

TRABAJO PRÁCTICO NÚMERO 3

Perceptrón Simple y Multicapa

72.27 - SISTEMAS DE INTELIGENCIA ARTIFICIAL

I N T E G R A N T E S

GRUPO 5

Kuchukhidze, Giorgi - 67262

Madero Torres, Eduardo Federico - 59494

Ramos Marca, María Virginia - 67200

Plüss, Ramiro - 66254



Agenda

01

INTRODUCCIÓN

02

EJERCICIO 1

03

EJERCICIO 2

04


EJERCICIO 3

05

CONCLUSIONES



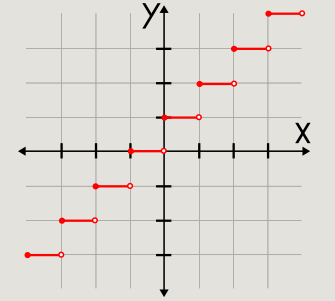
Introducción



En este trabajo se ha desarrollado una aplicación práctica que explora la evolución del perceptrón, abarcando sus variantes simple, lineal, no lineal y multicapa. Esta implementación se basa en los fundamentos establecidos por diversos matemáticos y científicos, permitiendo una comprensión profunda de estas estructuras neuronales fundamentales en la inteligencia artificial

A	B	Y
0	0	0
0	1	0
1	0	0
1	1	1

A	B	Y
0	0	0
0	1	1
1	0	1
1	1	1



Ejercicio 01

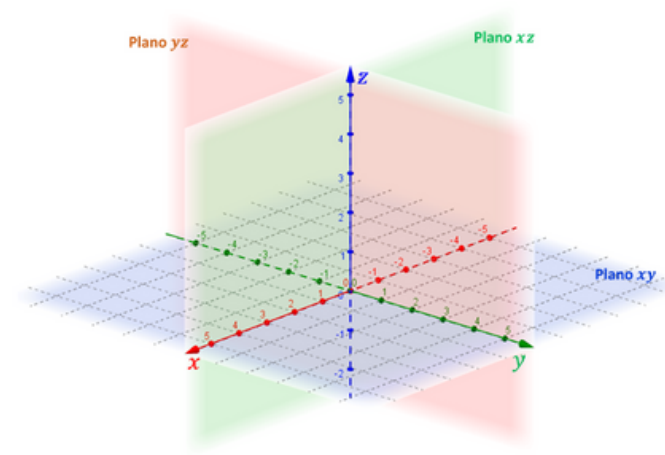
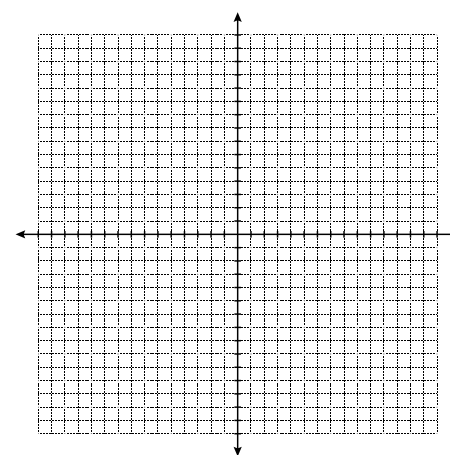
- Implementación del algoritmo de *perceptrón simple*.
- Se utiliza la función de escalón.
- Se resuelve para el conjunto de datos definido por la función **AND** y la función **XOR**.

Teorema de Convergencia del Perceptrón

- El perceptrón **converge** si los datos son **linealmente separables**.
- Ajusta los pesos hasta encontrar un hiperplano de separación.
- Con un número finito de iteraciones, clasifica correctamente todas las entradas.
- Si los datos **no son separables**, el perceptrón **no converge**.

Separabilidad de una función

Para demostrar que la función es *linealmente separable*, debemos mostrar que **podemos encontrar una línea recta** (en dos dimensiones) o un **hiperplano** (en más dimensiones) que **separe** los puntos de entrada según sus clases de salida.



Función Lógica AND

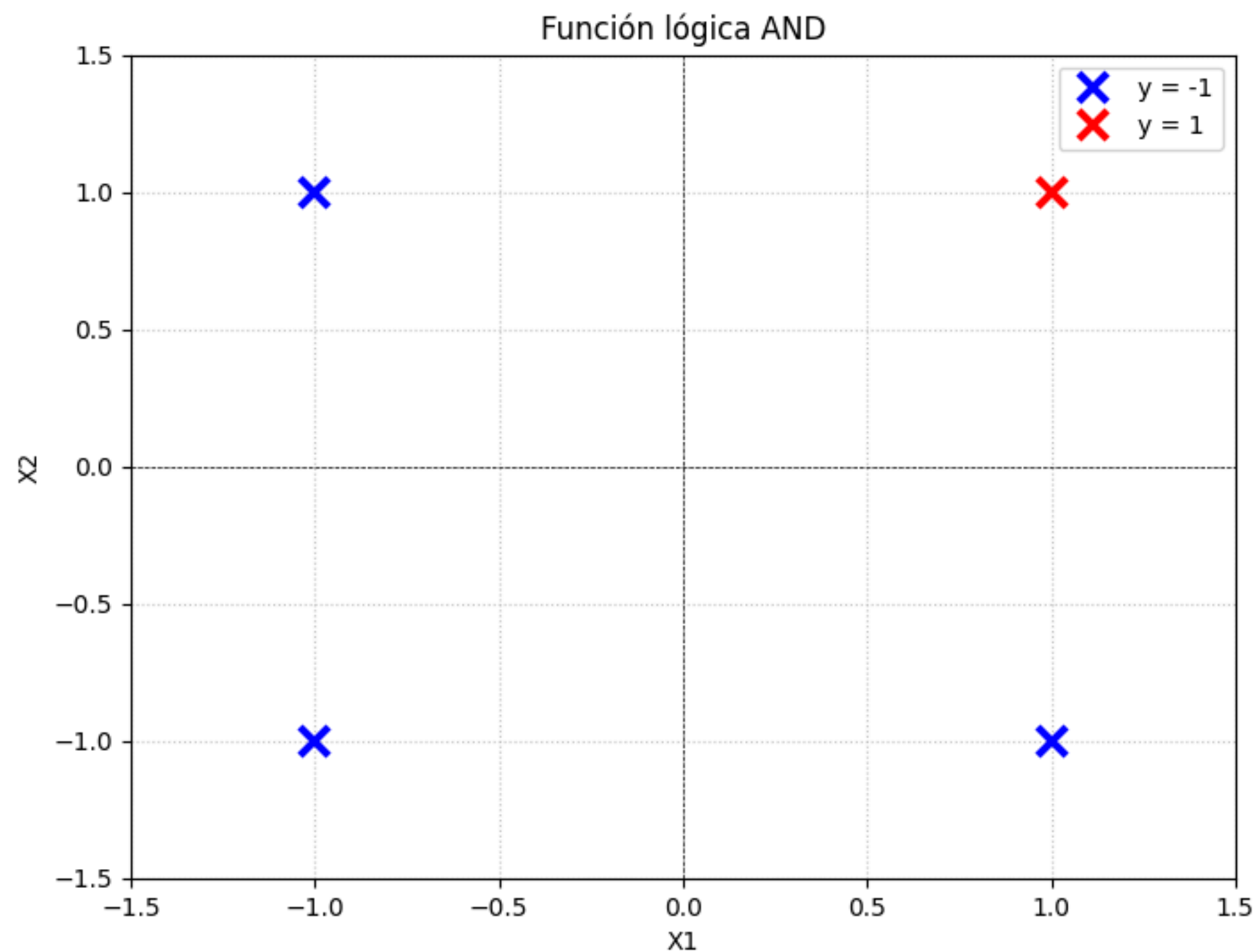
Definimos la función lógica **AND** como:

$$f_{\text{AND}}(x_1, x_2) = \begin{cases} 1 & \text{si } x_1 = 1 \text{ y } x_2 = 1 \\ -1 & \text{si } x_1 = -1 \text{ o } x_2 = -1 \end{cases}$$

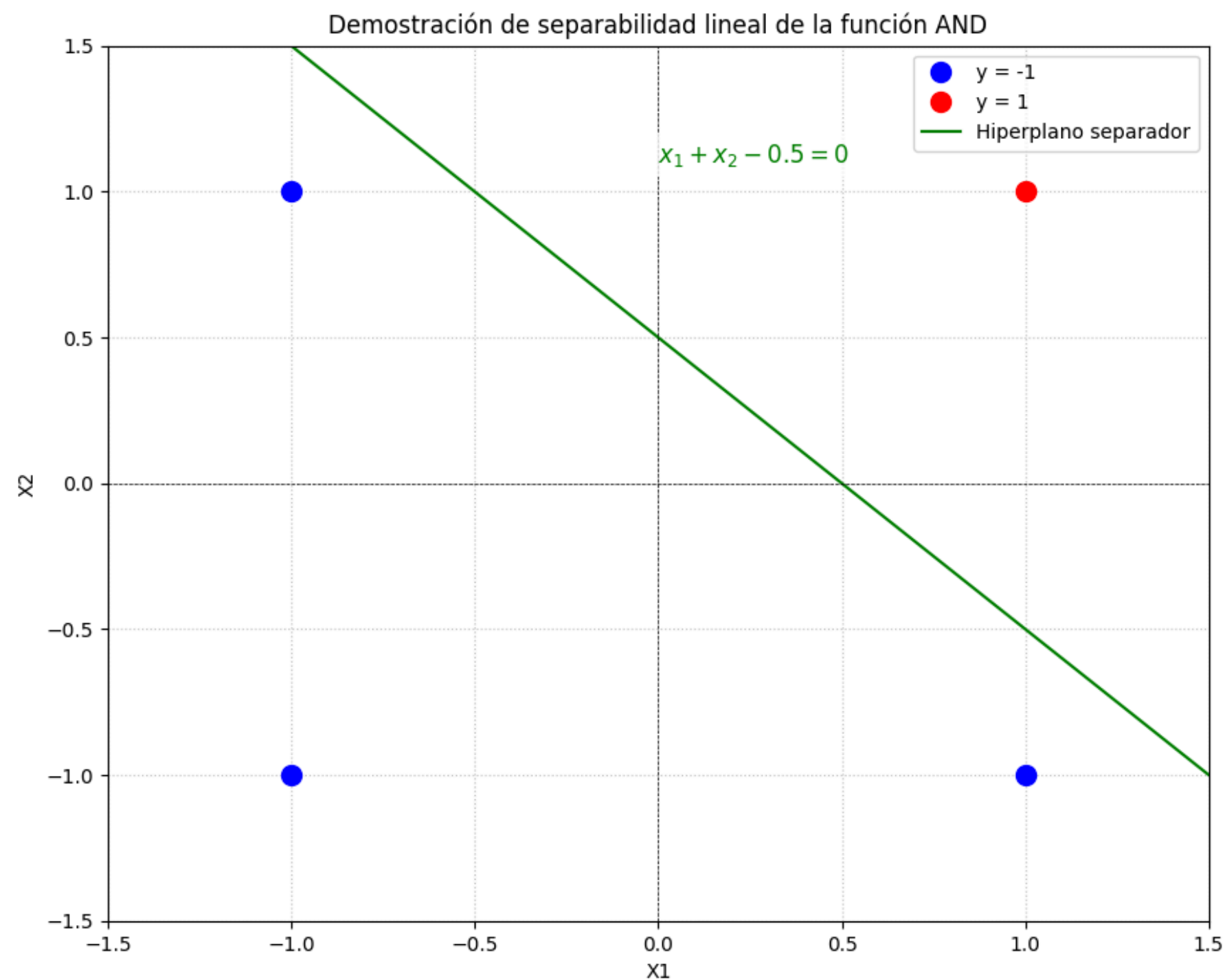
$$x = \{\{-1, 1\}, \{1, -1\}, \{-1, -1\}, \{1, 1\}\}$$

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

Función Lógica AND

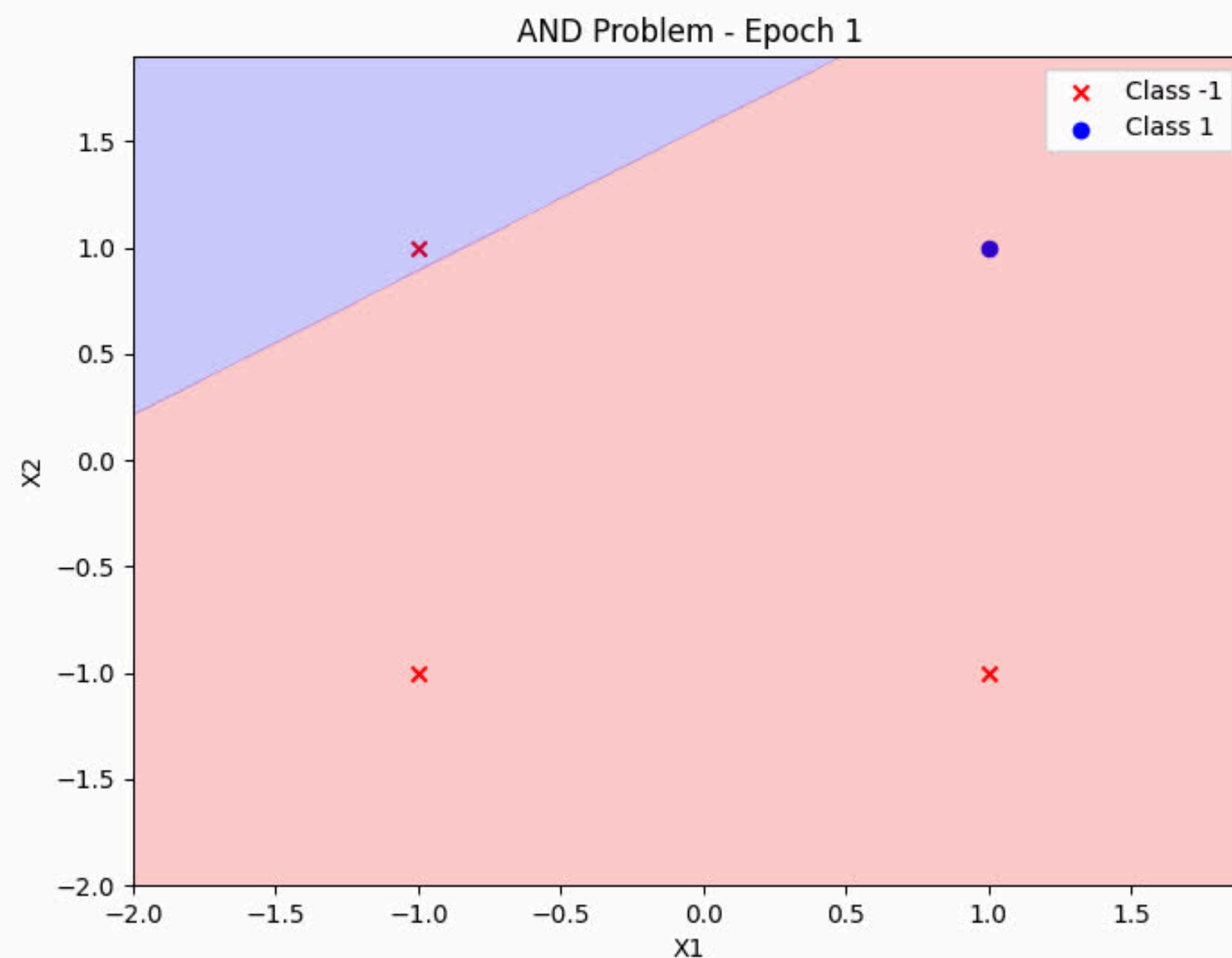


Separabilidad de Función Lógica AND



Resultados - AND

- $\eta = 0.1$
- $\varepsilon = 0.1$
- Epochs = 10



Resultados - AND

Resultados de **100** simulaciones del problema **AND**:

- $\eta = 0.1$
- $\varepsilon = 0.1$
- Epochs = 10

Media del número de épocas: 3.94

Mediana del número de épocas: 4.0

STD número de épocas: 1.55

AND se predice correctamente en todos los casos.



Ejercicio 01

Función Lógica XOR

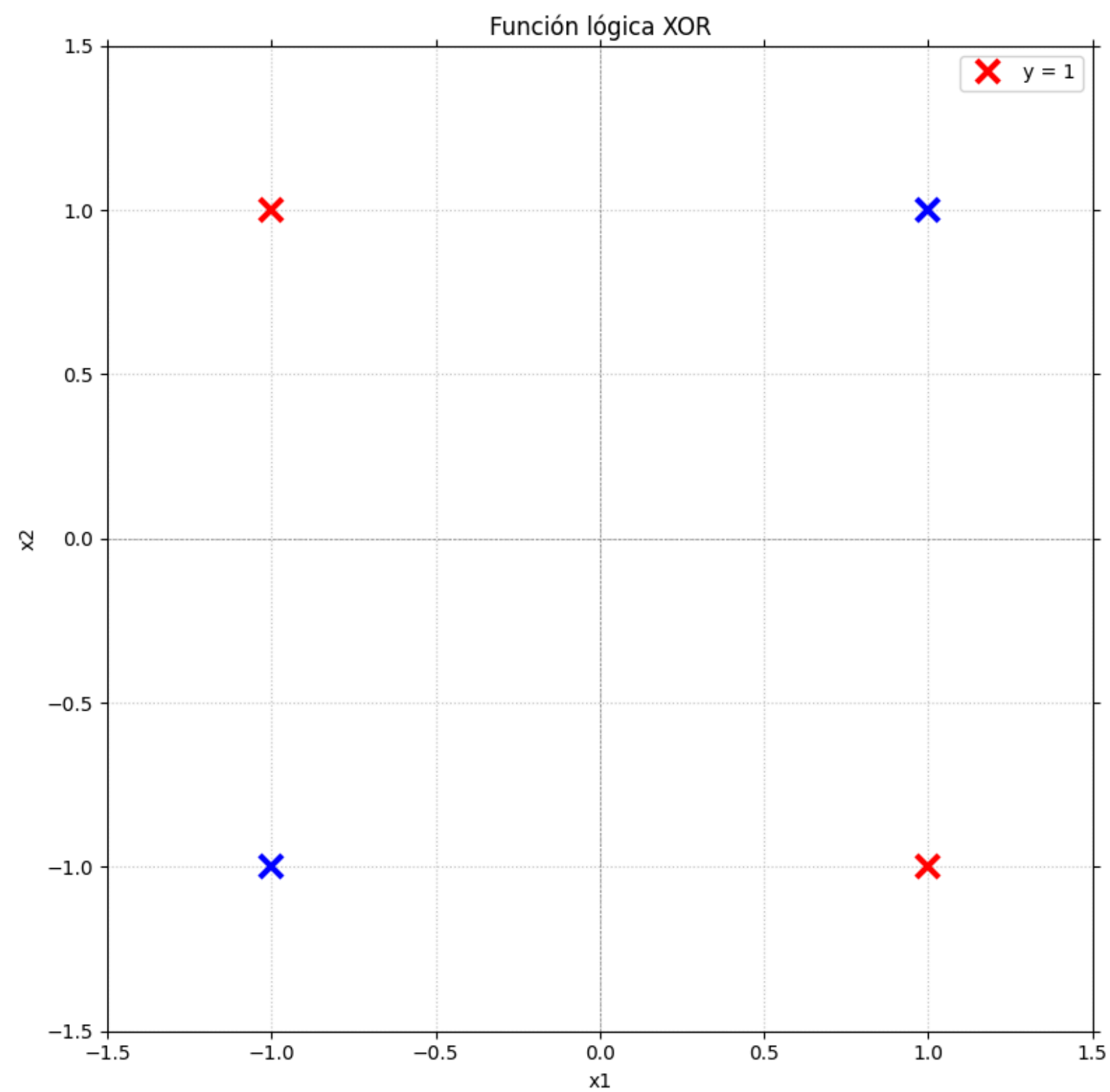
Definimos la función lógica **XOR** como:

$$f_{XOR}(x_1, x_2) = \begin{cases} 1 & \text{si } (x_1 \neq x_2) \\ -1 & \text{si } (x_1 = x_2) \end{cases}$$

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

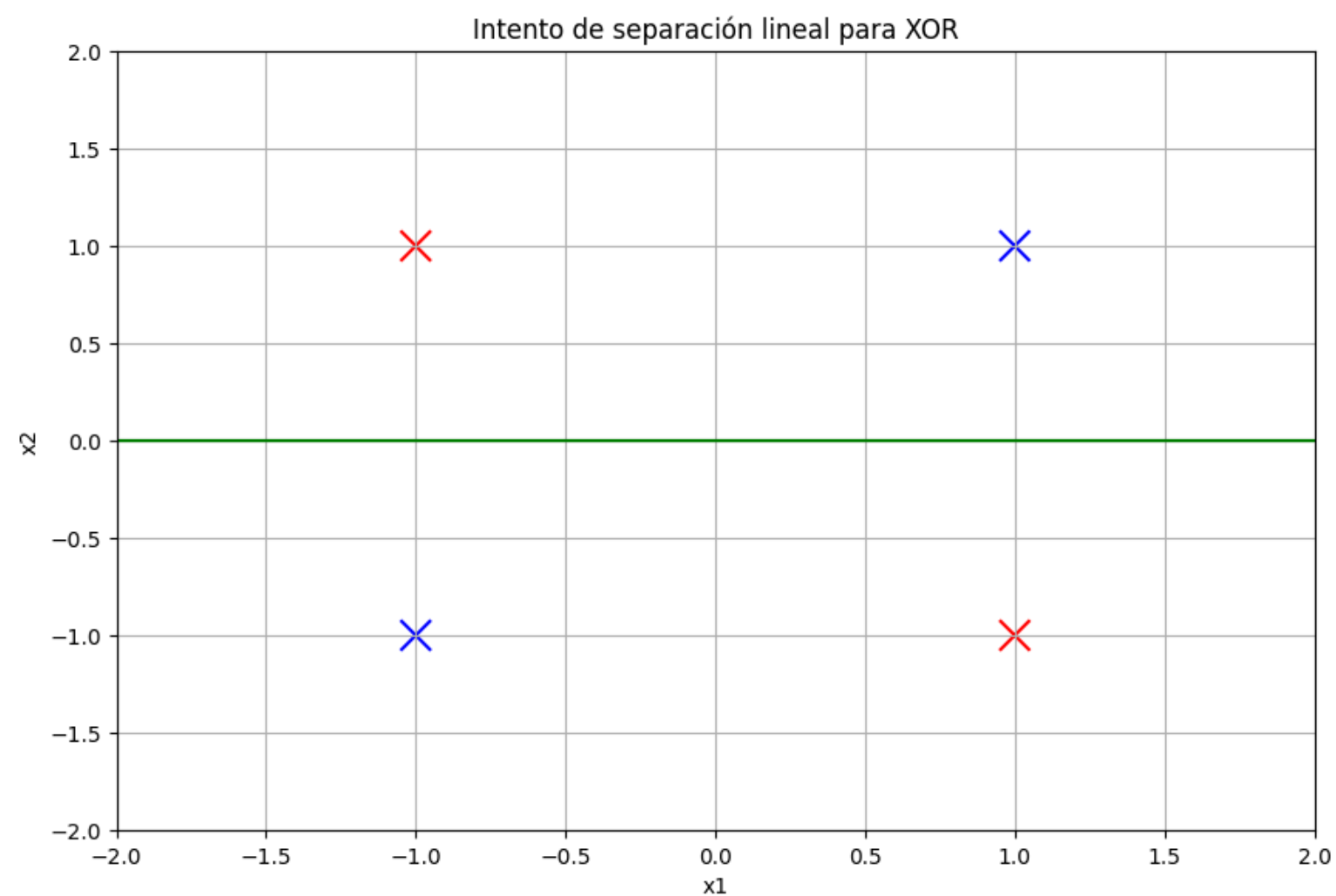
$$y = \{1, 1, -1, 1\}$$

Función Lógica XOR



Separabilidad de Función Lógica OR

¿Existirá un hiperplano que separe los puntos?



Pendiente 0

Intersección 0

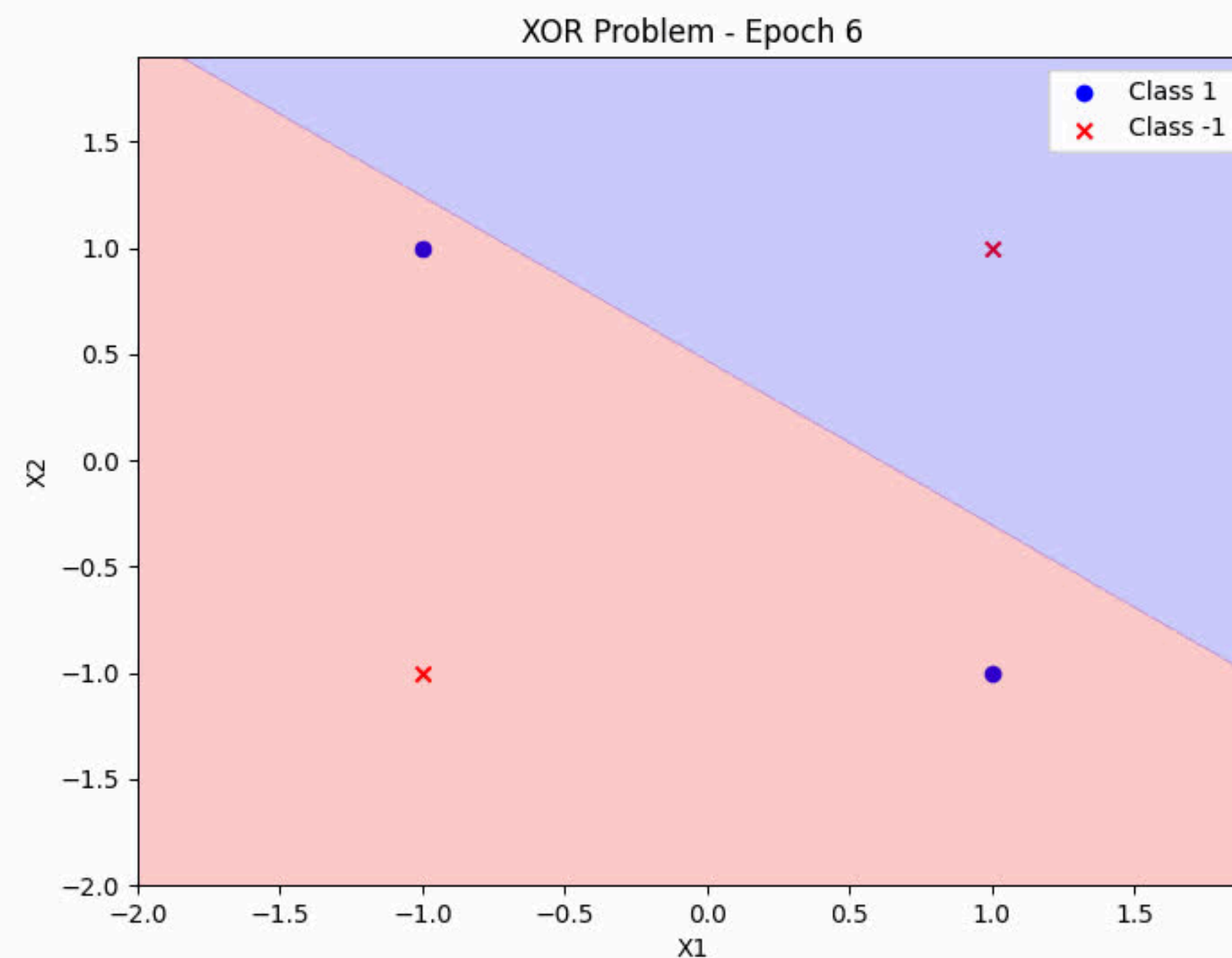
Función Lógica XOR no es linealmente separable

- La función XOR es no linealmente separable.
- Se puede demostrar que no existe un hiperplano que separe sus datos.
- Un perceptrón simple no puede clasificar correctamente todos los puntos de XOR.

