



SIA - TP3

Perceptrón Simple y Multicapa

Grupo 1

Alberto Bendayan
Tobias Ves Losada
Cristian Tepedino
Luca Bloise



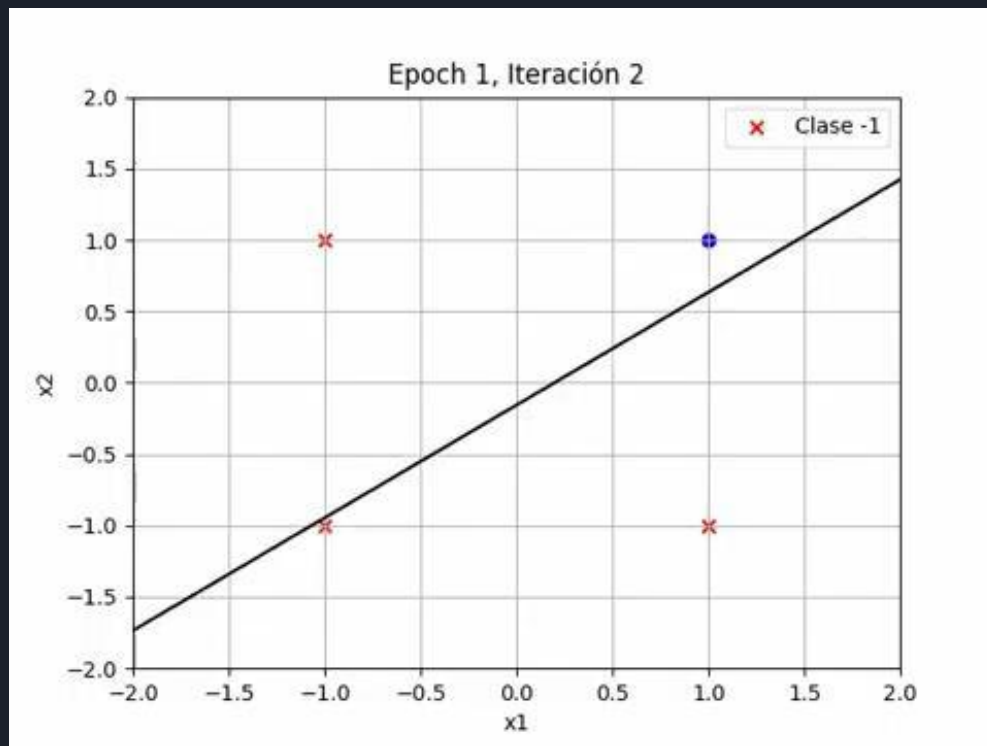
Ejercicio 1



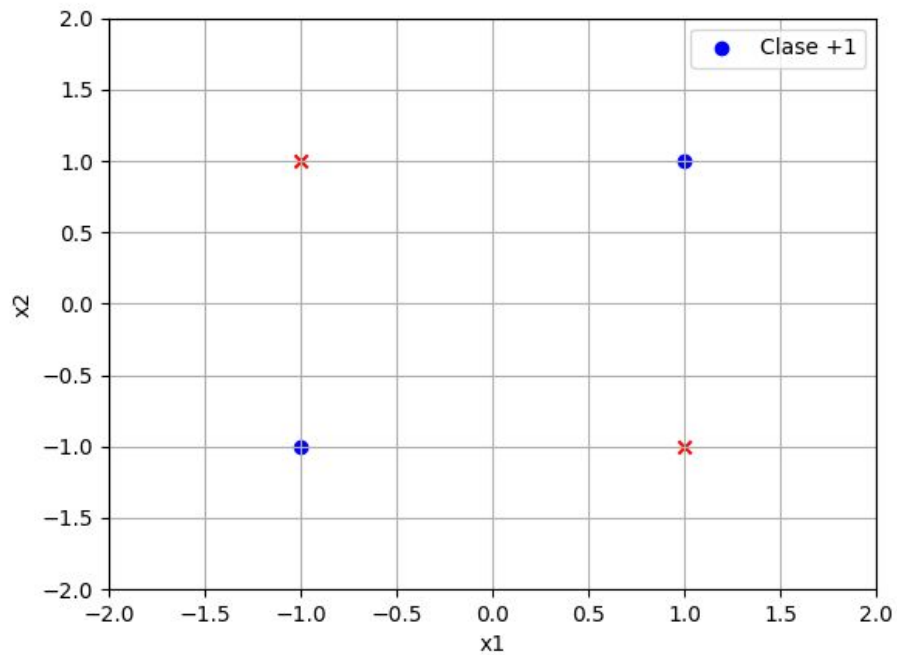
Definimos

- **Learning rate:** 0.1
- **Épocas máximas:** 1000
- **Bias y pesos iniciales:** random entre -1 y 1

