Sistemas de Inteligencia Artificial - TP2

Grupo Hamilton

Barmasch, Juan Martín (61033),
Bellver, Ezequiel (61268),
Castagnino, Salvador (60590),
Lo Coco, Santiago (61301),
Negro, Juan Manuel (61225).

Generalidades

Estructura

Se utilizó un perceptrón general que permitía modificar su **función de activación**, **optimizaciones**, **hiperparámetros**, **métodos** de entrenamiento, y sus **capas**.

Esto fue posible por la parametrización del código y el uso de productos matriciales.

Optimizaciones

- *Gradient Descent*
- *Momentum*
- *Root Mean Square Propagation*
- *Adaptive η*
- *Adam*
- *Adadelta*
- *Adamax*
- *Nadam*
- *Amsgrad*

Entrenamiento

- *Batch*
- *Online*

Regularización

- *Dropout*

Funciones de activación

- Identidad
- *Sign*
- *Logistic*
- Tangente Hiperbólica

Dataset Partitioning

- *Holdout*
- *K Fold Cross Validation*

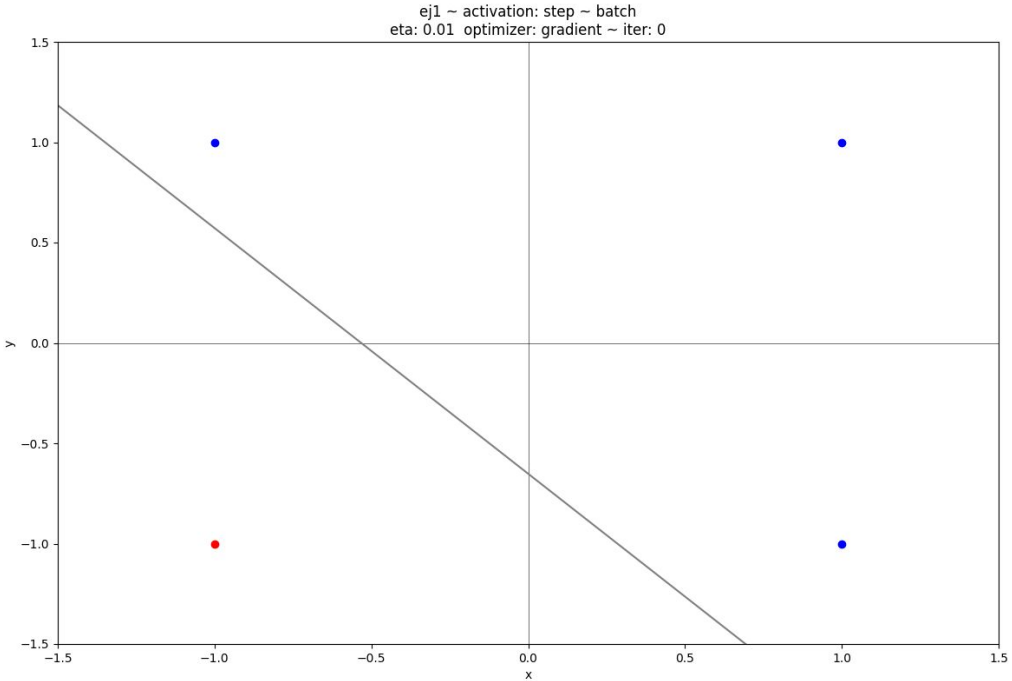
Ejercicio 1

Prueba

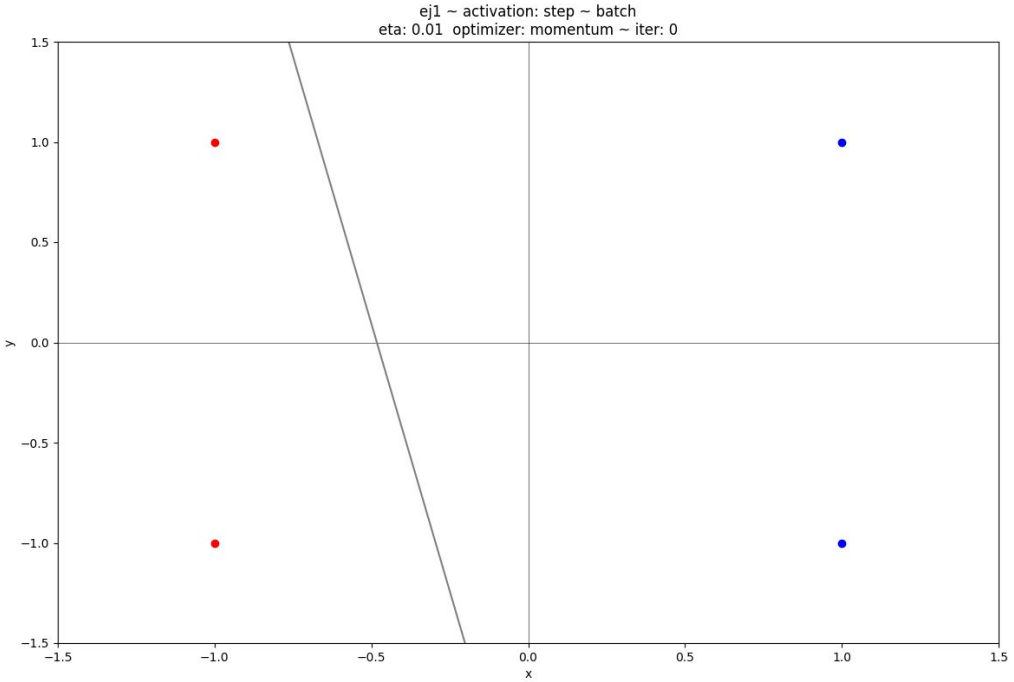
Siendo una estructura de **perceptrón simple** con **una sola neurona**, no se utilizó *dropout*. Además, como el *dataset* son 4 datos, tampoco usamos *dataset partitioning*.

Por la consigna, se utilizó la función de activación **escalón** y se probaron distintos métodos de entrenamiento y optimizadores.