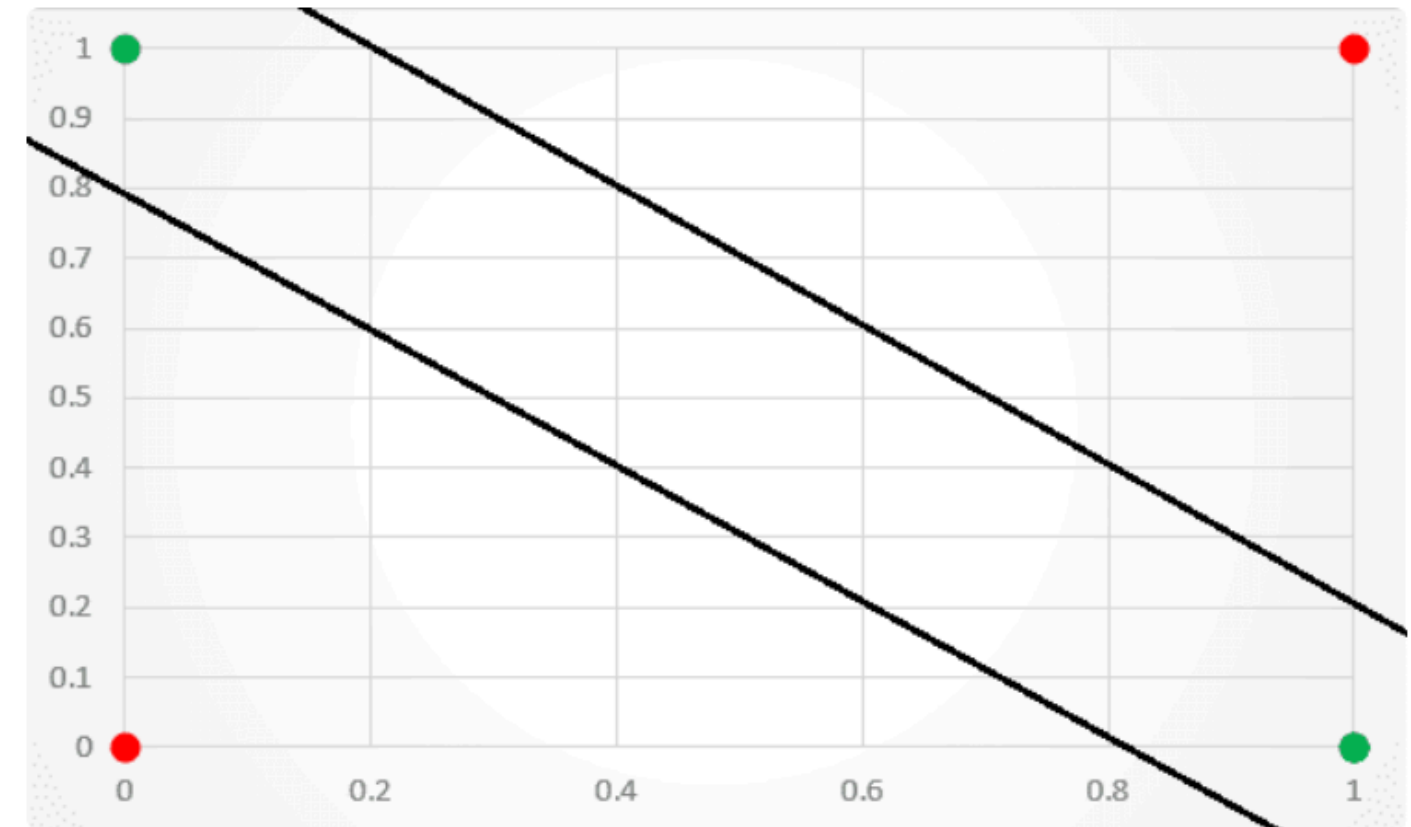
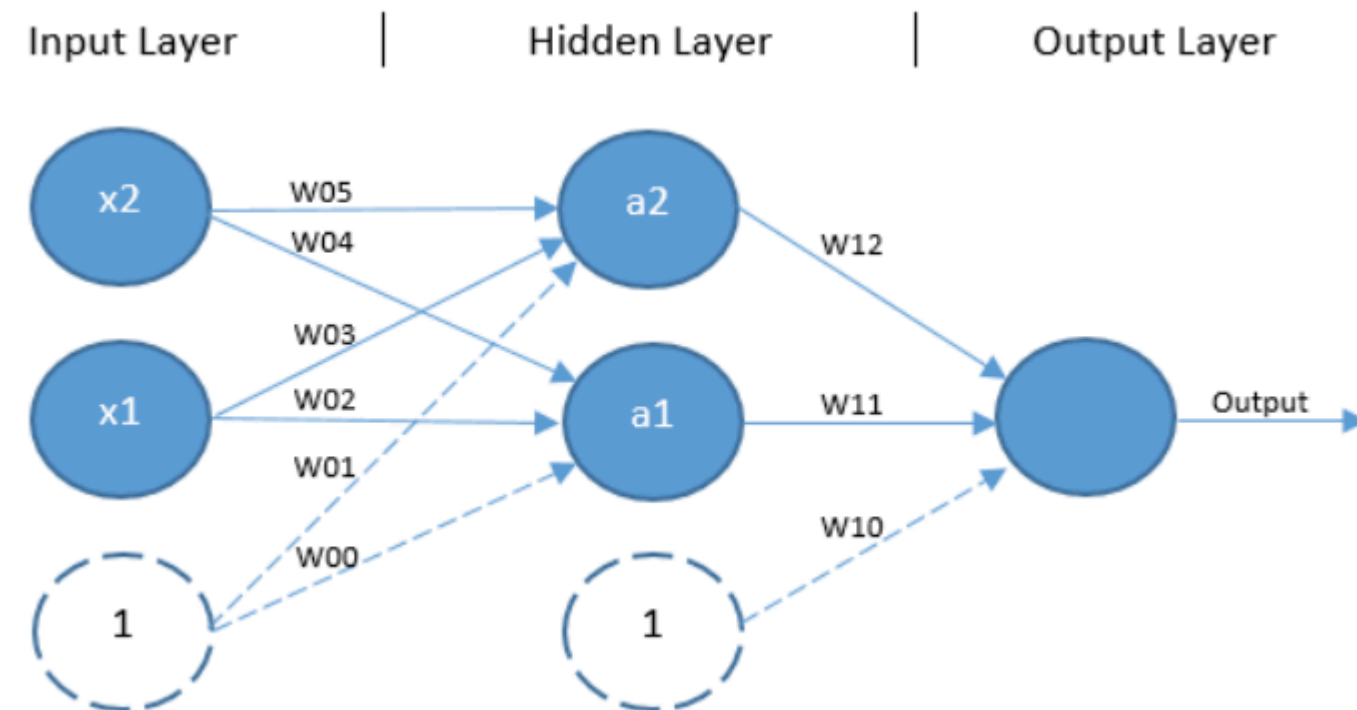
Resultados - XOR

- $\eta = 0.1$
- $\varepsilon = 0.1$
- Epochs = 10

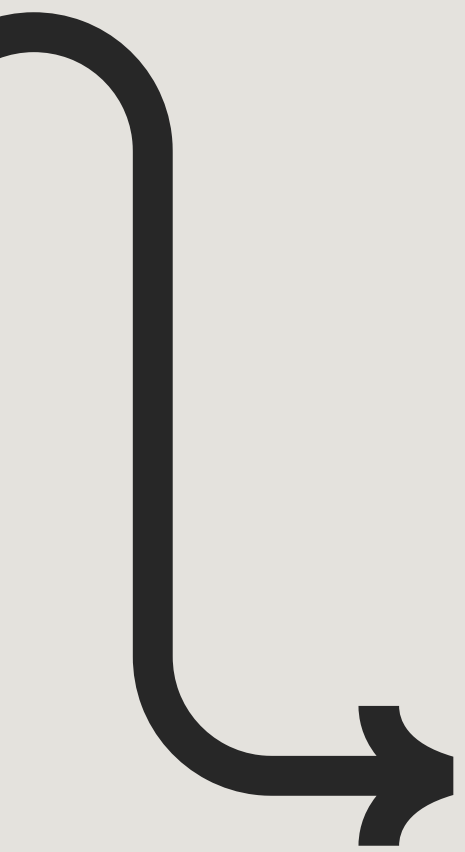
XOR siempre es diferente y a menudo incorrecto para 10 épocas.



Solución - XOR



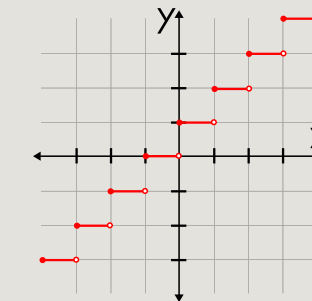
La solución es introducir el perceptrón multicapa
Esta arquitectura, aunque más compleja que la de la red clásica de perceptrones, es capaz de lograr una separación no lineal.



Ejercicio 02

A	B	Y
0	0	0
0	1	0
1	0	0
1	1	1

A	B	Y
0	0	0
0	1	1
1	0	1
1	1	1



- Implementación del algoritmo de *perceptrón simple LINEAL* y *perceptrón simple NO LINEAL*.
- Se resuelve para el conjunto de datos definido en el archivo **TP3-ej2-conjunto**.

Funciones de activación

Perceptrón Lineal

$$\Theta(h) = h$$

Perceptrón No Lineal

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

$$Im = (-1, 1)$$

Dataset para el perceptron lineal y no lineal

	x1	x2	x3	y
0	1.2	-0.80	0.00	21.755
1	1.2	0.00	-0.80	7.176
2	1.2	-0.80	1.00	43.045
3	0.0	1.20	-0.80	2.875
4	7.9	1.00	0.00	26.503
5	0.4	0.00	2.70	68.568
6	0.0	0.40	2.70	61.301
7	-1.3	3.23	3.00	23.183
8	0.4	2.70	0.00	2.820
9	0.4	2.70	2.00	17.654
10	-1.3	0.00	3.23	72.512

La salida es un **float**



Problema de **regresión**

Podemos utilizar tres métricas para analizar y comparar la capacidad de aprendizaje de los perceptrones **lineales** y **no lineales**.

- **MSE** (*Error Cuadrático Medio*): Mide la diferencia promedio cuadrada entre los valores predichos y los reales; penaliza más los errores grandes.
- **MAE** (*Error Absoluto Medio*): Calcula la diferencia absoluta promedio entre los valores predichos y los reales.
- **R²** (*Coeficiente de Determinación*): Indica la proporción de la varianza en la variable objetivo explicada por el modelo; valores cercanos a 1 indican un mejor ajuste.



Ejercicio 02

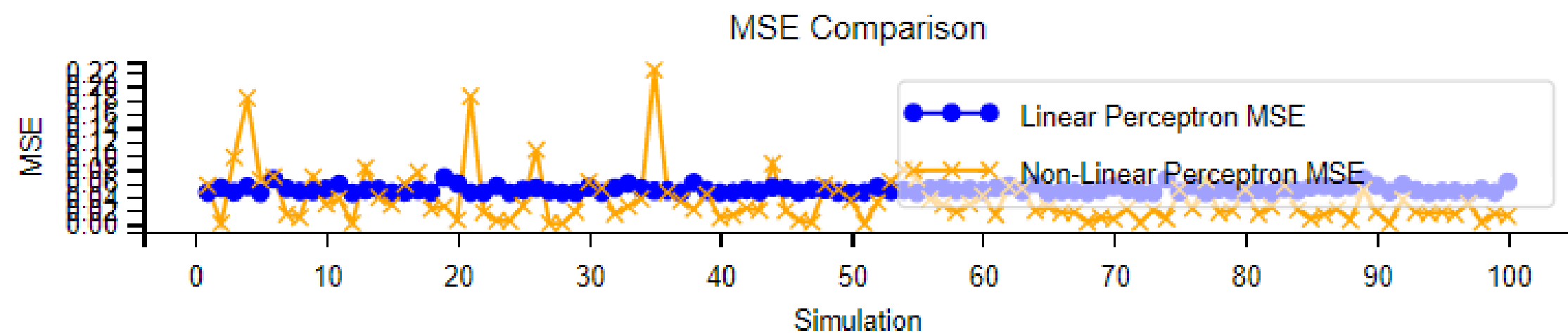
Método de Análisis

- Se realizaron 100 simulaciones de :
 - Entrenamiento de ambos perceptrones en el conjunto de datos completo.
 - Guardado de los valores de MSE, MAE y R^2 .
- Creación de gráficos de las 100 simulaciones.
- Comparación de los valores promedios.

Configuración

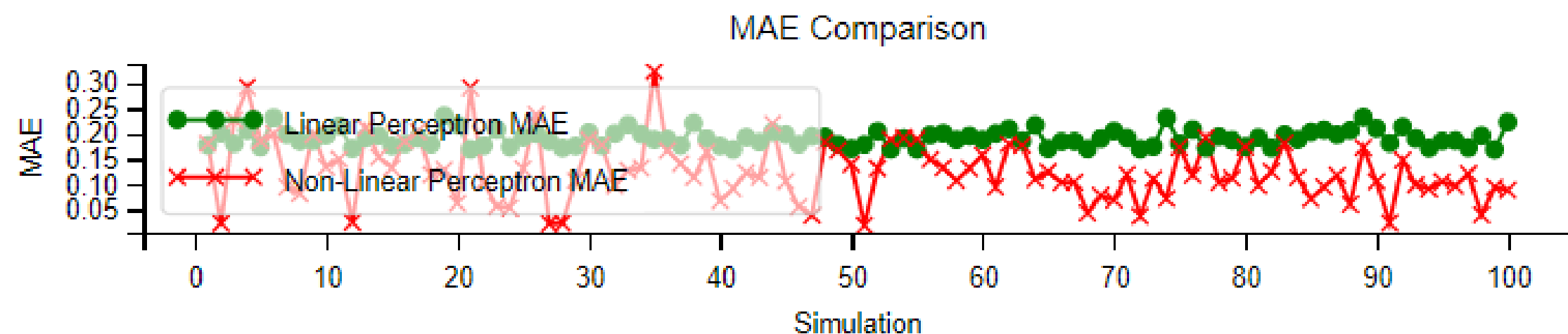
- epochs = 20
- learning rate = 0,01
- $\varepsilon = 1e-5$

↑ Ejercicio 02



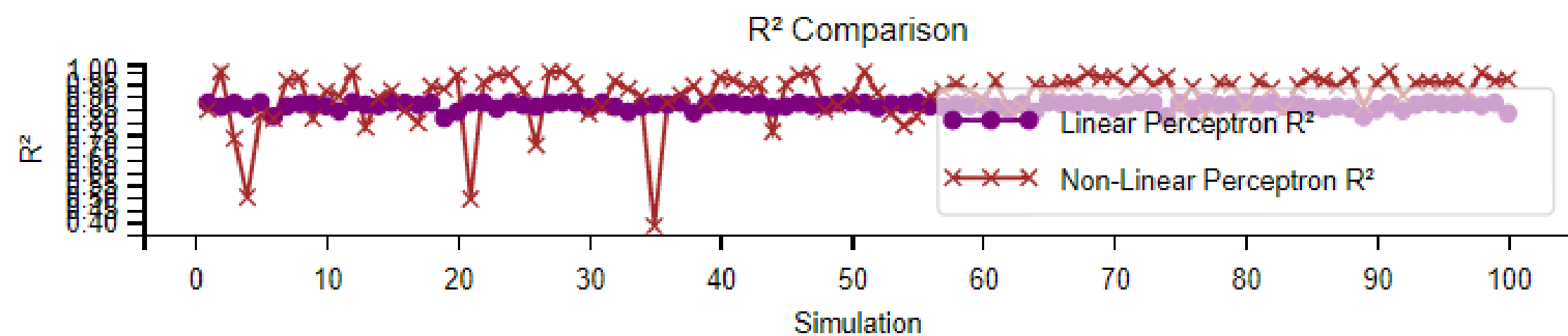
Average MSE sobre 100 simulaciones :

- Lineal: 0.051
- No lineal: 0.039



Average MAE sobre 100 simulaciones :

- Lineal: 0.192
- No lineal: 0.133



Average R² sobre 100 simulaciones :

- Lineal: 0.863
- No lineal: 0.919

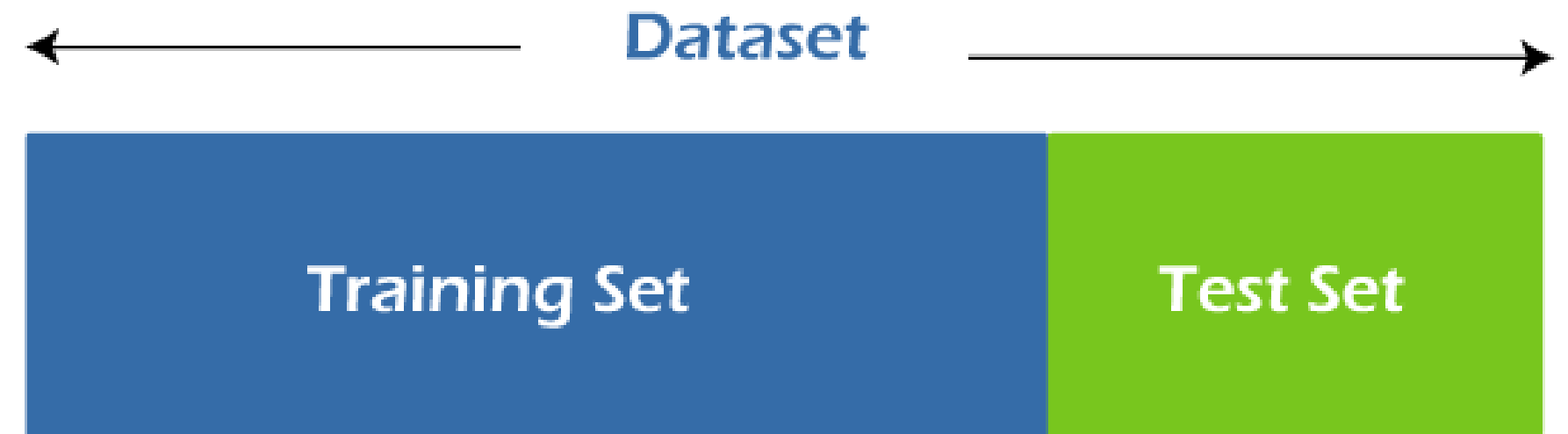
Lineal vs No Lineal

- El perceptrón **lineal** ofrece un rendimiento aceptable porque los datos no son demasiado complejos.
- Sin embargo, el perceptrón **no lineal** logra mejores resultados, ya que puede capturar patrones más complejos y ajustarse mejor a los datos.

Por esta razón, elegimos el perceptrón **no lineal** para la segunda parte del análisis.

Elegir el mejor conjunto de entrenamiento

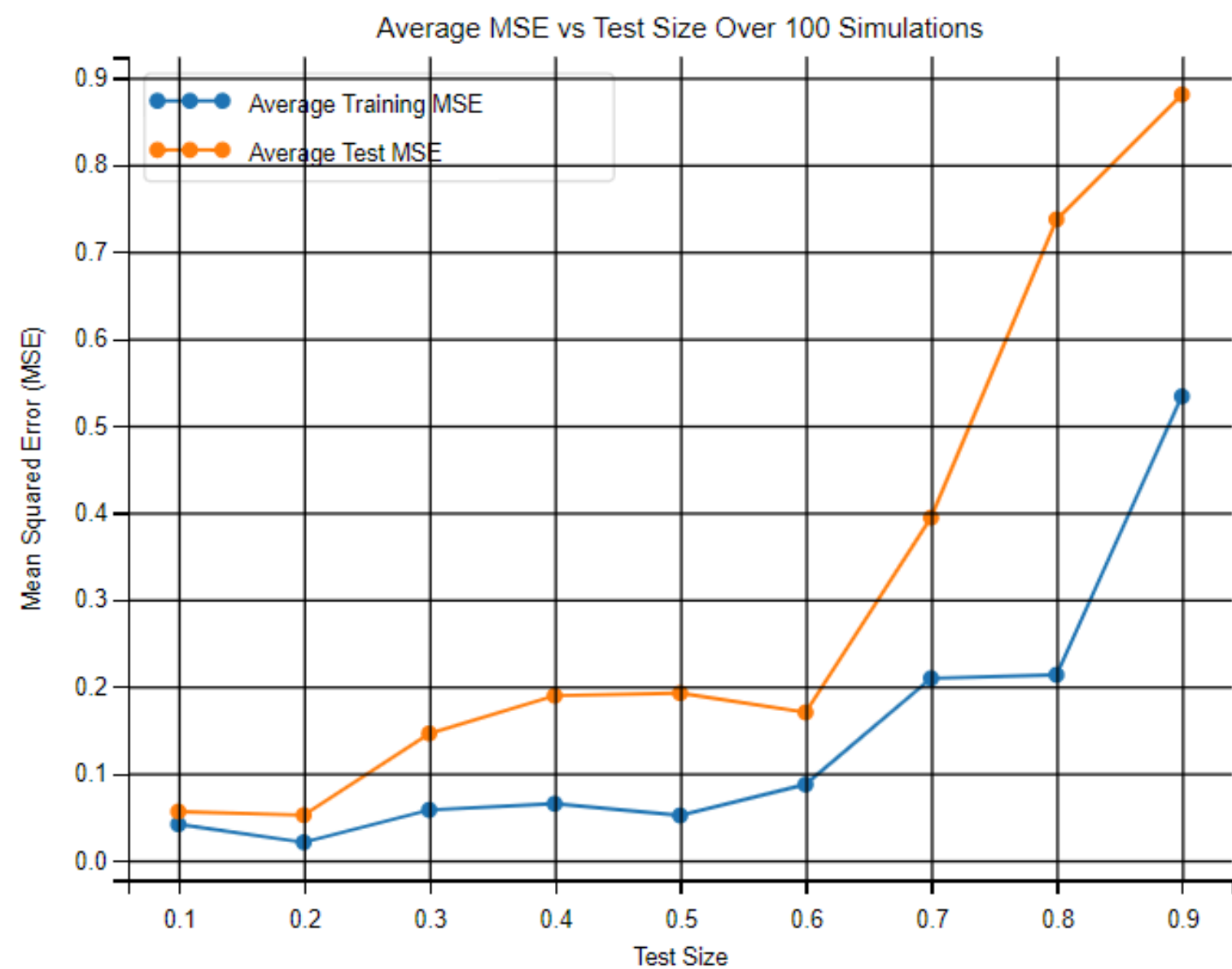
División fija de datos



División fija:

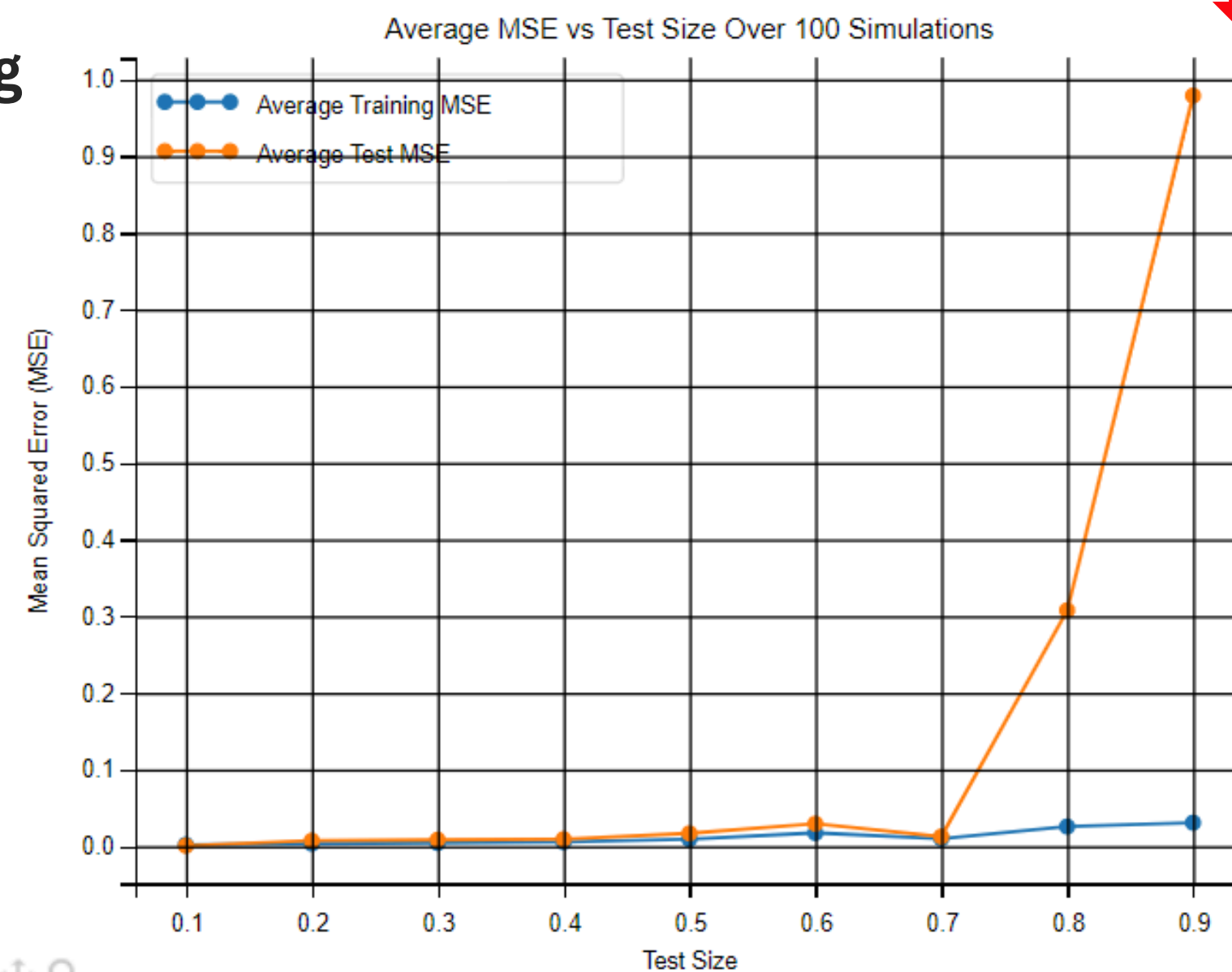
- El conjunto de datos se divide una sola vez en entrenamiento y prueba.
- La proporción de división es fija, por ejemplo, 80% para entrenamiento y 20% para prueba.
- Esta es la primera idea que viene a la mente para elegir el mejor conjunto de entrenamiento

