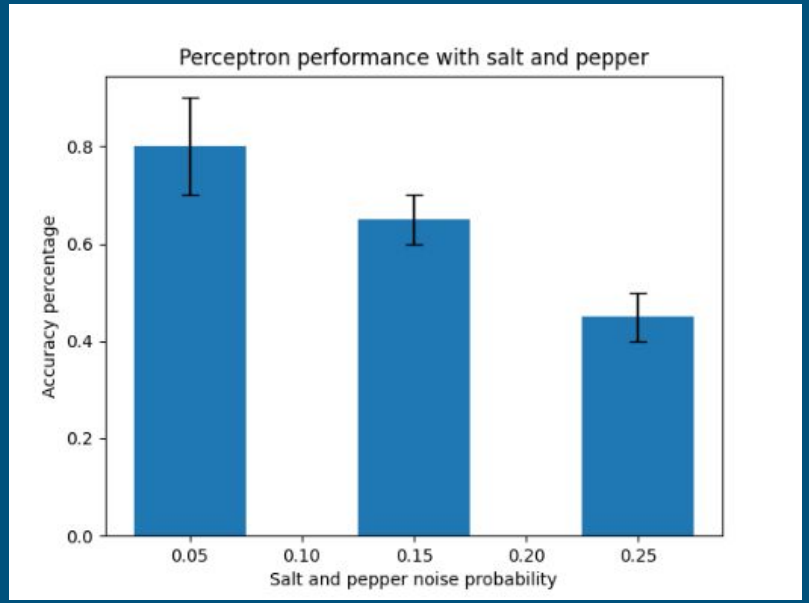
[35, 15, 15, 10]

Generalización y comparación de estructuras (II)
Funciones de ruido (I)
Salt and Pepper



[35, 15, 15, 15, 10]

num_runs = 2



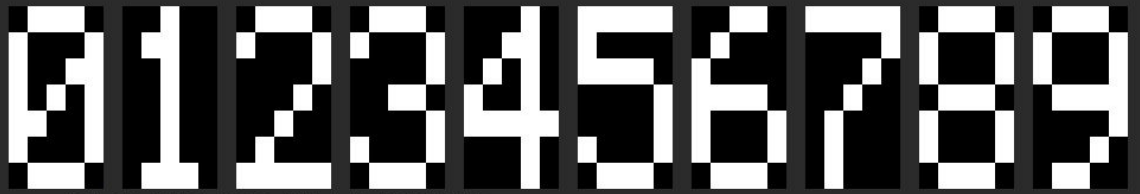
[35, 15, 15, 15, 15, 10]

Output (I)

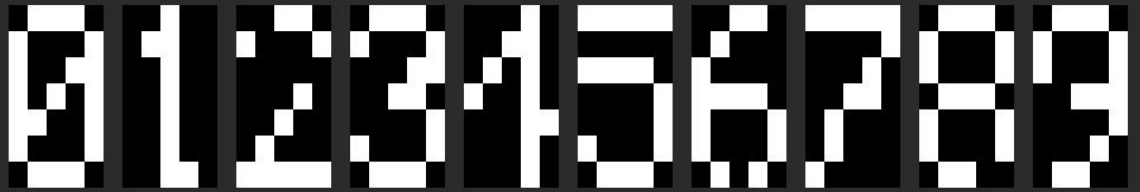
Salt and Pepper

prob = 0.05

```
----- Original Images -----
```



```
----- Salt and Pepper Images with 0.05 probability-----
```



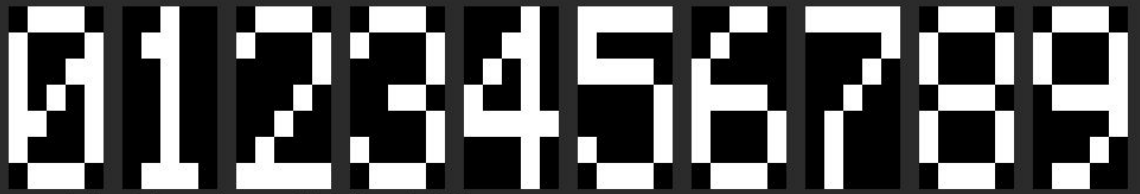
```
----- Evaluating after training -----

For the image of the 0 digit, the perceptron interpreted a 0
For the image of the 1 digit, the perceptron interpreted a 1
For the image of the 2 digit, the perceptron interpreted a 2
For the image of the 3 digit, the perceptron interpreted a 3
For the image of the 4 digit, the perceptron interpreted a 4
For the image of the 5 digit, the perceptron interpreted a 5
For the image of the 6 digit, the perceptron interpreted a 6
For the image of the 7 digit, the perceptron interpreted a 7
For the image of the 8 digit, the perceptron interpreted a 6
For the image of the 9 digit, the perceptron interpreted a 9
```