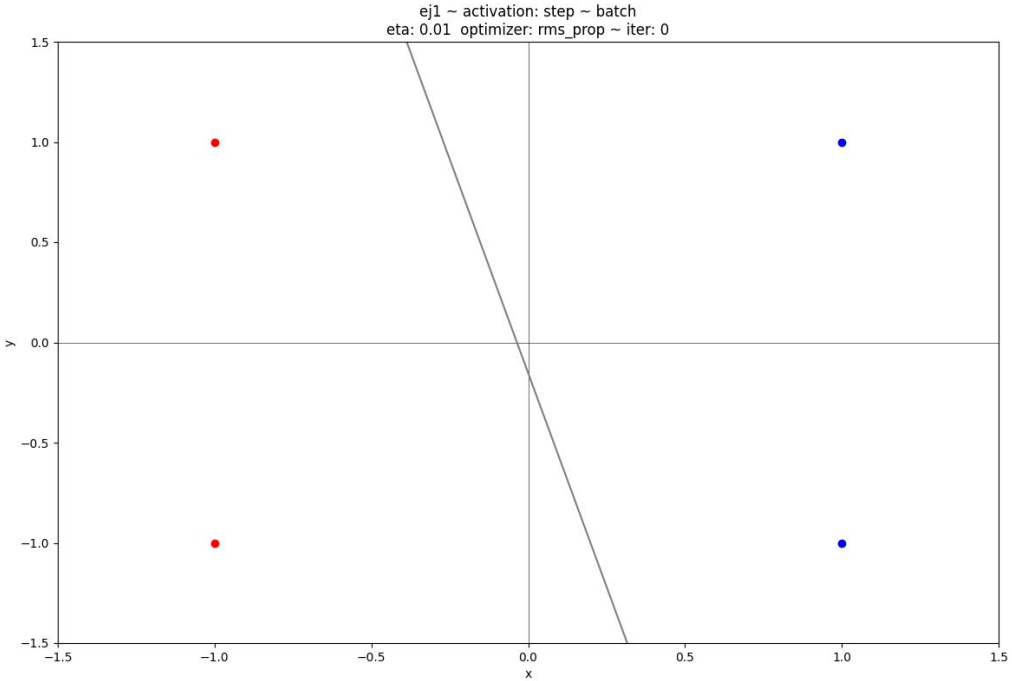
Resultados - *AND - Batch - Gradient*



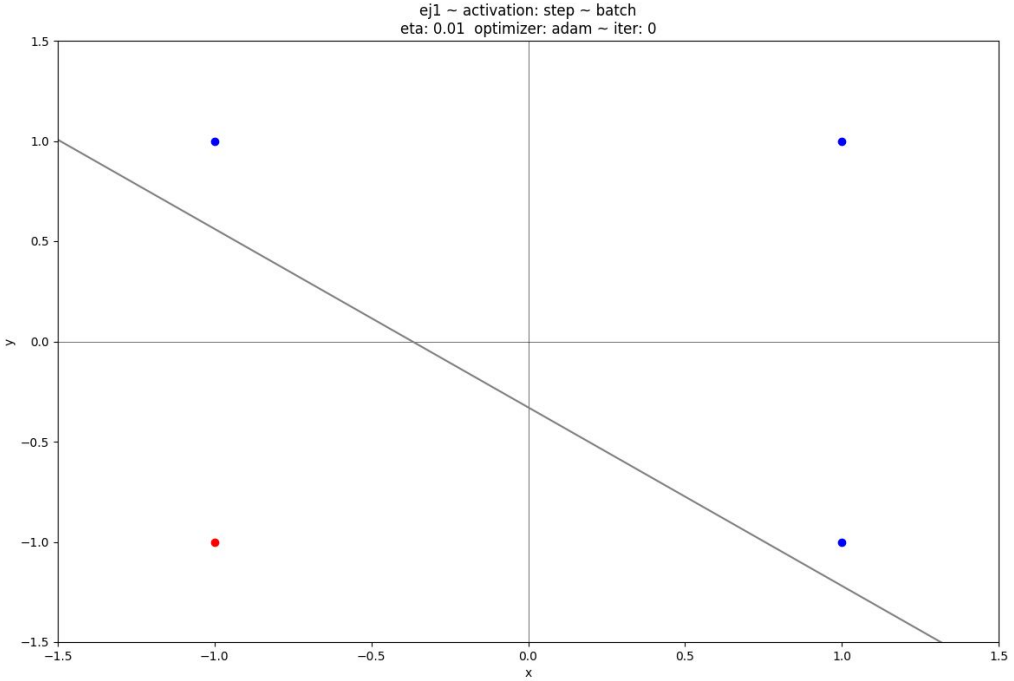
Resultados - *AND - Batch - Momentum*



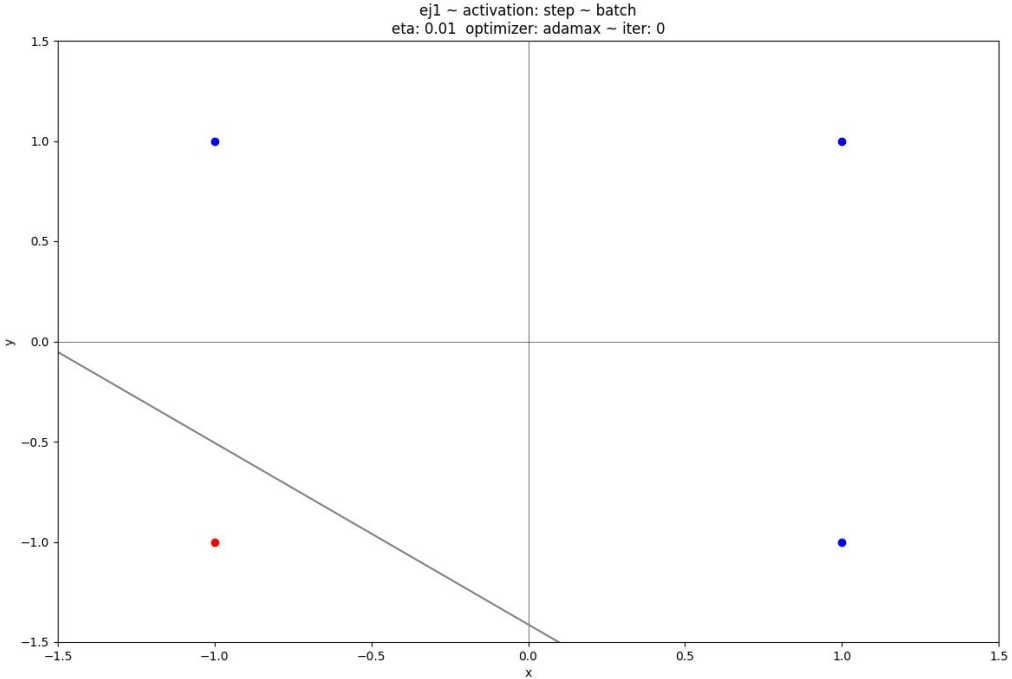
Resultados - *AND - Batch - RMS Prop*



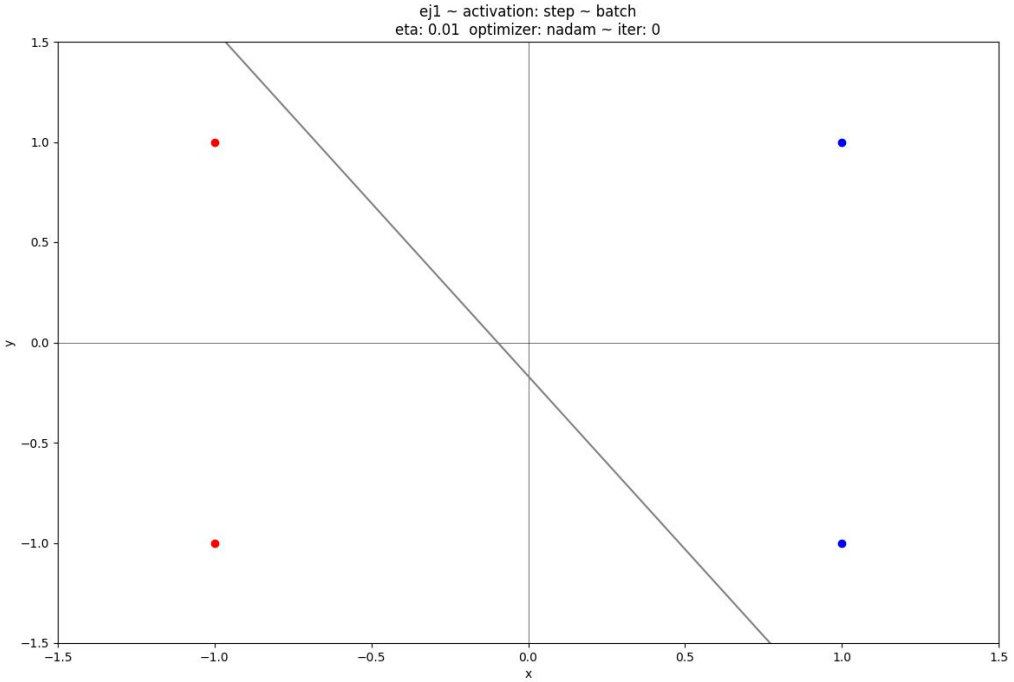
Resultados - *AND - Batch - Adam*



Resultados - *AND - Batch - Adamax*

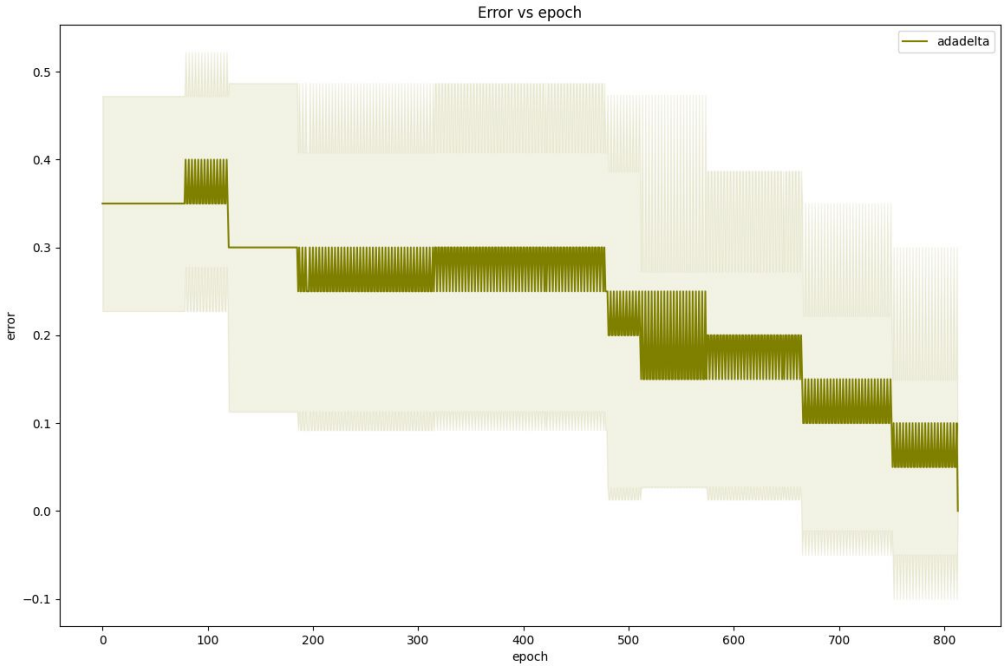


Resultados - *AND - Batch - NAdam*

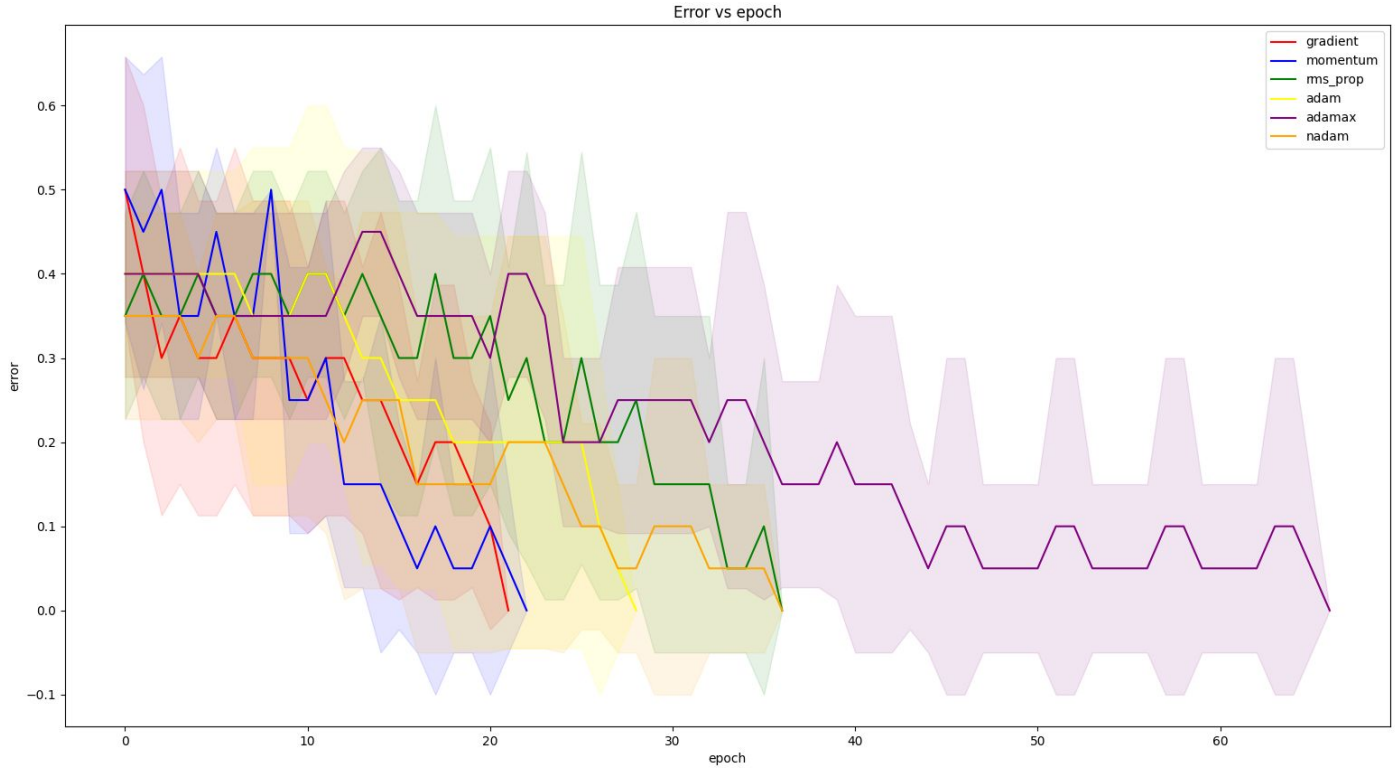


Resultados - *AND - Batch - Adadelta*

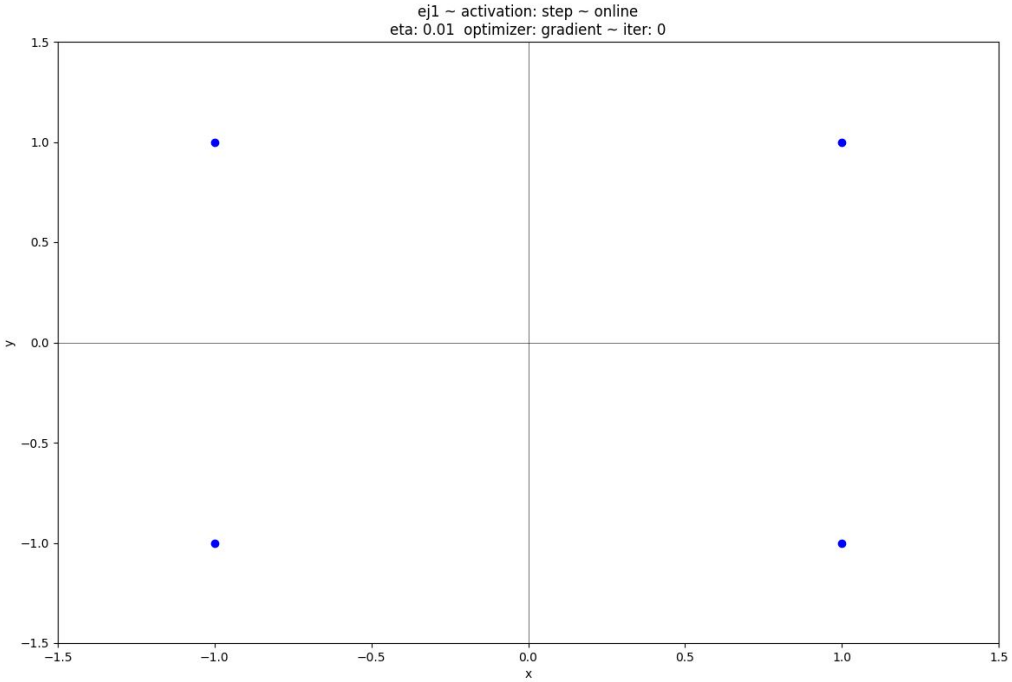
Resultados - *AND - Batch - Adadelta*



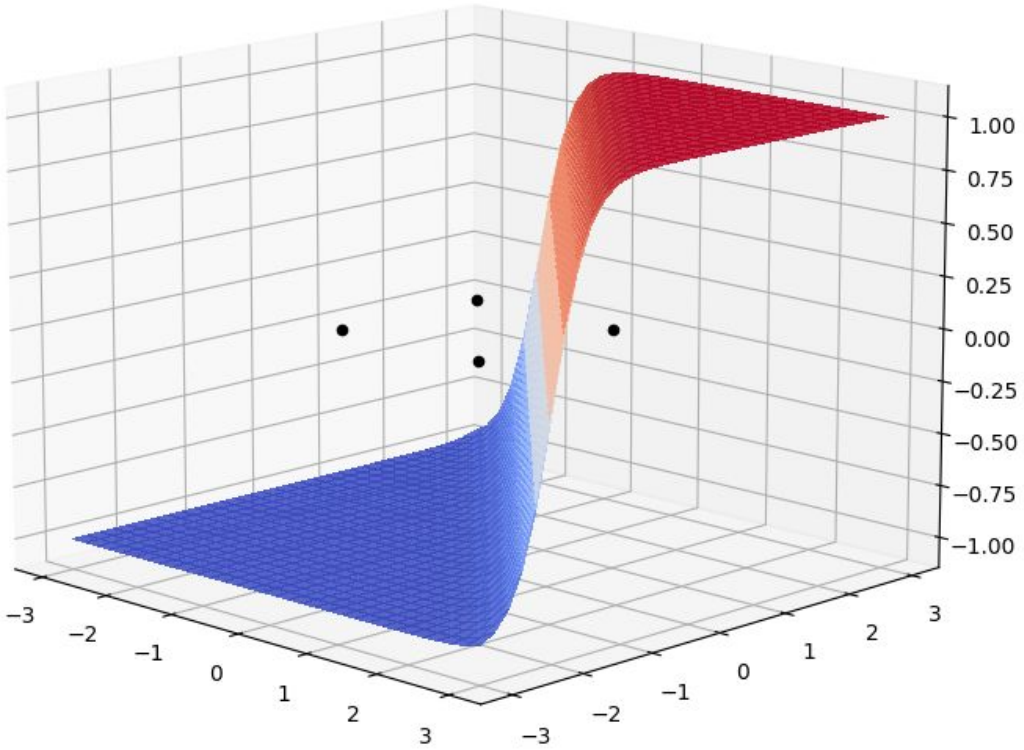
Resultados - *AND - Batch - Escalón*



Resultados - *AND - Online - Gradient - Escalón*



Resultados - *AND - Batch - Gradient - tanh*

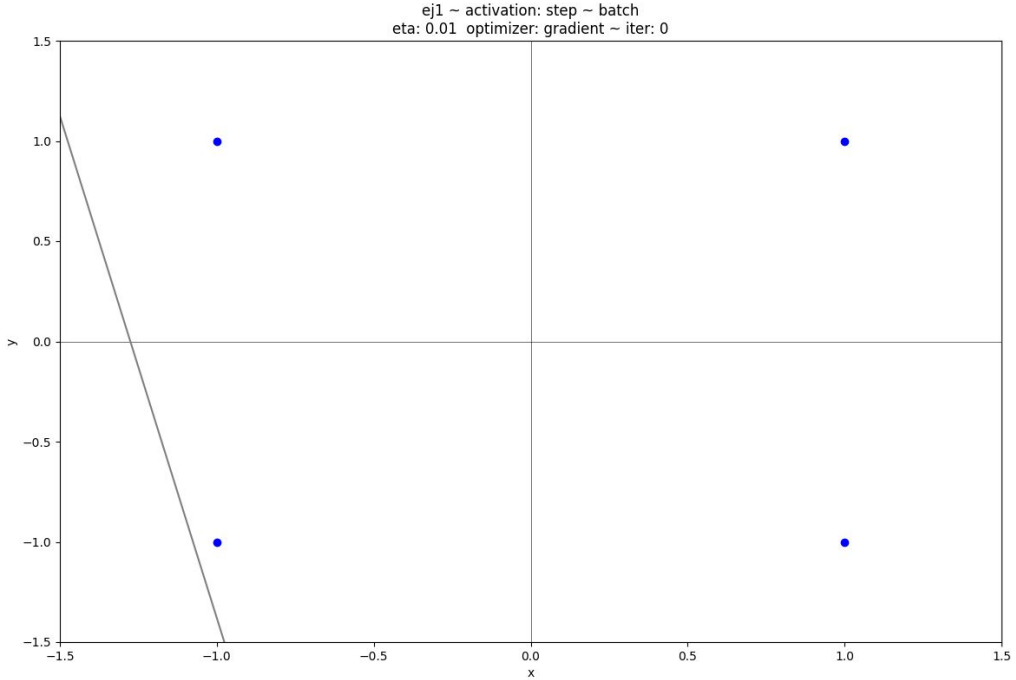


Análisis - *AND*

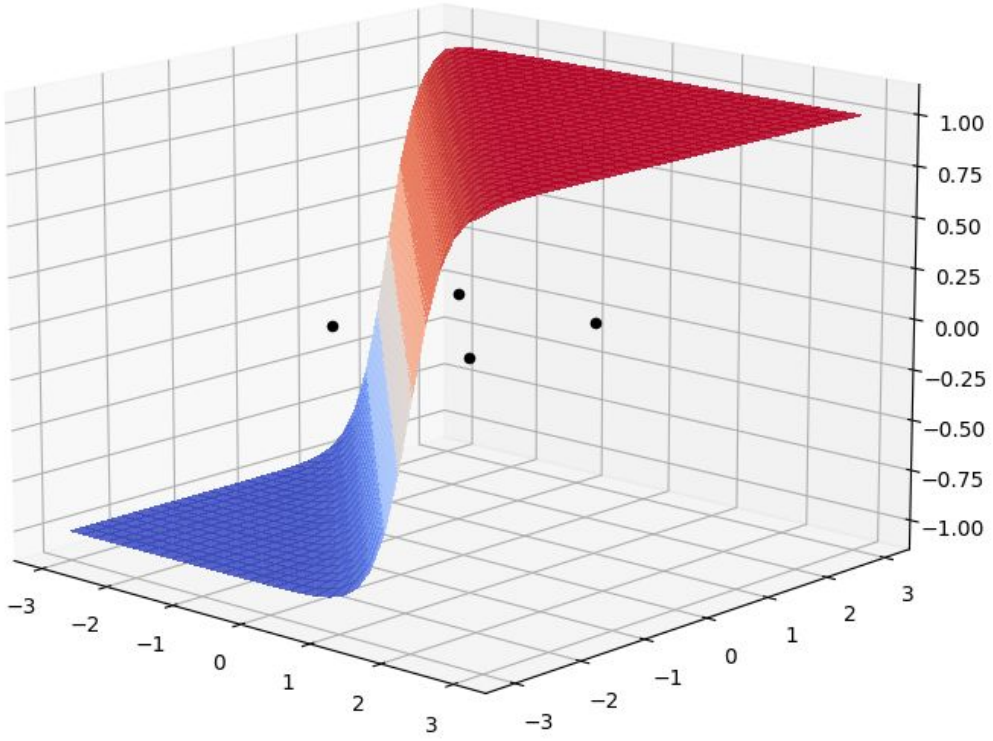
- El problema es **linealmente separable**, por lo que encontramos soluciones con todos los optimizadores implementados.
- Por la simplicidad de la estructura, optimizaciones “simples” son más efectivas, como *Momentum* o *RMS Prop*, mientras que optimizaciones más complejas, como las basadas en *Adam*, son más lentas en encontrar la solución.
- *Online* resulta muy similar a *Batch* en cantidad de épocas. Pero en el caso de Online, los pesos se actualizan más veces

Resultados - *XOR - Batch - Gradient*

El problema **no es linealmente separable**, por lo que el perceptrón simple con función de activación escalón no podrá encontrar una solución con error menor a 0,25.



Resultados - *XOR - Batch - Gradient*



Ejercicio 2

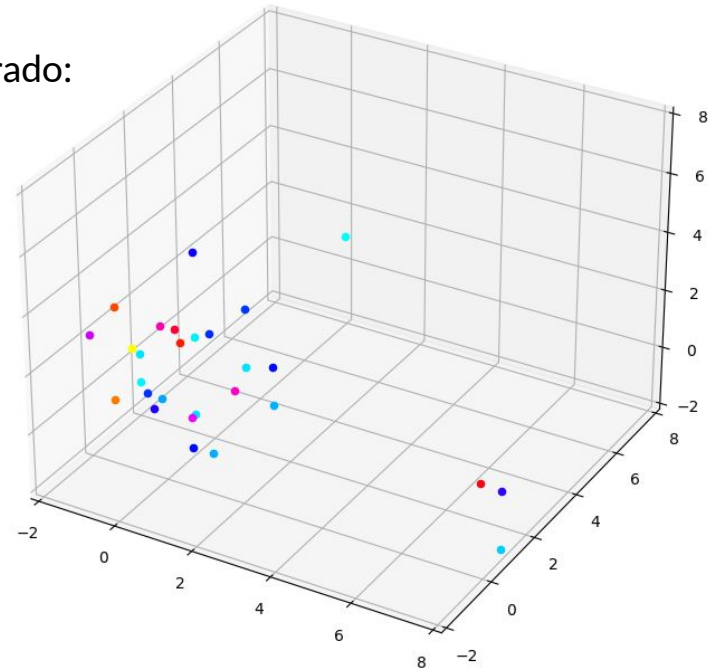
Prueba

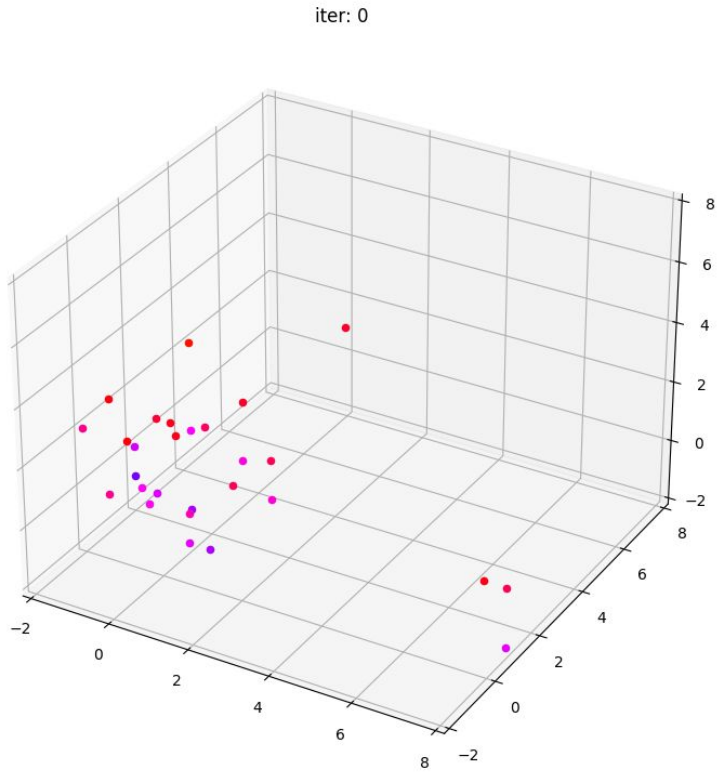
Siendo una estructura de **perceptrón simple** con **una sola neurona (según la consigna)**, no se probó con *dropout*. Aunque de todas formas se hizo la prueba con X capas ocultas, en cuyo caso el dropout mostró peores resultados que los obtenidos sin su uso.

Sin embargo, se probó con todos los optimizadores, métodos de entrenamiento, y particionamiento de datos implementados.

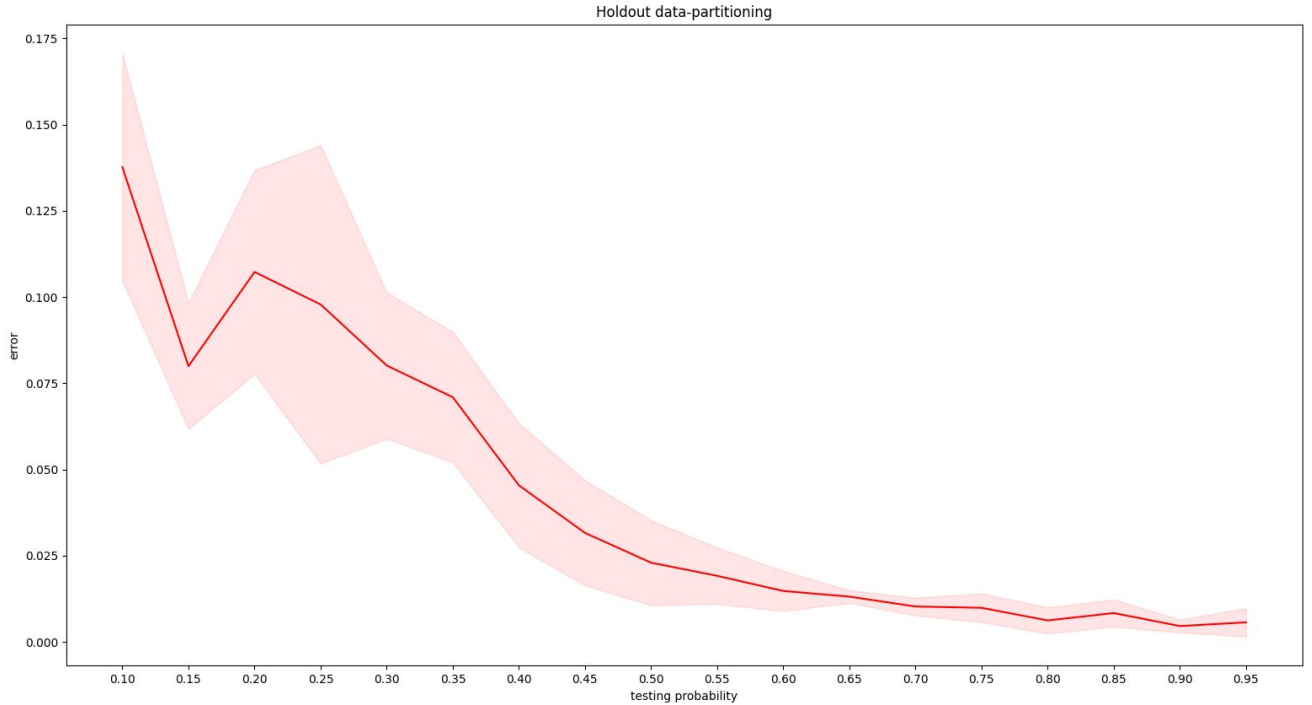
Resultados - *Batch*, *Adam*, *tanh*, $\eta = 10^{-4}$

Esperado:

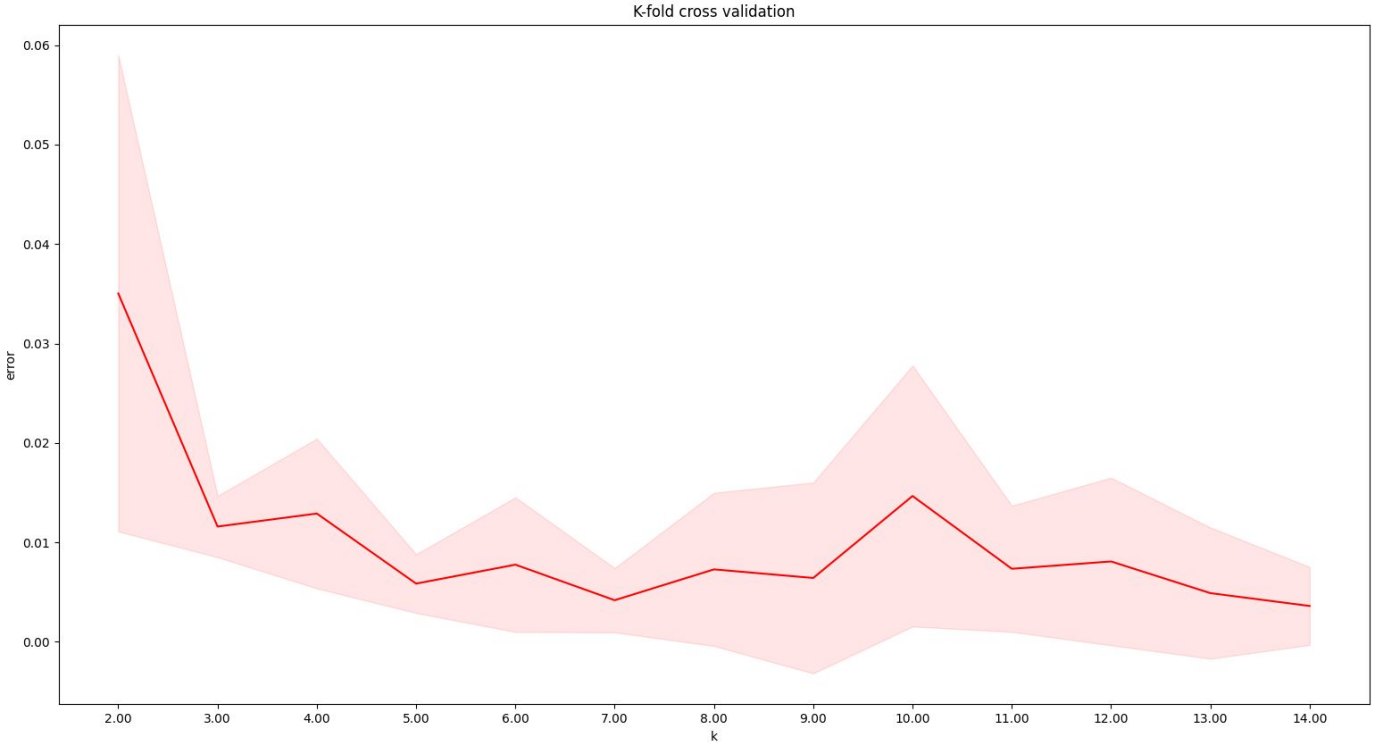




Holdout - $\eta = 10^{-4}$, $\beta = 1$, batch, Adam, tanh



K-Fold


K-fold cross validation
error
0.06
0.05
0.04
0.03
0.02
0.01
0.00
2.00
3.00
4.00
5.00
6.00
7.00
8.00
9.00
10.00
11.00
12.00
13.00
14.00
k

Análisis

Los valores de **holdout** contienen al principio un alto nivel de variación y recién a partir de un 60% de los datos se estabiliza la curva de los errores. El overfitting no pareciera presentarse cuando el tamaño de training set es mayor que 50%.

Los valores obtenidos por medio de **k-fold** contienen poca variación y se estabilizan rápidamente. Para valores mayores que k = 5 la curva no se aleja mucho del error obtenido con este parámetro, mostrando así que es el punto óptimo entre cómputo y precisión.

El método **k-fold** es conveniente en este caso ya que el dataset es chico. Este método aun así contiene un trade off que es que es demandante en cuanto a cómputo requerido.

Partitioning arbitrario

Se probó utilizando datos que tuvieran los resultados distribuidos uniformemente entre el mínimo y el máximo. De esta forma se logró separar 9 datos que cumplían la propiedad de que la diferencia entre sus resultados contiguos es menor a 16.

Se realizó esta discriminación buscando reducir la cantidad de datos con resultado parecido que se utilizaban en el entrenamiento de la red neuronal.

Tabla

Con los mismos parámetros que las corrida anteriores se obtuvieron:

- Media del error de predicción: 0.0218306
- Desviación estándar: 0.0023388

Podemos considerar esta división satisfactoria. Se puede observar en el gráfico de **holdout** como al usar 35% del *dataset* aleatoriamente, se obtiene una media de error de 0,7 y una desviación estándar de 0,2 (aproximadamente)
Pero usando 9 de 28 datos (32%) logramos resultados mucho mejores.

x1	x2	x3	y	Δy
0	7,9	1	0,32	—
-0,5	0,6	0	7,871	7,551
0,4	2,7	2	17,654	9,783
7,9	1	0	26.503	8,849
-2	0	2	40,131	13,628
-0,5	0.6	2,5	51,000	10,869
0	0,4	2,7	61,301	10,301
-1,3	0	3,23	72,512	11,211
0	-1,3	3,23	88,184	15,672

Ejercicio 3a
XOR

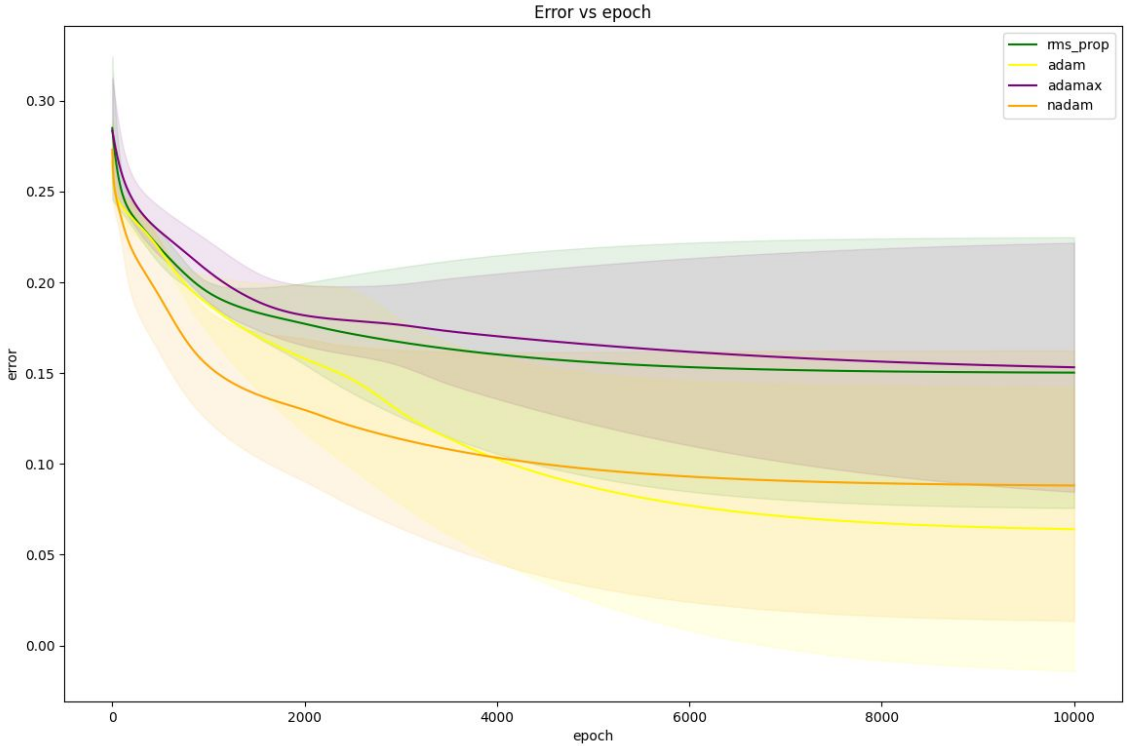
Prueba

Se utilizó *Batch* como entrenamiento, y se probaron muchos hiperparámetros y optimizadores.

Además, se varió la cantidad de neuronas por capa oculta y se mostraron los resultados.

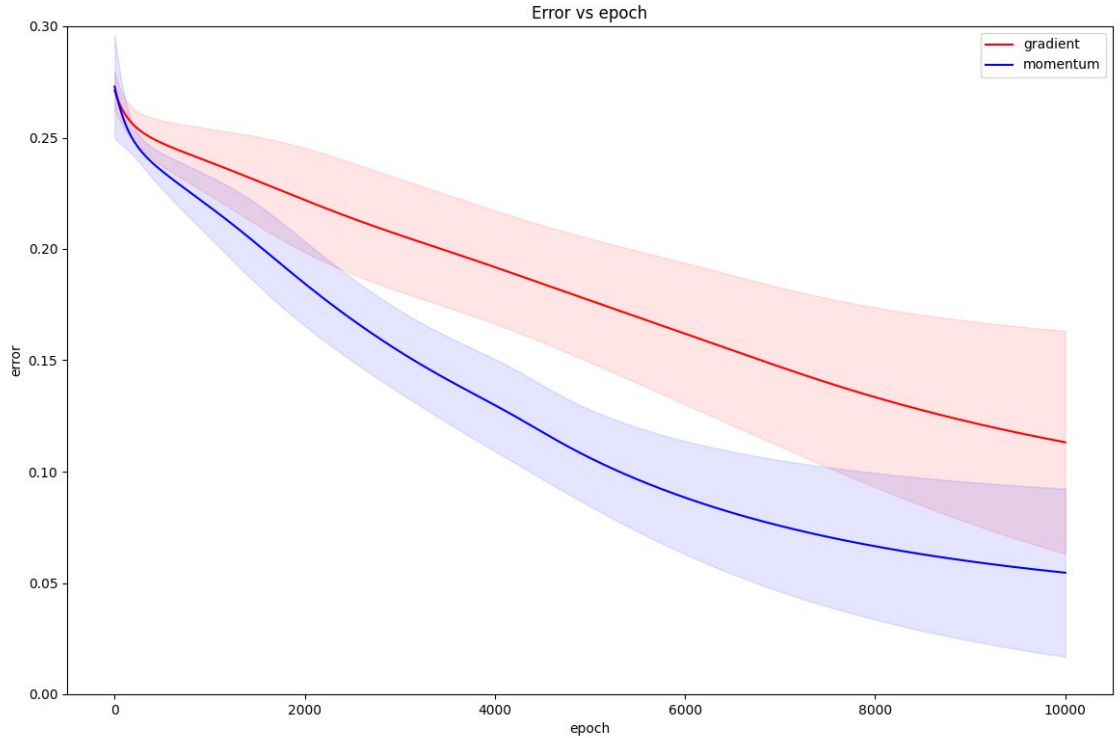
Resultados - *Batch*, *tanh*, $\eta = 10^{-4}$, $\beta = 1$

[2,1]



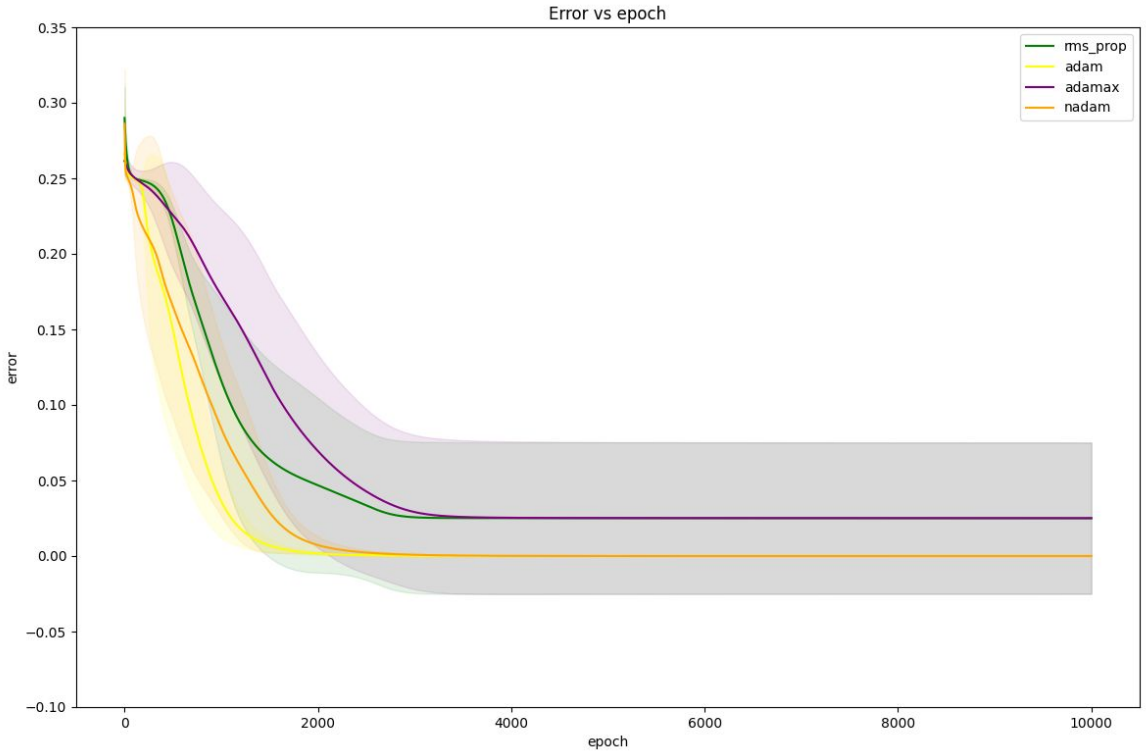
Resultados - *Batch*, *tanh*, $\eta = 10^{-4}$, $\beta = 1$

[2,1]



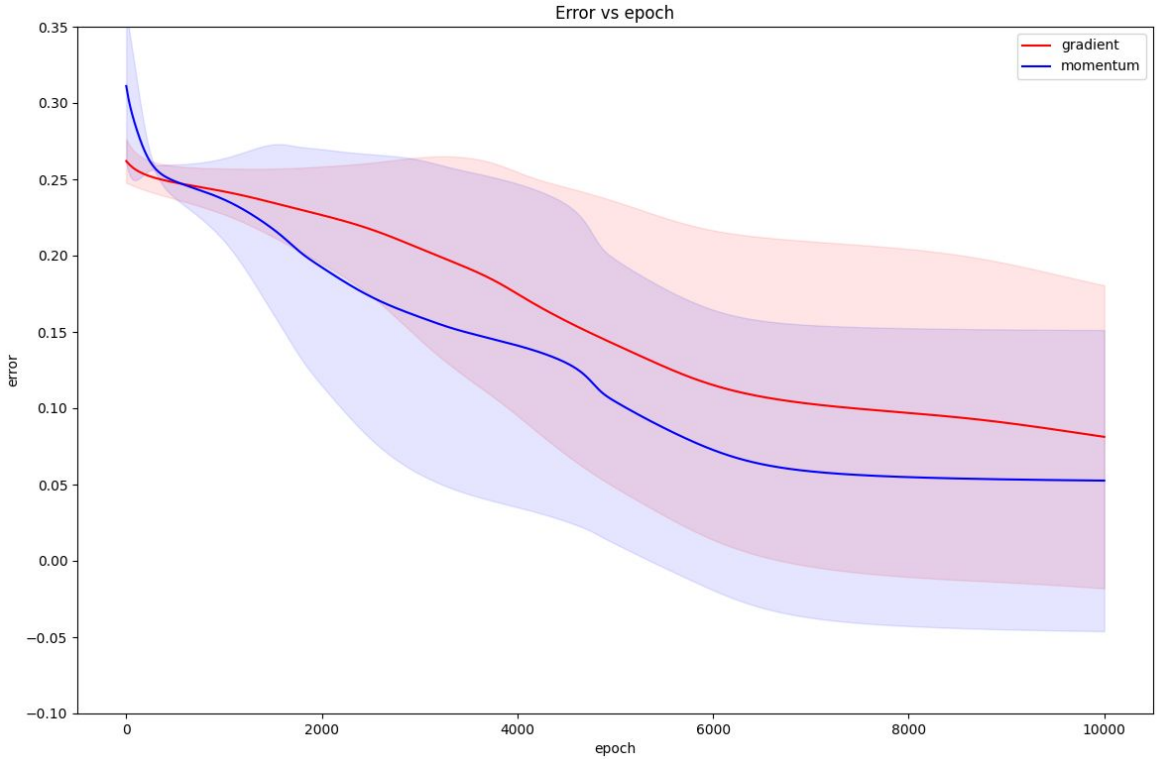
Resultados - *Batch*, *tanh*, $\eta = 10^{-4}$, $\beta = 1$

[5,1]



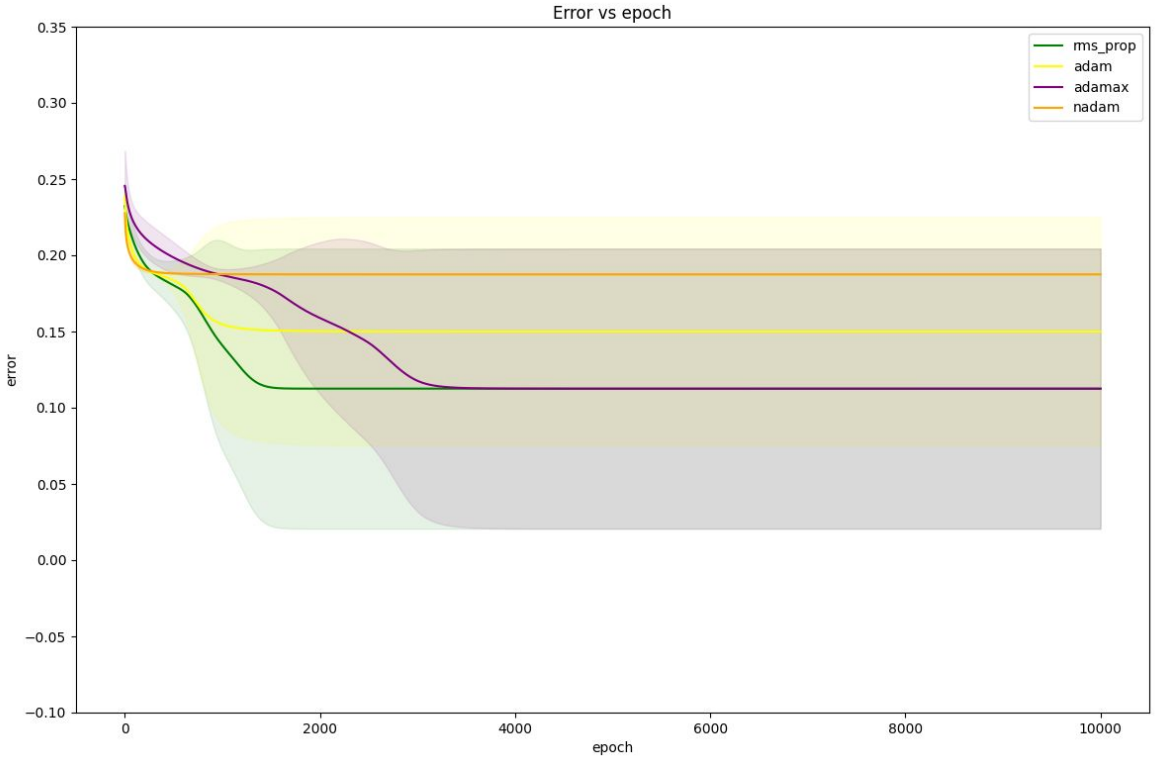
Resultados - *Batch*, *tanh*, $\eta = 10^{-4}$, $\beta = 1$

[5,1]