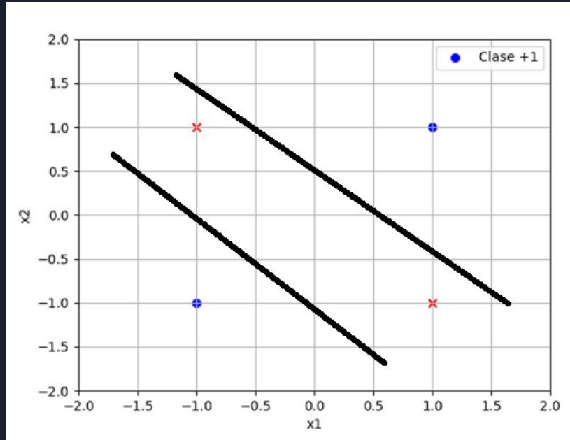
AND



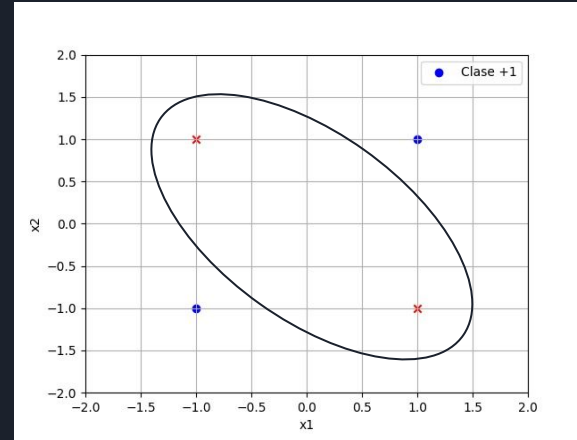
XOR - Problema



XOR - Soluciones



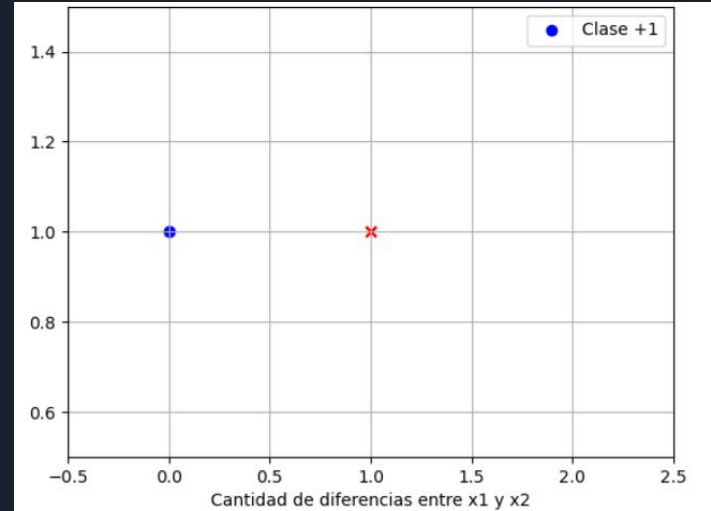
Combinación de perceptrones
simples



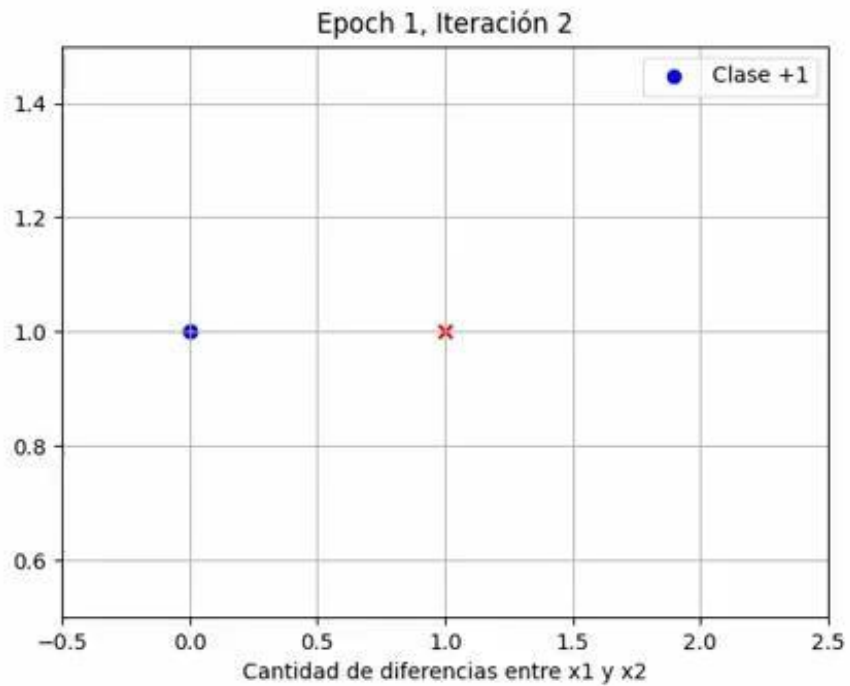
Perceptrón simple no lineal

XOR - Nuestra solución

- Transformamos la entrada
- Entran dos valores y la salida es la cantidad de diferencias
- Buscamos una recta que divida estos valores



XOR





Ejercicio 2



Parámetros

Función de activación	Identidad (Lineal)	Sigmoidea	Tanh
Learning Rate	0.001		
Max Epochs	100000		
Beta	-	1	
Inicialización de pesos	Aleatorios en (-1, 1)		



Normalización de inputs

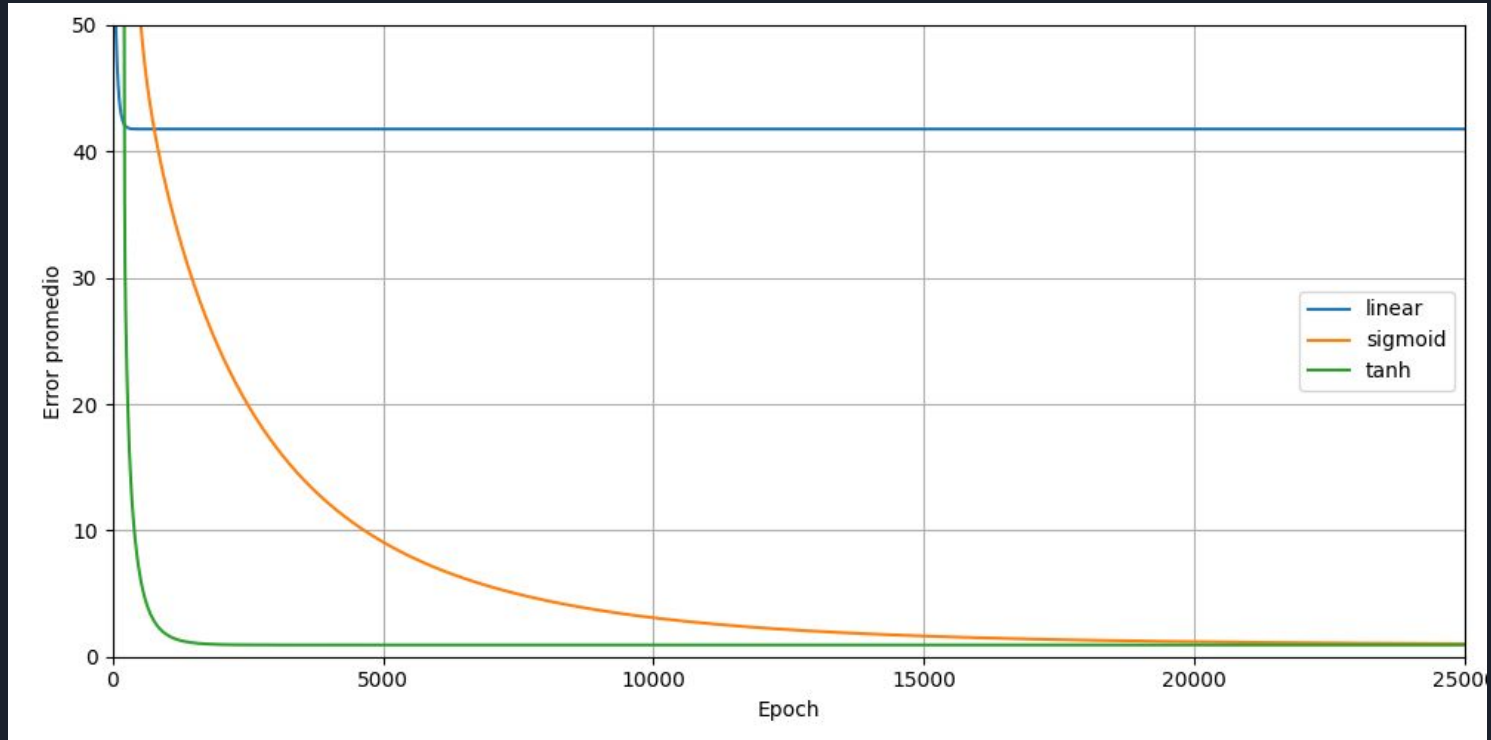
Función de activación	Lineal	Sigmoidea	Tanh
Rango	$(-\infty, +\infty)$	$(0, 1)$	$(-1, 1)$

Para poder usar las funciones no lineales con el dataset, tenemos que normalizar las salidas al rango de la función (transformación lineal)