Output (I)

Salt and Pepper

prob = 0.15

```
----- Original Images -----
```



```
----- Salt and Pepper Images with 0.15 probability-----
```

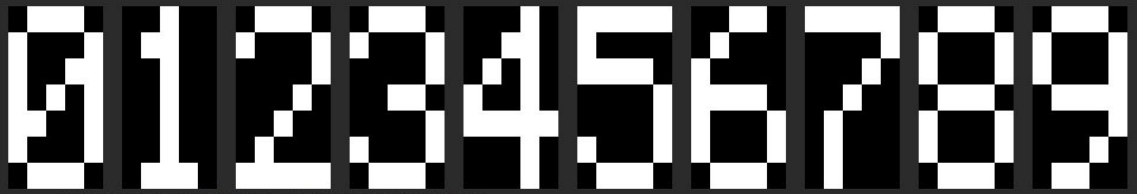
```
----- Evaluating after training -----

For the image of the 0 digit, the perceptron interpreted a 0
For the image of the 1 digit, the perceptron interpreted a 1
For the image of the 2 digit, the perceptron interpreted a 2
For the image of the 3 digit, the perceptron interpreted a 3
For the image of the 4 digit, the perceptron interpreted a 4
For the image of the 5 digit, the perceptron interpreted a 9
For the image of the 6 digit, the perceptron interpreted a 1
For the image of the 7 digit, the perceptron interpreted a 7
For the image of the 8 digit, the perceptron interpreted a 4
For the image of the 9 digit, the perceptron interpreted a 0
```

Output (I)

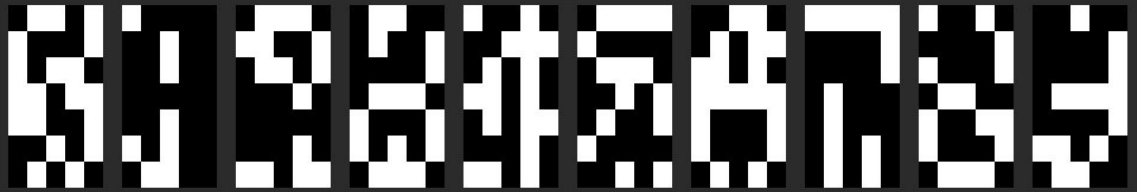
Salt and Pepper

prob = 0.25

```
----- Original Images -----
```



```
----- Salt and Pepper Images with 0.25 probability-----
```

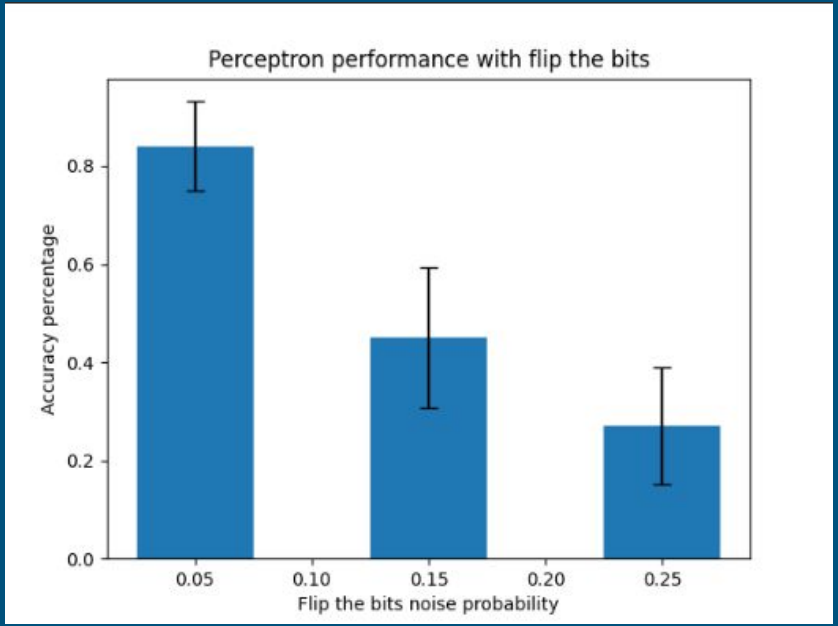


```
----- Evaluating after training -----

For the image of the 0 digit, the perceptron interpreted a 7
For the image of the 1 digit, the perceptron interpreted a 5
For the image of the 2 digit, the perceptron interpreted a 2
For the image of the 3 digit, the perceptron interpreted a 8
For the image of the 4 digit, the perceptron interpreted a 4
For the image of the 5 digit, the perceptron interpreted a 9
For the image of the 6 digit, the perceptron interpreted a 6
For the image of the 7 digit, the perceptron interpreted a 7
For the image of the 8 digit, the perceptron interpreted a 8
For the image of the 9 digit, the perceptron interpreted a 6
```