División fija - La proporción de división



20 epochs

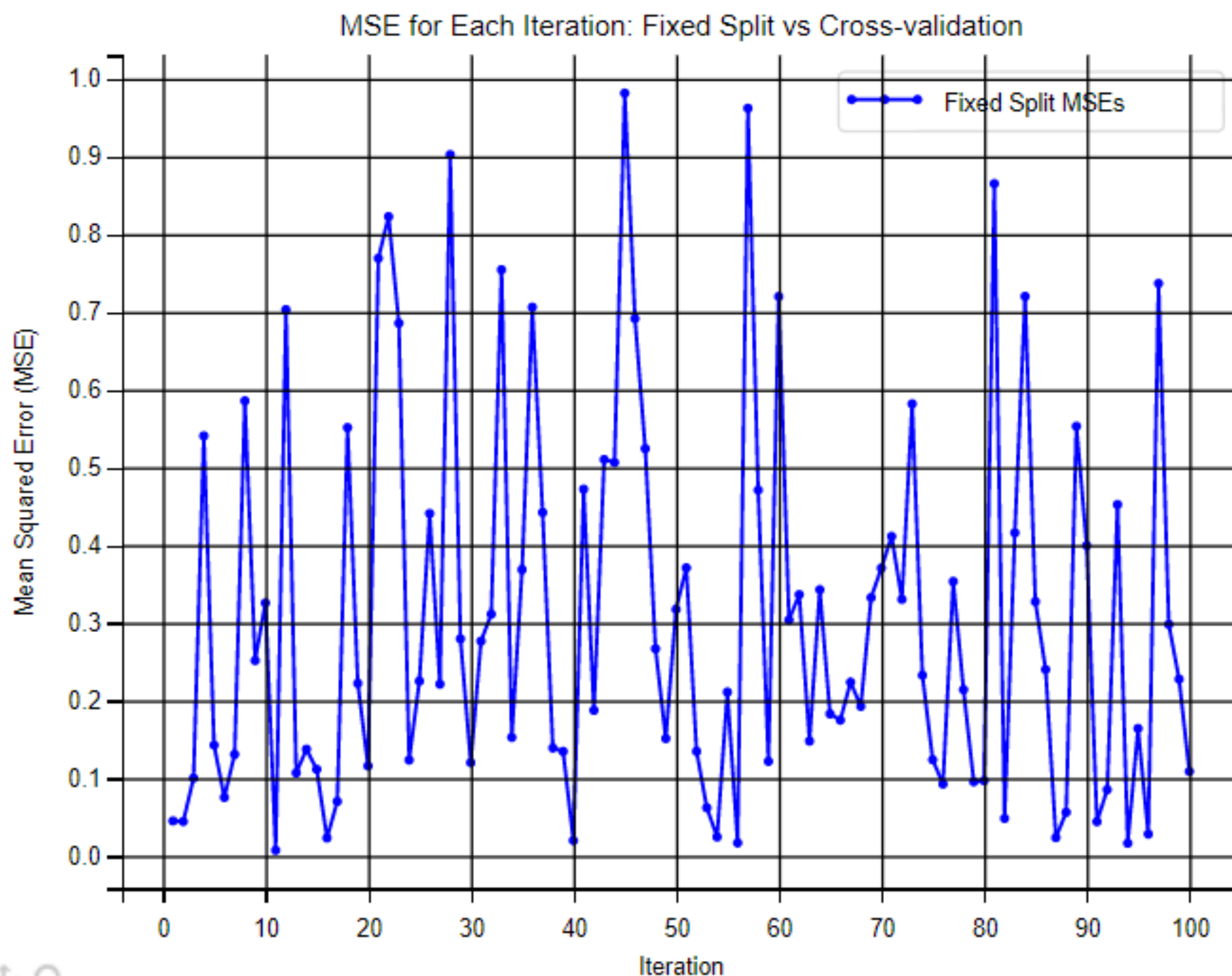
Underfitting



100 epochs

Overfitting

Division fija con la mejor proporción de división (Test size = 0.2)



Fixed split **average** MSE: 0.3083

Fixed split **worst** MSE: 0.9815

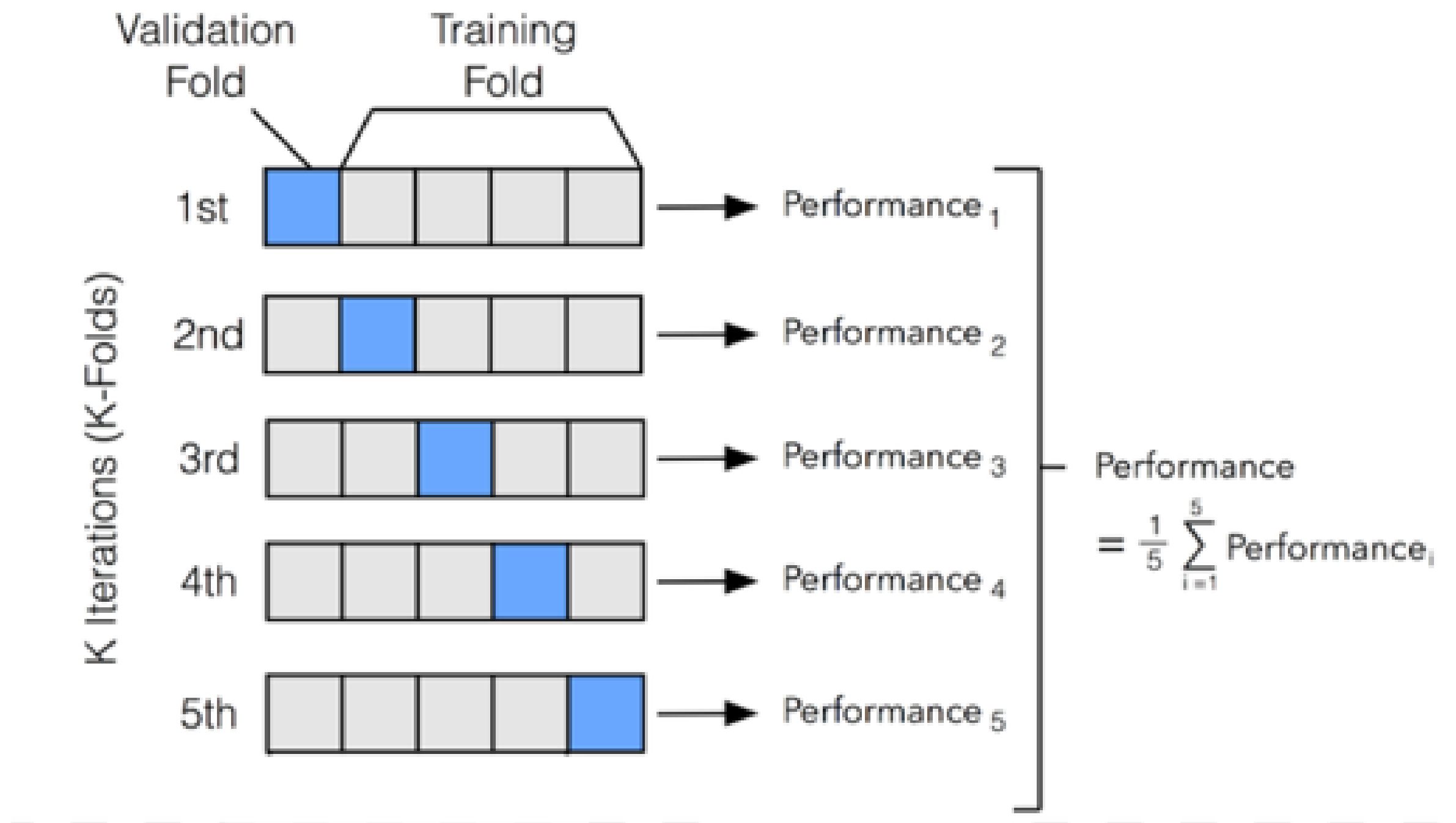
Varianza muy alta

- Datos de entrenamiento sesgados.
- Datos de entrenamiento con demasiado ruido.

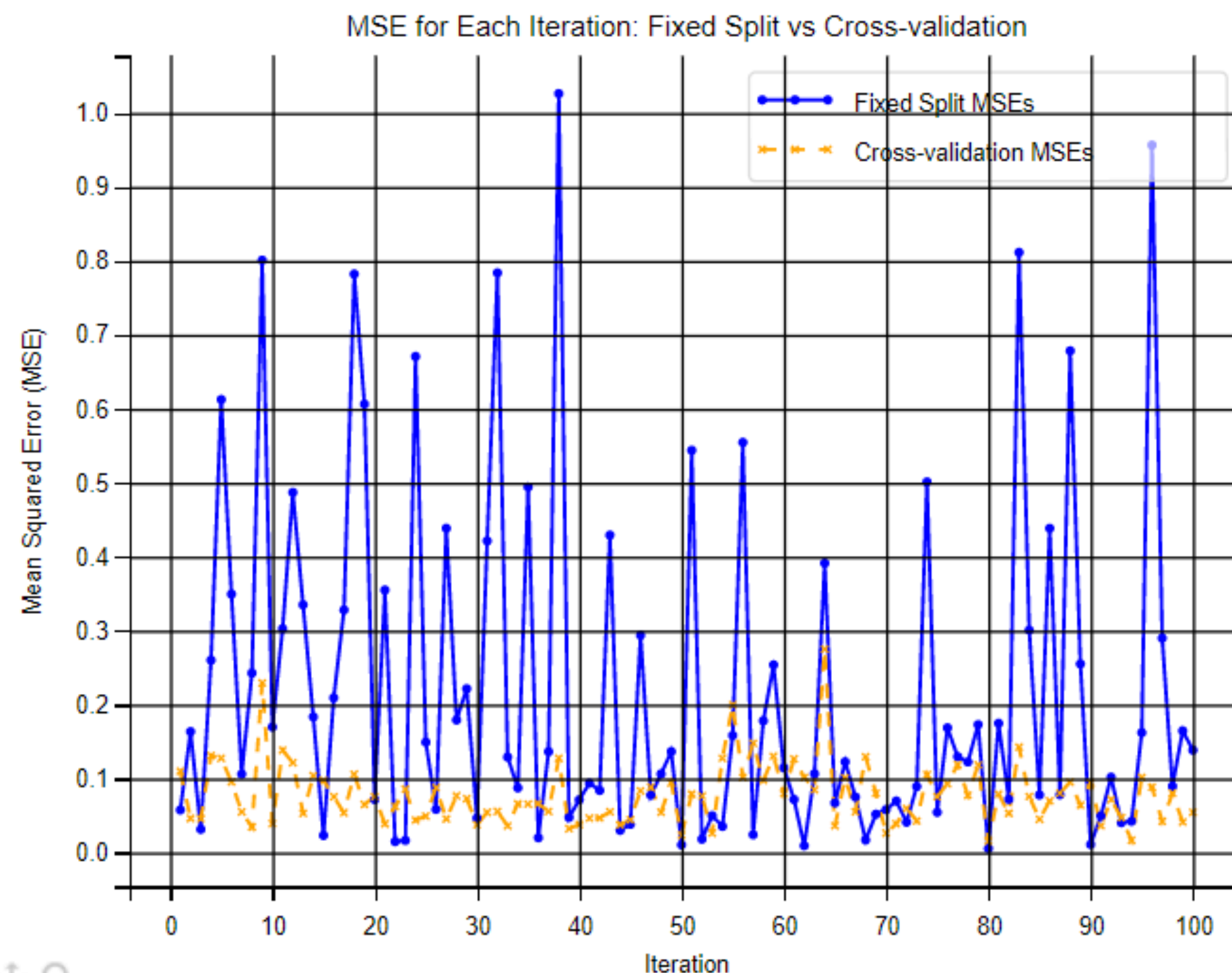
Conclusión: incluso con el mejor tamaño de prueba, esta no es una buena manera de seleccionar los datos de entrenamiento.

Elegir el mejor conjunto de entrenamiento

Cross-validation



Division fija vs Cross validation



- Cross-validation average MSE: 0.0778
- Fixed split average MSE: 0.2197
- Cross-validation worst MSE: 0.2749
- Fixed split worst MSE: 1.0262



Conclusión de Cross-validation:

- Una varianza muy baja.
- Mejores resultados generales.
- Evita el sobreajuste.

Conclusiones

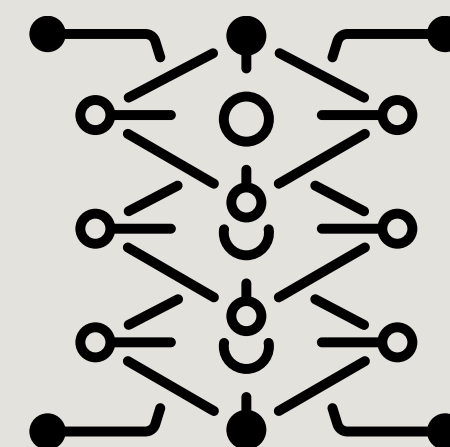
¿Cómo elegir el mejor conjunto de entrenamiento?

- Usar **cross-validation** para entrenar en diferentes subconjuntos.
- se selecciona un conjunto de entrenamiento que representa bien la distribución global de los datos.

Efecto en la capacidad de generalización:

- Mejora la capacidad de generalización al asegurar que el modelo aprenda patrones que se **generalicen a datos no vistos**.
- permite al modelo entrenar en diferentes porciones del conjunto de datos y **evitar el sobreajuste** a una sola partición

Resultado → Utilizando validación cruzada, nuestro perceptrón no lineal predice nuevos datos de manera efectiva con un error promedio muy bajo y un buen **capacidad de generalización**



Ejercicio 03

- Implementación del algoritmo de **perceptrón *MULTICAPA***.
- Se resuelve para el conjunto de datos definido en el archivo **TP3-ej3-digitos**

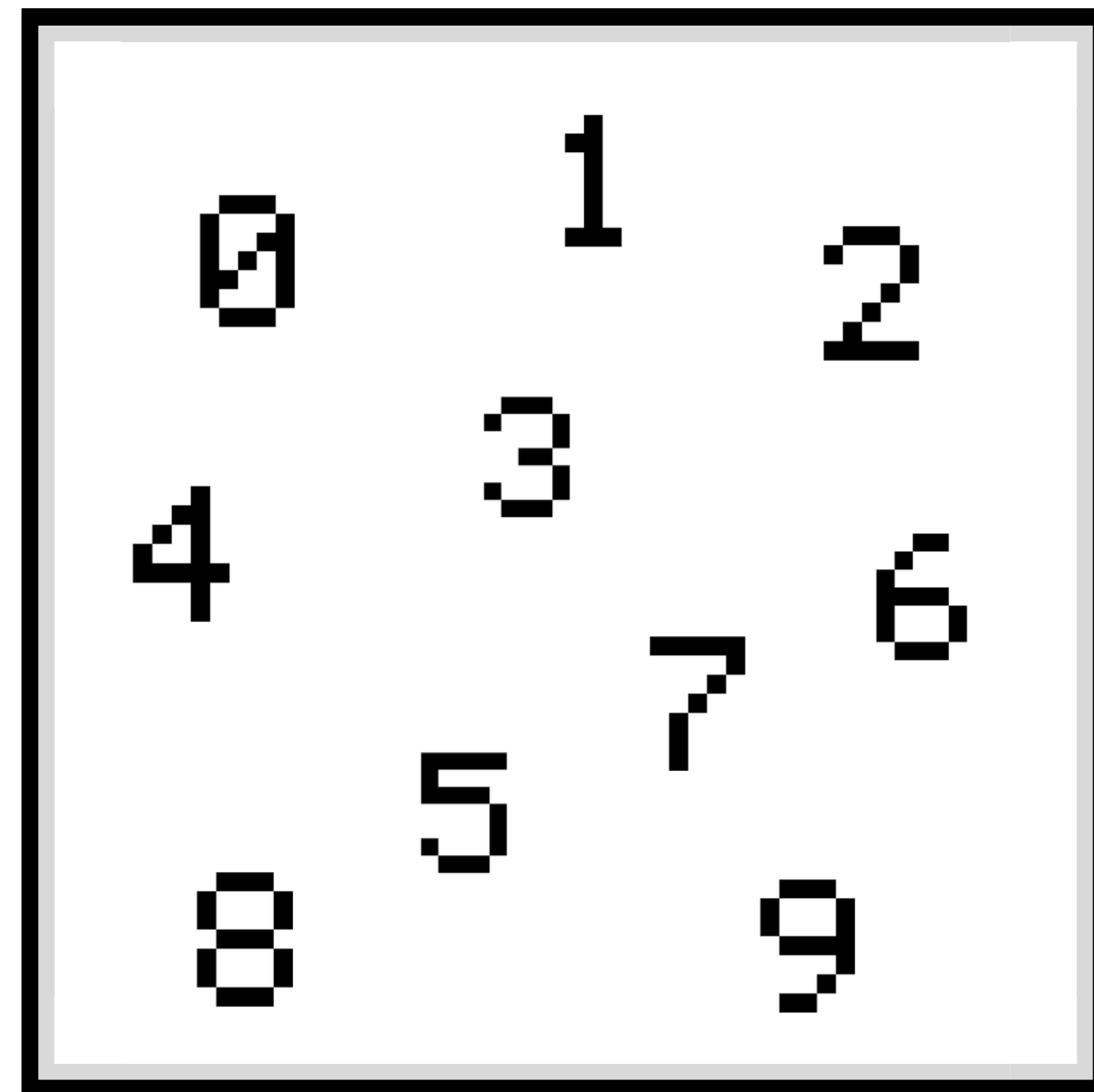


Aplanamiento de datos

De una estructura de datos multidimensional a
un vector unidimensional

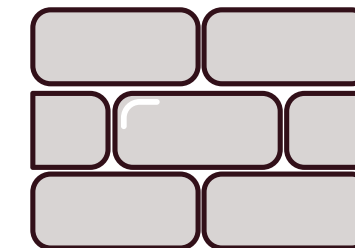
[70x5] → [10x35]

Cada fila corresponde a un número





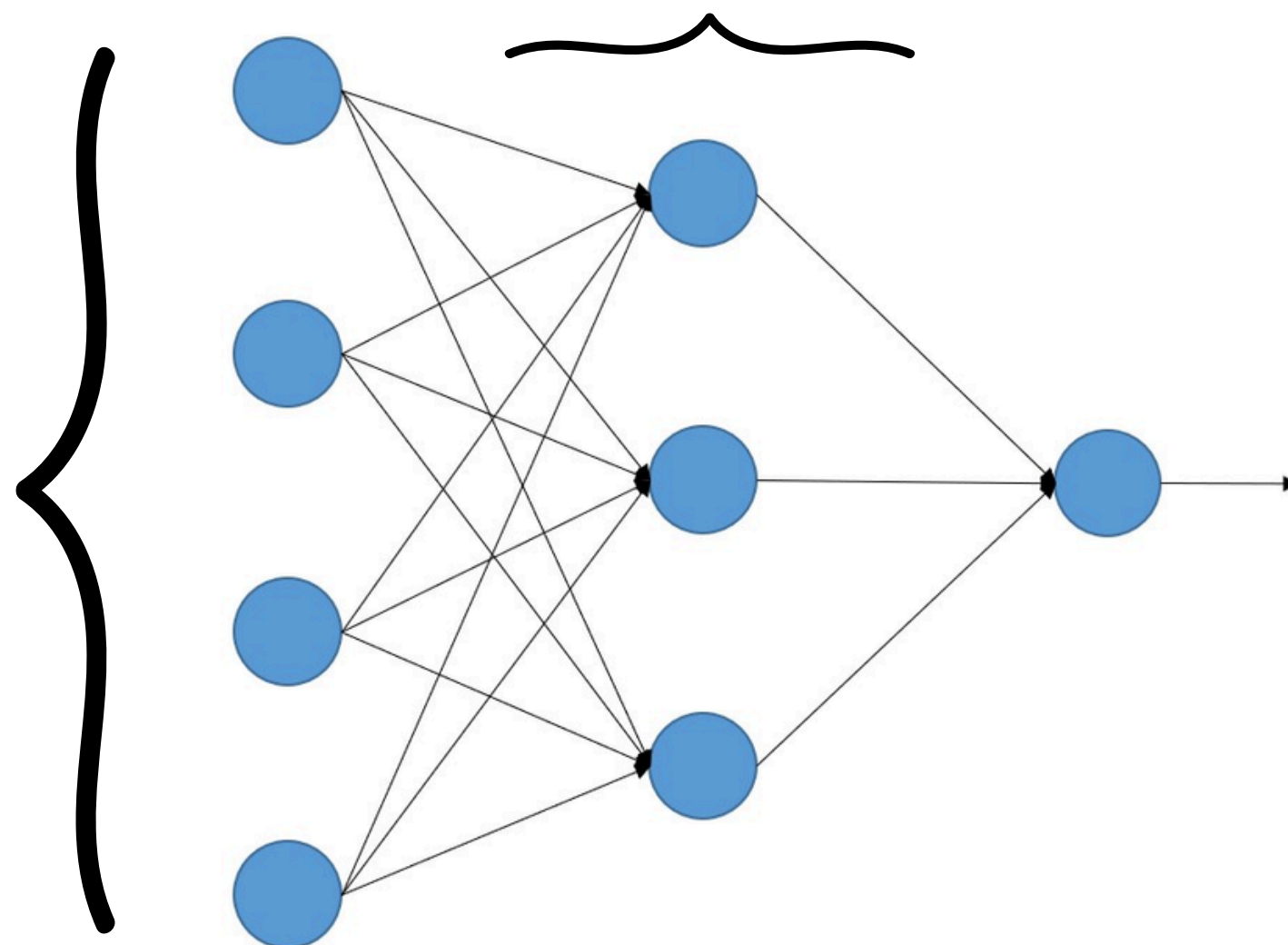
Ejercicio 03 - Configuración Base



Arquitectura

Las intermedias las probaremos
en los ejercicios

35 neuronas en la capa
inicial



Neuronas de salida
Clasificación binaria: 1
Clasificación multiclase: 10



Inicialización de pesos

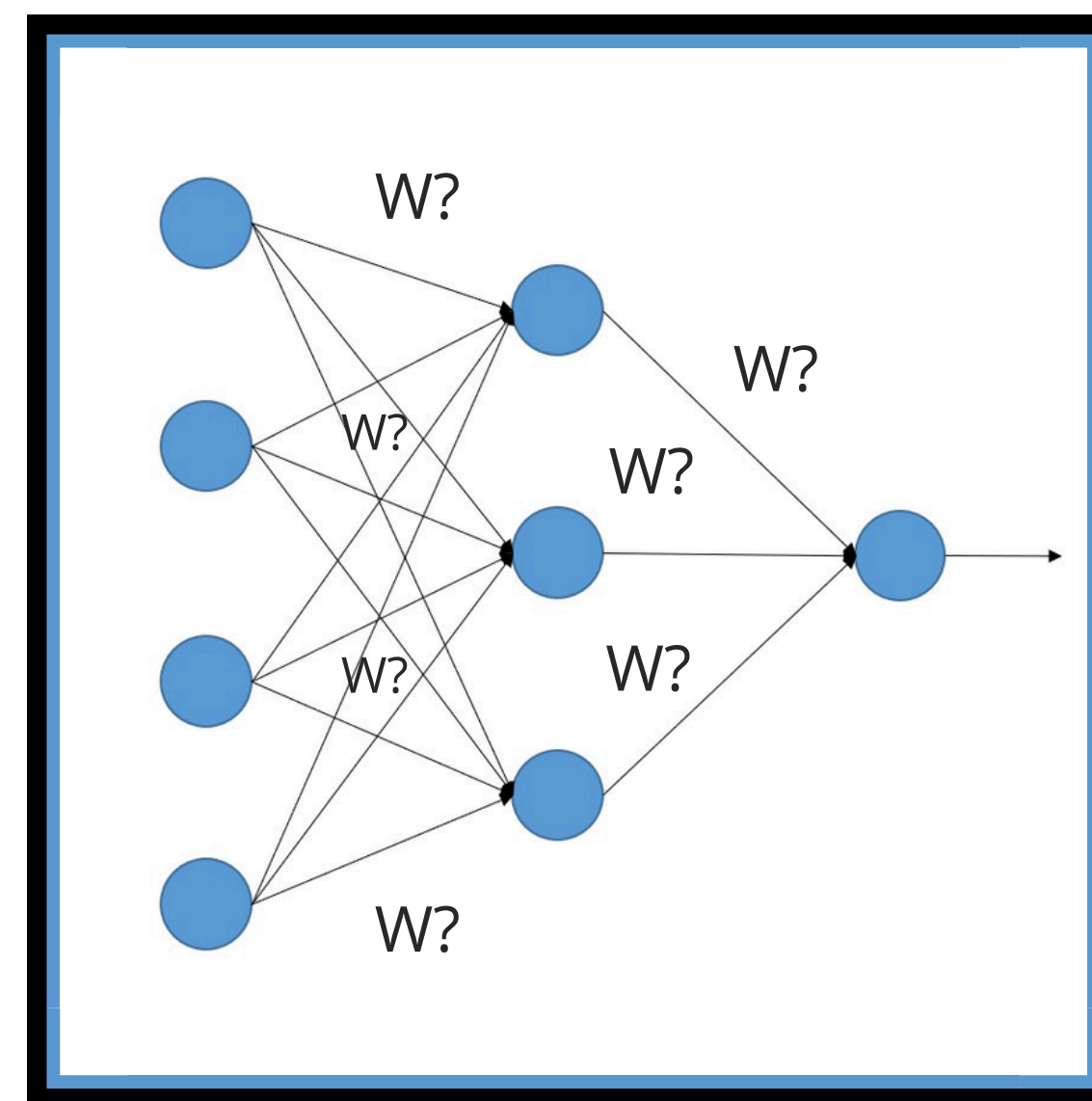
Aleatoria: Pesos en un rango pequeño aleatorio $(-0.01, 0.01)$