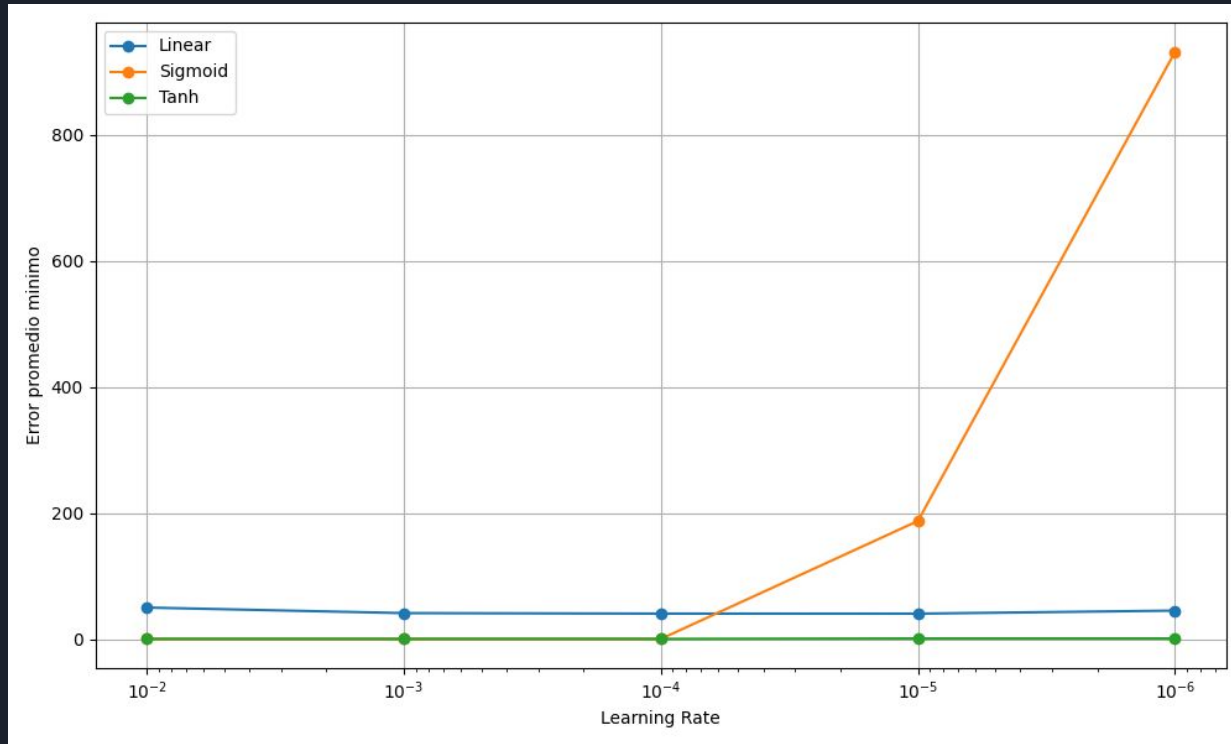
Luego, se hace el proceso opuesto para obtener la salida predicha en los rangos del dataset

El error se calcula con los valores sin normalizar, a fin de poder compararlo entre perceptrones

Comparación de errores por epoch



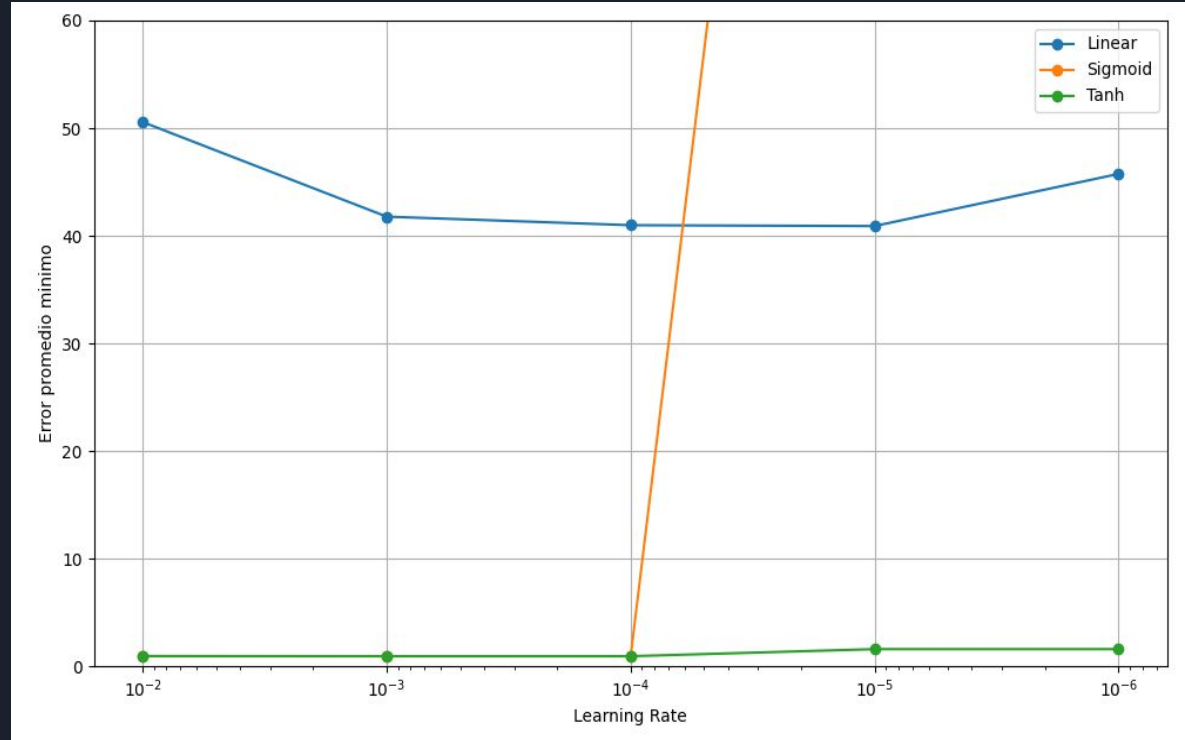
Comparación de errores por learning rate



Comparación de errores por learning rate

Al bajar el learning rate, vemos que permite alcanzar un menor error mínimo alcanzado.

Pero, como tarda mucho más en converger, para valores muy bajos, incluso con muchas epochs no se llega al error de learning rates más altos.





Conclusiones (del aprendizaje)

- El perceptrón lineal no puede aprender bien el dataset
- Tanh llega a un error bajo de forma rápida
- Sigmoidea tarda más epochs, pero eventualmente se acerca al error que se obtiene con tanh
- Un learning rate más bajo hace que eventualmente se llegue a un error muy ligeramente menor, pero eso solo ocurre tras muchas épocas



