Sistemas de Inteligencia Artificial

Redes Neuronales

1er Cuatrimestre 2019

Grupo 08

- Diego Bruno
- Tomás Ferrer
- Matías Heimann
- Marcos Lund

Índice General

Objetivo

Se propuso desarrollar una red neuronal multicapa que aproxime lo mejor posible un terreno dado en base a coordenadas del formato ‘latitud, longitud, altitud’ que lo describen. Para esto se buscó implementar dos métodos de entrenamiento distintos (incremental y batch), con el fin de poder comparar los resultados obtenidos. Asimismo, se quiso implementar mejoras al algoritmo utilizado con el fin de optimizar el proceso de aprendizaje.

Desarrollo del Proyecto

Configuración de la red

Se implementó un archivo de configuración con el fin de que todos aquellos aspectos relevantes de la red neuronal fueran parametrizables.

Una de las características fundamentales a determinar fue la arquitectura de la misma. Por lo tanto, se logró que un usuario pudiera configurar la cantidad de capas ocultas presentes en la red y la cantidad de neuronas presentes en ellas. Por ejemplo, una red neuronal con 2 capas ocultas y 5 neuronas presentes en cada una se indicaría de la manera [5, 5]. En base a esto, los tamaños de ciertas variables (los pesos, deltas, etc.) son asignados dinámicamente. La cantidad de entradas de la red fue prefijada en base al problema dado, en este caso, a dos: las variables de longitud y latitud para un punto dado del conjunto de patterns. Por su parte, la cantidad de salidas fue establecida en una, valor que luego se compara directamente con la salida esperada en el archivo de datos.

Para la etapa de entrenamiento de la red se utilizó el algoritmo de backpropagation visto en clase, en el cual se calcula en primer lugar la salida obtenida partiendo de la primera capa de entrada, y luego se propaga el error hacia atrás modificando los pesos de cada conexión según el delta necesario. Para cada patrón del conjunto de entrenamiento, se compara el error cuadrático medio obtenido con el cuadrado de un epsilon configurable para determinar si hubiera que realizar dicha modificación. Es posible fijar un número máximo de iteraciones mediante la variable max_epochs, la cual propiamente limita cuántas veces se cicla por todo el conjunto de entrenamiento. Si no se quisiera establecer un límite, se le asignaría el valor -1.

Para inicializar el proceso de entrenamiento fue necesario establecer un porcentaje requerido de patrones a utilizar. Por ejemplo, para un total de 441 puntos se podría setear la variable patterns a 0.25 con el fin de utilizar 110 entradas, un 25% de la cantidad total. En base al resultado obtenido con ellos, fue luego posible

generalizar para el conjunto entero de datos como se verá a continuación. Dicha elección de patrones se realiza al azar. Asimismo, fue posible dar al usuario la opción de utilizar ciertos puntos elegidos por el mismo en base a su número de línea en el archivo .data, con el fin de evitar que la red elija luego al azar entradas no representativas (aunque lo mismo se podría implementar para que nunca suceda mediante ciertos chequeos de las altitudes correspondientes a los puntos elegidos). Por ejemplo, para elegir los primeros tres puntos de la lista se debería setear la variable input_patterns a [1, 2, 3].

En cuanto a la inicialización de pesos para la red neuronal, se procuró no elegir valores muy grandes como para que la misma no se sature y se lo hizo de manera tal que sean del orden de $\frac{1}{\sqrt{k_i}}$ donde k_i corresponde al número de entradas de la capa inferior a la actual.

Se implementaron dos funciones de activación diferentes para utilizar en la red entera: la tangente hiperbólica y la exponencial. Además, se implementó el beta utilizado por ambas funciones, el cual es configurable.

$$g(x) = tanh(\beta x) \qquad g(x) = \frac{1}{1 + e^{-2\beta x}}$$

En base a qué función es elegida (mediante la variable fn_index), fue necesario normalizar los datos de entrada para que coincidan con la imagen de la misma. También fue necesario desnormalizar la salida de la red neuronal para poder comparar de manera correcta con la salida esperada del patrón elegido.

Finalmente, si se quisiera modificar alguna variable antes de que el proceso finalice sería posible interrumpiendo la ejecución y seteando la variable **stop_now** a true. Si se continuara la ejecución mediante el comando dbcont, el programa se detendría nuevamente y en ese momento el usuario podría reasignar valores garantizando que se haga al comienzo de una nueva época con el fin de evitar posibles imprevistos.