Ceros: Todos los pesos empiezan en cero (mal para entrenar)

Distribución Normal: Pesos de una distribución normal con media 0

Xavier: Ajusta según el tamaño de la capa, ideal para sigmoide/tanh

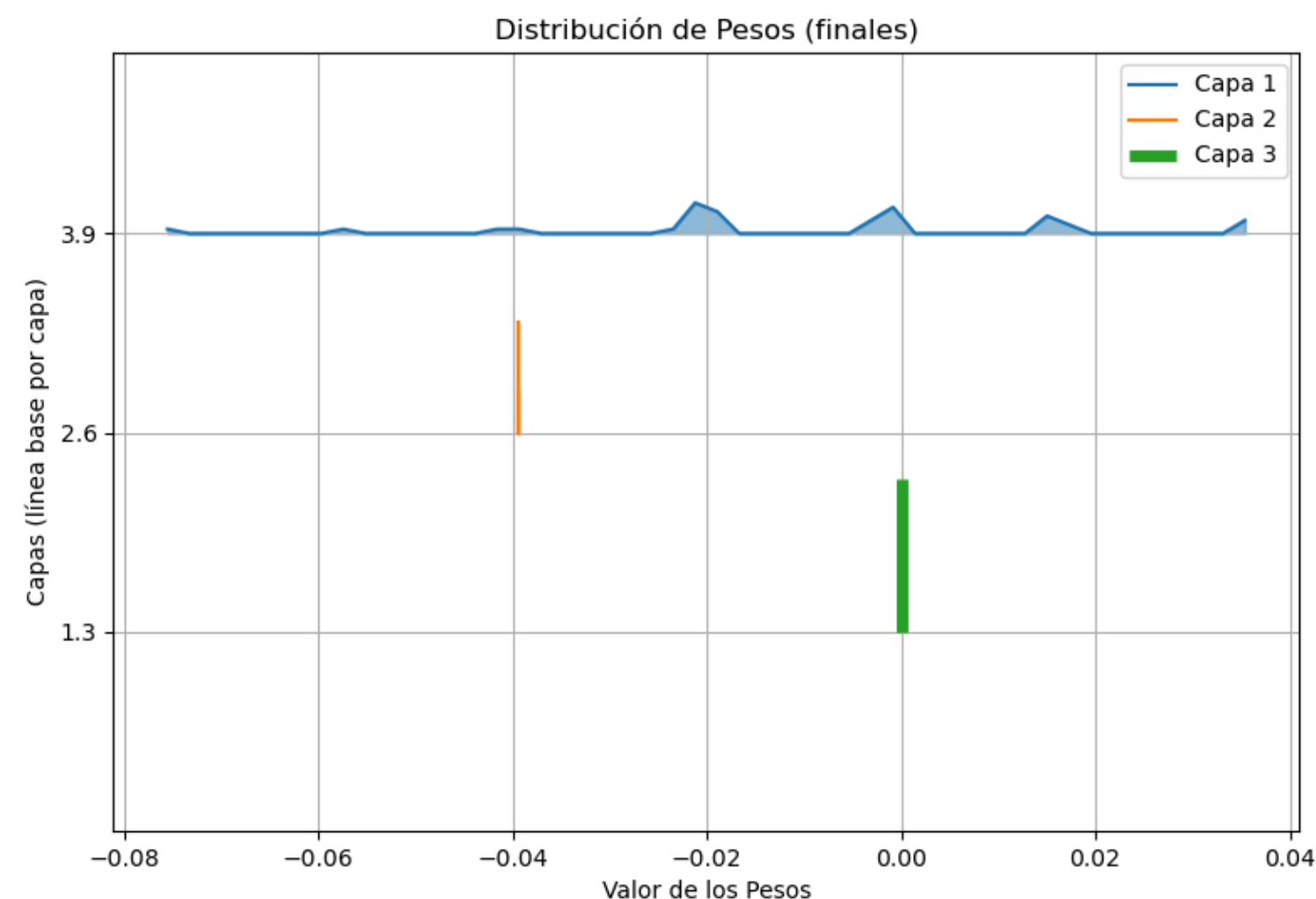
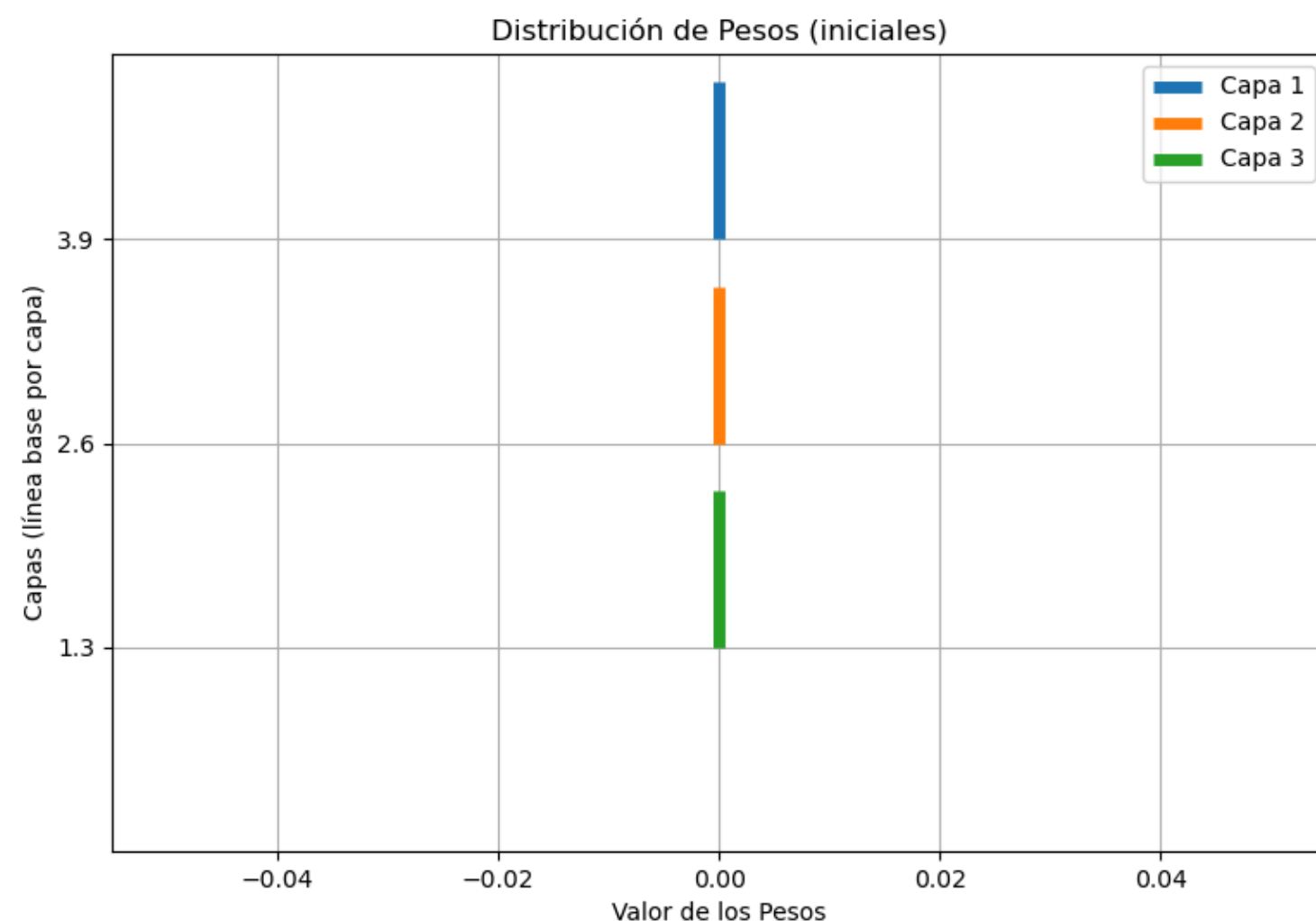
He: Escala para ReLU, eficiente en redes profundas





¿Cómo afectan a los pesos de la última capa del entrenamiento?

Comparación de Pesos Iniciales y Finales con Inicialización ZERO



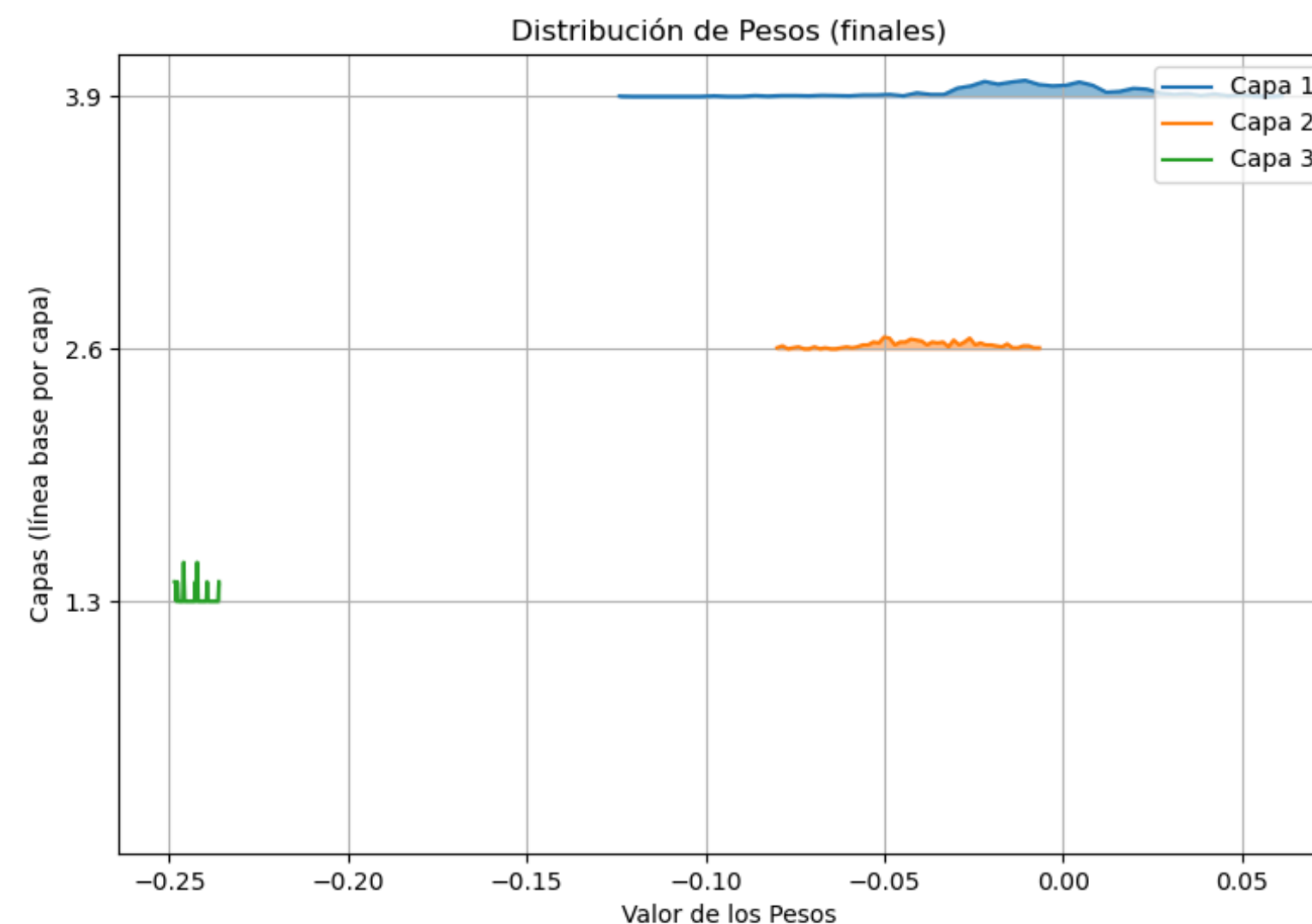
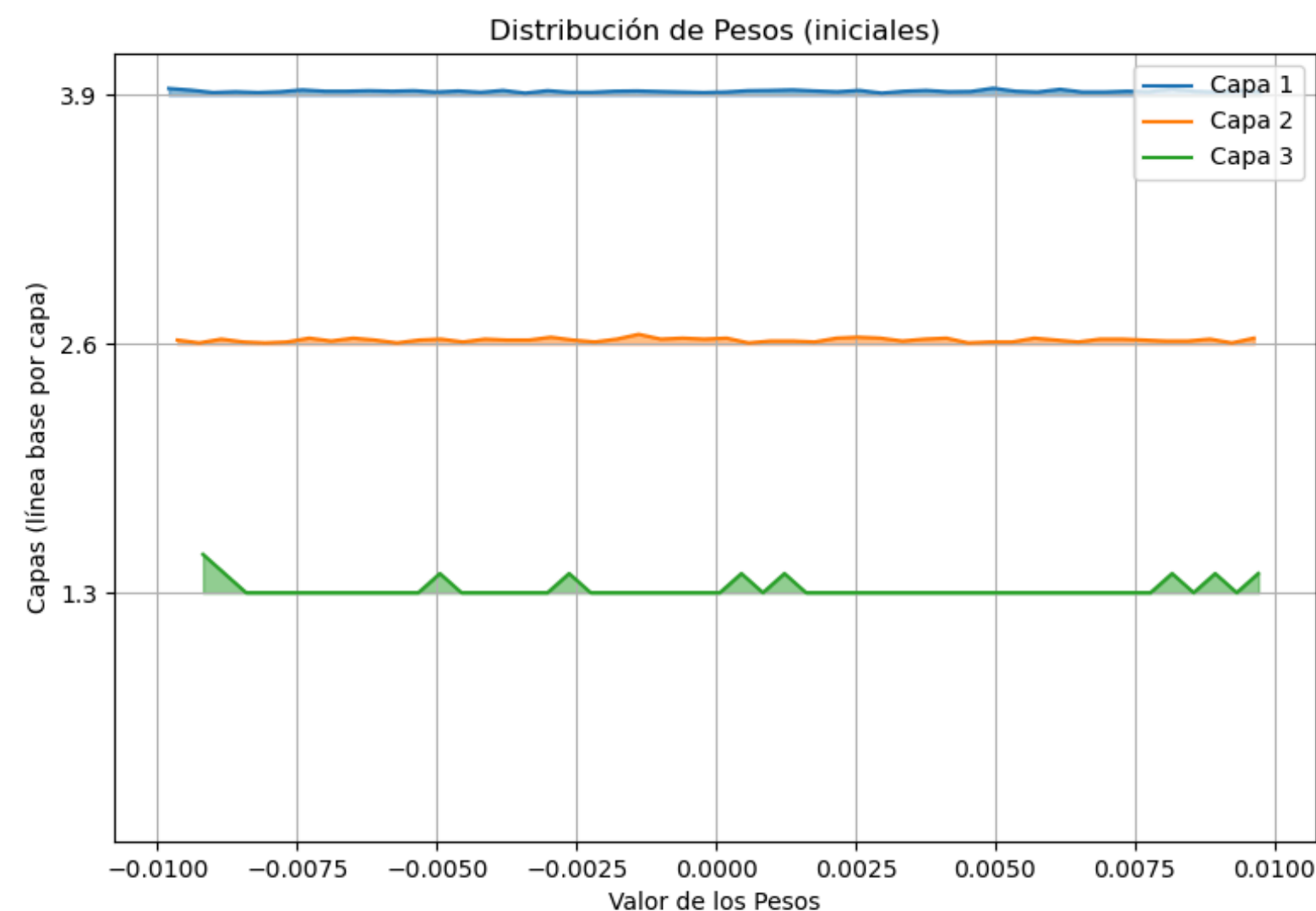
Todas las neuronas en una capa actualicen sus pesos de manera idéntica

Los pesos apenas han cambiado tras el entrenamiento, especialmente en las capas 2 y 3, lo que sugiere que el modelo podría no haber capturado las características necesarias para realizar la clasificación deseada.



¿Cómo afectan a los pesos de la última capa del entrenamiento?

Comparación de Pesos Iniciales y Finales con Inicialización RANDOM

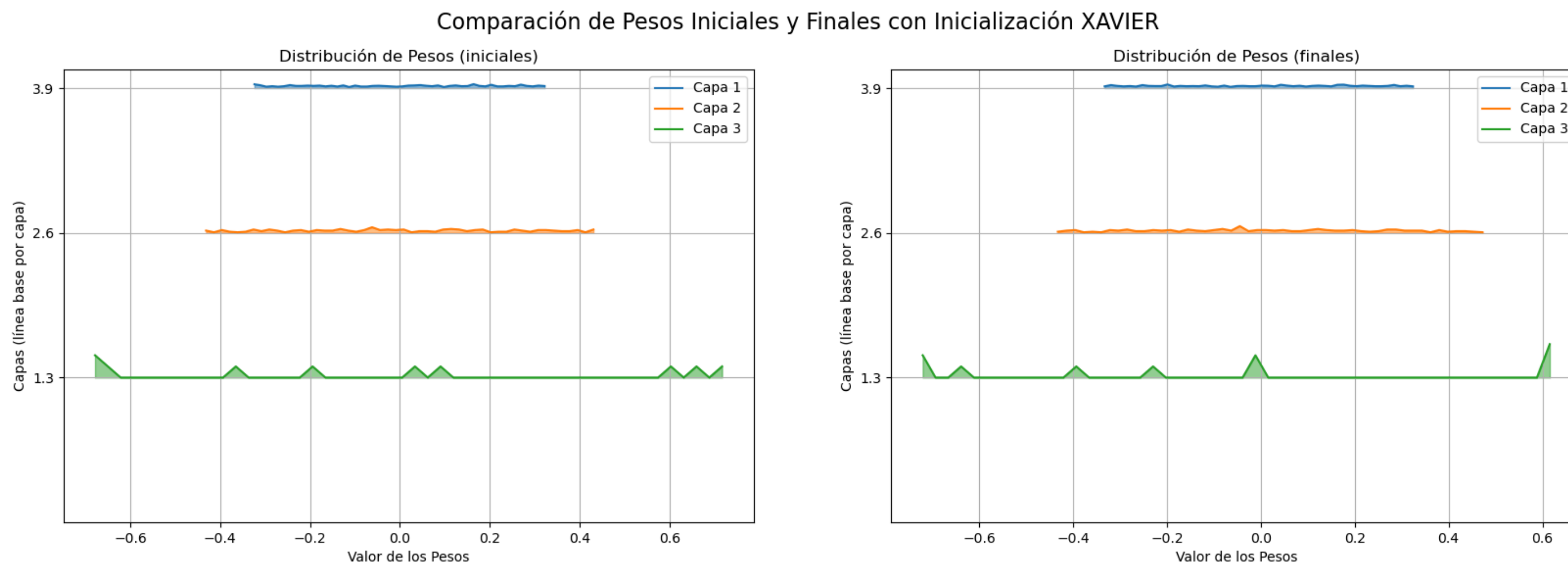


Los pesos de las capas más profundas tienden a concentrarse en valores específicos debido al desvanecimiento del gradiente durante el entrenamiento.

La distribución de pesos en capas iniciales es más dispersa, mientras que en las capas profundas los pesos tienden a agruparse, afectando su capacidad de aprendizaje



¿Cómo afectan a los pesos de la última capa del entrenamiento?



Los pesos se mantienen en un rango controlado, lo que ayuda a evitar saturar las activaciones en redes profundas

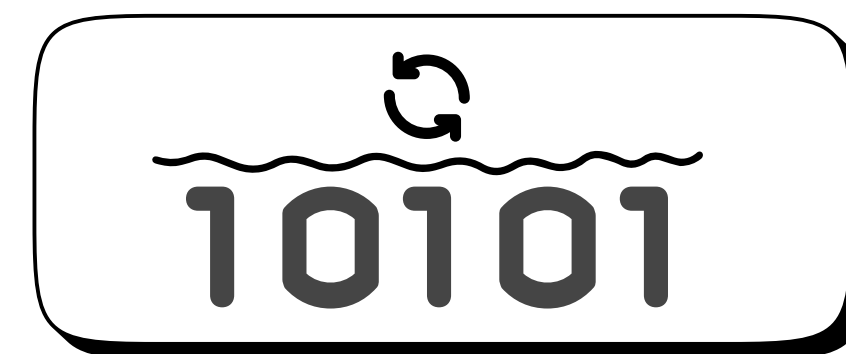
La falta de variación significativa en los pesos al inicio es normal y está diseñada para prevenir problemas de desvanecimiento del gradiente



Modos de entrenamiento

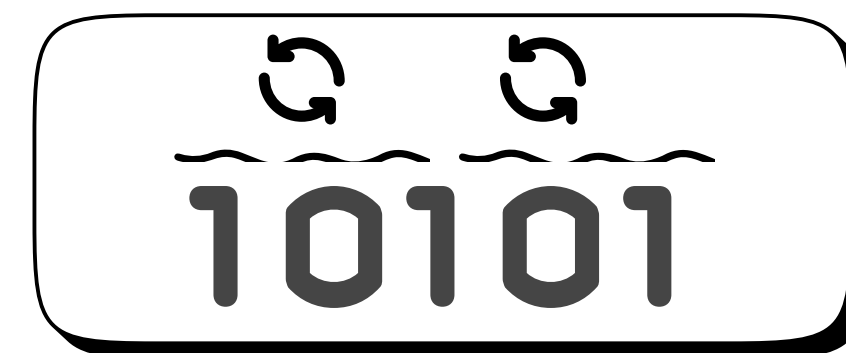
Batch: Procesa todo el dataset completo en cada iteración antes de actualizar los pesos de la red.

Es bueno porque ofrece gradientes más estables y actualizaciones precisas, pero puede ser ineficiente en términos de memoria y tiempo con datasets grandes.



Mini-batch: Divide el dataset en pequeños lotes (mini-batches) y actualiza los pesos después de procesar cada lote.

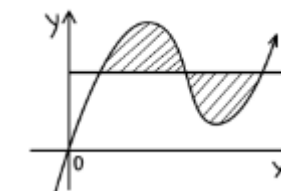
Es un buen equilibrio entre precisión y eficiencia, ideal para manejar datasets grandes sin consumir demasiados recursos, aunque los gradientes son menos estables que en batch.



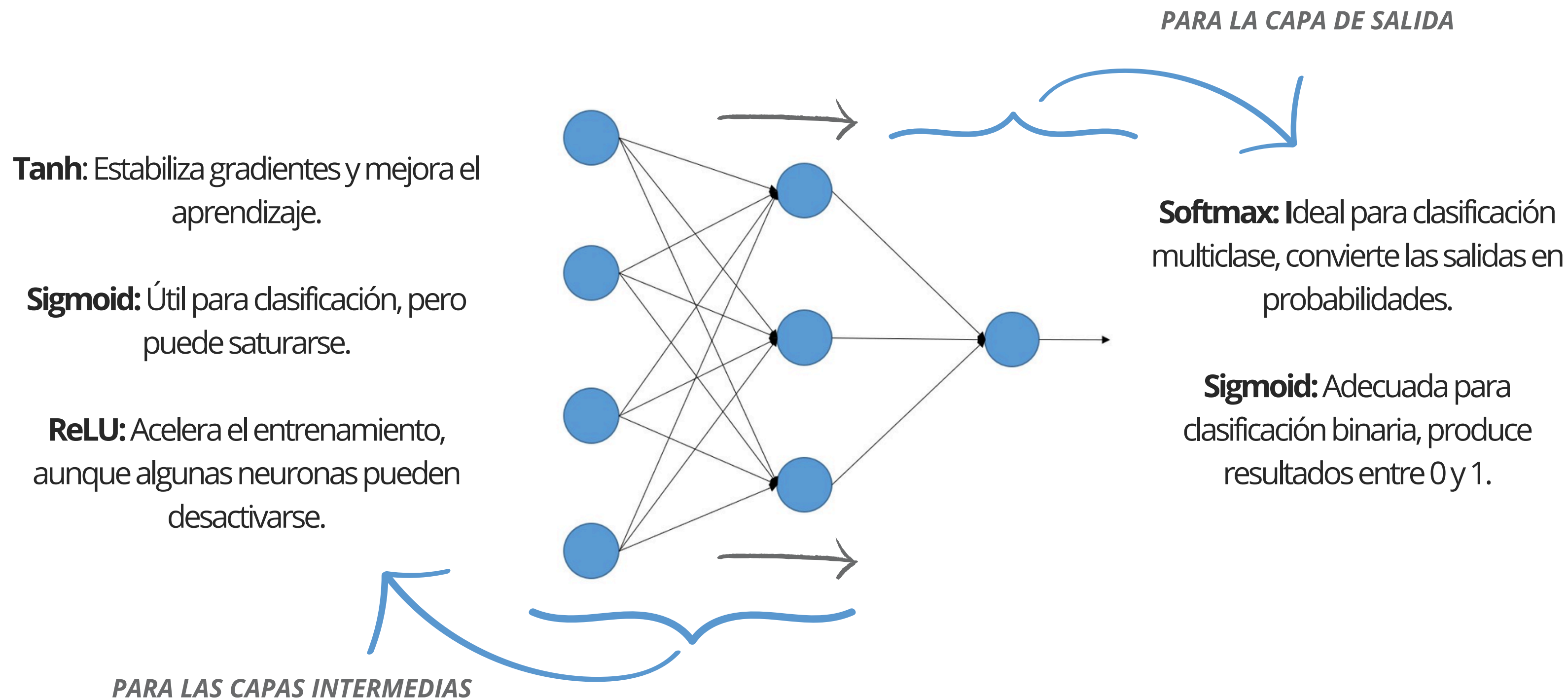
Online: Actualiza los pesos después de procesar cada ejemplo individual del dataset.