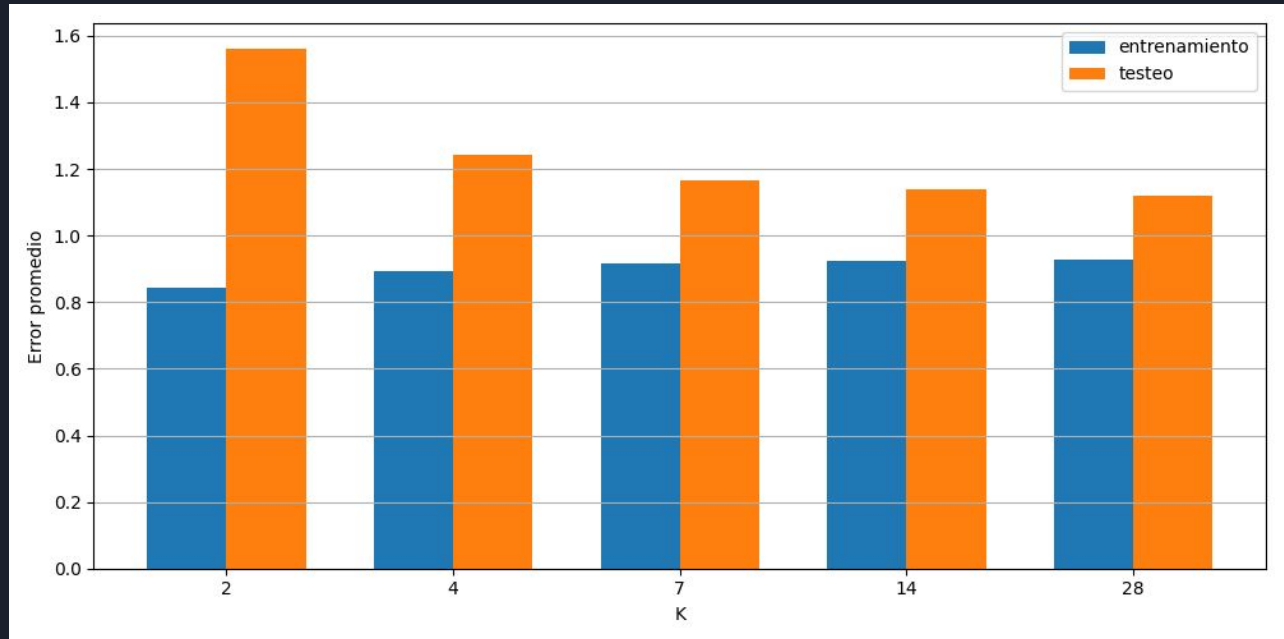
Capacidad de generalización

- Vamos a probar que tan bien pueden generalizar el dataset los perceptrones no lineales
- Se usará K-fold cross validation, tomando como métrica el error cuadrático medio, y se variara el valor de K para ver cómo influye en la capacidad de generalización
- Para obtener resultados representativos, se tomará la media de 100 ejecuciones de la validación, y la distribución del dataset en las particiones será aleatoria

Comparación de error de testeo según cantidad de particiones

Parámetros:

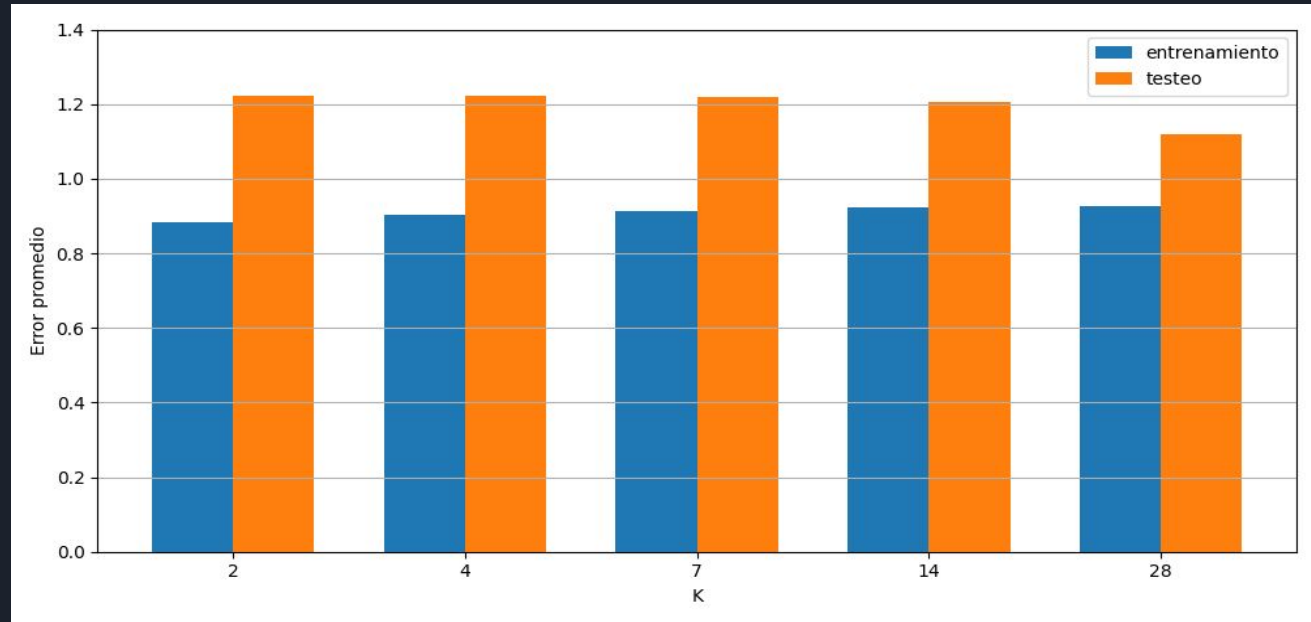
- Activación: Tanh
- Epochs: 100000
- Learning Rate: 0.001



Comparación de error de testeo según cantidad de particiones

Parámetros:

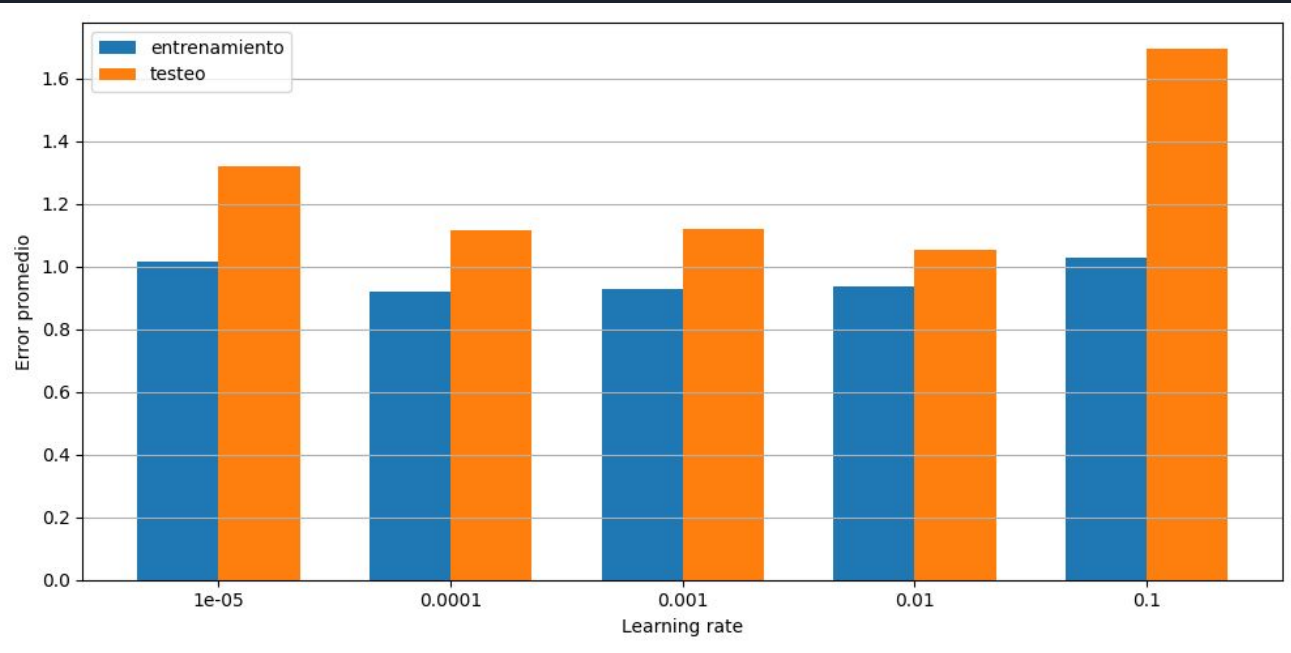
- Activación: Sigmoid
- Epochs: 100000
- Learning Rate: 0.001