Implementación de Entrenamiento Incremental

En la ejecución incremental del proceso de entrenamiento, se eligió al azar un orden para los distintos patrones a utilizar, y se fueron modificando los pesos de cada conexión en la red neuronal basándose en la salida producida por cada entrada. Una vez que la diferencia entre las salidas calculadas y las salidas obtenidas para cada una es menor a epsilon, finalizará el proceso.

Implementación de Entrenamiento Batch

En la ejecución del proceso de entrenamiento, se realizó el mismo criterio que se utilizó en el entrenamiento incremental. Este método consiste en corregir los pesos de las neuronas teniendo en cuenta el error en cada patrón. Los pesos se vuelven a calcular utilizando la fórmula $\Delta w_{pq} = \Sigma (\delta_p V_q)$. La sumatoria es por cada patrón.

Mejoras al Algoritmo de Aprendizaje

Eta adaptativo

Los parámetros adaptativos permiten ajustar el valor del learning rate durante el aprendizaje con la siguientes reglas:

- Si varios pasos disminuyen el error se aumenta el learning rate de manera constante
- Si aumenta el error después de un paso, este paso se deshace y se disminuye geométricamente el learning rate.

Parámetros:

- Eta_a (utilizado para aumentar el learning rate)
- Eta_b (utilizado para reducir el learning rate)

$$\Delta\eta = \begin{cases} +a & si \;\; \Delta E < 0 \;\; consistentemente \\ -b\eta & si \;\; \Delta E > 0 \\ 0 & en \; otro \; caso \end{cases}$$

Momentum

Esta mejora permite realizar correcciones más grandes cuando la función de costo avanza de forma consistente. Utiliza el salto anterior para intentar salvar mínimos locales pero un momentum alto podría generar un gran salto el cual puede omitir el mínimo global.

Parámetro:

- Momentum_alpha

ADAM

ADAM utiliza los gradientes al cuadrado para corregir el learning rate. Este computa un learning rate particular para distintos parámetros. ADAM utiliza la primera y la segunda estimación del momento del gradiente para adaptar el learning rate para cada peso de la red neuronal.

Parámetro:

- adam_beta1
- adam_beta2

Resultados

En primer lugar, se realizaron varias pruebas para determinar la arquitectura de red neuronal óptima a utilizar. Para dichas pruebas no se implementó ningún algoritmo de mejora, por lo cual no se debe tener en cuenta el error más que para comparar entre estos casos. Se atribuyó el ruido inicial a las correcciones de pesos elegidos al azar; a mayor número de neuronas, mayor número de pesos. Se observó de las *Figuras 1, 2 y 3* qué sucede al variar el número de capas ocultas a una, dos y tres respectivamente para el mismo número de neuronas en cada una (diez). Tomando un máximo de épocas de 500, se determinó que no variaron mucho los resultados. Sin embargo, fue posible apreciar que el error con el que finalizaron las pruebas fue significativamente mayor para una red de una capa oculta (0.0015 vs 0.0008 para dos capas y 0.0004 para tres capas). Si bien para tres capas se logró un mejor error, se pudo observar cómo se logró una disminución más rápida del mismo para dos capas ocultas. En la *Figura 14* se puede observar del promedio de épocas (cuando no hay un límite para las mismas) que una red con dos capas ocultas resultó ser superior al alcanzar valores dentro del margen de error en un número de pasos menor al límite de épocas. Asimismo y en este sentido, se pudo determinar que la función de activación exponencial brindó mejores resultados que la tangente hiperbólica.

En segundo lugar, se compararon los resultados obtenidos al aplicar mejoras al algoritmo de aprendizaje:

- Se modificó el valor del alfa de momentum para distintos valores. En la *Figura 7* se utilizó un valor de 0.5 y se observó que el error disminuye muy lentamente; en la *Figura 8* se utilizó un valor de 0.99 y se observó una rápida caída del error aunque de manera irregular; por último, en la *Figura 6* se fijó el alfa en 0.9 (que suele ser el valor elegido por defecto) y se apreció una suave disminución del error con tiempos promedio.
- Para la mejora de eta adaptativo, se graficó en la *Figura 9* la evolución del valor del mismo y se observó su disminución escalonada a medida que varía el error a través de las épocas. Esto resulta conveniente porque de esta manera podemos comenzar con un eta alto, que rápidamente pueda llevar a los pesos a un estado más balanceado, y una vez que se requiere de un ajuste más fino de los pesos, el eta debería ir decreciendo según sea necesario. La disminución rápida del error, visible en la *Figura 10*, es causada por esta optimización. También se podría observar qué sucede cuando eta es inicializado con un valor pequeño: en este caso, el eta aumentaría escalonadamente. Esto se debe a que, como el error comienza disminuyendo, el eta va creciendo, pero cuando el error crece el eta va decreciendo.
- Al utilizar la mejora ADAM, se observó (*Figura 11*) que en un principio no hubo mucha diferencia con el eta adaptativo. Sin embargo, luego de cierto punto el