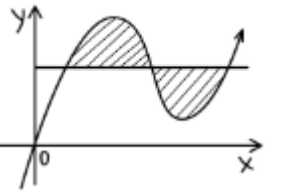
Es excelente para datasets muy grandes o para entrenar en tiempo real, pero puede generar actualizaciones ruidosas y tardar más en converger de manera estable.





Función de activación





PERO, ¿SE NECESITA NORMALIZAR PARA LAS FUNCIONES?

El conjunto de datos ya está en el rango de 0 a 1 (valores binarios), no es necesario aplicar normalización adicional.

Las funciones de activación de salida **sigmoid y softmax** están diseñadas para controlar salidas dentro de este rango, lo que las hace ideales para este tipo de problema.

Las funciones de activación en las capas intermedias, como **tanh y ReLU**, no requieren normalización previa, ya que pueden manejar eficientemente entradas en este rango sin comprometer su rendimiento.

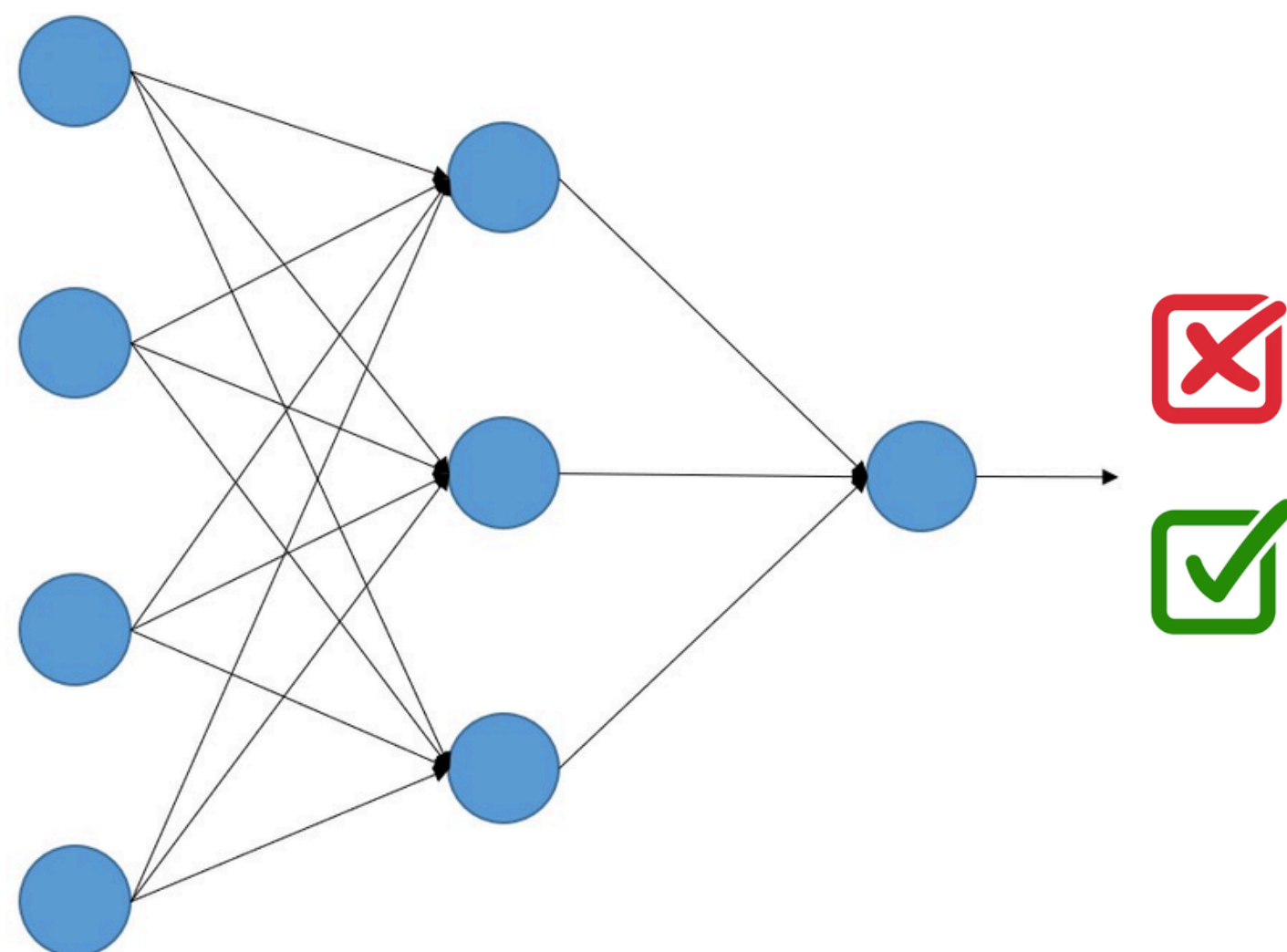


Función de error

Mean Squared Error

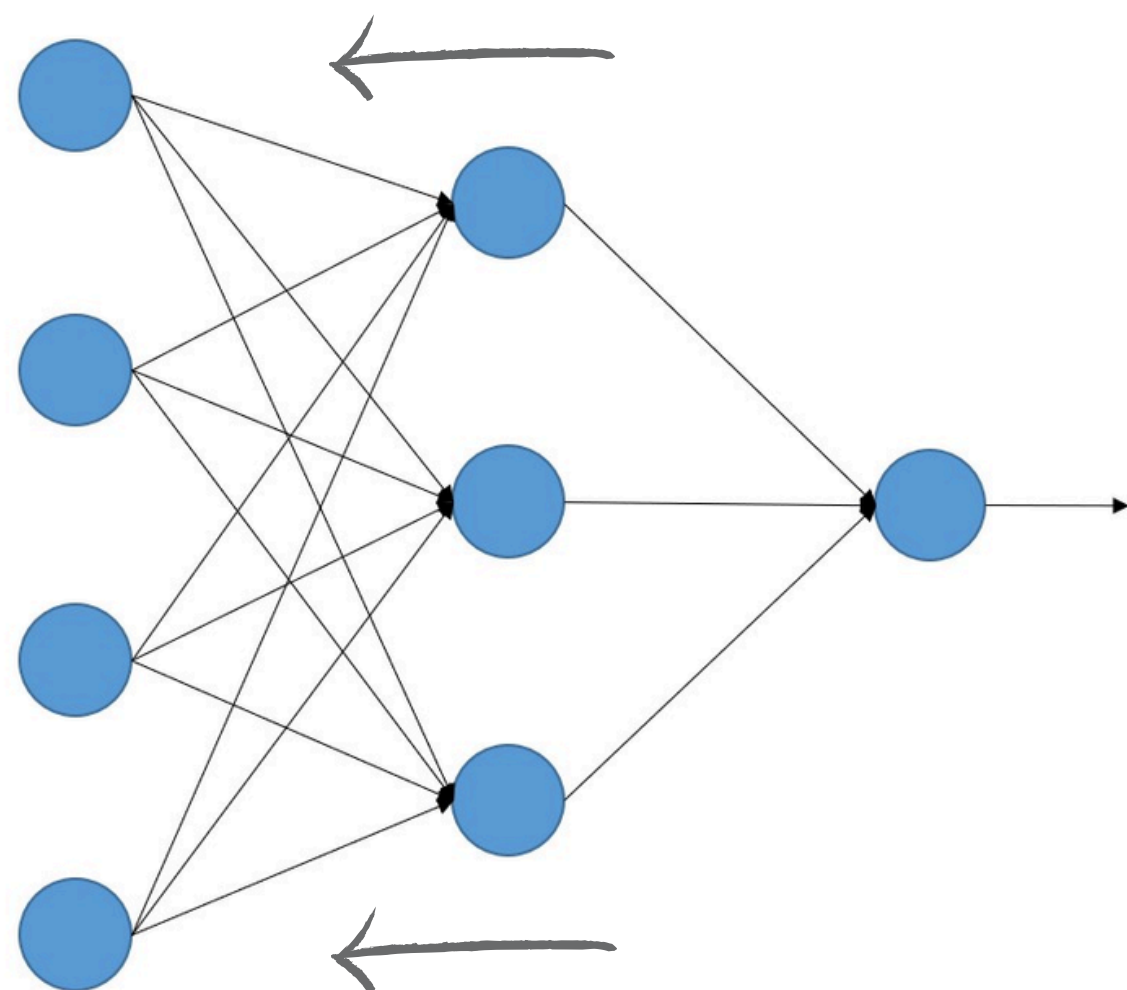
Se utiliza comúnmente en problemas de **regresión** y **clasificación binaria** donde las salidas son continuas o probabilidades

MSE mide la diferencia entre los valores predichos (en probabilidades) y las etiquetas reales, penalizando más los errores grandes, lo que ayuda a ajustar el modelo de manera precisa a las predicciones.





Optimizadores



Adam: Combina lo mejor de varios métodos, ajusta los pesos eficientemente y es ideal para datos complejos .

Momentum: Acelera el aprendizaje al evitar mínimos locales, útil cuando los gradientes son pequeños.

Gradient Descent: Método básico, ajusta los pesos lentamente, adecuado pero menos eficiente que Adam o Momentum.

Adaptive LR para el ajuste automático de la la tasa de aprendizaje, ayudando a evitar que el aprendizaje sea demasiado lento o se estanque.

1 0 1 0 1 0
0 1 0 1 0 1
1 0 1 0 1 0
0 1 0 1 0 1



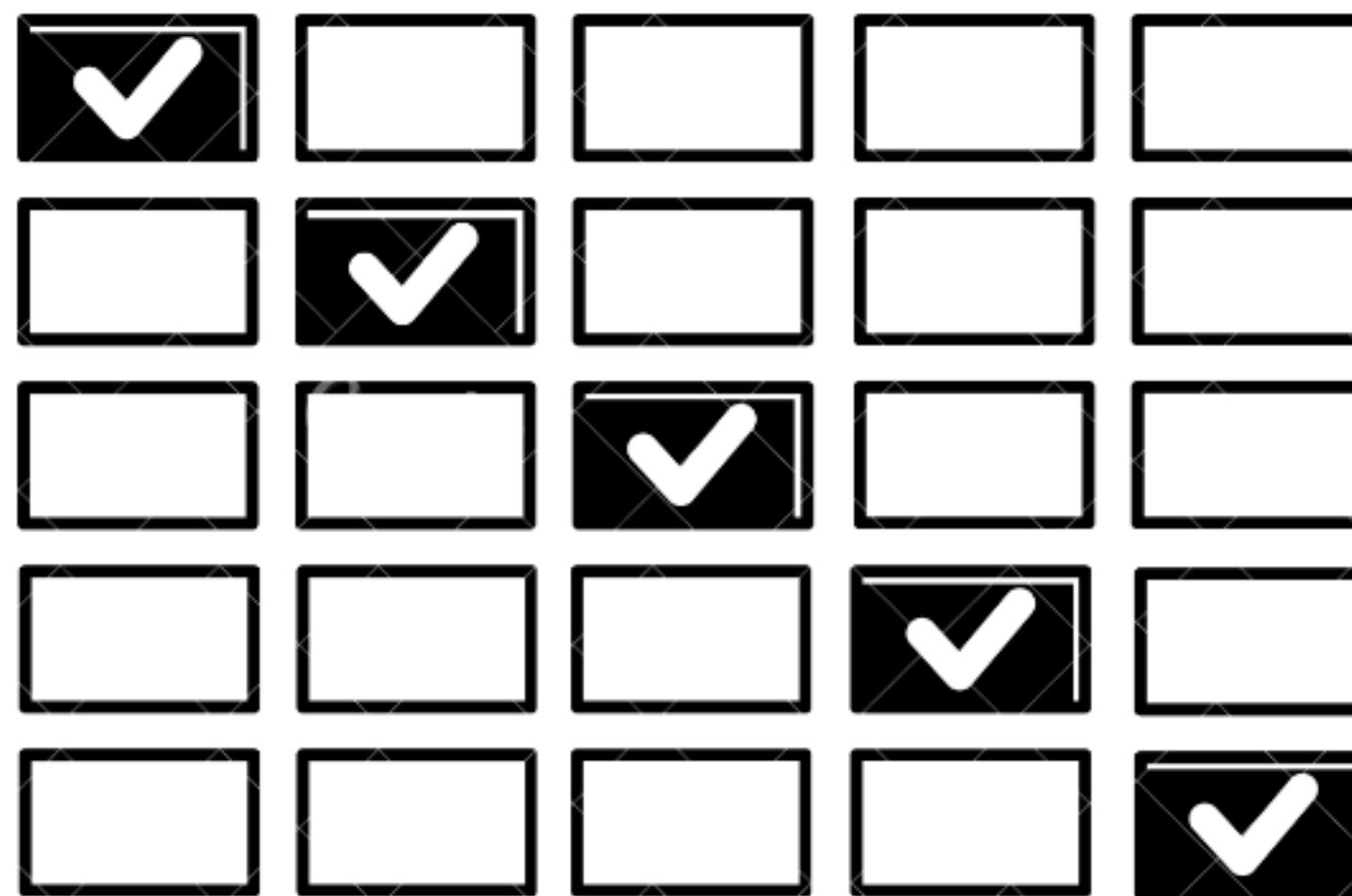
Ejercicio 3B

Reconocimiento de Paridad

Training vs Testing

5-FOLD CROSS VALIDATION

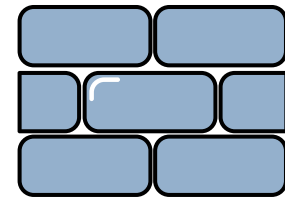
***5 partes
iguales***



***2 números
aleatorios***



Ejercicio 03B - Configuración Base



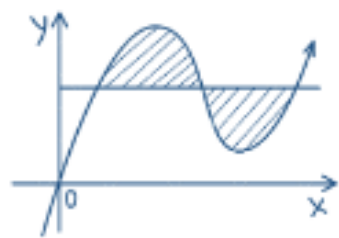
[35,20,10,1]



Xavier



Batch



**Tanh (hidden)
Sigmoid (output)**



Momentum



$\epsilon = 0$



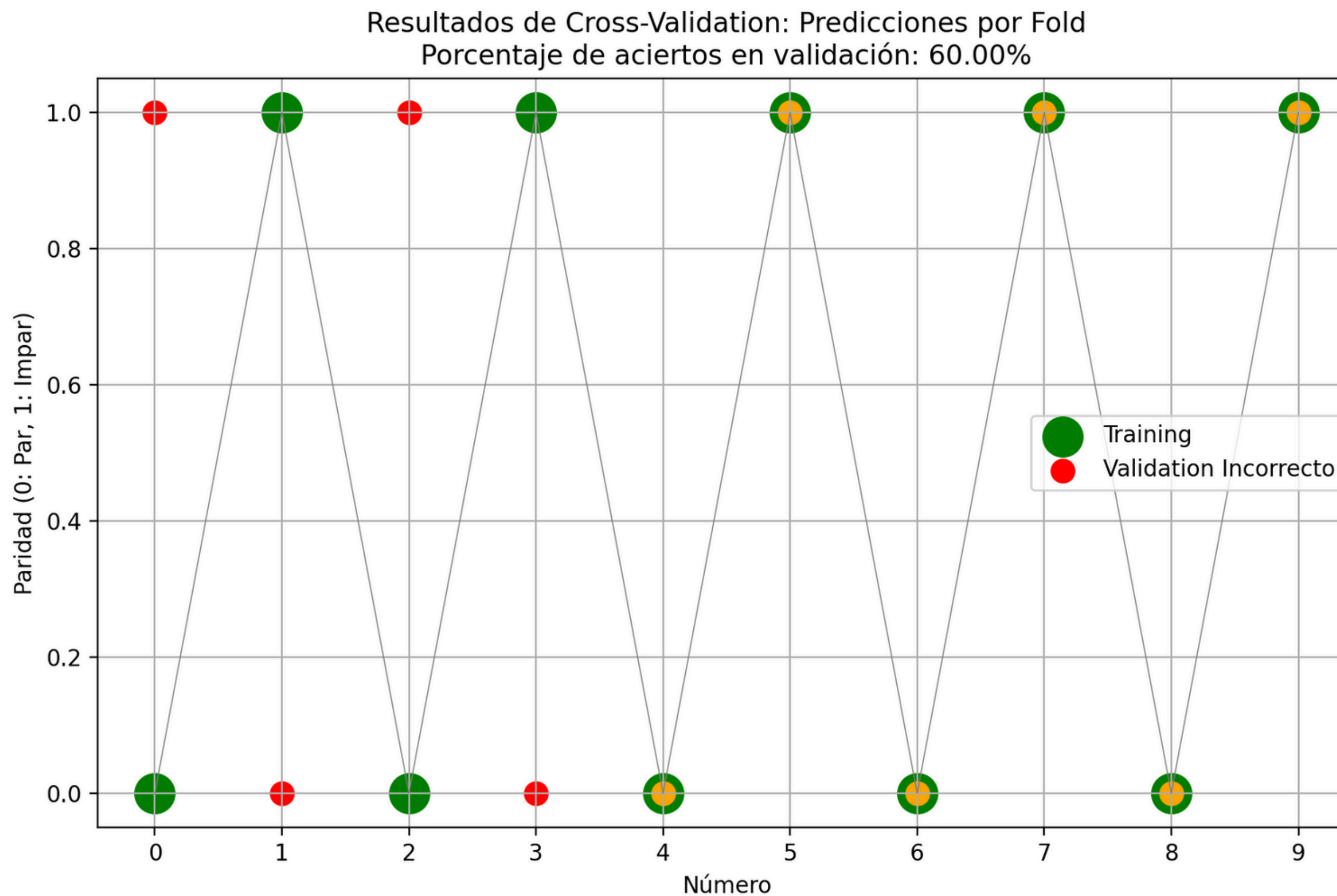
**LR adaptive
LR inicial = 0.01**



Ejercicio 03B - Análisis de datos

**Buena combinación teórica*

6/10

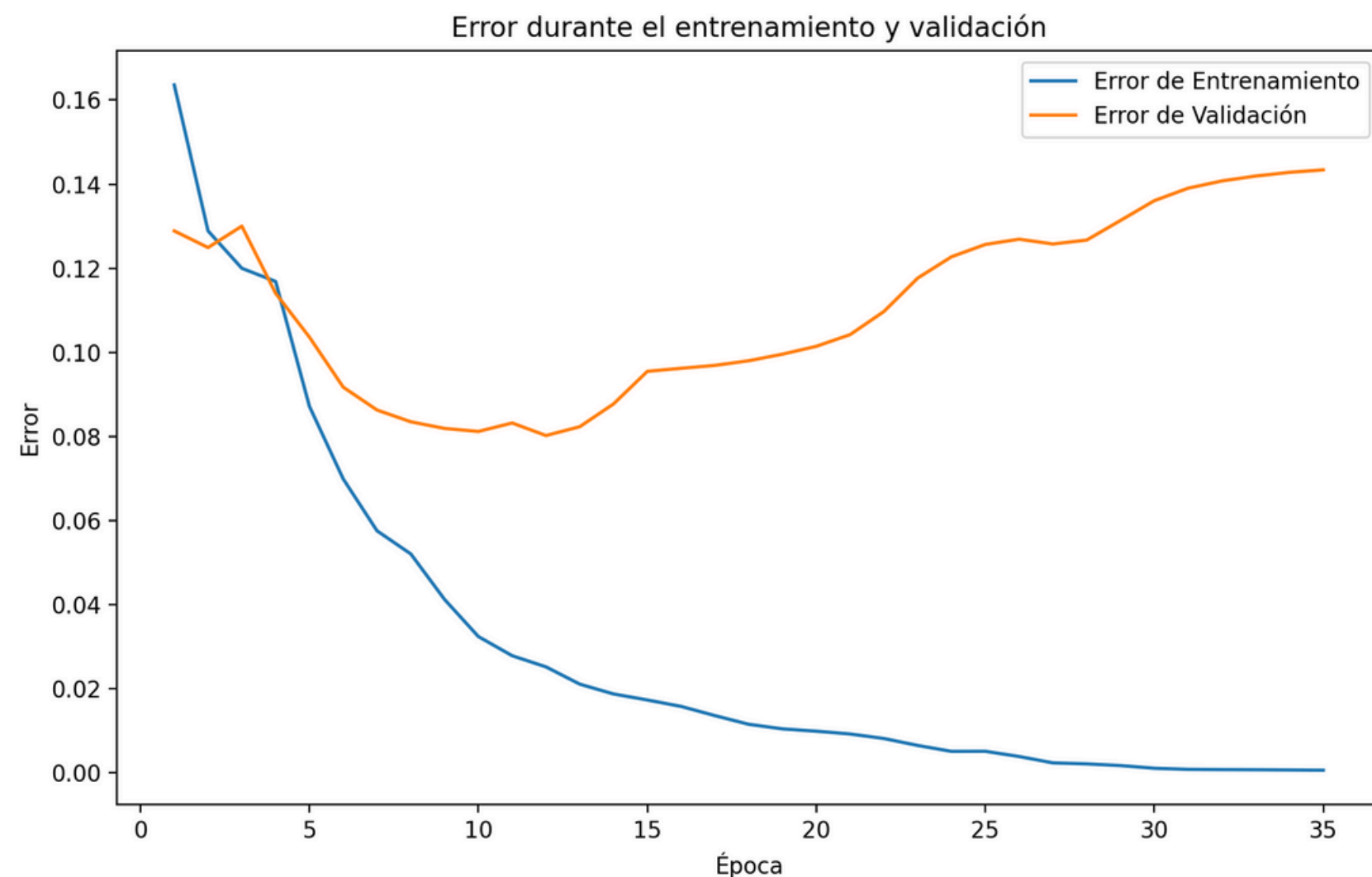




Ejercicio 03B - Análisis de datos

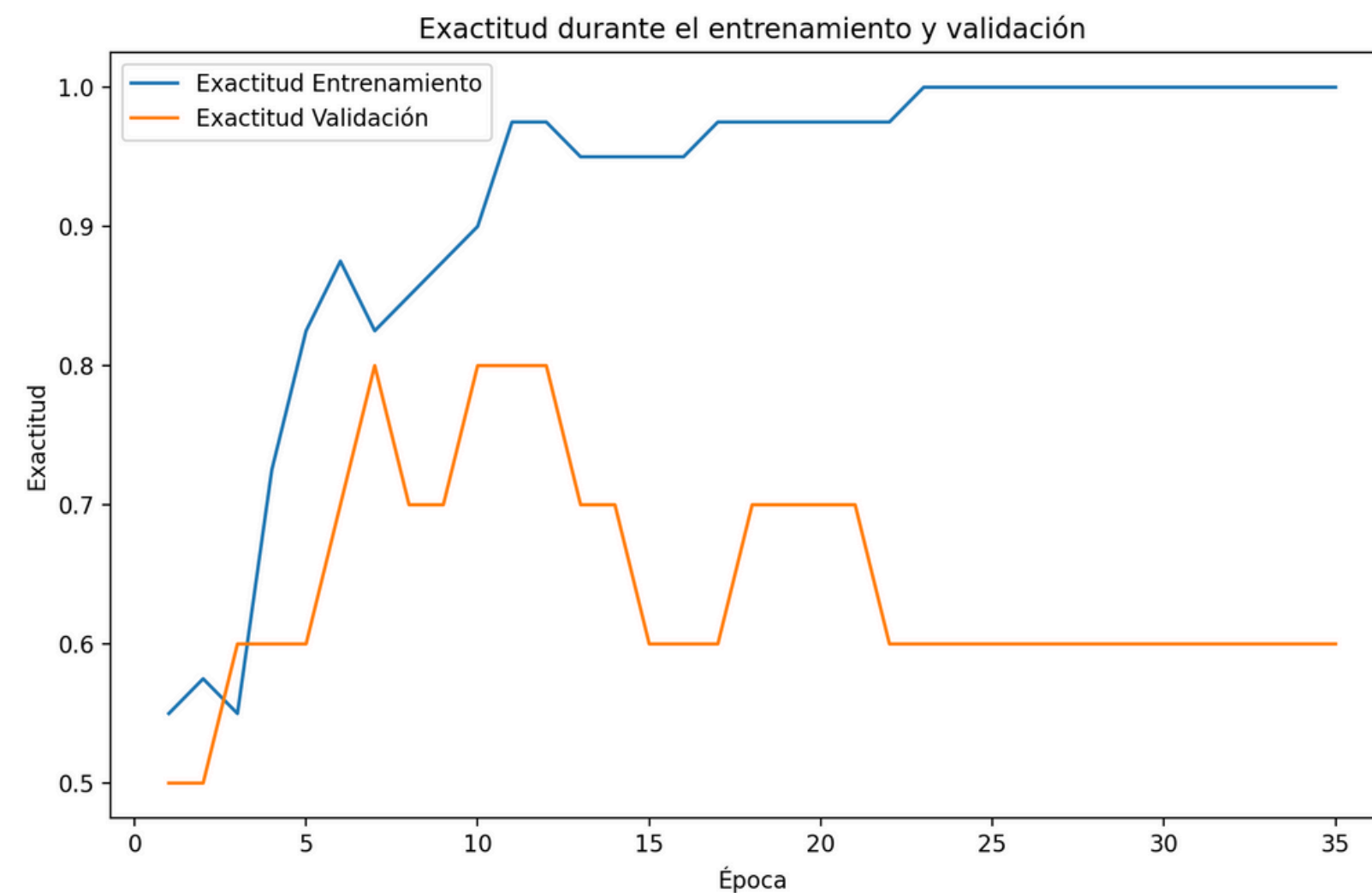
**Buena combinación teórica*

● Testing
● Training



ERROR:

el modelo está aprendiendo "demasiado bien" los ejemplos que ya ha visto.
Pierde la capacidad de generalizar a nuevos ejemplos

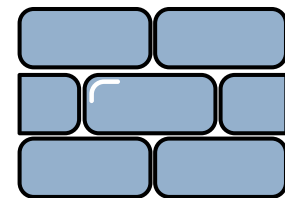


ACCURACY::