Comparación de error de testeo según learning rate

Parámetros:

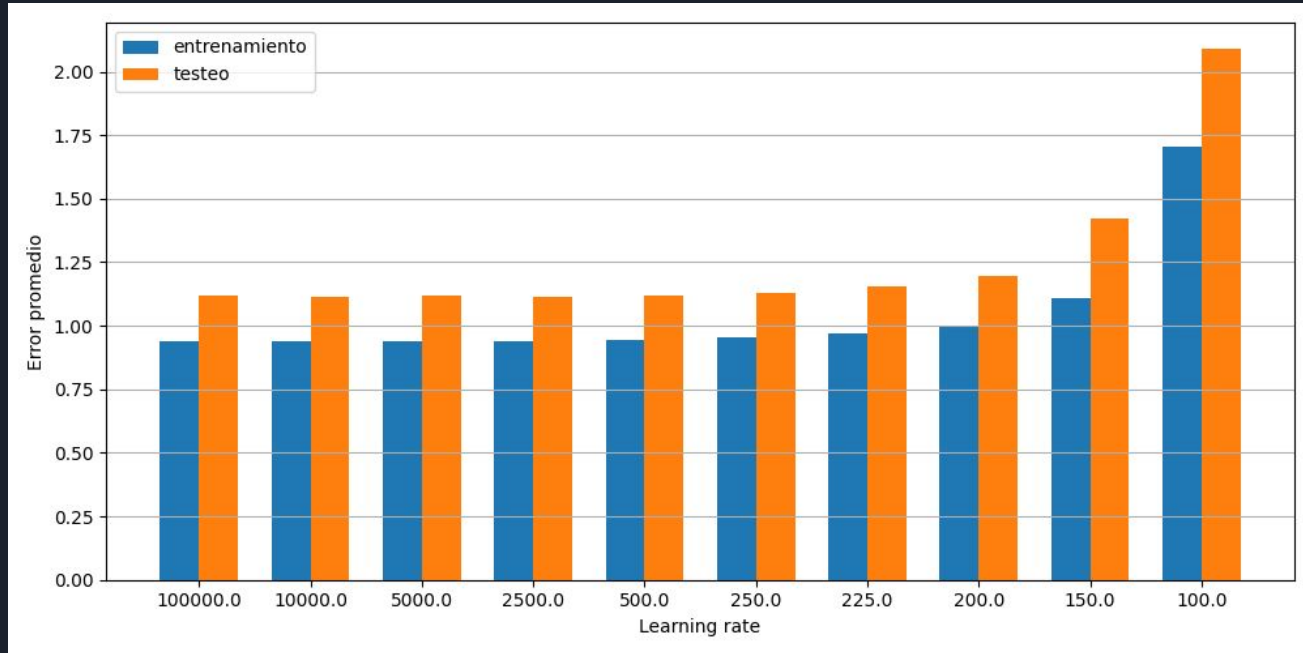
- Activación: Tanh
- Epochs: 100000
- K: 28



Comparación de error de testeo según epochs

Parámetros:


- Activación: Tanh
- Learning Rate: 0.01
- K: 28





Conclusiones

- Para este dataset, aumentar la cantidad de datos que se usan para entrenar conduce a mejores predicciones para los valores faltantes
- El learning rate que produce un menor error de generalización es 0.01. Bajar el learning rate lo aumenta
- Un número alto de epochs (al menos en el rango probado) no produce overfitting, pero correr demasiado pocas epochs produce underfitting



Elección de un conjunto de entrenamiento específico

Si fuéramos a entrenar un perceptrón con una sola parte X del dataset, debemos determinar cual posible partición produce la mejor generalización

Para esto, se pueden realizar repeticiones del K-fold cross validation con particiones aleatorias del dataset, y evaluar qué partición específica provoca un menor error de entrenamiento

En general, el dataset ideal tendrá una buena distribución de los posibles valores de salida. Un conjunto de entrenamiento muy específico causara que no pueda aprender la función entera



Ejercicio 3



Discriminacion de paridad

- **Learning rate:** 0.1
- **Épocas máximas:** 1000
- **Funcion de transformacion:** tanh