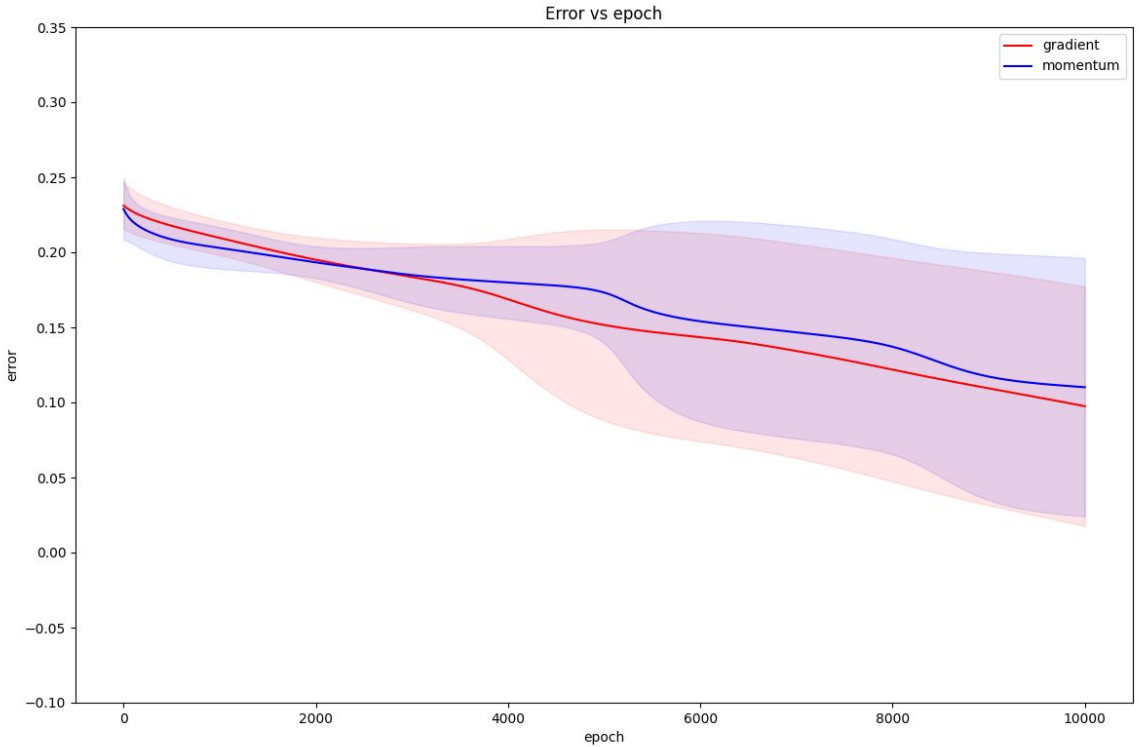
Resultados - *Batch*, *tanh*, $\eta = 10^{-4}$, $\beta = 1$

[4,2,1]



Resultados - *Batch*, *tanh*, $\eta = 10^{-4}$, $\beta = 1$

[4,2,1]



Análisis

Podemos ver que mientras más neuronas se agregan:

- Aumento en la desviación estándar de *gradient* y *momentum*.
- Disminución de la desviación estándar en los métodos *adam*-like dando, además, una convergencia mucho más rápida.

Por otro lado, aumentar las capas ocultas no produce mejores resultados.

- *Gradient* y *momentum* tardan en converger.
- Los métodos *adam*-like se estancan en un mínimo local.

Ejercicio 3c
Identificación de dígitos

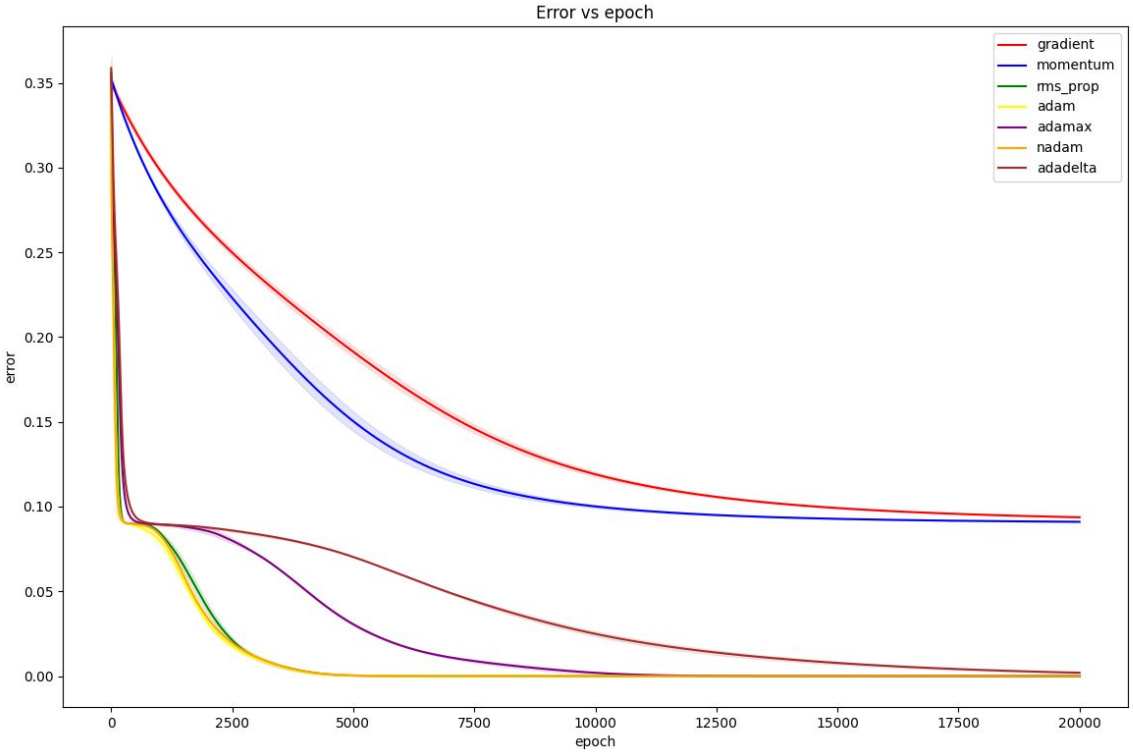
Prueba

Se utilizó *Batch* como entrenamiento, y se probaron muchos hiperparámetros y optimizadores. Se probó [20, 10] y [20, 20, 10] pero se decidió por [20, 10].

Además, se utilizaron distintas formas de agregar ruido al *dataset* y evaluar cómo los distintos algoritmos se comparaban.

Resultados - *Batch*, *tanh*, $\eta = 1^{*}10^{-3}$, $\beta = 5^{*}10^{-2}$

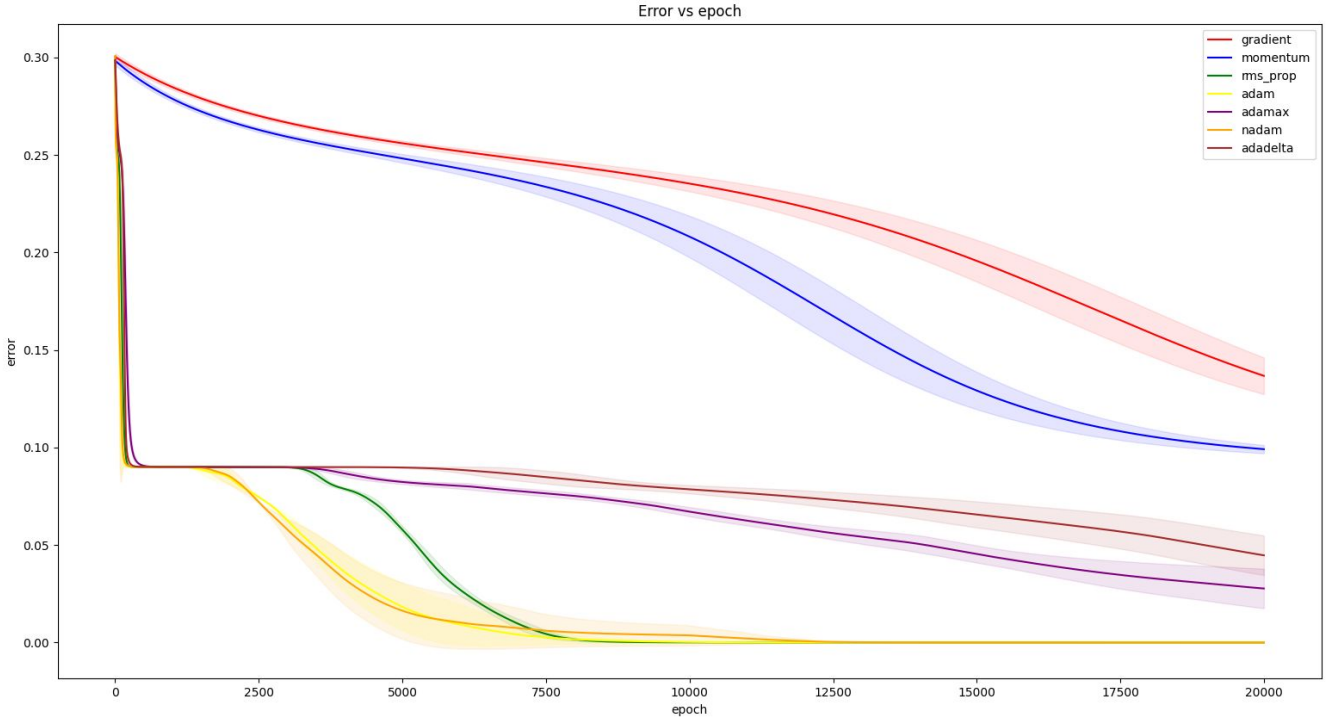
[20, 10]



- Gradiente y *Momentum* son lentos, disminuyendo poco el error comparado con los otros. Además, se quedan en mínimos locales.
- El resto de los optimizadores encuentra el mínimo local, y lentamente busca otros mínimos aún menores.
- Tanto *RMS Prop* y *Adam-like* son más rápidos al disminuir el error, pero entre ellos se destacan *Adadelta* y *Adamax* como los más lentos.

Resultados - *Batch*, *tanh*, $\eta = 1^{*}10^{-3}$, $\beta = 5^{*}10^{-2}$

[20, 20, 10]

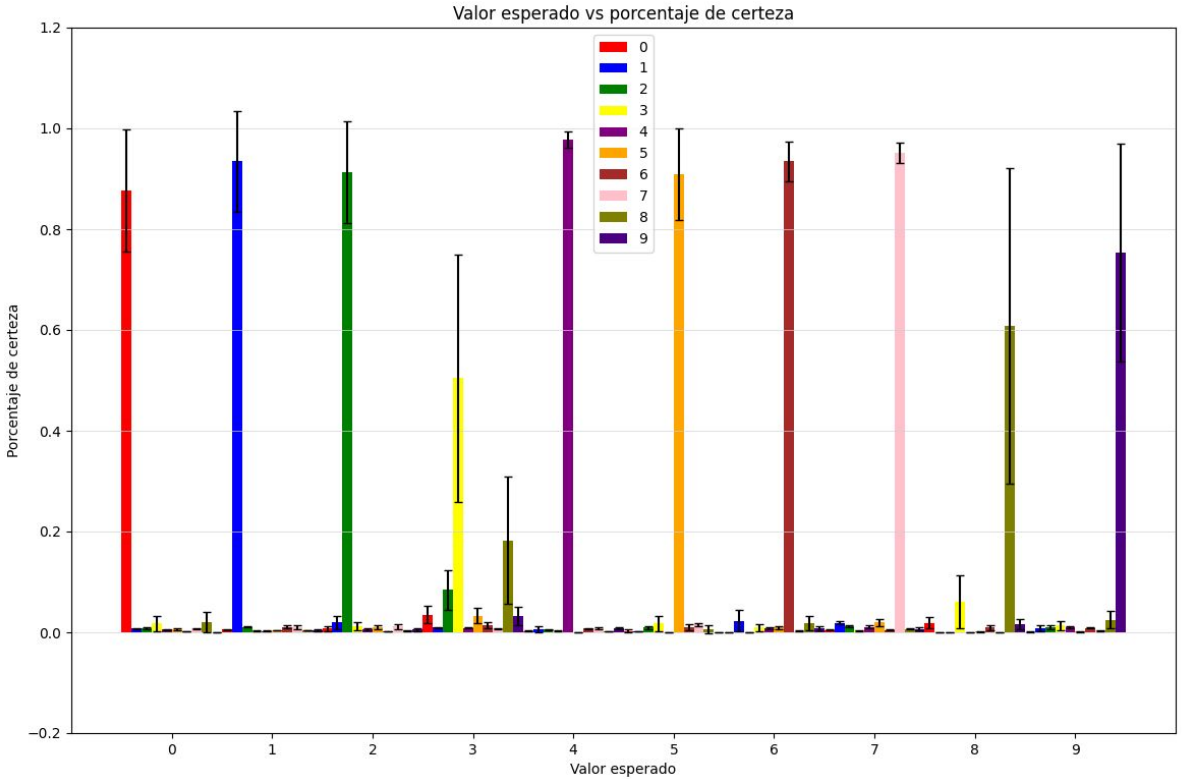


Ruido - Uniforme

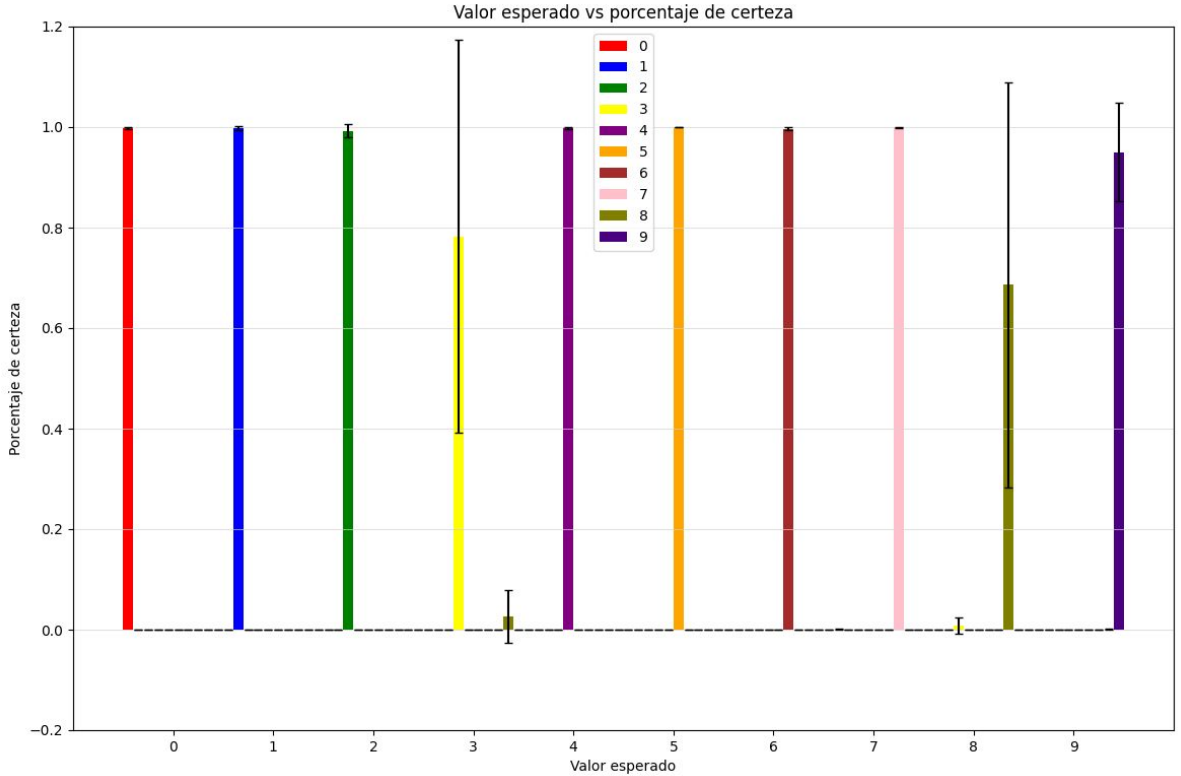
Se utilizó una función de probabilidad uniforme para invertir el color de un punto con probabilidad 0,05.

Aquí se corrió con la siguiente configuración: batch, tanh, η = 5e-3, β = 0.05 y el máximo de iteraciones 5e4.