Validación se estanca en 60%

CLARO EJEMPLO DE SOBREAJUSTE

↑ Ejercicio 03B - Análisis de datos



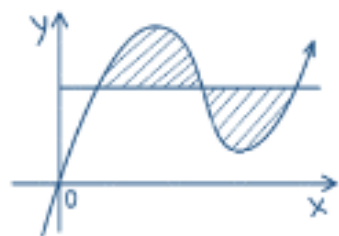
[35,20,10,1]



Random



Batch



**Tanh (hidden)
Sigmoid (output)**



Momentum



$\epsilon = 0$

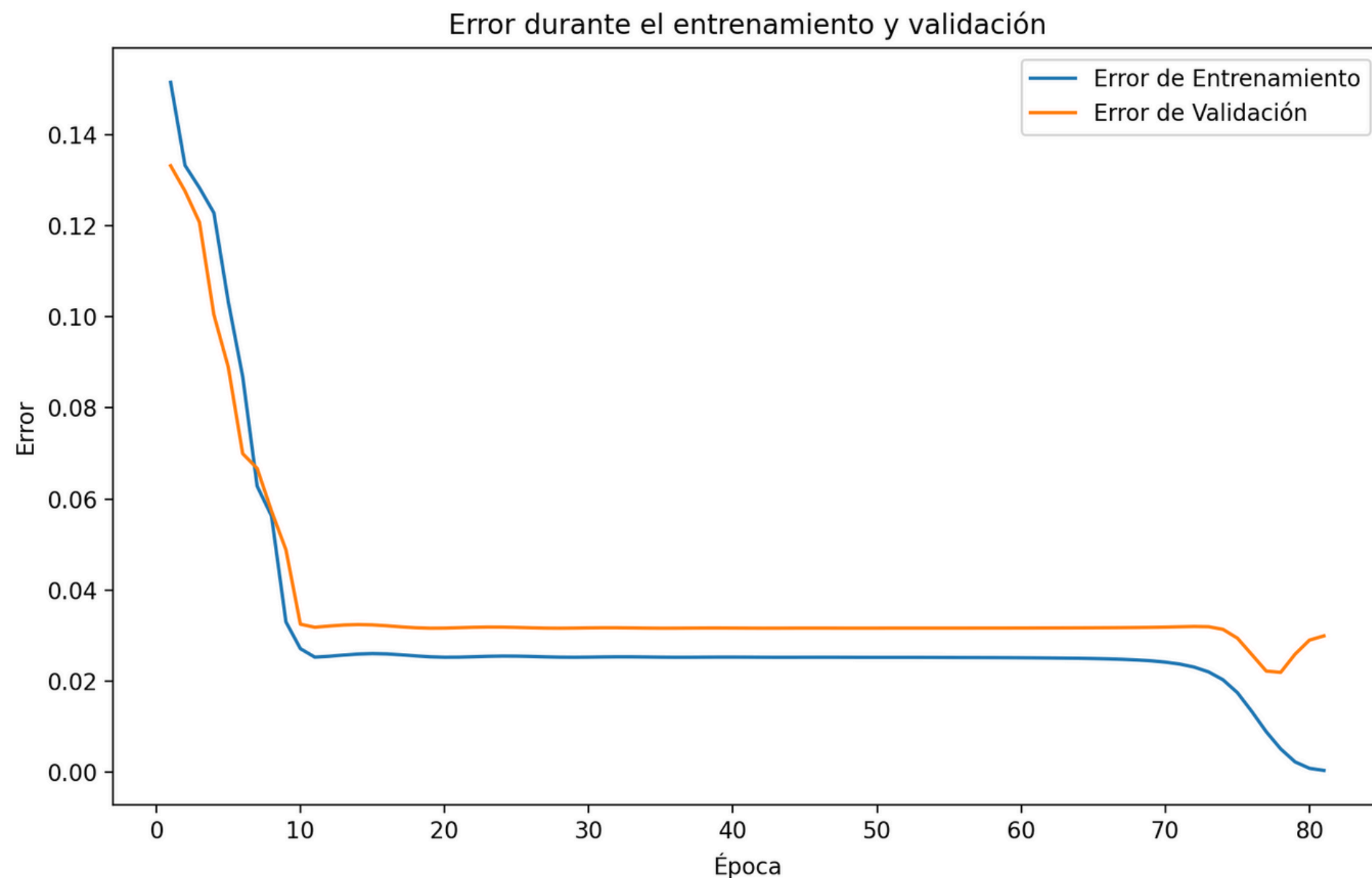


**LR adaptive
LR inicial = 0.01**



Ejercicio 03B - Análisis de datos

**Inicialización de pesos y error*



9/10

Las curvas se juntan y bajan de manera consistente
La generalización es buena y no hay sobreajuste

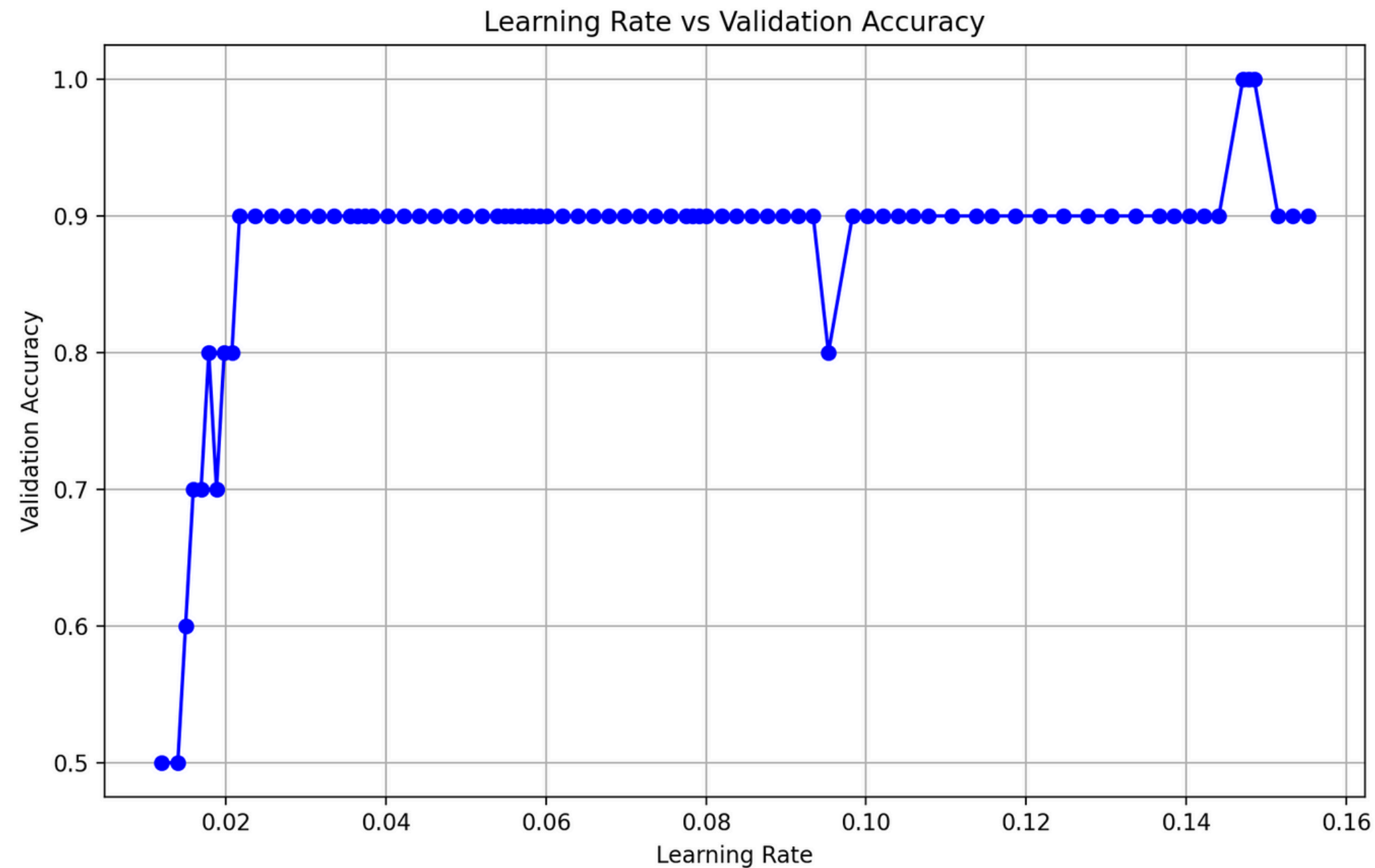
Lo epochs se aumentan de 12 a 80

Modelo más preciso pero mayor tiempo de entrenamiento



Ejercicio 03B

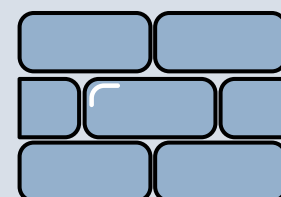
*Modelo funciona bien con la mayoría de los LR que se adaptan
No esta sobreajustando ni subntrenando*



Modelo necesitaba mayor LR para aprender mejor

**Inicialización de pesos y error*

↑ Ejercicio 03B - Análisis de datos



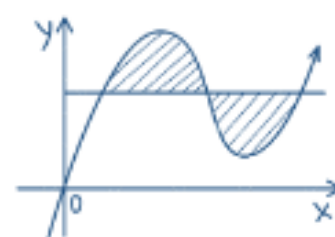
[35,64,16,8,1]



Xavier



Batch



**Tanh (hidden)
Sigmoid (output)**



Momentum



$\epsilon = 0$



**LR adaptive
LR inicial = 0.01**

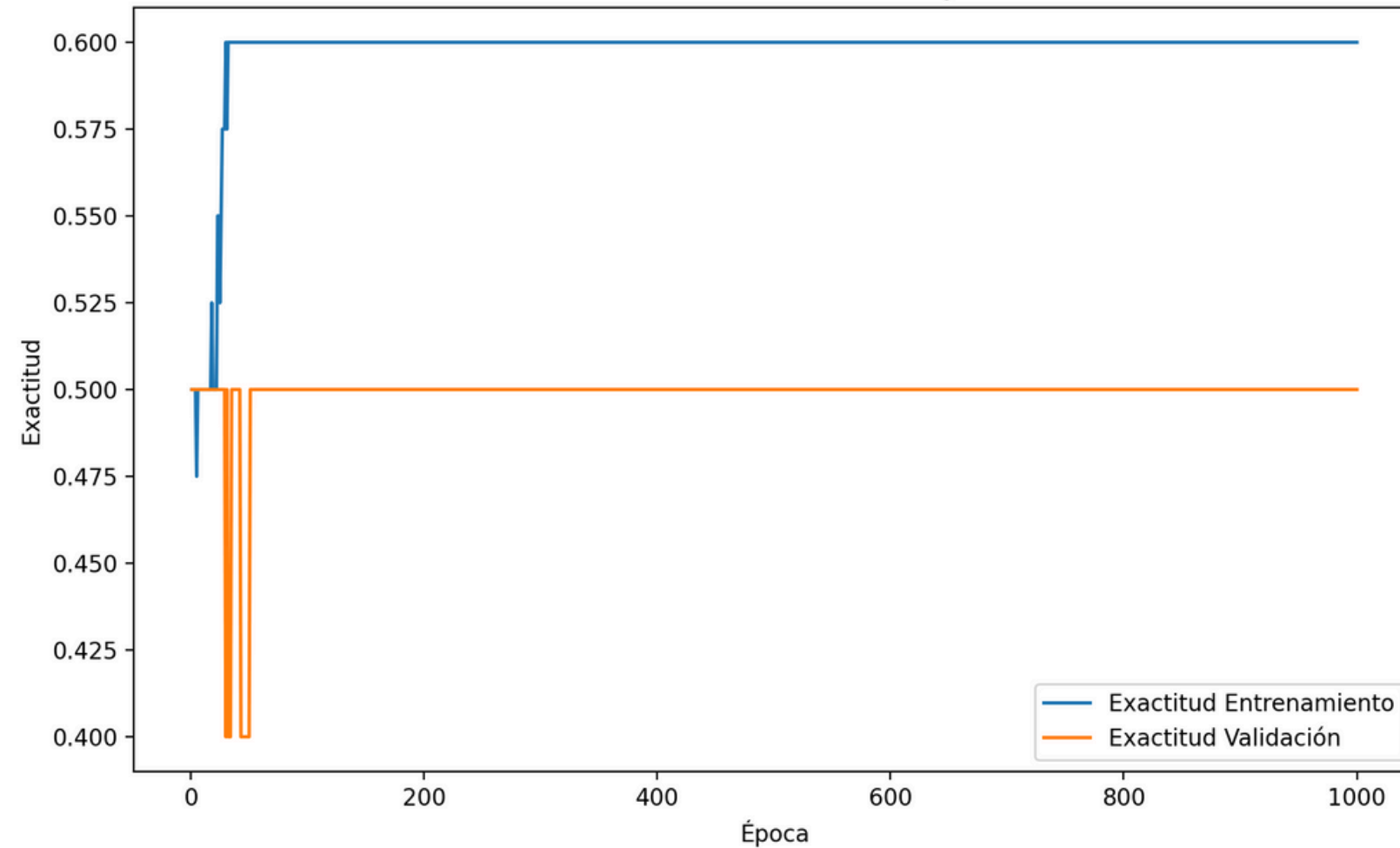


Ejercicio 03B - Análisis de datos

5/10

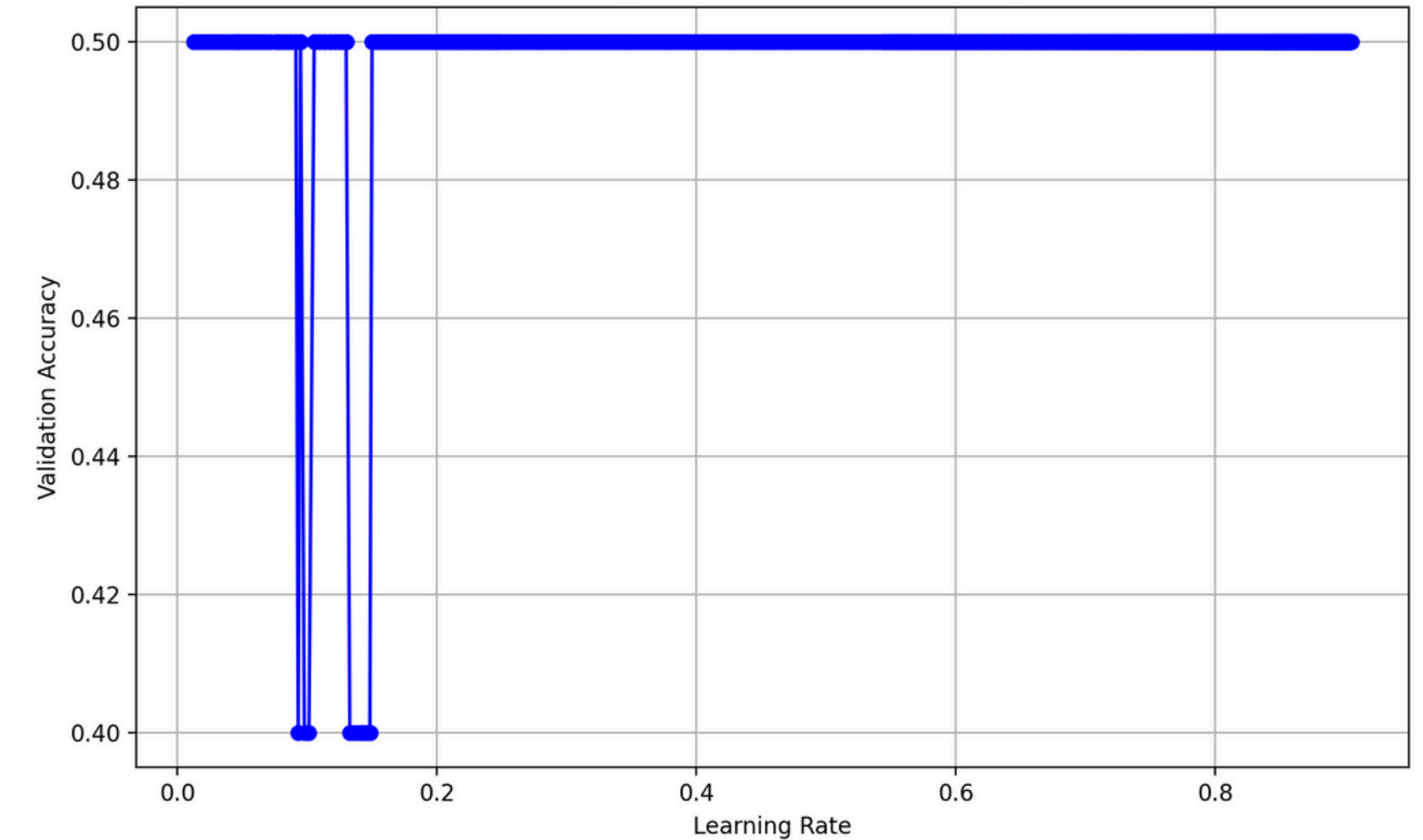
ACCURACY VS EPOCHS

Exactitud durante el entrenamiento y validación



LR VS ACCURACY

Learning Rate vs Validation Accuracy



Una arquitectura más compleja no necesariamente mejora el rendimiento

Aumentar la arquitectura no mejora el rendimiento porque los datos no son suficientes para una red tan compleja, lo que provoca sobreajuste y dificultades en la propagación del gradiente en redes profundas.



Conclusiones generales

La inicialización random en un rango limitado mejora la precisión al 90%, pero requiere más epochs para converger. Los pesos iniciales pequeños generan un proceso de aprendizaje más lento, pero estable.

Sin el ajuste del LR, las gráficas de entrenamiento y validación se separan más, requiriendo más epochs para alcanzar convergencia. Esto indica que la tasa de aprendizaje no está optimizada y el modelo no puede adaptarse bien durante el entrenamiento.

Aumentar la **complejidad de la arquitectura** no ayuda debido a la cantidad limitada de datos.



Conclusiones generales


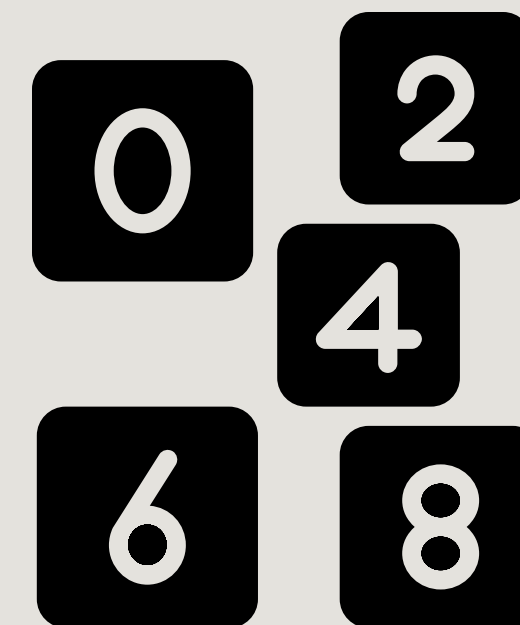
Al cambiar a **gradient descent**, se observa una mayor separación en el error y una precisión menor.

Esto sugiere que el modelo tiene más dificultades para ajustar los pesos de forma eficiente, ya que gradient descent no tiene la capacidad de "suavizar" las actualizaciones como lo hace momentum.

Con valores más bajos de **beta para momentum**, la actualización de los pesos pierde el efecto de "memoria" que provee momentum, lo que lleva a una caída en la precisión.

El modo batch es el más eficiente en esta configuración, permitiendo un aprendizaje más estable y rápido.

En contraste, **el modo online** es el peor, ya que introduce mucha variabilidad en cada paso de actualización, lo que se refleja en las gráficas separadas y un menor rendimiento general.



Ejercicio 3C

Reconocimiento de Dígitos



Optimización de Hiperparámetros

Al elegir la configuración inicial de una red neuronal, lo que hacemos es establecer un conjunto de hiperparámetros clave que controlan cómo la red aprenderá de los datos.

El proceso de optimización de hiperparámetros es un equilibrio entre **exploración** y **ajuste fino**.

La elección inicial **puede no ser la óptima**, pero proporciona una base sobre la cual se pueden realizar ajustes y evaluaciones para mejorar el rendimiento del modelo.



Conjunto de Datos y Preprocesamiento

Comenzamos con la decisión de utilizar el conjunto completo de datos aplanados para el entrenamiento

Simplificación del proceso: El aplanamiento de los datos (de 7x5 a 35 características) permite una entrada uniforme a la red, simplificando la arquitectura inicial.

Maximización de la información: Al usar todos los datos disponibles, aseguramos que la red tenga acceso a la mayor cantidad posible de ejemplos y variaciones de dígitos.

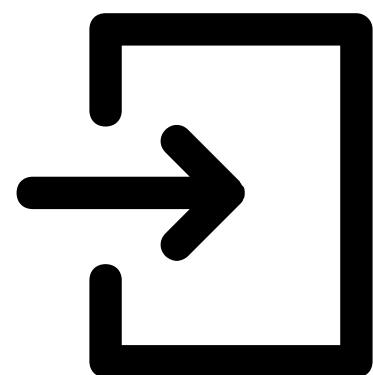


Arquitectura de la Red

Para la arquitectura, optamos por una red de **cuatro capas**:

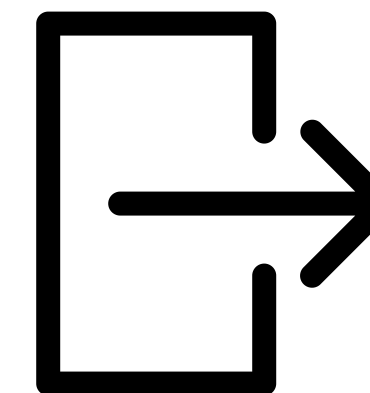
[35, 64, 32, 10]

Función de Activación



ReLU (Rectified Linear Unit)
como función de activación
para las capas ocultas.

Softmax como función
de activación para la capa
de salida.



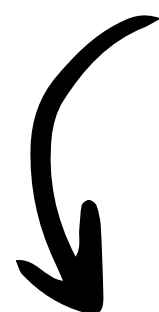
Mitiga el problema del desvanecimiento del gradiente
Eficiencia Computacional
Induce Sparsity

Ideal para clasificación multiclase, ya que normaliza las
salidas a una distribución de probabilidad sobre las
clases.

Inicialización de Pesos

Optamos por la inicialización He, diseñada específicamente para redes con **ReLU**:

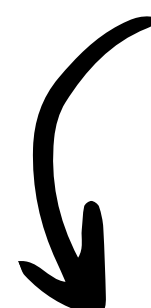
$$\text{Var}(W) = \frac{2}{n_{\text{in}}} \quad W \sim \mathcal{N}(0, \sqrt{\frac{2}{n_{\text{in}}}})$$




Previene la saturación:
Facilita el entrenamiento

Optimizador y Learning Rate

Elegimos el optimizador **Adam** con un learning rate inicial de 0.001



Adam: Combina las ventajas de RMSprop y momentum, adaptando el learning rate para cada parámetro.



Learning rate de **0.001**: Un valor conservador que permite un aprendizaje estable sin riesgo de divergencia.



Tamaño de Batch y Modo de Entrenamiento

Optamos por el modo **mini-batch** con un tamaño de **32**.

Número de Epochs

Establecimos inicialmente **500** épocas.

Evaluación y Generalización

Para evaluar la **robustez** y **capacidad** de **generalización**, decidimos probar el modelo entrenado agregando RUIDO al conjunto de datos de generalización.

- **Evaluación de robustez:** Nos permite medir qué tan bien el modelo maneja entradas degradadas o distorsionadas.
- **Simulación de condiciones reales:** El ruido simula imperfecciones que podrían encontrarse en aplicaciones del mundo real.