Matriz de confusión

	Impar	Par
Impar	500	0
Par	0	500

100 repeticiones




Métricas

Accuracy: 100%

Precision: 100%

Recall: 100%

Tasa de Verdaderos Positivos: 100%



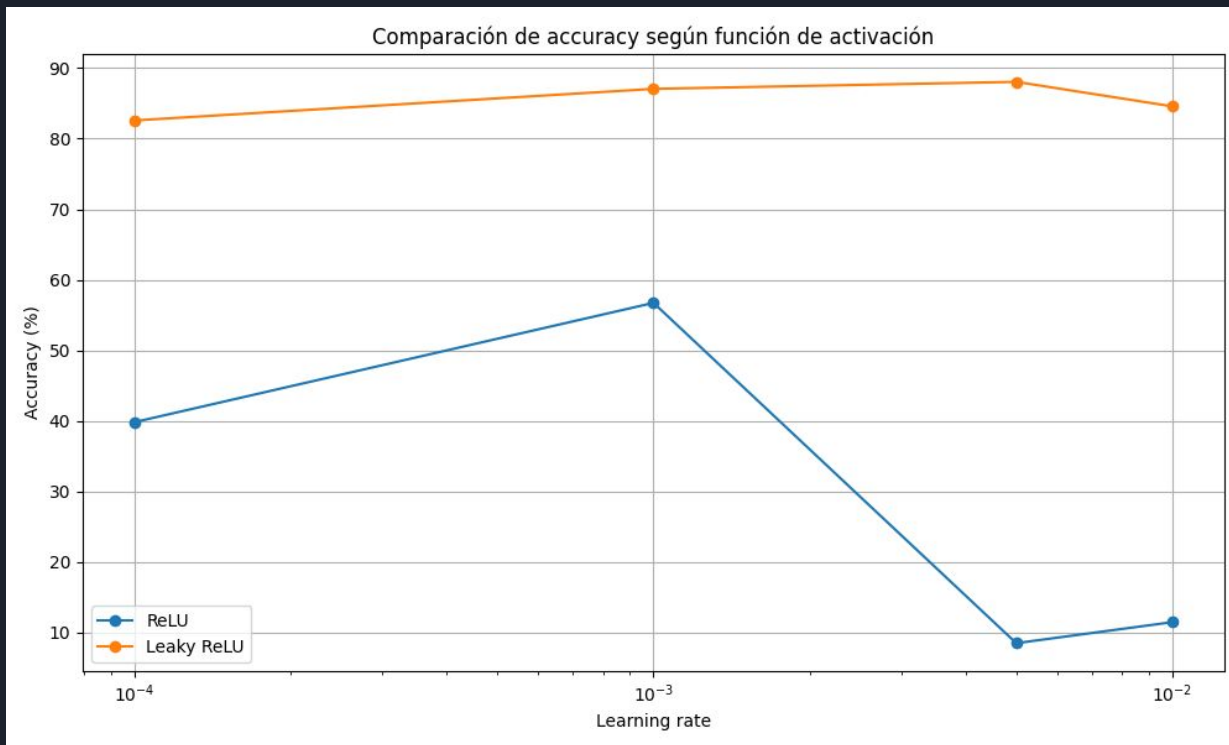
Ejercicio 3

Discriminacion de digitos

A decorative graphic on the left side of the slide. It consists of a blue parallelogram and a light green parallelogram, both tilted at an angle. The blue shape is in the foreground, and the green shape is partially behind it. They are set against a dark blue background with faint, lighter blue diagonal stripes.

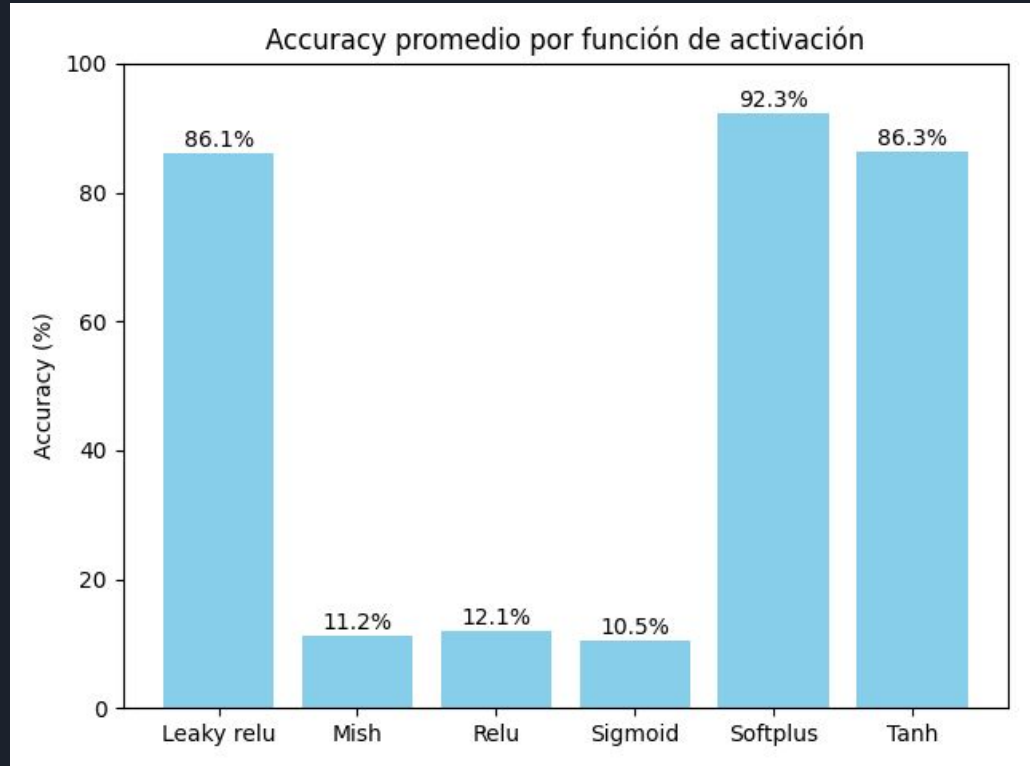
Análisis de hiperparametros

ReLU vs Leaky ReLU



Accuracy por función de activación

- Learning rate: 0.01
- 10 ejecuciones
- 1000 épocas
- Sin optimizadores





Costo computacional

Softplus: Extremadamente costosa (tardaba entre el doble y el triple de tiempo que Leaky ReLU)

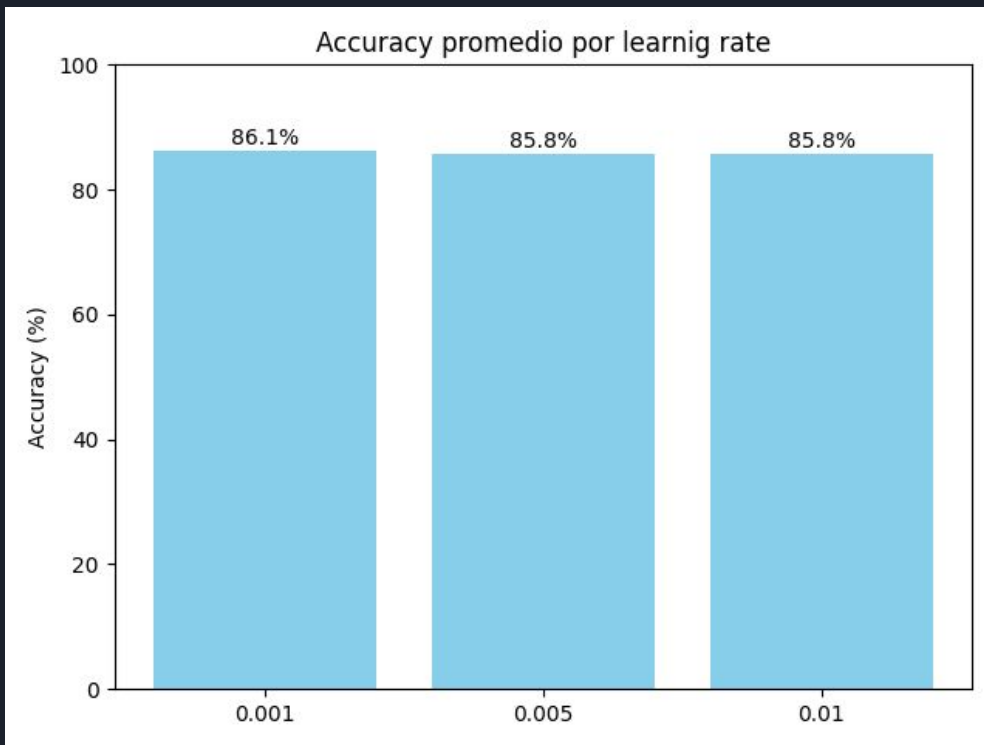
Tanh: También costosa debido a sus operaciones exponenciales

Leaky ReLU: Barata, una resta y una multiplicación (opcional).