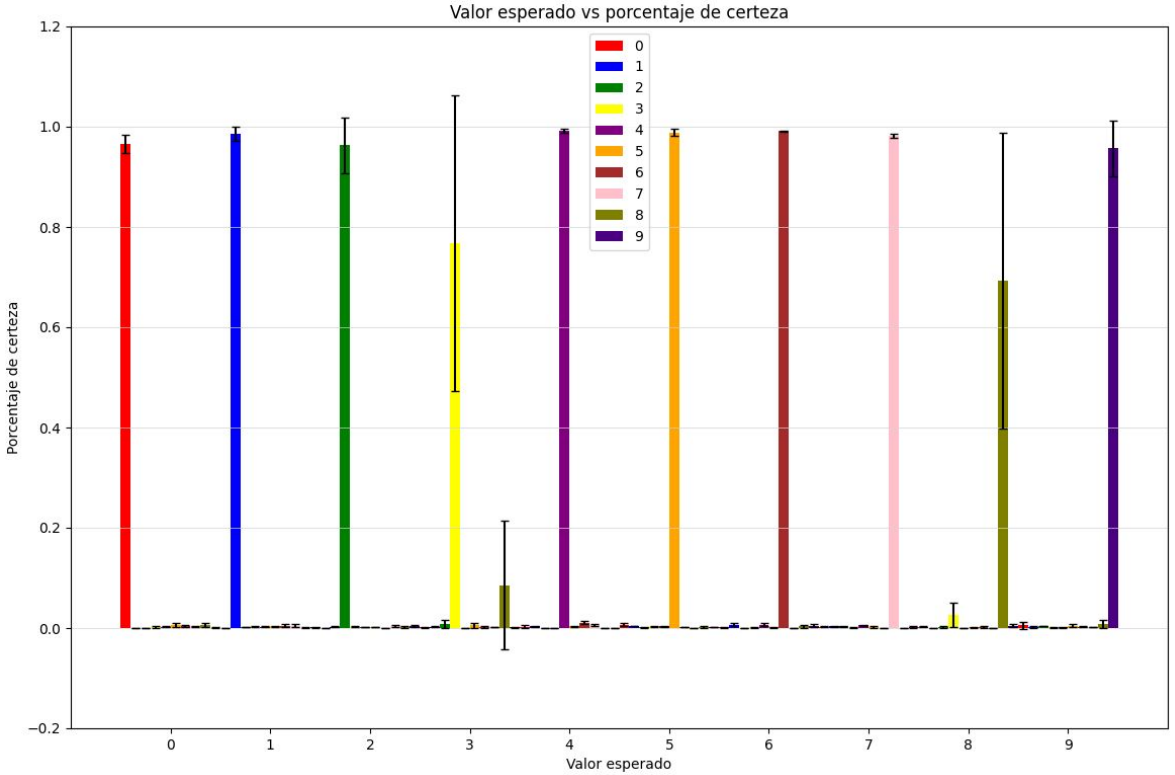
Resultados - Uniforme - *Adam*



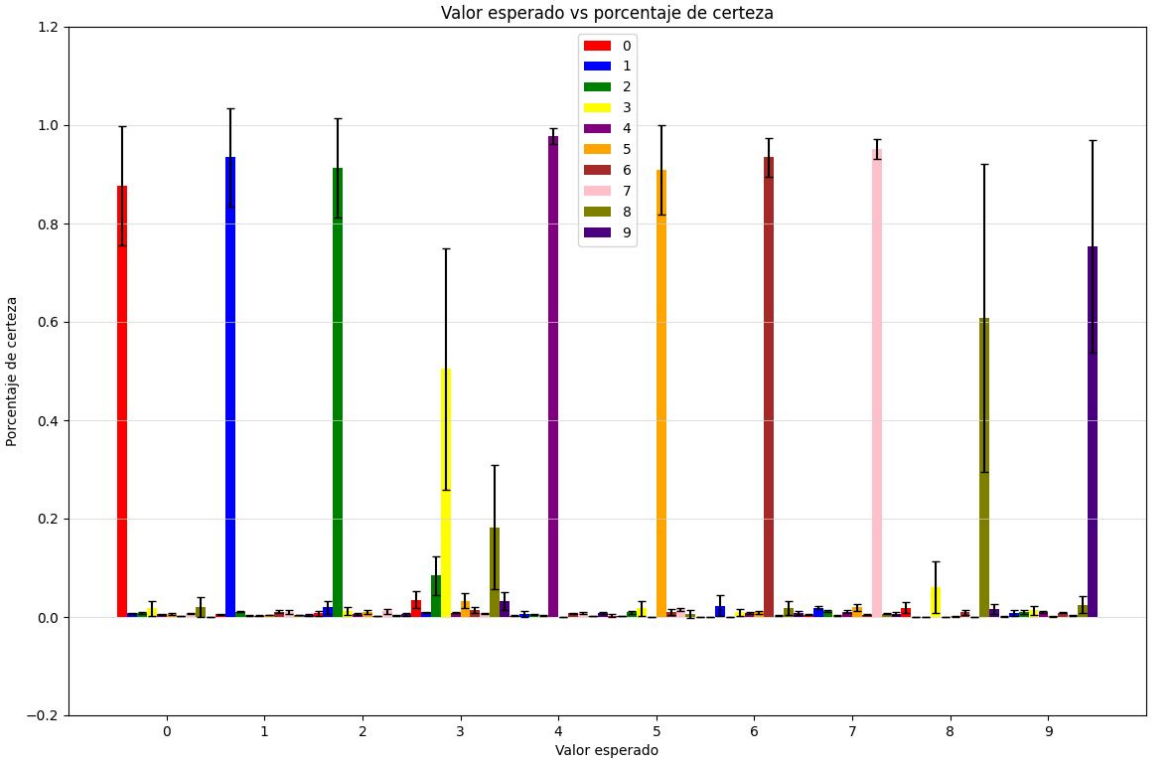
Resultados - Uniforme - *Nadam*



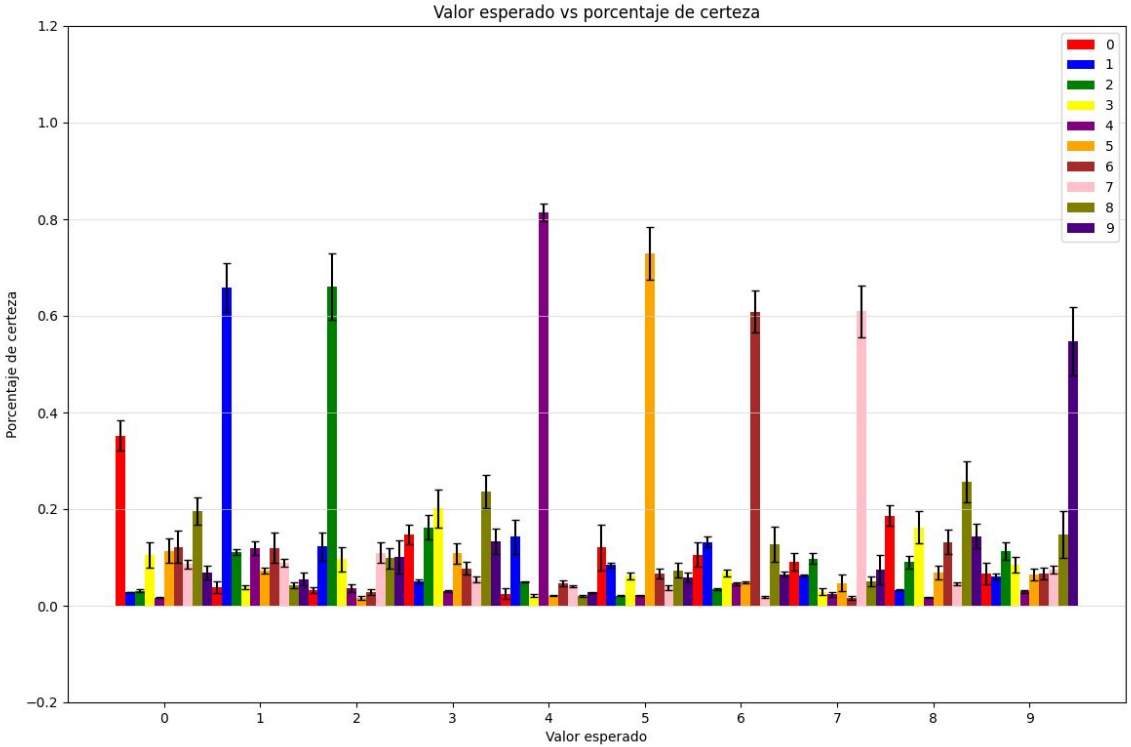
Resultados - Uniforme - *RMS Prop*



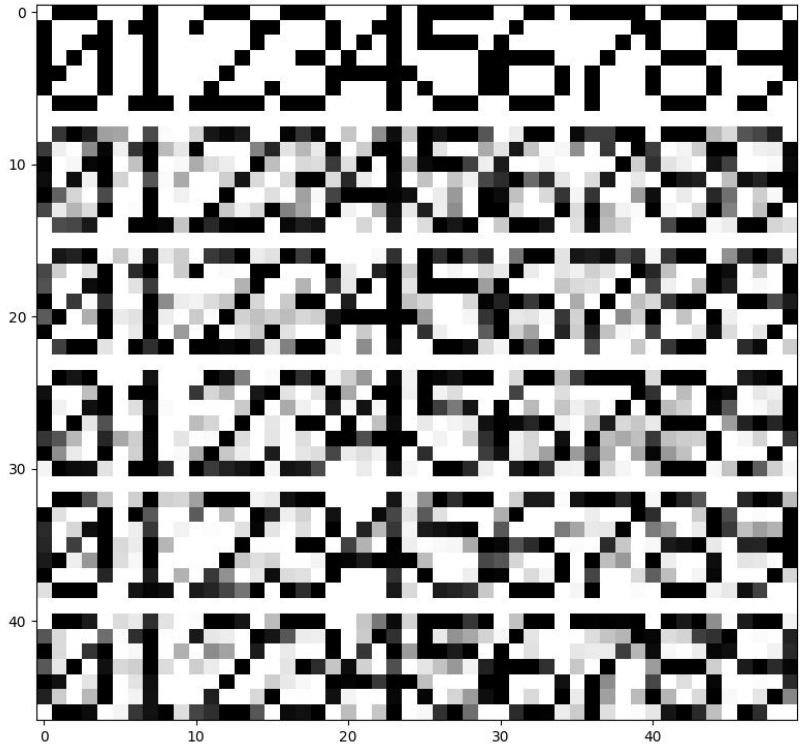
Resultados - Uniforme - *Adamax*



Resultados - Uniforme - *Adadelta*

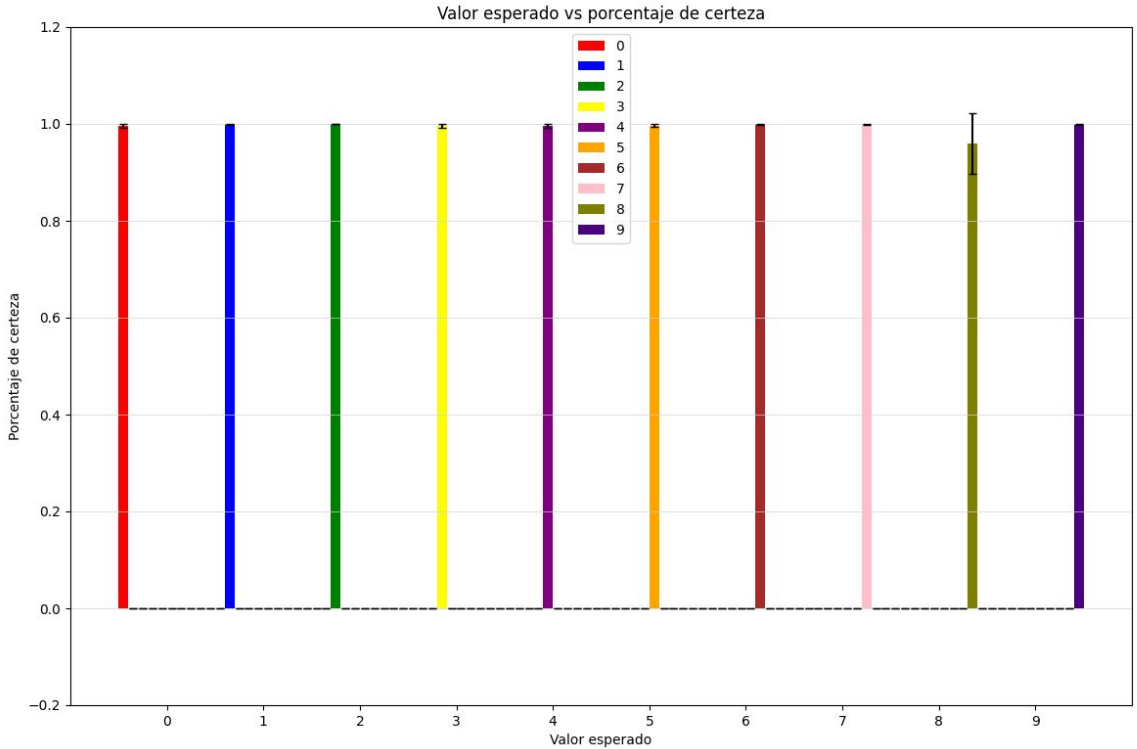


Ruido - Gaussiano

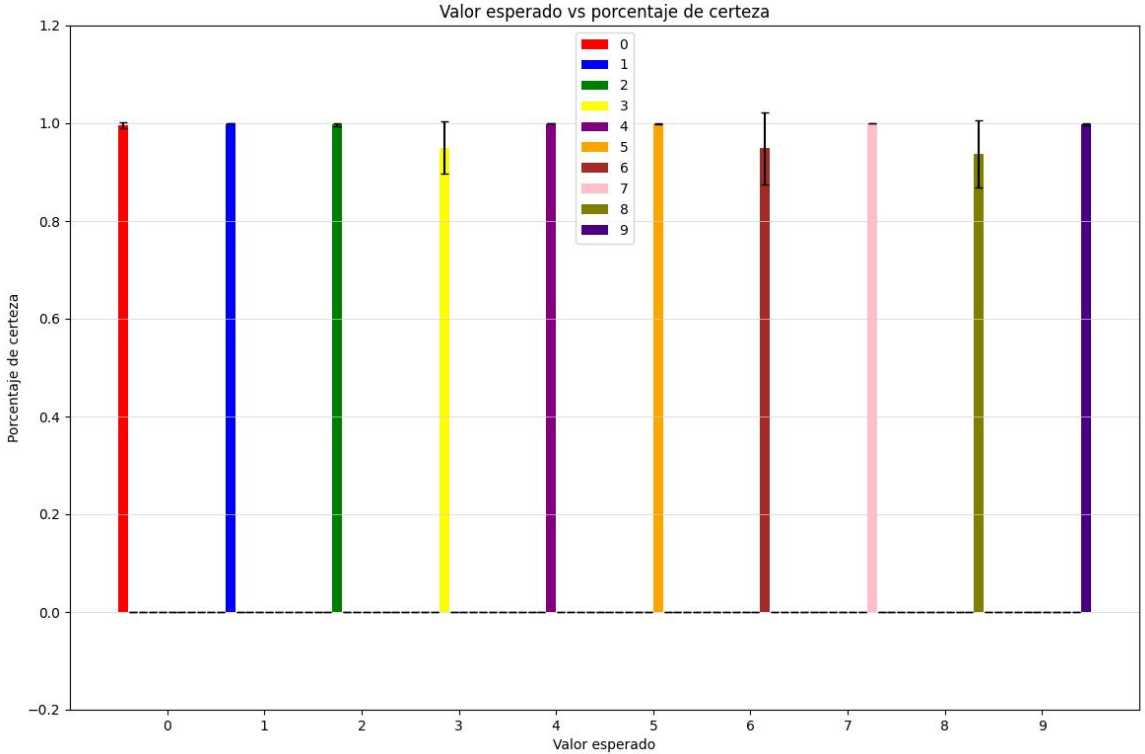


Se utilizó una función de probabilidad normal para invertir el color de un punto con desviación de 0,2.