error disminuyó rápidamente hasta alcanzar el epsilon y la ejecución, por lo tanto, terminó antes del límite de épocas.

A continuación se comparó ambas implementaciones utilizadas: batch e incremental en la *Figura 12* y *Figura 13* respectivamente. Para ambos casos se utilizó la mejora ADAM con un eta de 0.01, y se pudo observar en primer lugar que el método incremental logró terminar su ejecución a diferencia de batch, que llegó al límite de épocas sin alcanzar valores de error aceptables. Además, se pudo apreciar un mayor ruido en el método incremental debido a que cada patrón trata de sesgar los pesos a su favor en detrimento de los demás. En batch se observa una caída más suave debido a que los pesos de la red cambian utilizando un promedio del conjunto de entrenamiento. Es importante también resaltar que la implementación incremental, a pesar de obtener mejores resultados, tardó más tiempo que batch debido a que debe realizar cálculos de modificación de pesos para cada entrada mientras que batch lo hace una vez por iteración.

Por último, en la *Figura 15* se observan distintos resultados a los que se llegó cuando se varió el tamaño del conjunto de entrenamiento, y cuando se limitó la ejecución a distintos límites de épocas. Como era de esperar, al utilizar más patrones para el entrenamiento de la red los índices de generalización aumentaron al tener más información de la superficie a generar. Además, al dar más tiempo de entrenamiento a la red se consiguieron pesos más precisos lo cual se tradujo en mejores resultados. En la *Figura 16* y *Figura 17* se puede efectivamente comprobar que la superficie generada cuando el índice de generalización alcanzó un valor de 87% resultó ser muy parecida a la superficie descrita por el conjunto de datos provistos, y en la *Figura 18* se observaron qué áreas de la misma tuvieron las mejores aproximaciones.

En la *Figura 19* se puede observar un planteo similar comparando los resultados obtenidos de la generalización al utilizar distintas implementaciones. Como pudo observarse, con un método incremental se consiguieron representaciones más exactas.

Conclusiones

Tomando en cuenta los resultados obtenidos en la ejecución del proyecto, se pueden obtener múltiples conclusiones.

En primer lugar, se decidió que dos capas ocultas en la red neuronal dieron mejores resultados que una sola al disminuir significativamente el error obtenido y al alcanzar salidas dentro del margen de error en un menor número de iteraciones, debido al hecho de que resultó proveer una mejor aproximación a la capacidad del modelo desarrollado. A una red con menos capas ocultas de las suficientes se le dificulta aceptar un conjunto de entrenamiento, mientras que si se implementan más capas de las necesarias se pueden llegar a “memorizar” características del conjunto de prueba que no sean útiles a la hora de generalizar. Se decidió que 10 neuronas por capa oculta era la cantidad suficiente al observar que con un número menor y mayor de neuronas el error tardaba mayor cantidad de tiempo en disminuir. Esto se debió a que una capa muy pequeña resulta en un menor número de pesos y adaptar el conjunto de datos al mismo puede ser una tarea ardua; una capa muy grande disminuye la capacidad de generalizar.

Asimismo, se pudo asegurar que aplicar mejoras al algoritmo de aprendizaje efectivamente resultó positivo. Para momentum, se verificó que un valor de alfa de 0.9 es adecuado según lo propuesto por defecto. Además, el eta adaptativo resultó ser más eficiente en cuanto a una reducción de los errores. Pero sin embargo se determinó que ADAM resultó ser la mejor implementación dado que, a diferencia del resto y en igualdad de condiciones, logró finalizar la ejecución alcanzando valores de salida dentro del margen de error y luego de un número pequeño de épocas.