Decidimos utilizar Leaky ReLU dado que preferimos priorizar la eficiencia sobre la poca efectividad que nos otorgaba

Learning rates

- 1000 épocas
- Leaky ReLU
- Promedio de 10 ejecuciones
- Sin optimizadores





Discriminacion de digito

Demostración en vivo:

Learning Rate: 0.001

Funcion de activacion: Leaky ReLU

Épocas: 1000

Dataset: 1000 números con ruido medio



Inicialización de pesos

Xavier/Glorot

Distribución normal

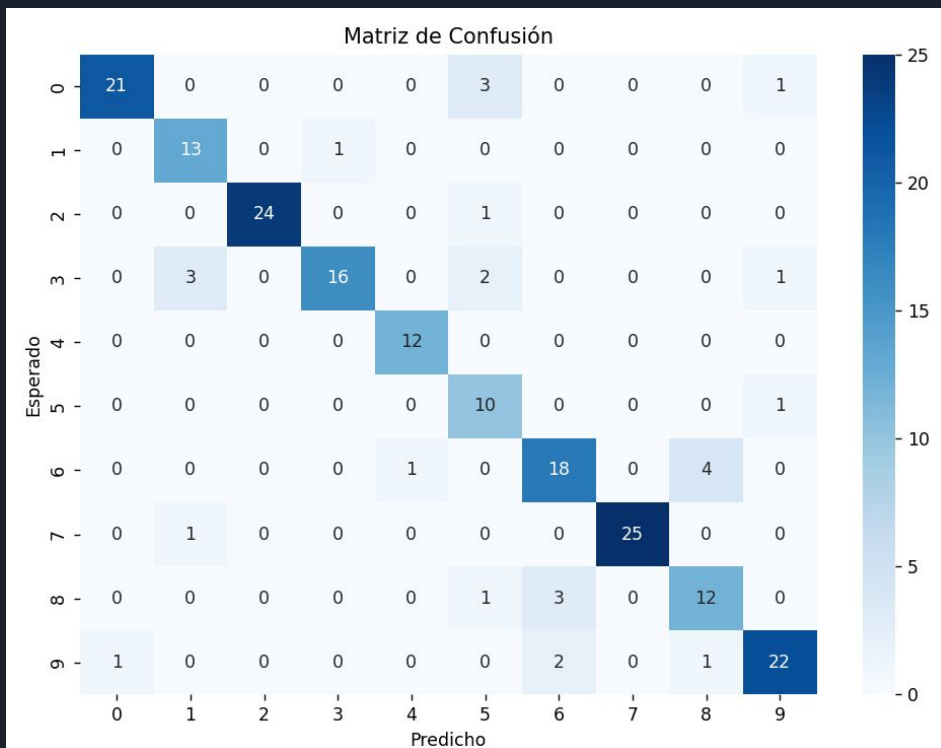
Evitar explosión o desaparición de gradientes

$$\text{Var}(W) = \frac{2}{n_{\text{in}} + n_{\text{out}}}$$

A decorative graphic on the left side of the slide. It consists of a blue parallelogram and a light green parallelogram, both tilted at an angle. The blue shape is in the foreground, and the green shape is partially behind it. They are set against a dark blue background with subtle diagonal lines.

Análisis de resultados

Matriz de confusión





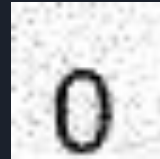
Métricas

Accuracy: 86.50%

	precision	recall	f1-score	support
0	0.84	0.95	0.89	22
1	0.93	0.76	0.84	17
2	0.96	1.00	0.98	24
3	0.73	0.94	0.82	17
4	1.00	0.92	0.96	13
5	0.91	0.59	0.71	17
6	0.78	0.78	0.78	23
7	0.96	1.00	0.98	25
8	0.75	0.71	0.73	17
9	0.85	0.88	0.86	25

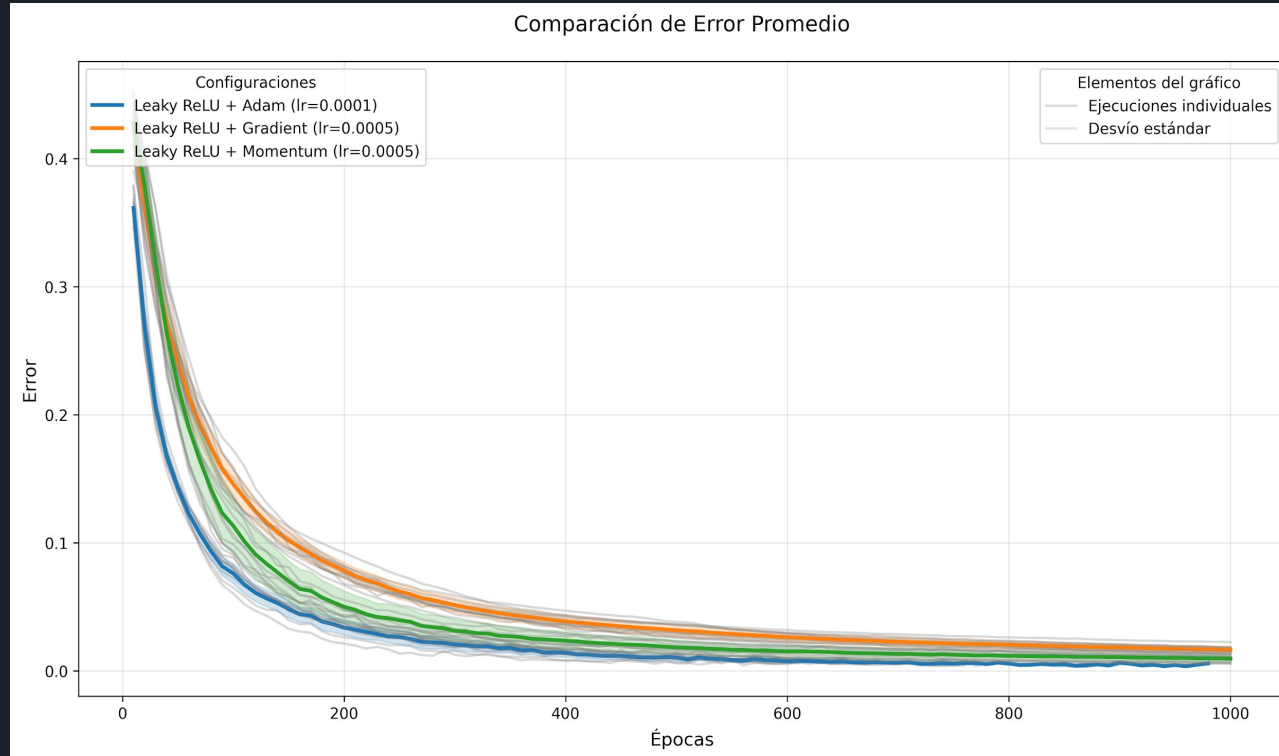
Accuracy segun training set y test set

Train \ Test	Sin ruido	Ruido medio	Ruido alto
Sin ruido	100.00	36.70	19.20
Ruido medio	100.00	99.70	35.80
Ruido alto	47.80	46.80	32.10