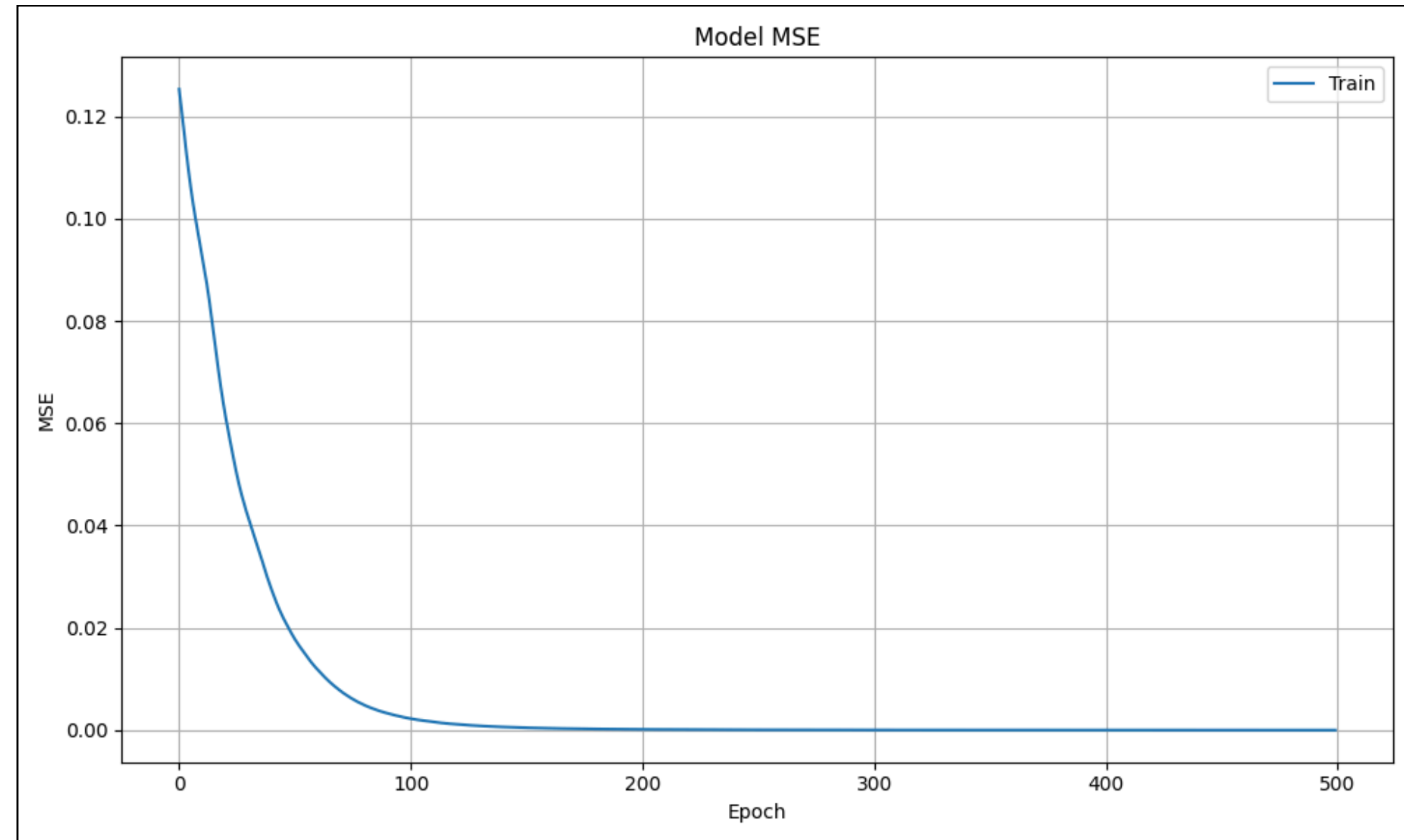
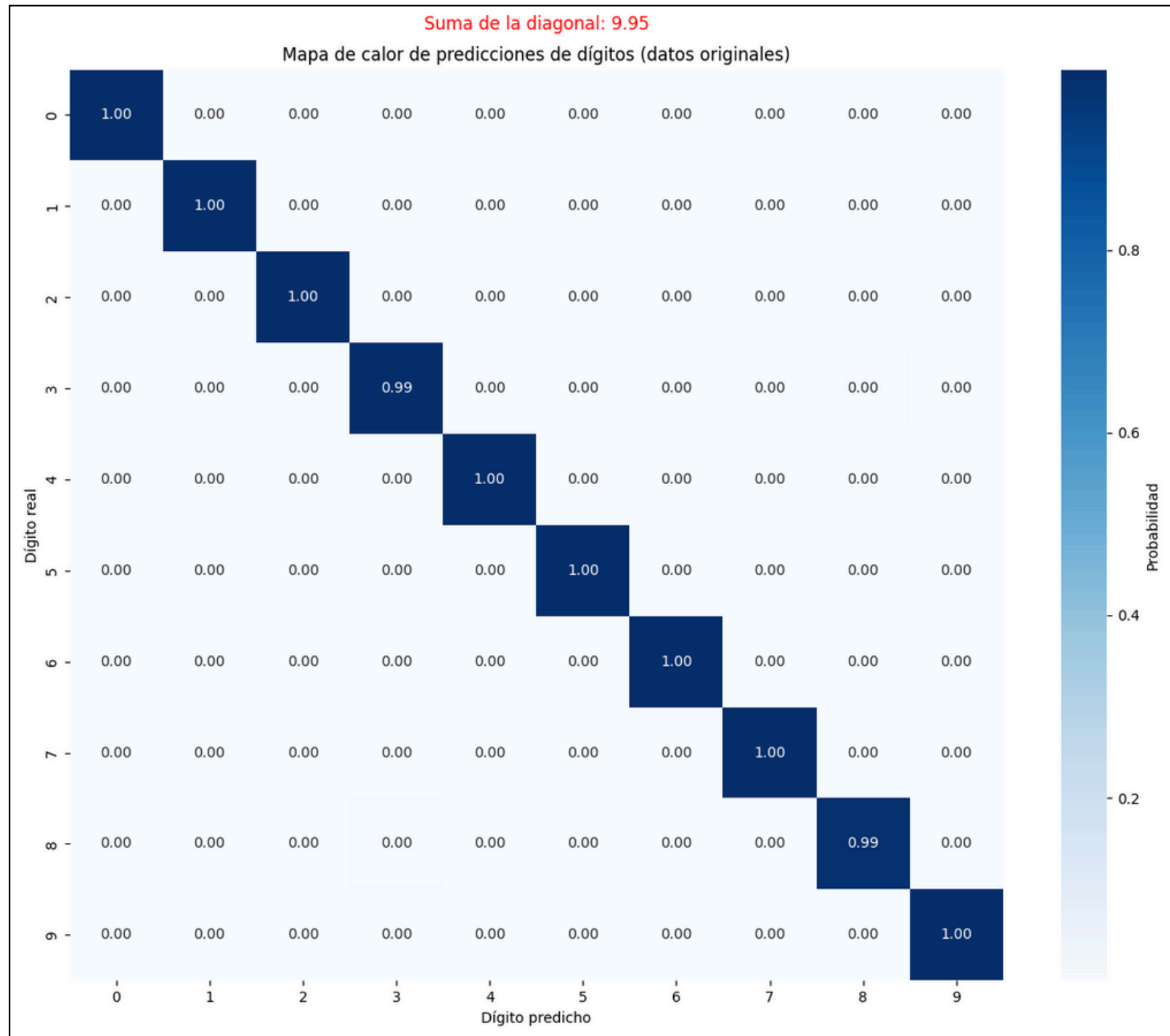
Entrenamiento con TODOS los datos (sin ruido)

En esta primera prueba, se entrena el modelo con todos los **datos originales** (sin agregar ruido ni perturbaciones) y luego se realiza el testing **con los mismos datos**. El objetivo es asegurarnos de que el modelo pueda **aprender todos los ejemplos correctamente**, lo que ayudará a establecer una línea base para las pruebas posteriores.

10 iteraciones, y se toma el PROMEDIO de las mismas.



Ejercicio 3C - Entrenamiento con TODOS los datos (sin ruido)



¿Qué ruidos implementamos?



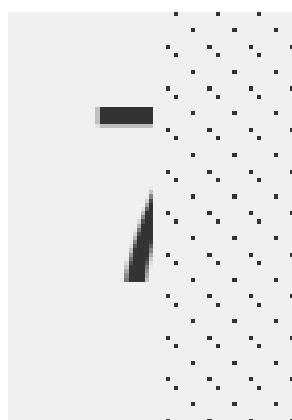
Original



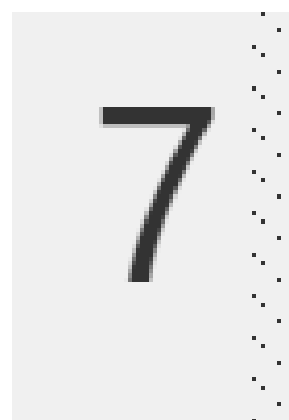
Normal



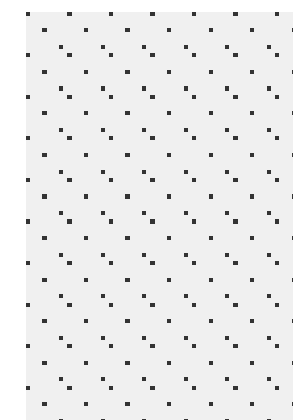
Salt and Pepper



50 Percent



20 Percent



100 Percent

¿Qué métricas implementamos?



Accuracy

Proporción de predicciones correctas



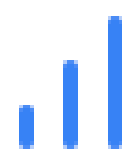
Precision

Exactitud de las predicciones positivas



Recall

Capacidad de identificar casos positivos



F1-Score

Balance entre precisión y recall



MSE

Magnitud promedio de errores de predicción

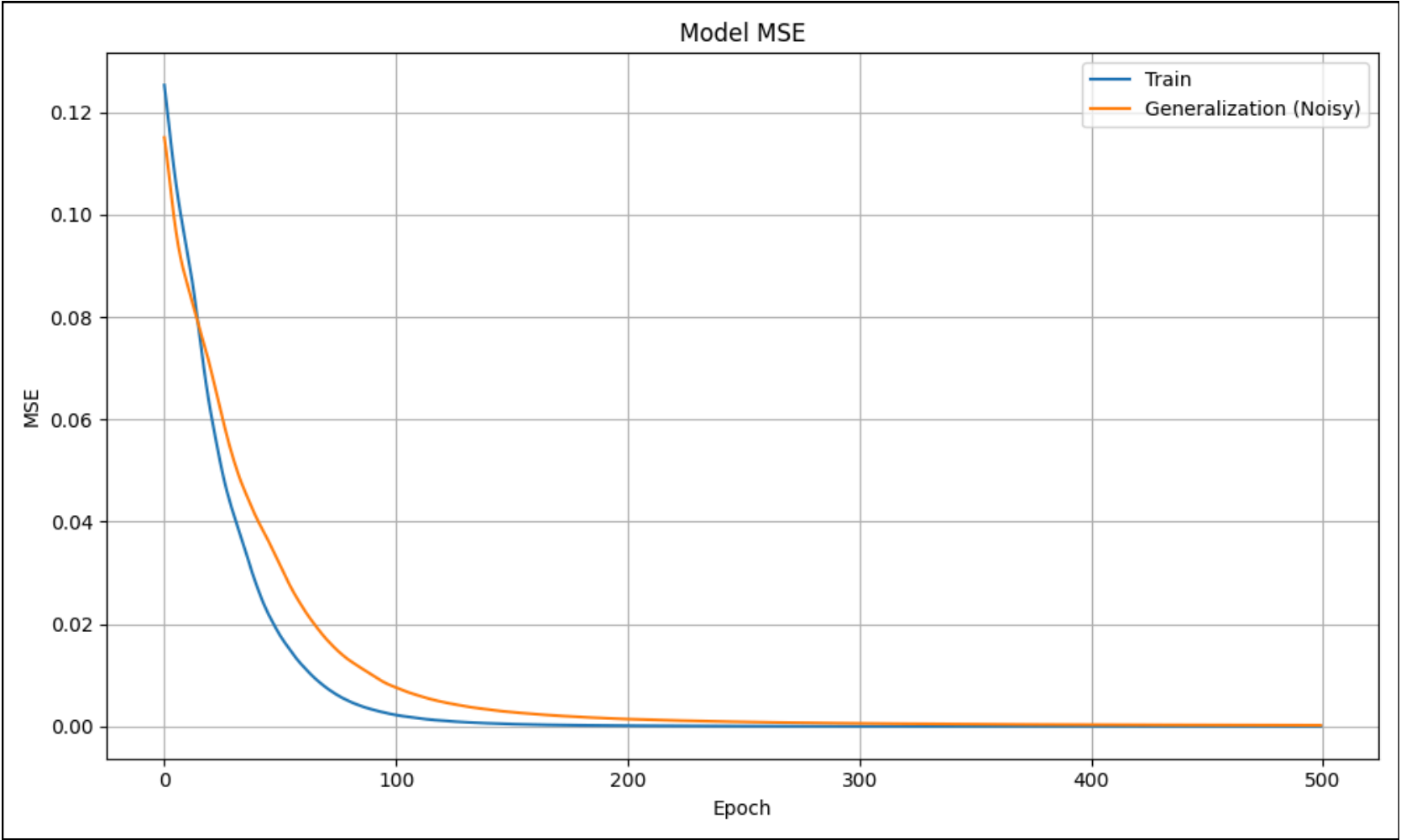
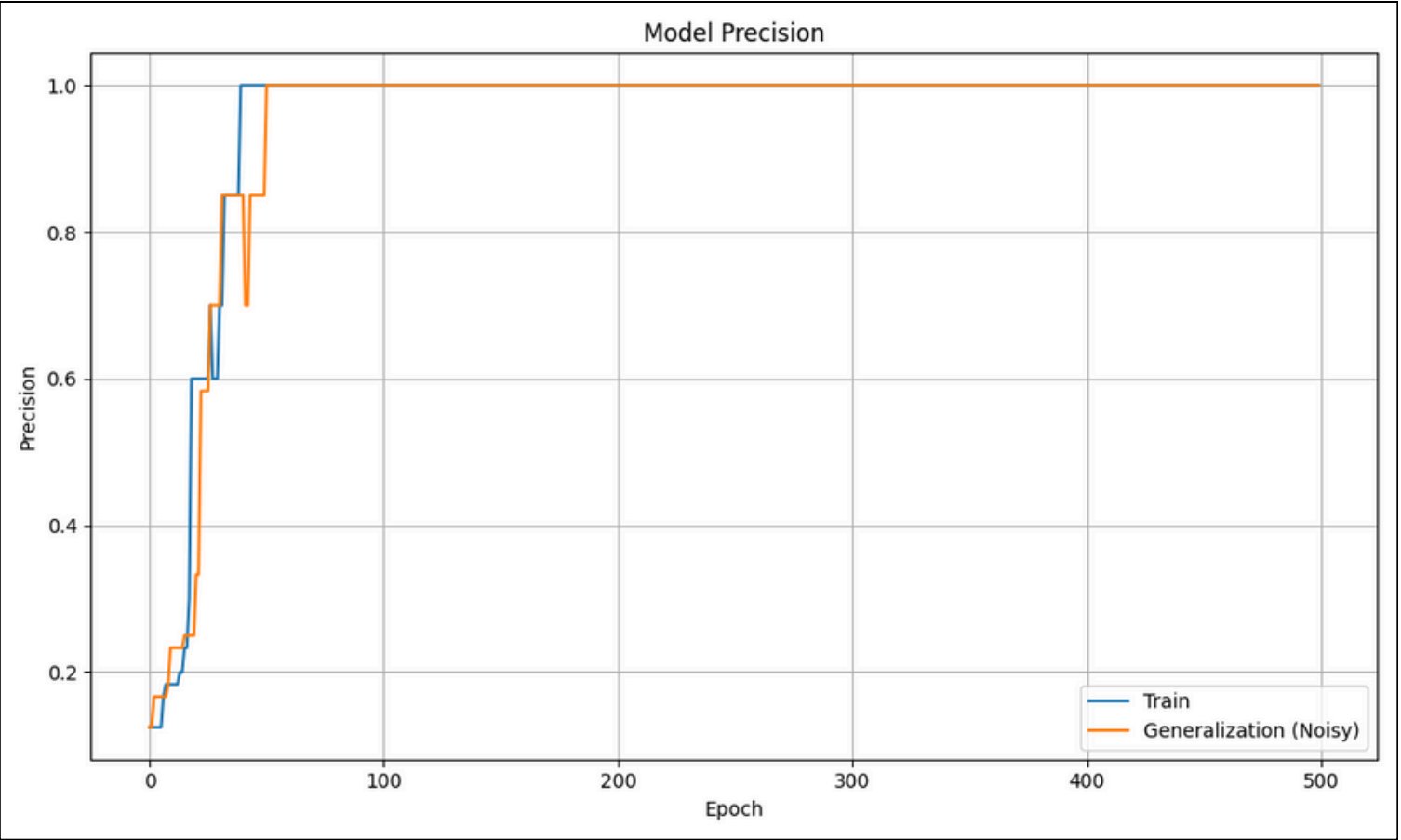
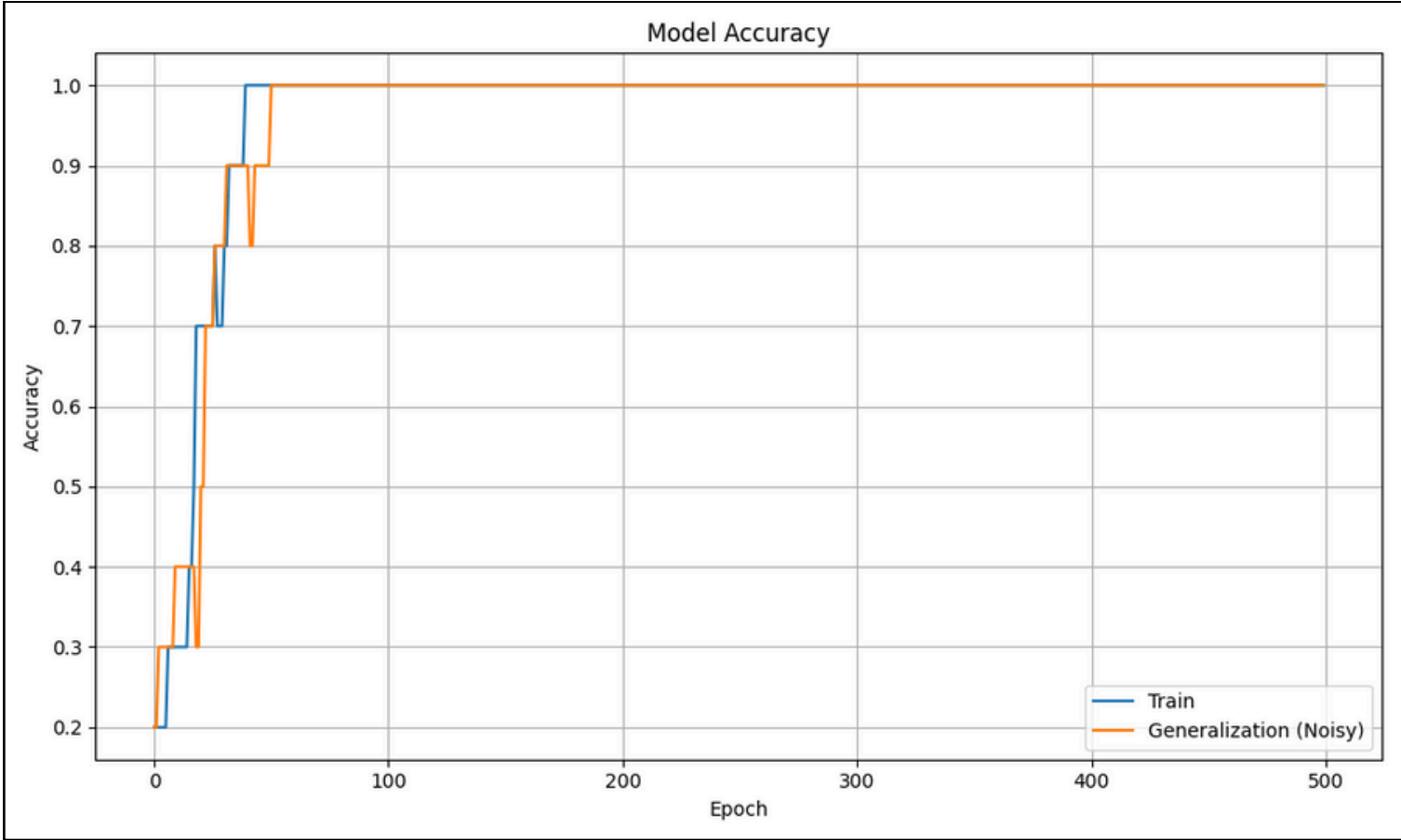
Pruebas de generalización con datos ruidosos.

Usamos el conjunto noisy para validar si el modelo puede generalizar en lugar de memorizar. La idea es ver cómo el modelo maneja pequeñas perturbaciones en los datos. Es importante ver cómo el ruido afecta la capacidad de generalización del modelo.

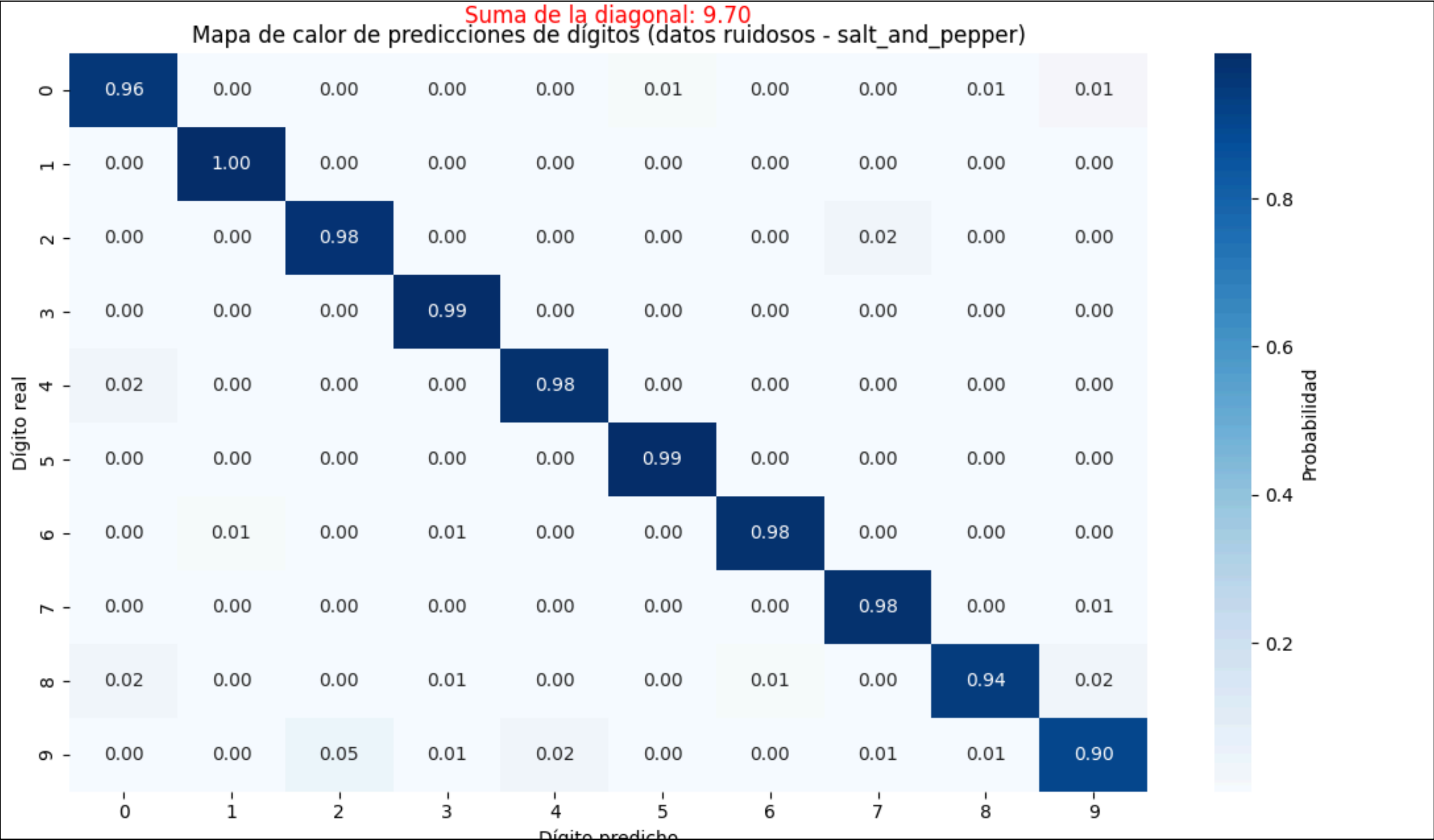
- a) Salt and pepper
- b) 50% de ruido
- d) 100% de ruido