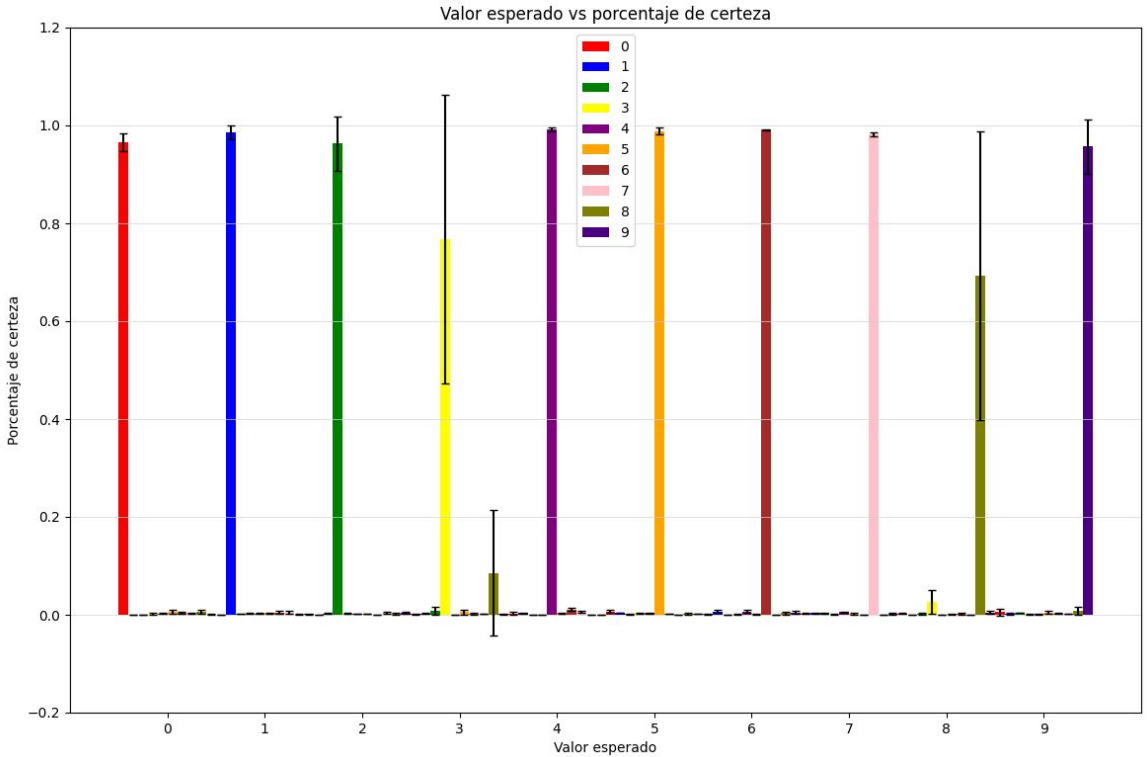
Ruido - Gaussiano 0,2 - *Adam*



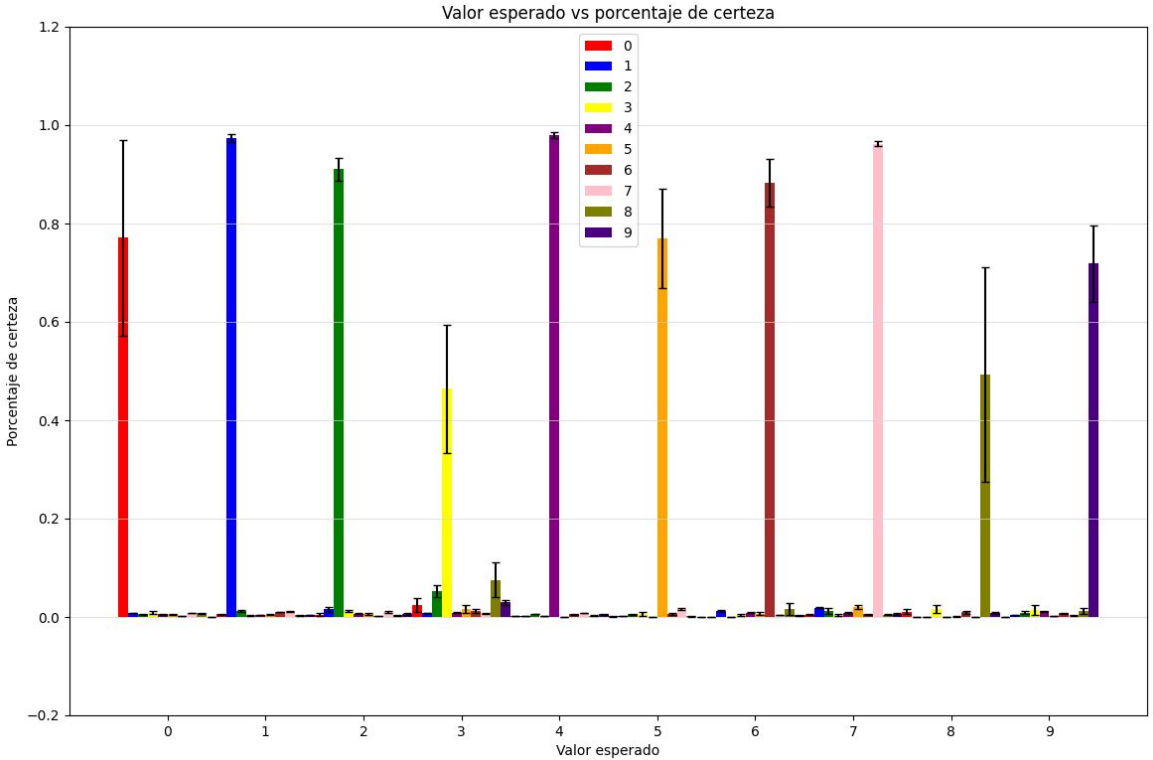
Ruido - Gaussiano 0,2 - *Nadam*



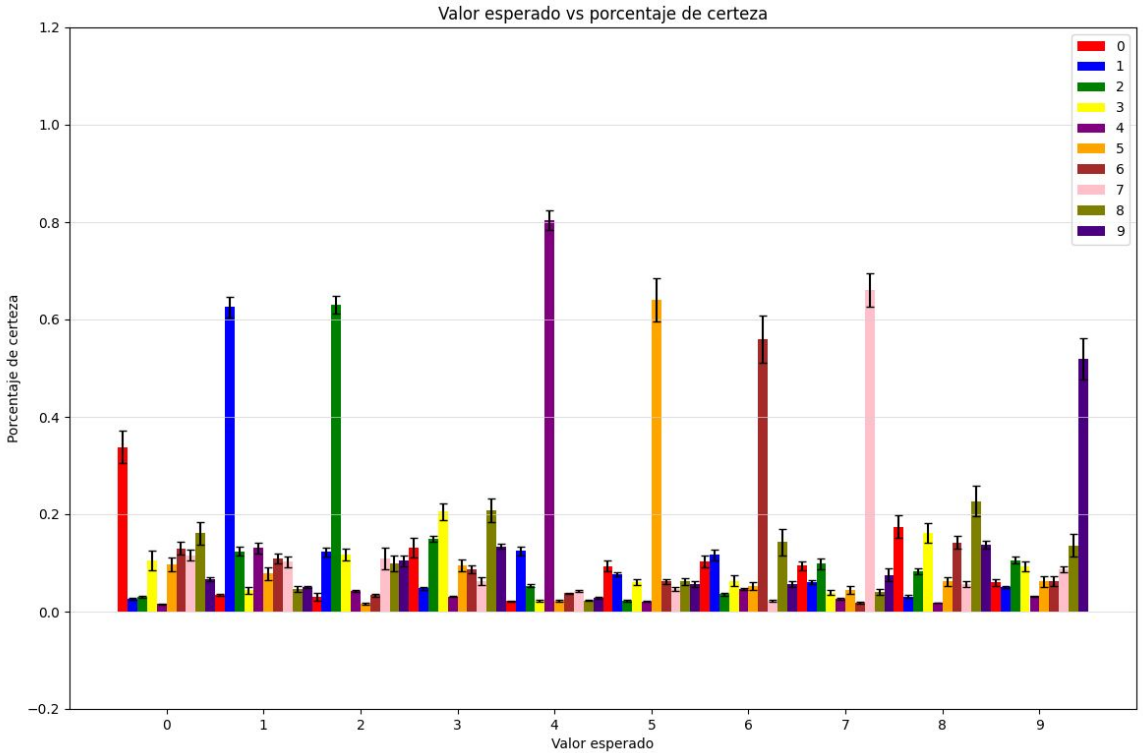
Ruido - Gaussiano 0,2 - *RMS Prop*



Ruido - Gaussiano 0,2 - *Adamax*



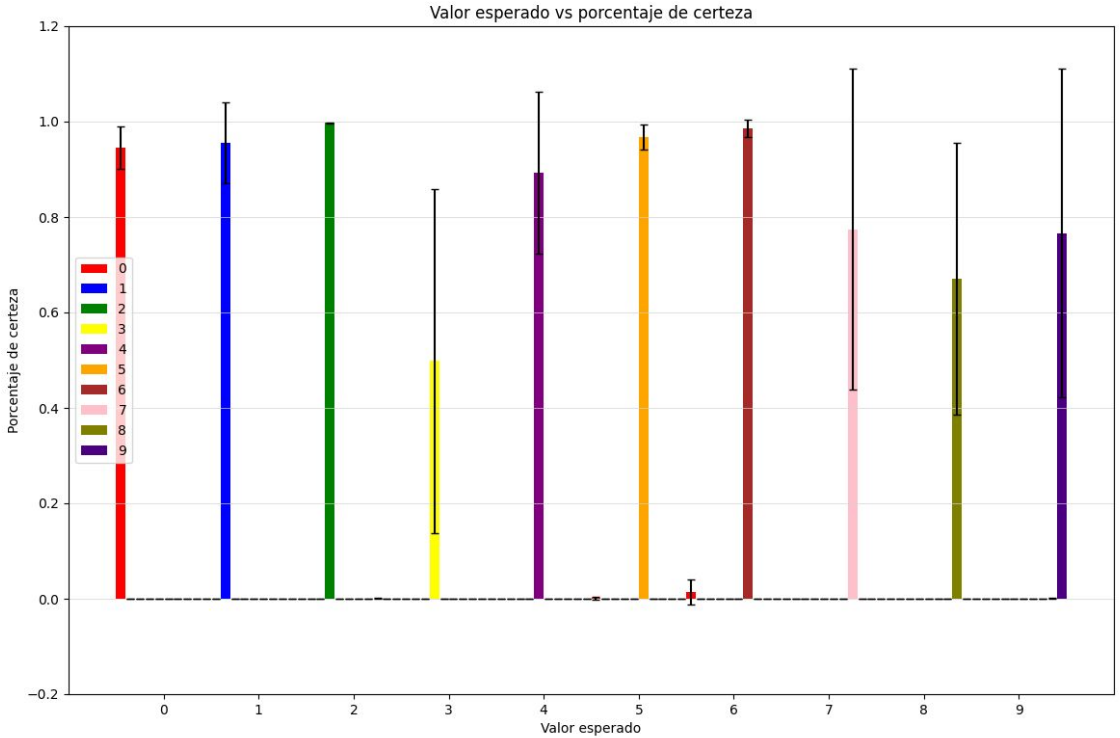
Ruido - Gaussiano 0,2 - *Adadelta*



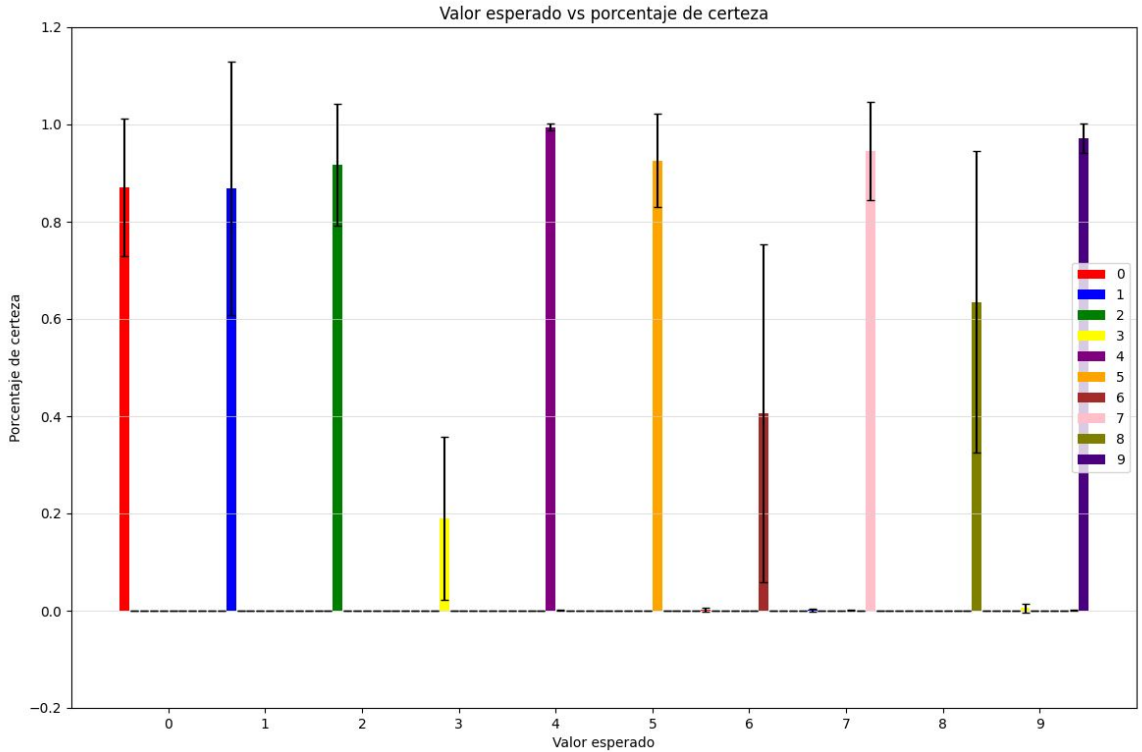
Ruido - Gaussiano

Se utilizó una función de probabilidad normal para invertir el color de un punto con desviación de 0,35.

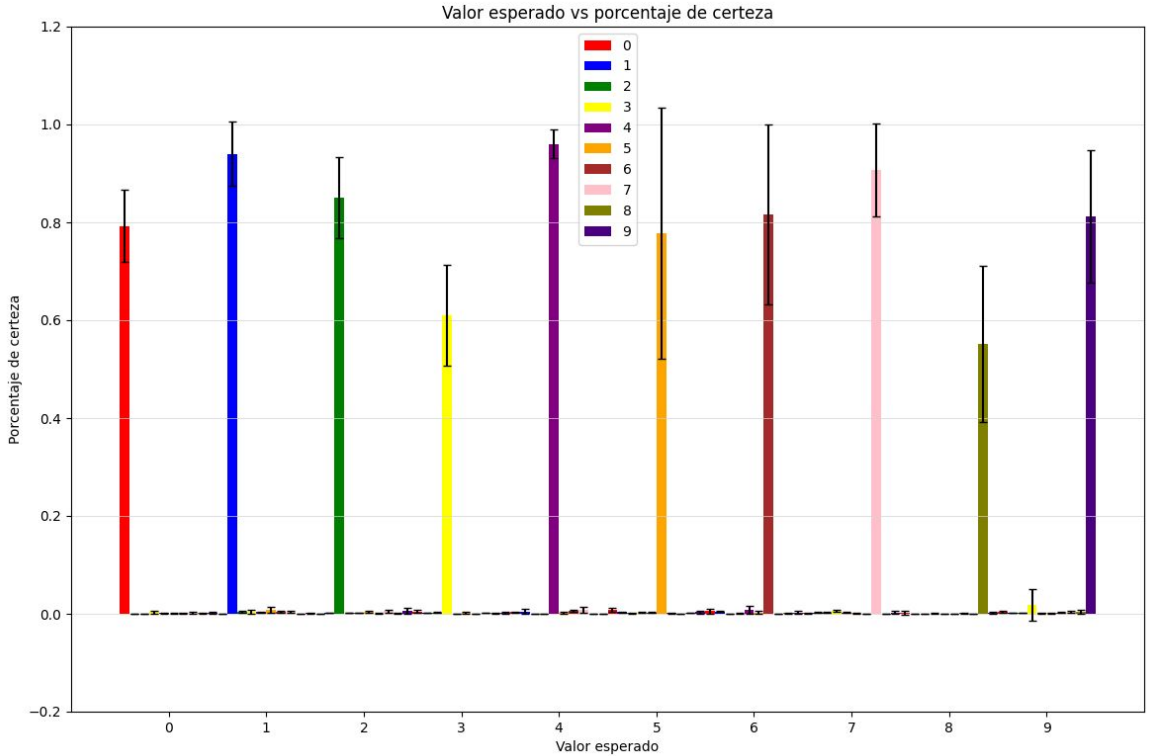
Ruido - Gaussiano 0,35 - *Adam*



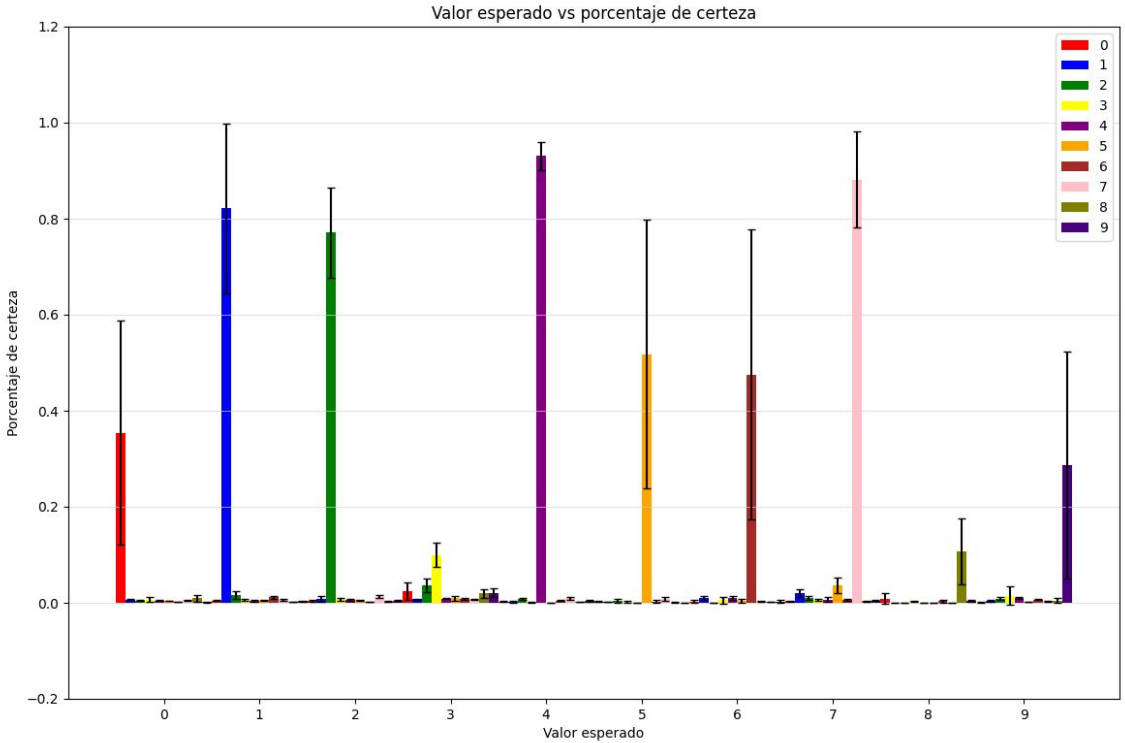
Ruido - Gaussiano 0,35 - *Nadam*



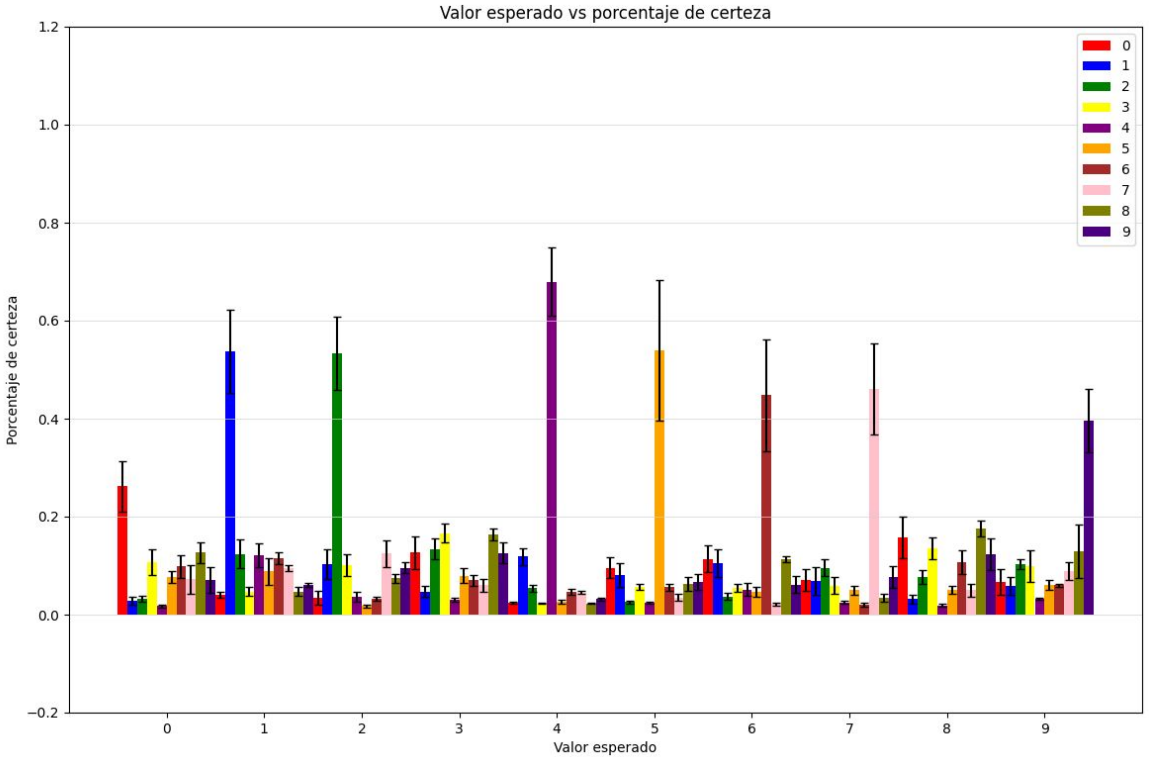
Ruido - Gaussiano 0,35 - *RMS Prop*



Ruido - Gaussiano 0,35 - *Adamax*



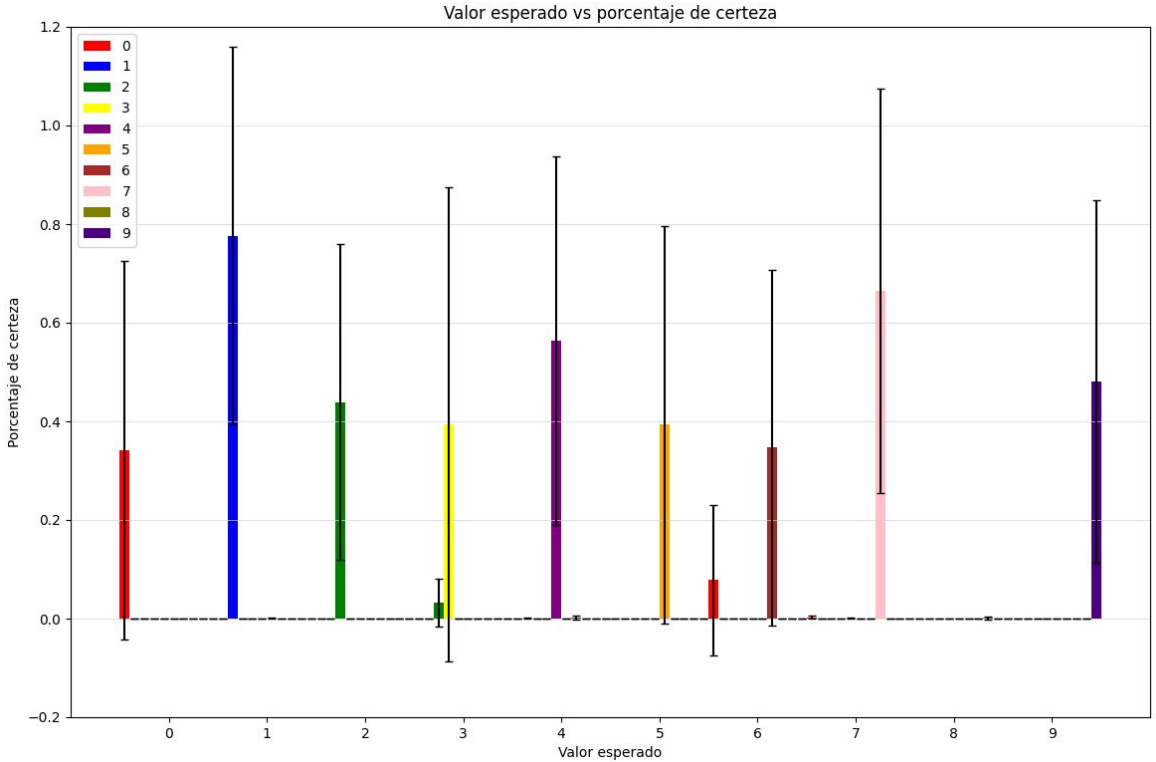
Ruido - Gaussiano 0,35 - *Adadelta*



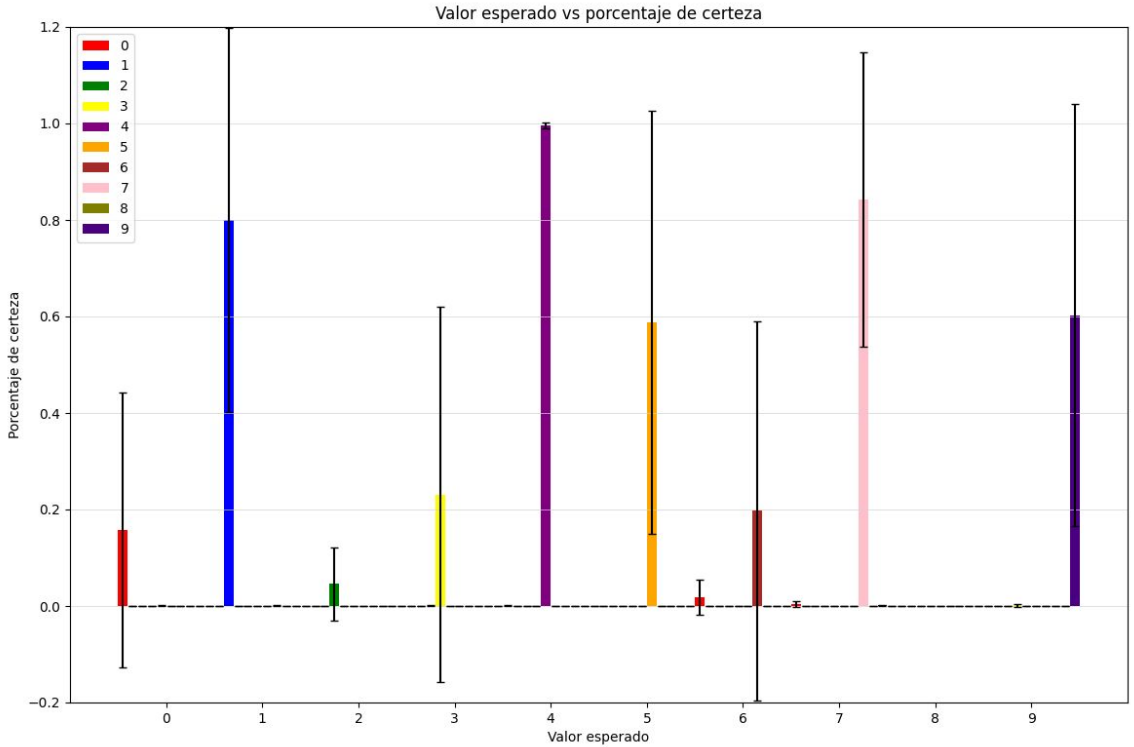
Ruido - Gaussiano

Se utilizó una función de probabilidad normal para invertir el color de un punto con desviación de 0,5.