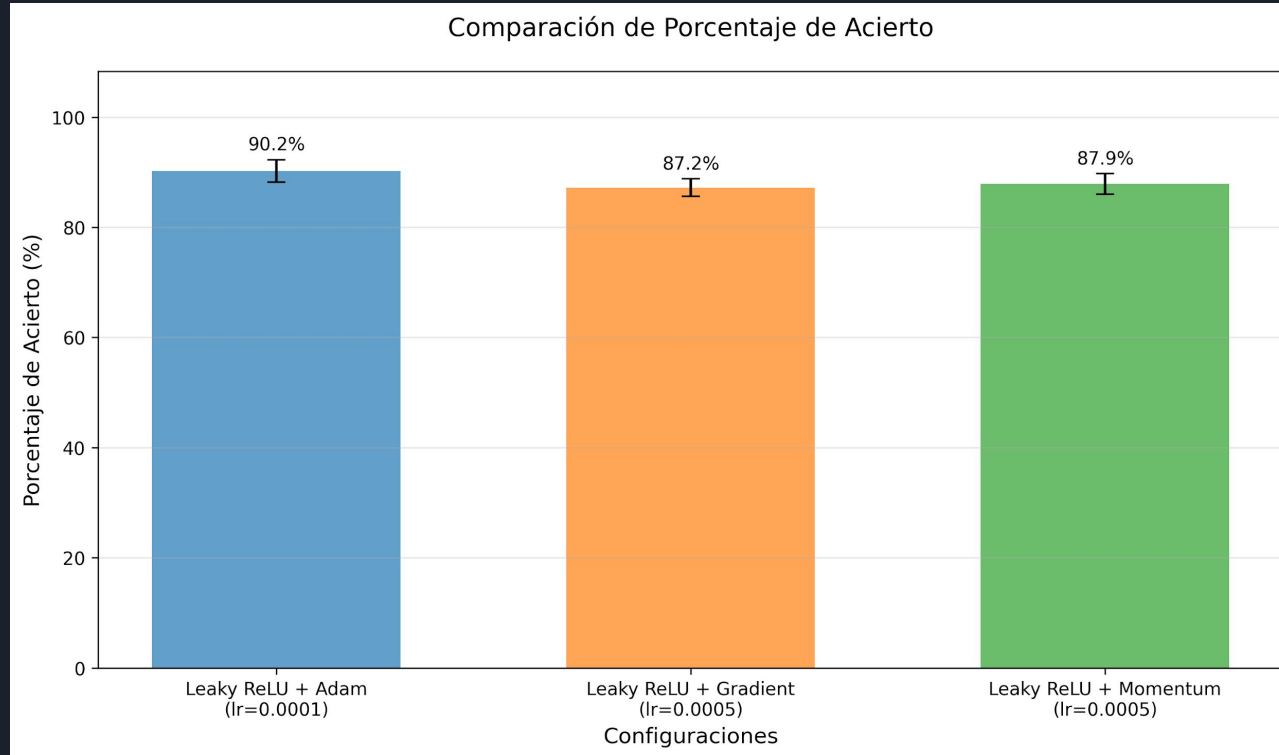


Los casos donde se testean imagenes de dataset iguales, se separa el 20% para testeo

Optimizadores



Optimizadores





Conclusiones

- El modelo fue capaz de generalizar correctamente la clasificación de dígitos con ruido medio, teniendo más errores con ruido alto.
- La comparación entre optimizadores mostró que Adam es el que mejor resultados obtuvo
- Leaky ReLU demostró ser la mejor función de activación en relación de costos y resultados



Ejercicio 4



Ejercicio 4

- DATASET MNIST: Colección de dígitos escritos a mano
- 60.000 imágenes de entrenamiento
- 10.000 imágenes de prueba



Idea inicial

Tenemos análisis de nuestros mejores hiperparametros -> usemoslos

```
Configuracion del entrenamiento:  
Arquitectura: [784, 128, 64, 10]  
Learning rate: 0.0001  
Epocas: 30  
Funcion de activación: leaky_relu  
Optimizador: adam
```



Obstáculo

Tiempo: resultados de 30 épocas con un subset de 10.000 entrenado a 30 épocas

```
Precisión final en prueba: 95.35%  
Total de imágenes correctas: 1907/2000  
Error final: 0.003710597379931923  
Tiempo de entrenamiento: 388.39 segundos  
Tiempo total de ejecución: 388.93 segundos
```



Nueva idea

- Aprovechemos cada ejecución y guardemos los pesos y parámetros de adam obtenidos
- Falta decidir cuando los parámetros obtenidos son mejores que los que teníamos antes
- Solución errónea: Evaluar en base al conjunto de prueba si si tenía mejor rendimiento
- Contaminación del entrenamiento



Idea final (Por ahora)

- “Simular” el conjunto de prueba
- Tomar primeras 10.000 imágenes como nuestra nueva prueba
- Entrenar en base a las 50.000 restantes

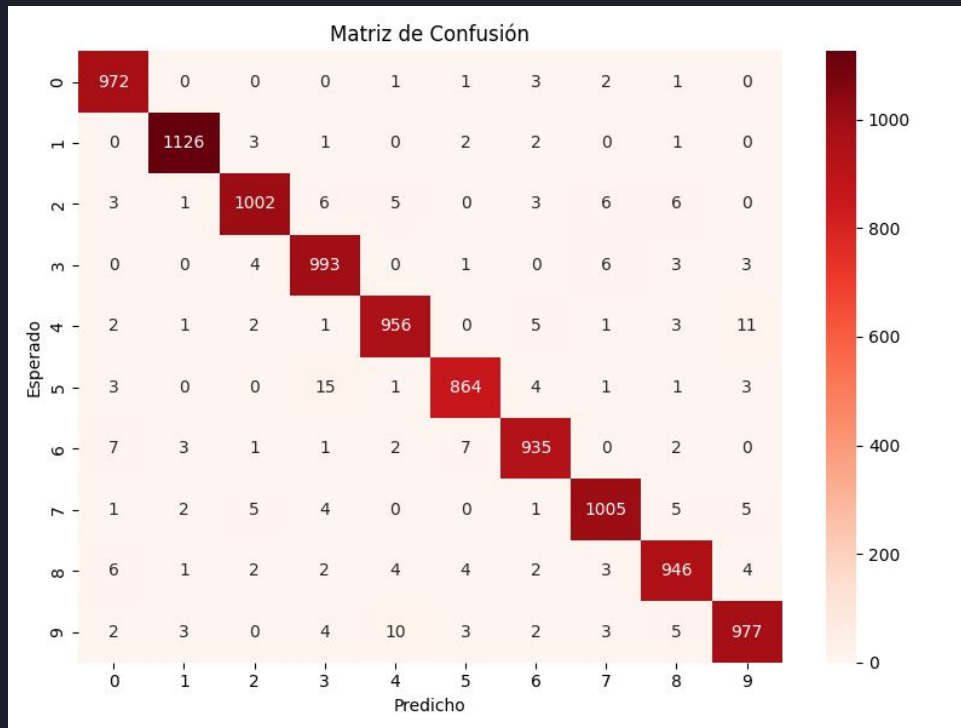


Resultados “finales” y posible problema

Precisión en validación: 97.85%
Precisión final en prueba: 97.85%
Total de imágenes correctas en prueba: 9785/10000
Error final: 0.005508682452115342
Tiempo de entrenamiento: 1059.24 segundos
Tiempo total de ejecución: 1065.54 segundos

Precisión en validación: 97.92%
Precisión final en prueba: 97.76%
Total de imágenes correctas en prueba: 9776/10000
Error final: 0.0025531050501176413
Tiempo de entrenamiento: 192.37 segundos
Tiempo total de ejecución: 198.19 segundos

Resultados





Posibles mejoras a implementar

- Variar el subset tomado como validación
- Implementar paralelismo para ejecuciones más rápidas
- Data augmentation
- Modificar la estructura de la red a una más robusta



Muchas gracias!