10 iteraciones y se promedia

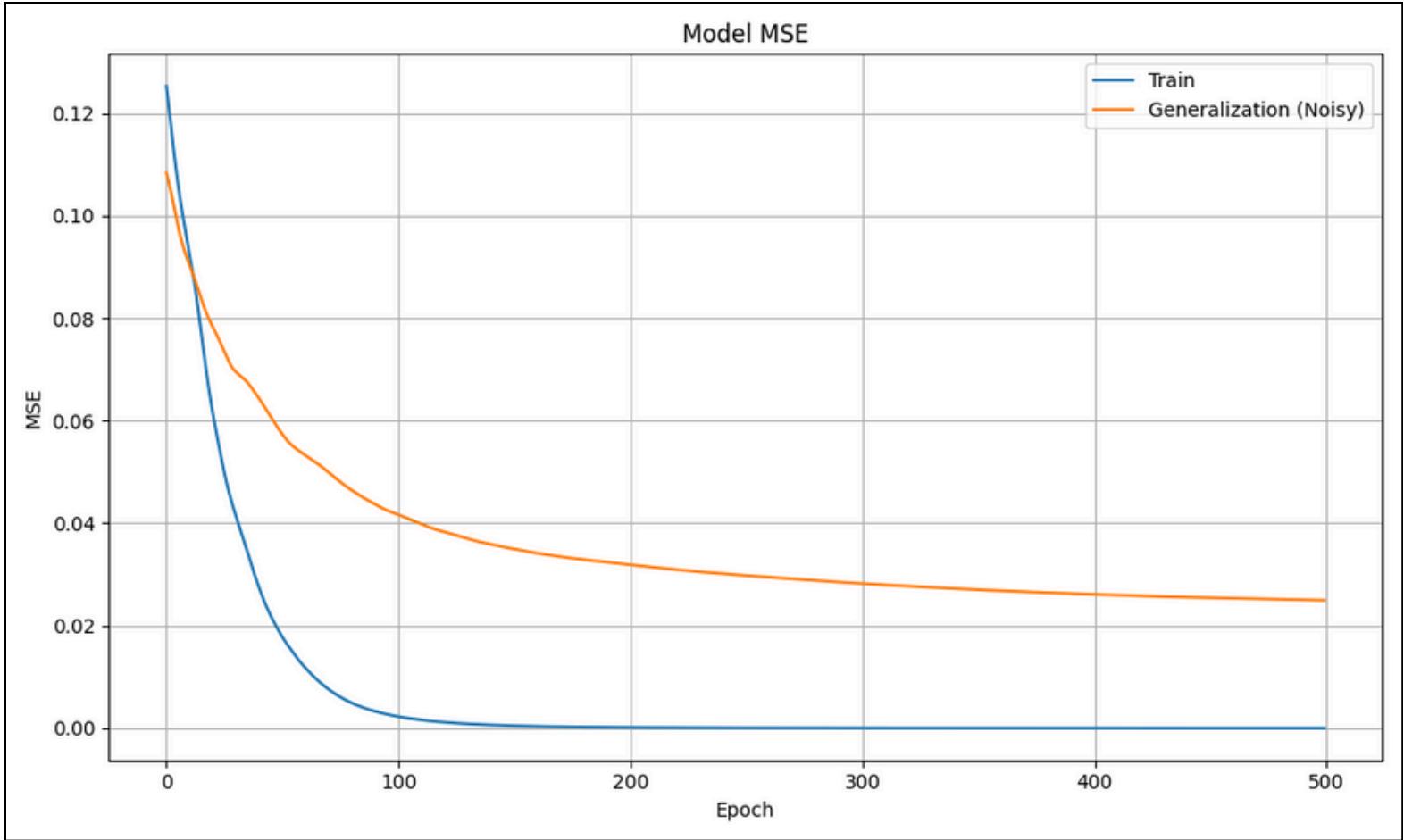
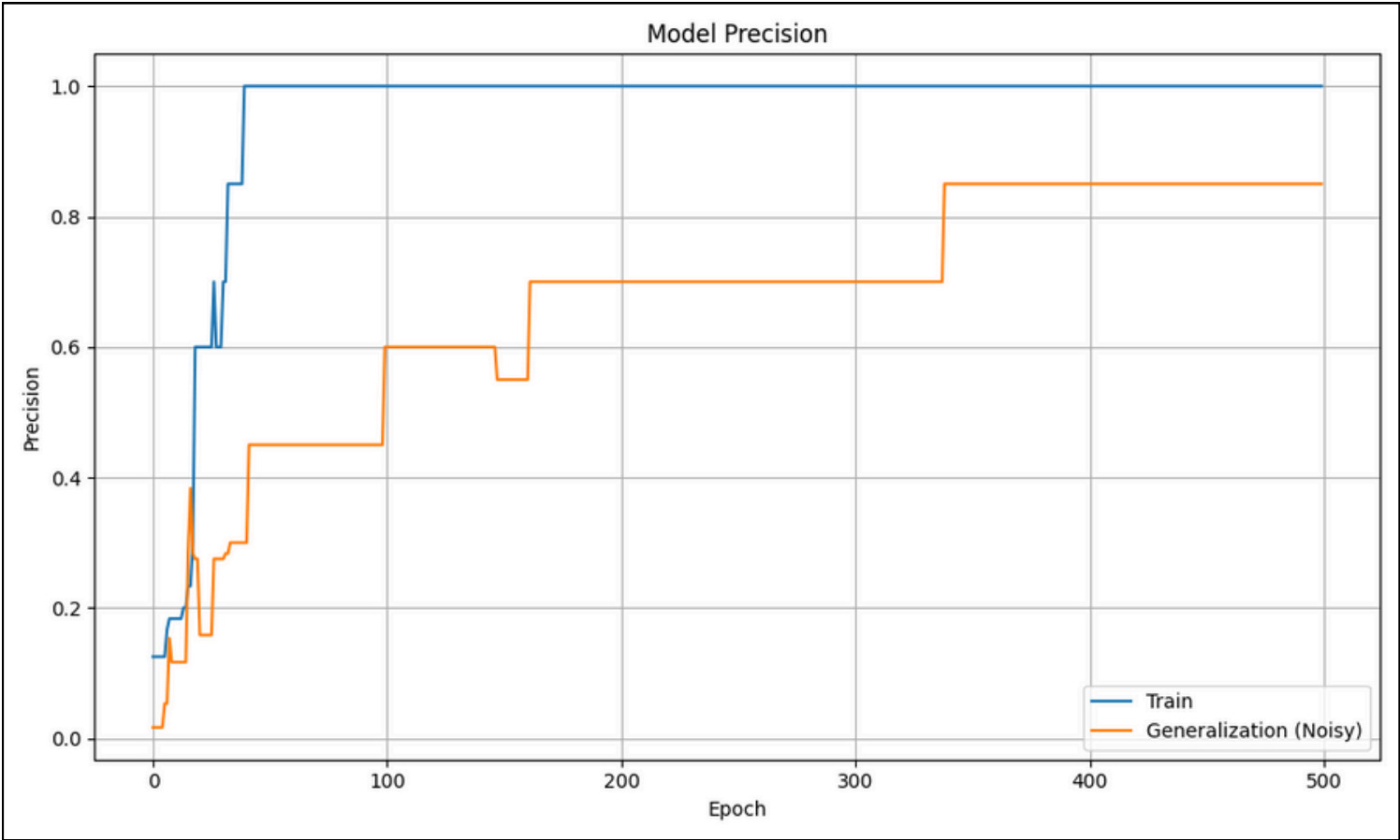
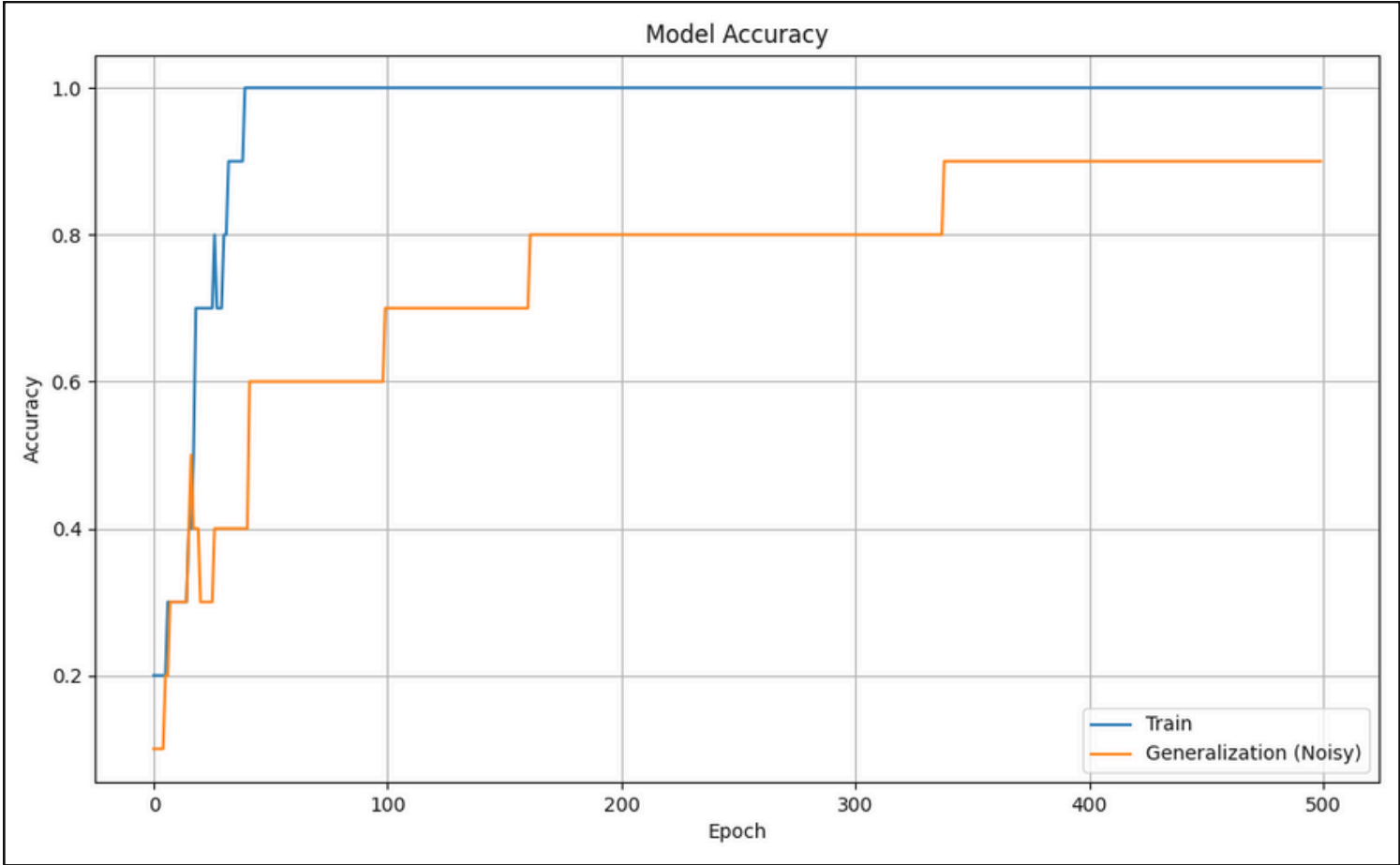
a)Salt and Pepper - Ruído



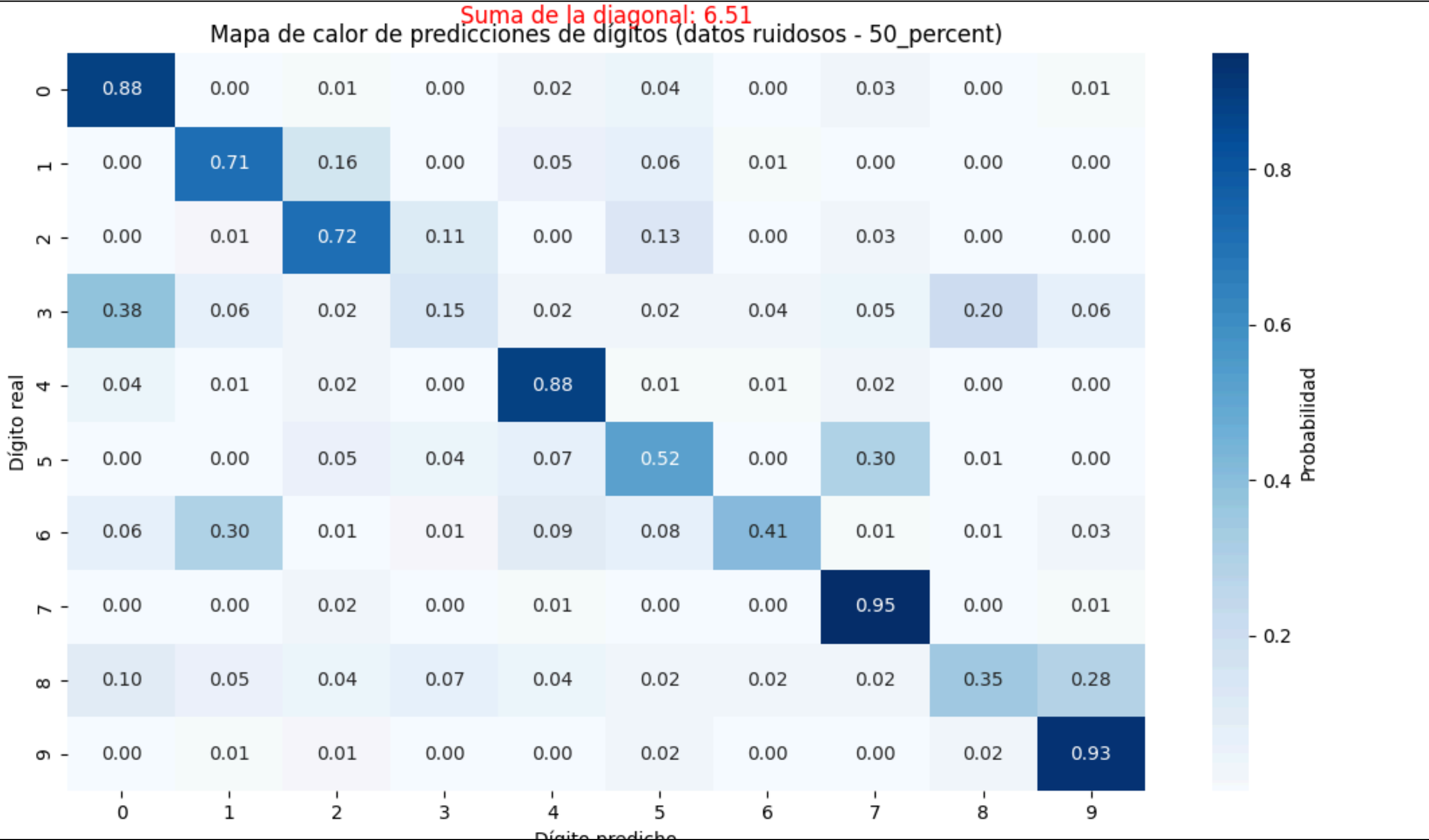
a) Salt and Pepper - Ruido



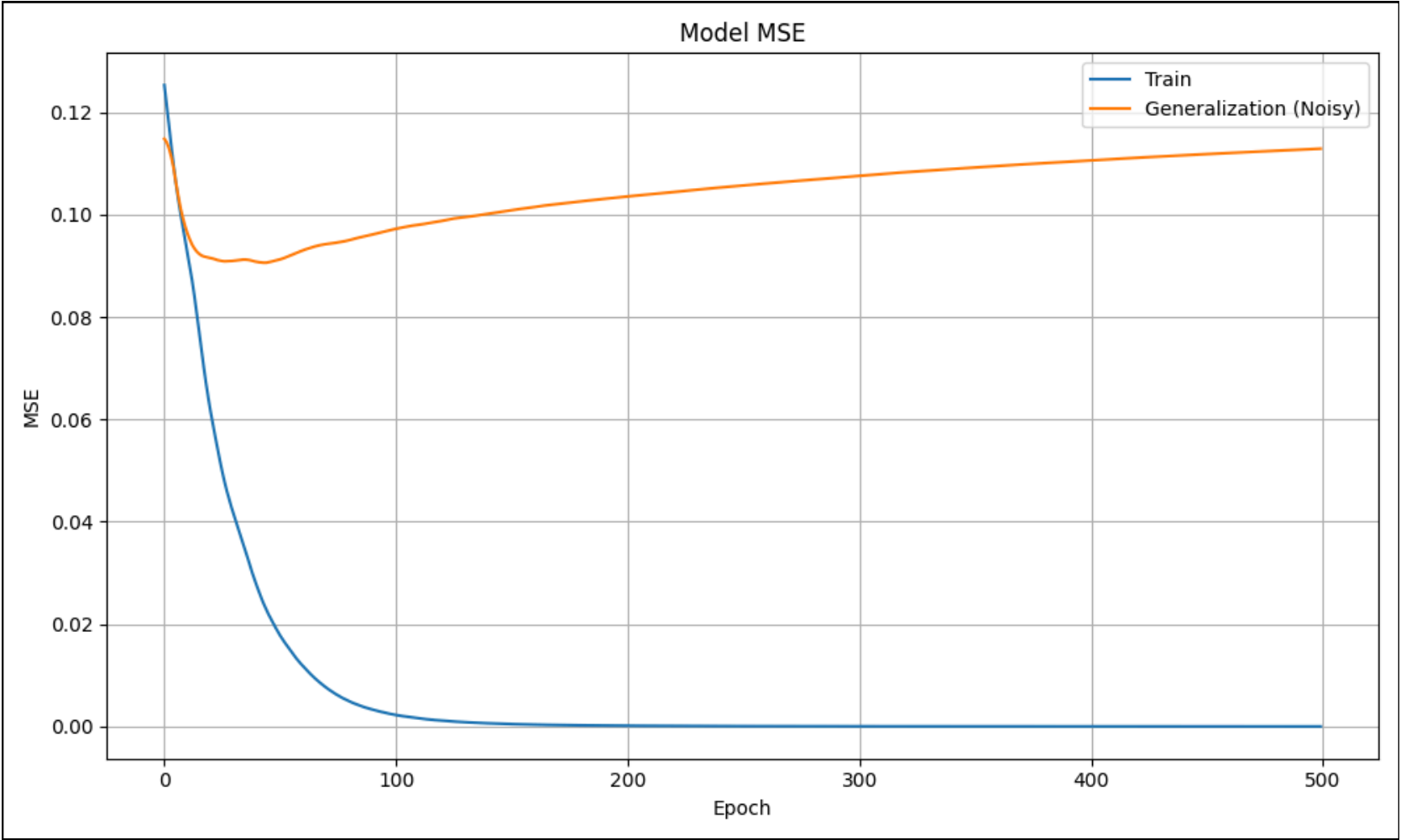
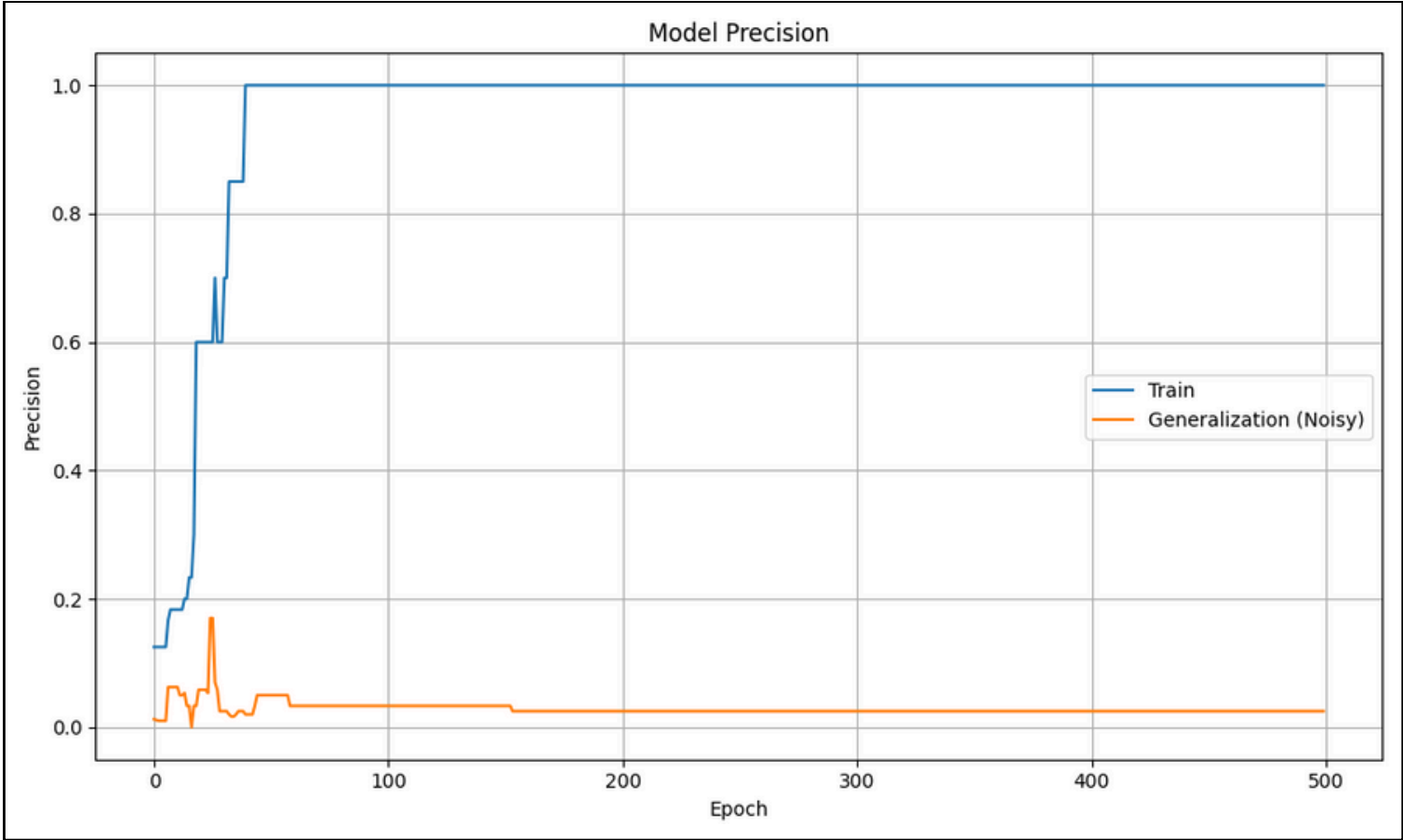
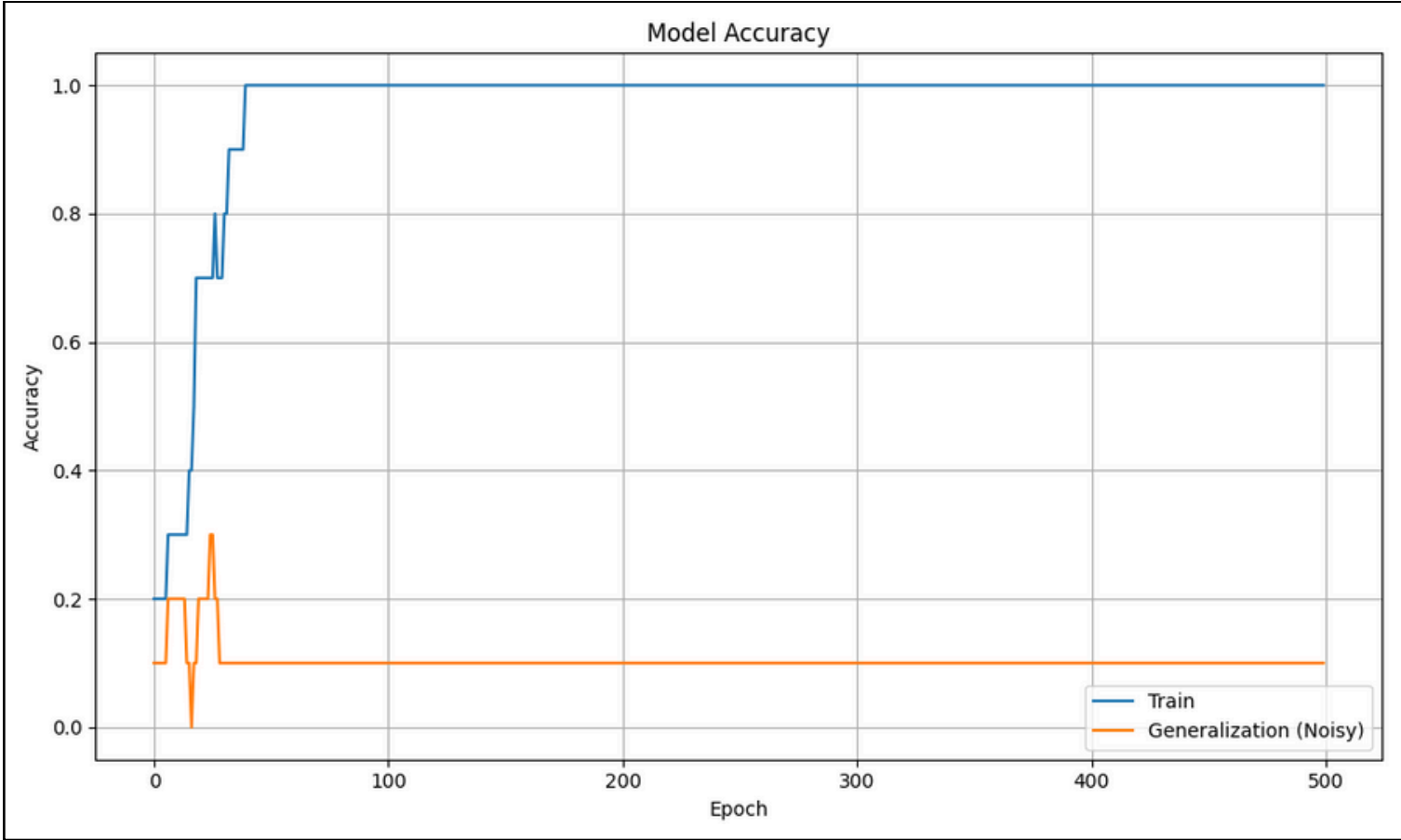
a) 50% - Ruido



a) 50% - Ruido

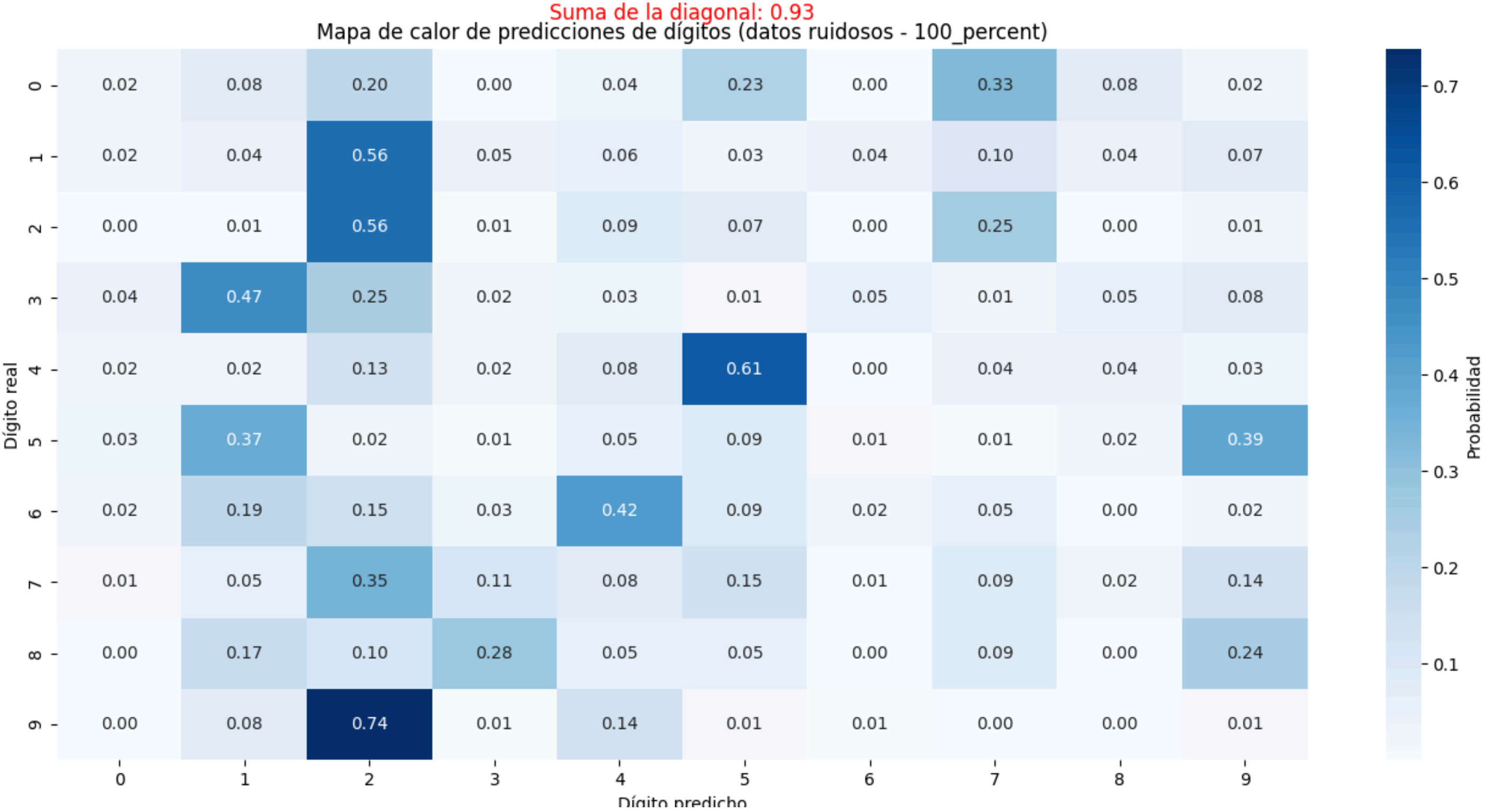


a) 100% - Ruido



Sobreajuste

a) 100% - Ruido



¿Qué observamos?

- El perceptrón mostró cierto nivel de generalización cuando el **ruido era bajo**, pero **perdió capacidad de predicción** a medida que el ruido se incrementaba.
- La robustez disminuye con mayor nivel de ruido, lo que señala la importancia de mejorar la capacidad de generalización del modelo.
- Confirmamos que uno de los factores del sobreajuste es tener un conjunto de datos de entrenamiento con mucho ruido.

¿Preguntas?

¡Gracias por la atención!