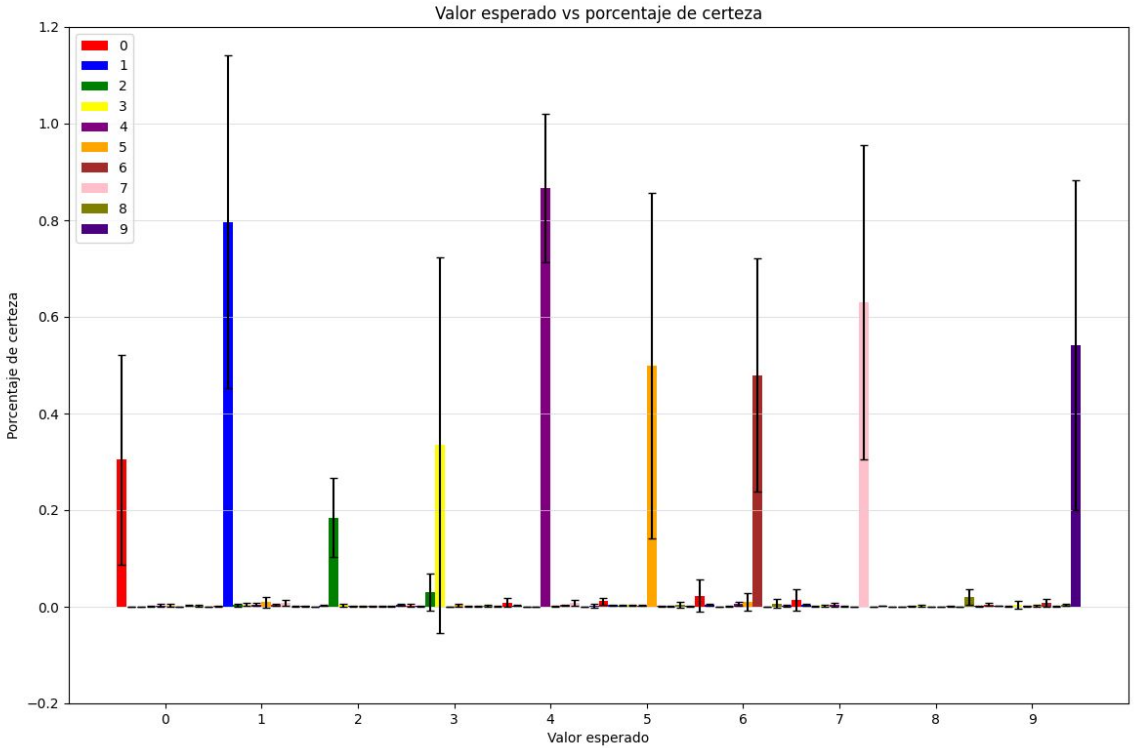
Ruido - Gaussiano 0,5 - *Adam*



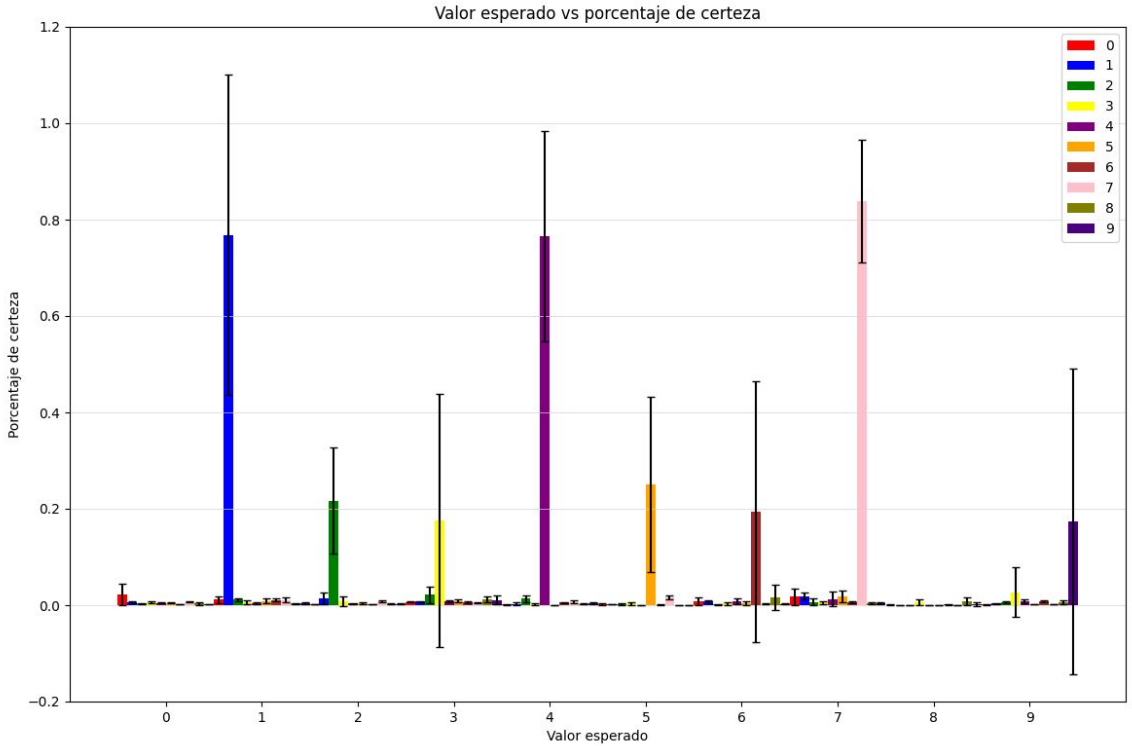
Ruido - Gaussiano 0,5 - *Nadam*



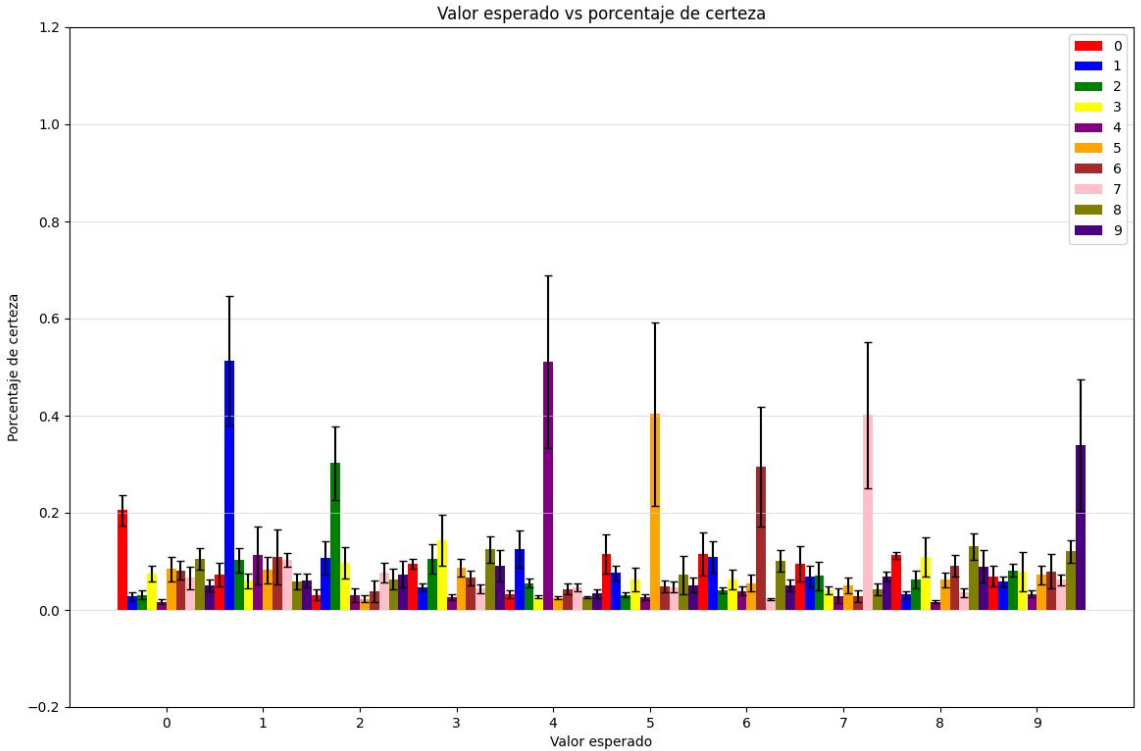
Ruido - Gaussiano 0,5 - *RMS Prop*



Ruido - Gaussiano 0,5 - *Adamax*



Ruido - Gaussiano 0,5 - *Adadelta*



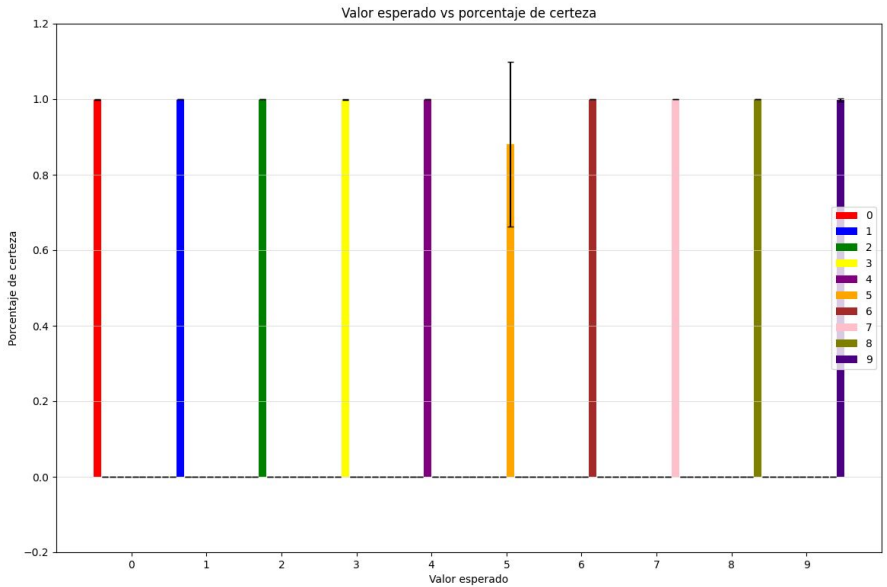
Caso overfitting

Se entrenó un perceptrón multicapa con dimensiones [20, 20, 10] con el optimizador Adam.

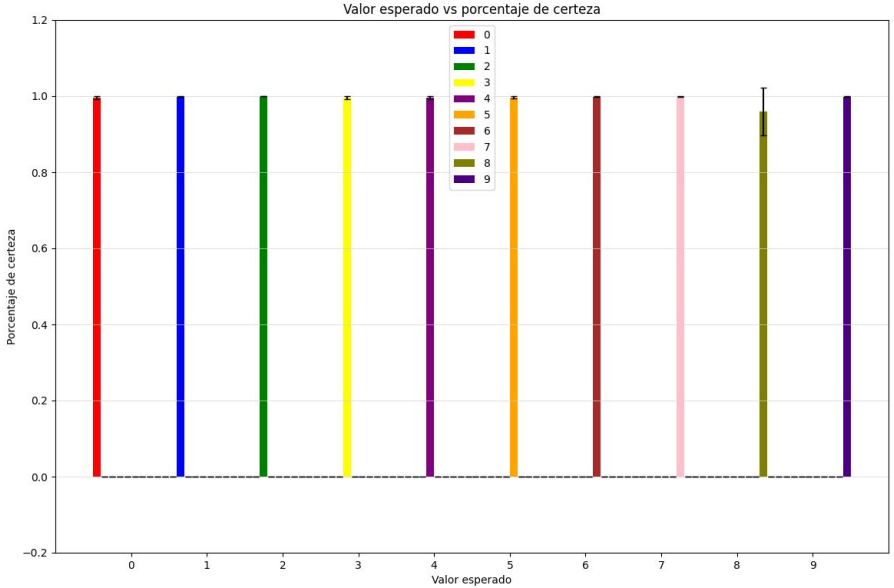
Al contar con una capa oculta más, no solo tarda más en converger sino que además se genera overfitting; lo cual significa que no obtendrá resultados consistentes fuera de los datos del *dataset*.

Ruido - Gaussiano (std=0.2) - *Adam*

[20, 20, 10]

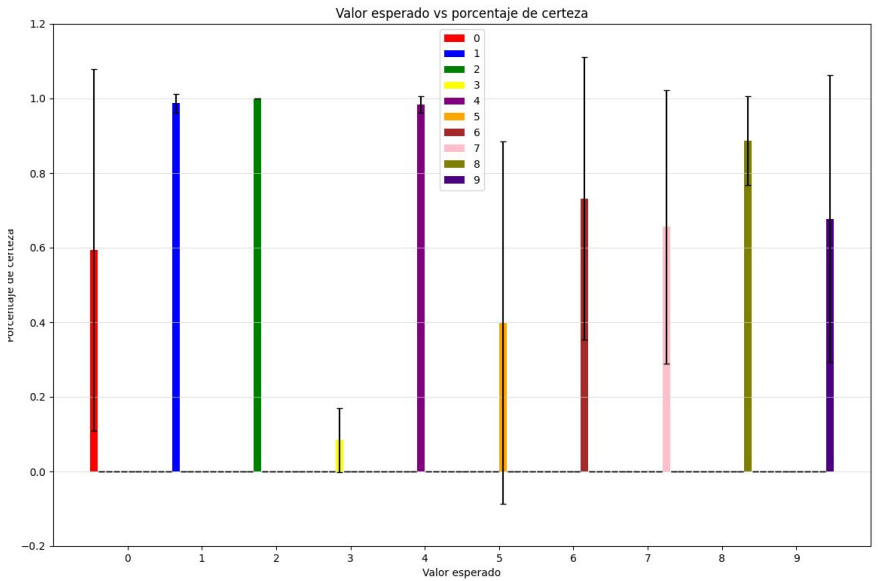


[20, 10]

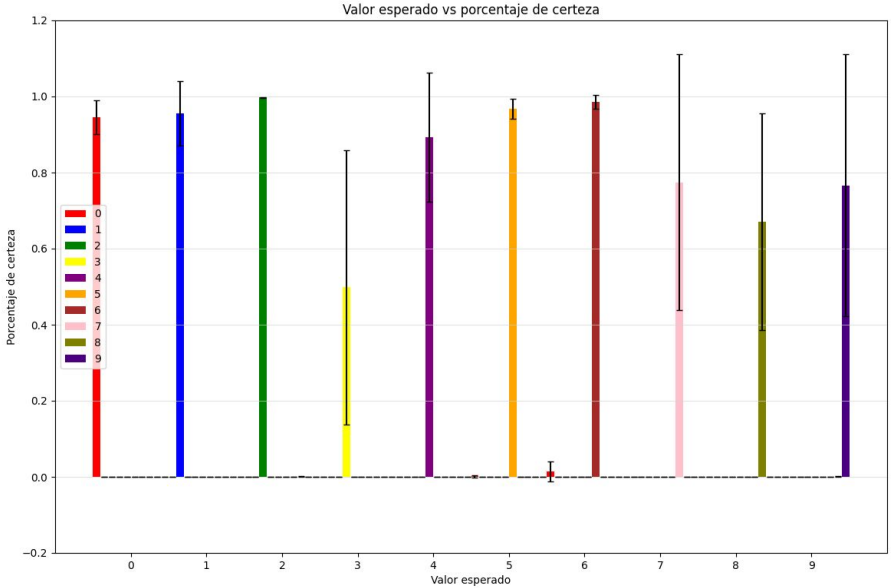


Ruido - Gaussiano (std=0.35) - *Adam*

[20, 20, 10]

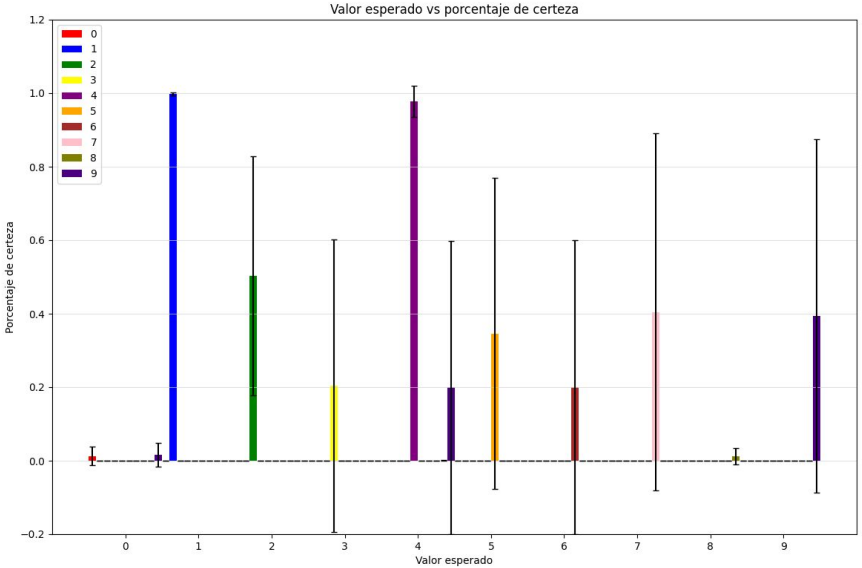


[20, 10]