Funciones de ruido (II)

Flip the bits

num_runs = 10

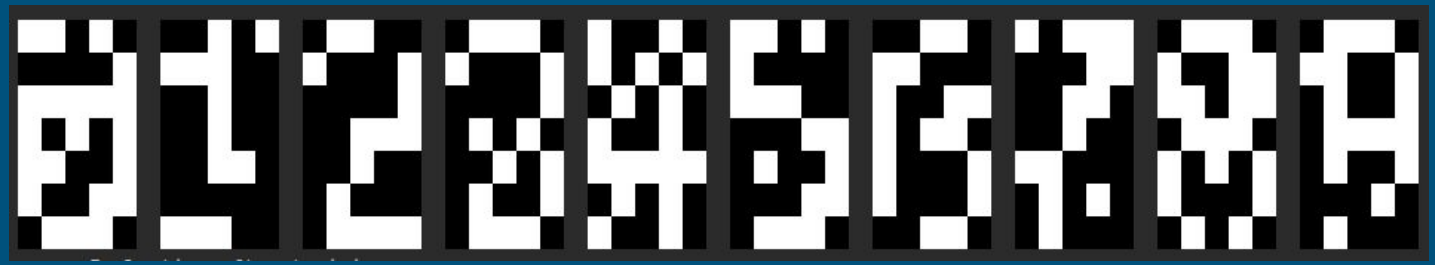


[35, 15, 10]

Output (II)

Flip the bits

prob = 0.05



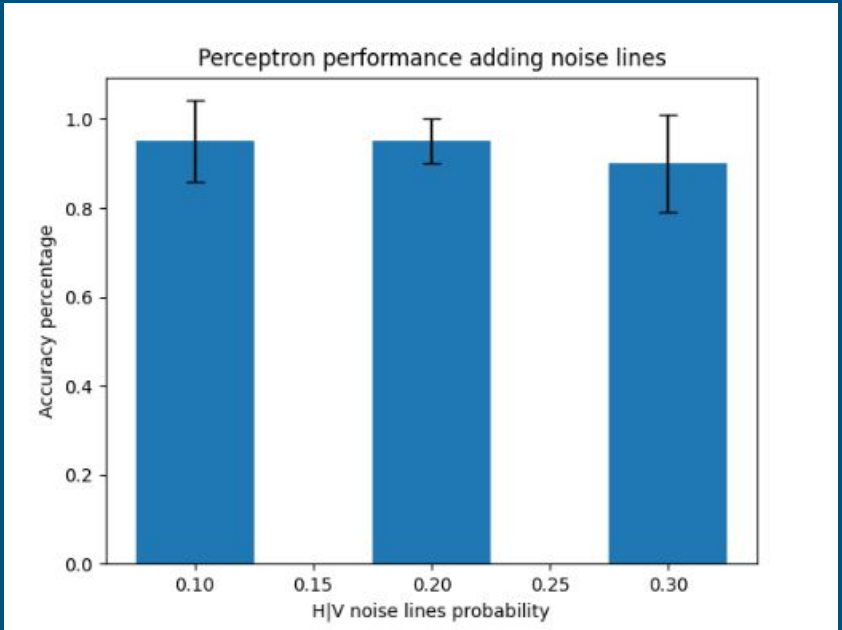
prob = 0.15

prob = 0.25

Funciones de ruido (III)

Add noise lines

num_runs = 10

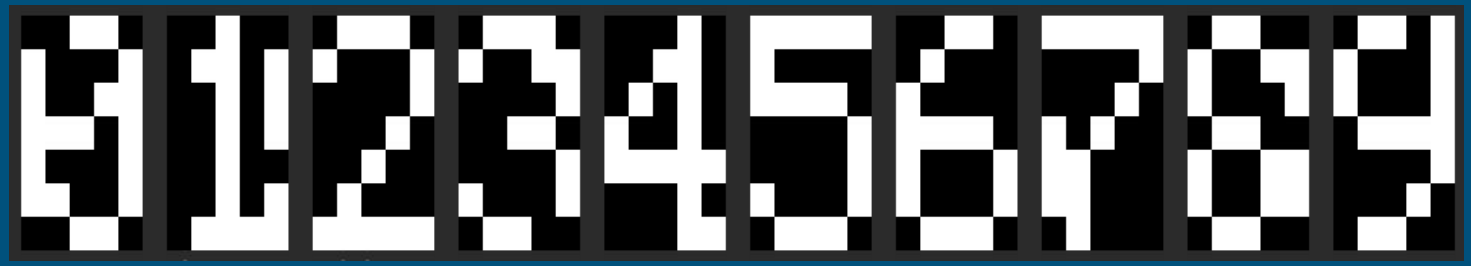


[35, 15, 10]