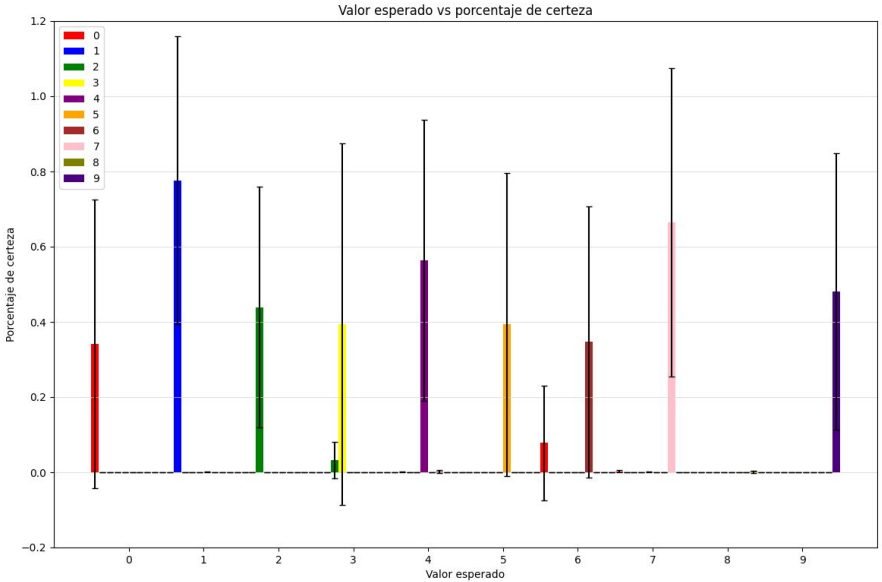


Ruido - Gaussiano (std=0.5) - *Adam*

[20, 20, 10]

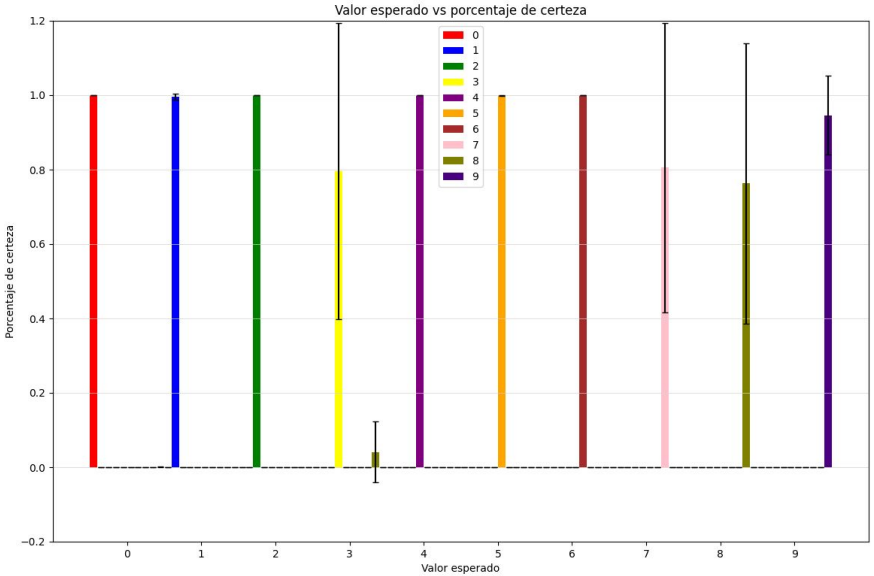


[20, 10]

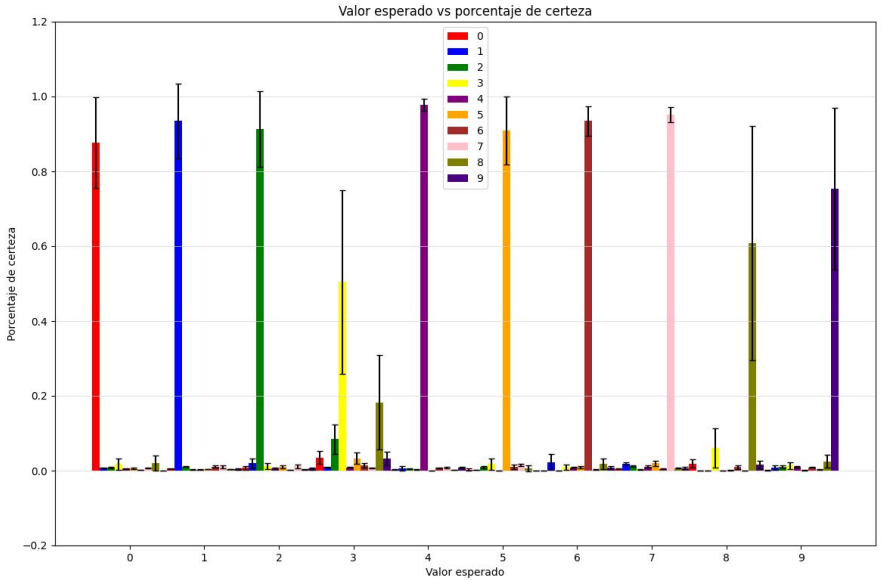


Ruido - Uniforme - *Adam*

[20, 20, 10]



[20, 10]



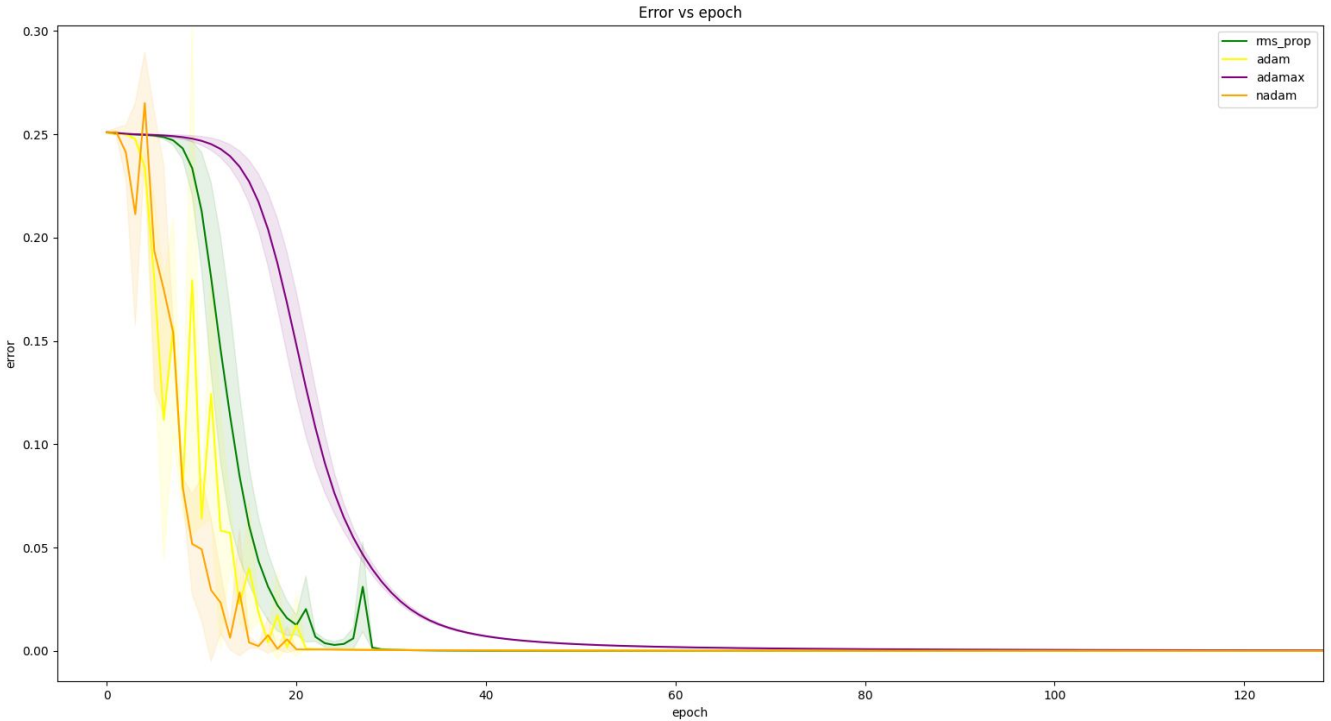
Ejercicio 3b
Paridad

Prueba

Se utilizó *Batch* como entrenamiento, y se probaron muchos hiperparámetros y optimizadores. Se probó [20, 10, 2] y [15, 2].

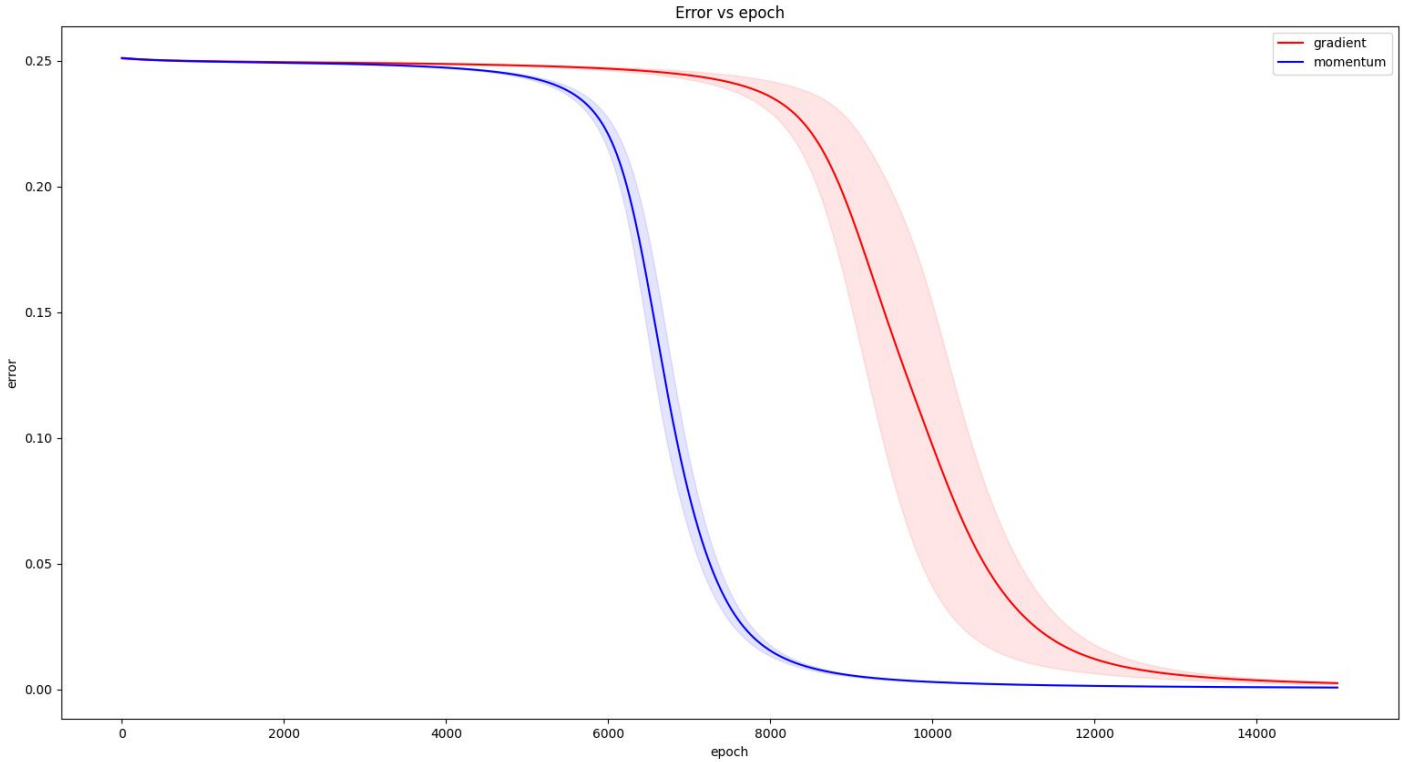
Resultados - *Batch*, *tanh*, $\eta = 0.05$, $\beta = 0.05$

[20, 10, 2]



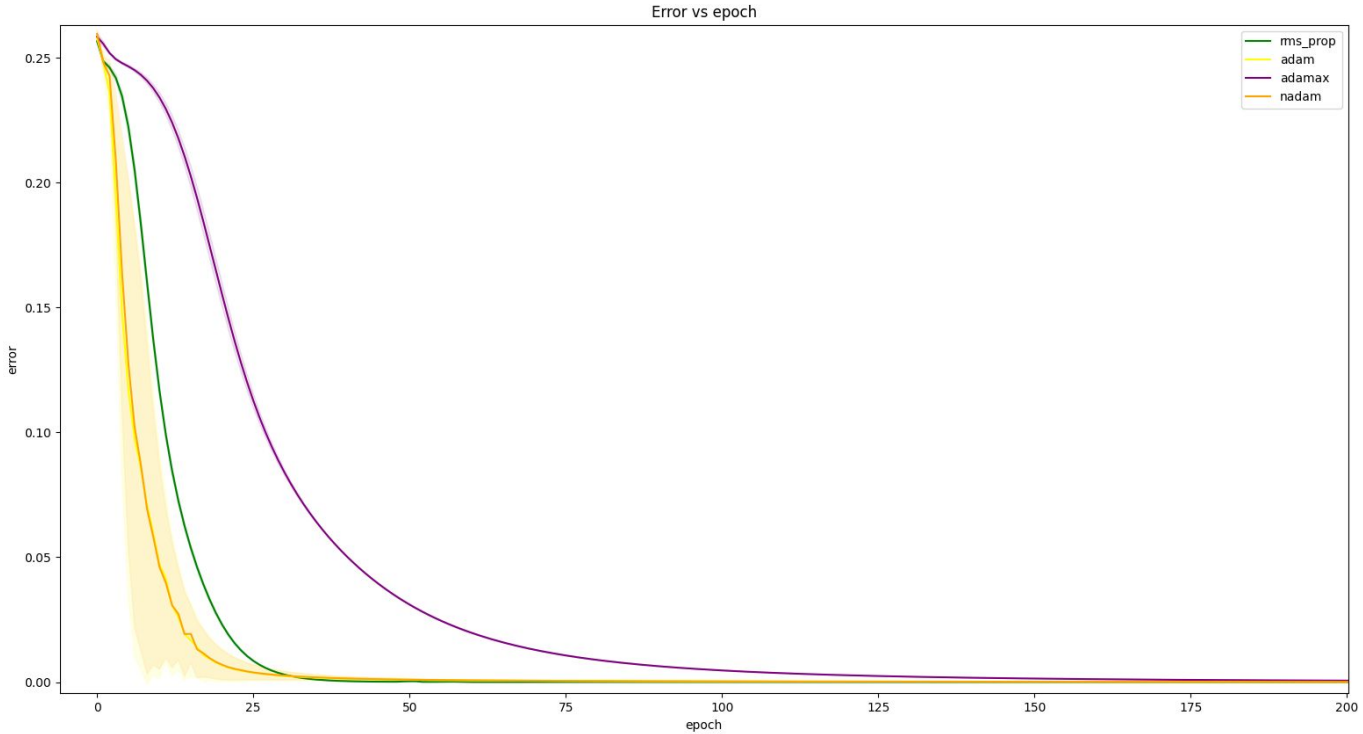
Resultados - *Batch*, *tanh*, $\eta = 0.05$, $\beta = 0.05$

[20, 10, 2]



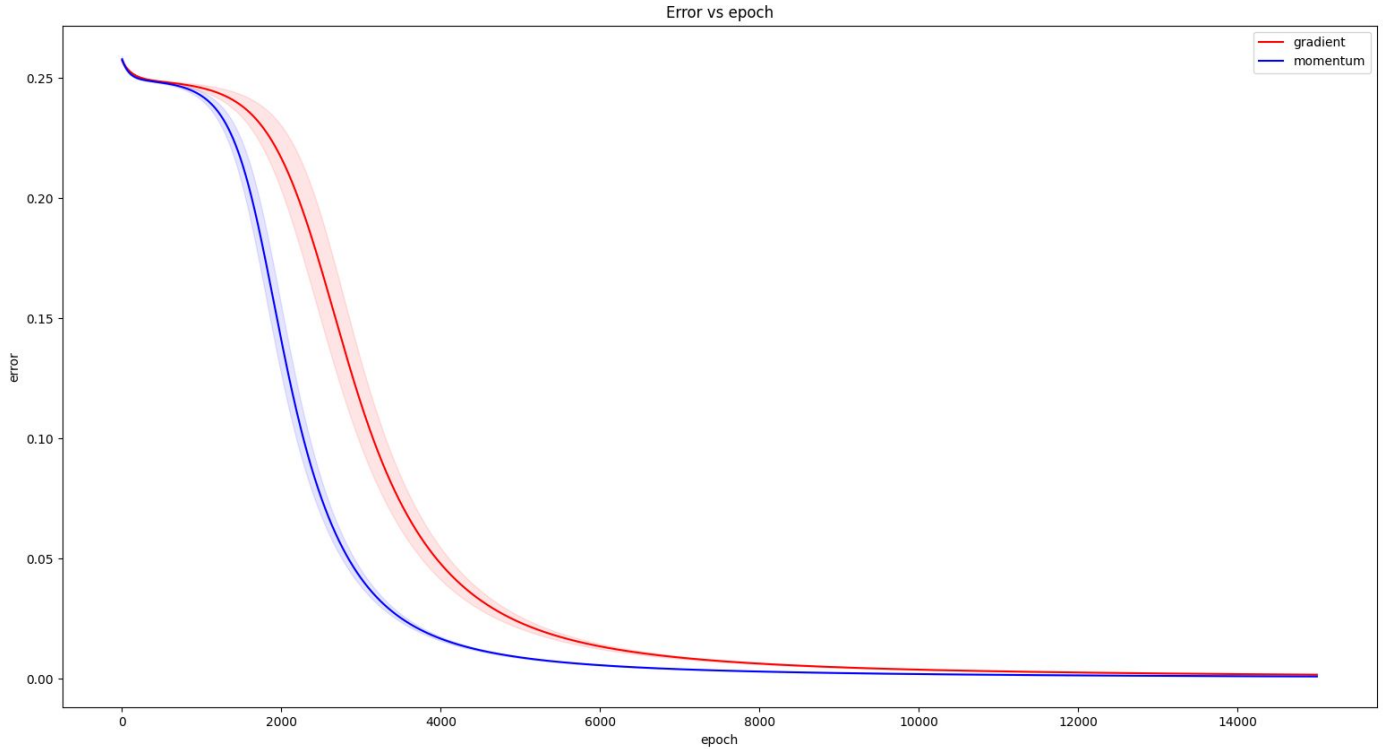
Resultados - *Batch*, *tanh*, $\eta = 0.05$, $\beta = 0.05$

[15, 2]



Resultados - *Batch*, *tanh*, $\eta = 0.05$, $\beta = 0.05$

[15, 2]



Conclusiones