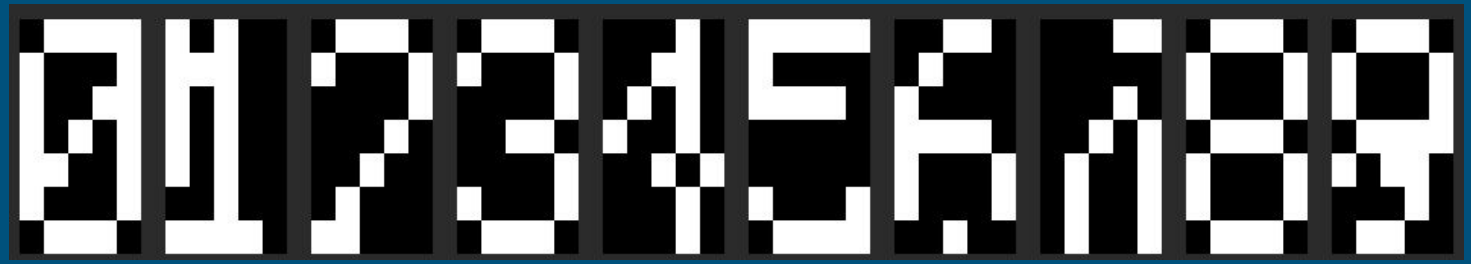
Output (III)

Add noise lines

prob = 0.1



prob = 0.2

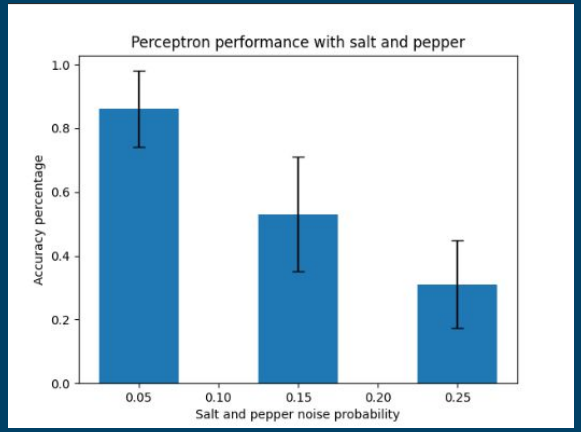


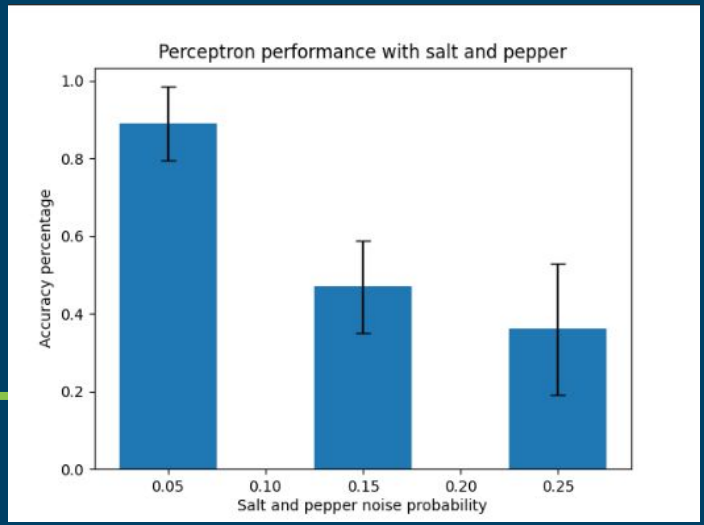
prob = 0.3

Más entrenamiento

Imágenes originales
+
30 imágenes con Add Lines
probabilidad = 0.2

Más entrenamiento no quiere decir mejores resultados. Depende de muchas variables.





Conclusiones

- A partir de un número de hidden layers, y cantidad de perceptrones en estas, no tiene sentido seguir aumentando estas cantidades para obtener un accuracy mejor, proporcionalmente al tiempo que se gasta al aumentar esas cantidades.

- Este problema si es generalizable, igualmente se debería seguir entrenando al perceptrón con imágenes con ruido que efectivamente se pueda entender de qué dígito se habla, para que el perceptrón pueda comportarse mejor en futuros testeos. Sin embargo, esto puede conducir a un sobreajuste, por lo que se deberán usar estrategias para evitar esto.

- Tener varias funciones de ruido y que estas sean parametrizables, para poder evaluar imágenes con ruido, es importante para comprobar cómo se comporta el perceptrón luego de ser entrenado y hasta qué punto da unos resultados “aceptables”.

Preguntas?