Finalmente, se realizó un balance general al comparar la cantidad de datos a utilizar en la etapa de entrenamiento con los parámetros tiempo y precisión. Utilizando un alto porcentaje de patrones para el conjunto de entrenamiento se llegó, lógicamente, a una mejor generalización. Sin embargo, el tiempo requerido para dicha tarea fue muy superior al deseado. Al contrario, al utilizar un bajo porcentaje de entradas se lograron tiempos muy acotados pero con generalizaciones pobres. Resulta entonces de gran importancia poder elegir un porcentaje adecuado que implique un balance entre ambos aspectos.

Anexo

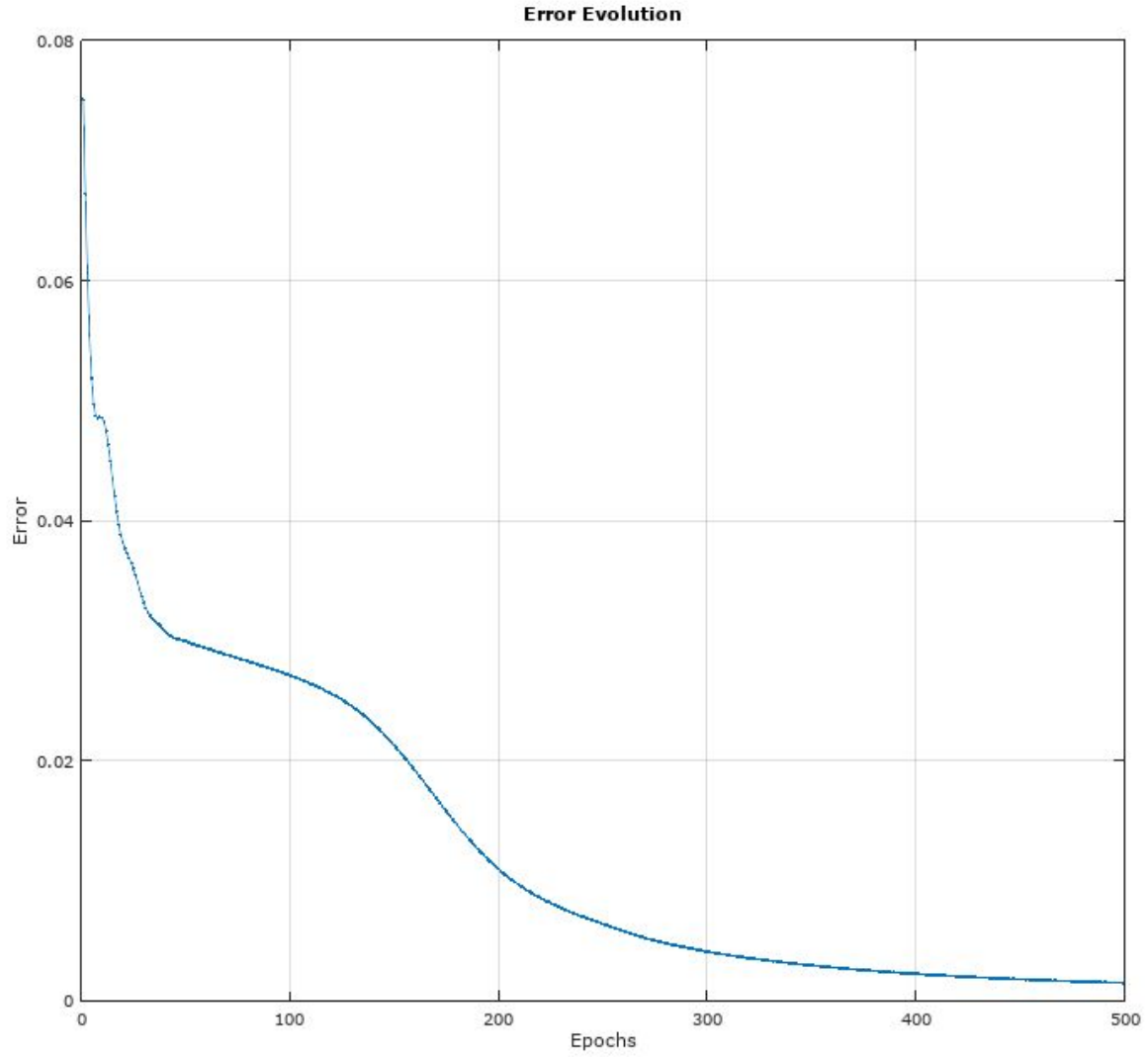


Figura 1: Gráfico de error a través de épocas para una capa oculta de 10 neuronas cada una (batch, eta 0.01, exponencial, sin ADAM)

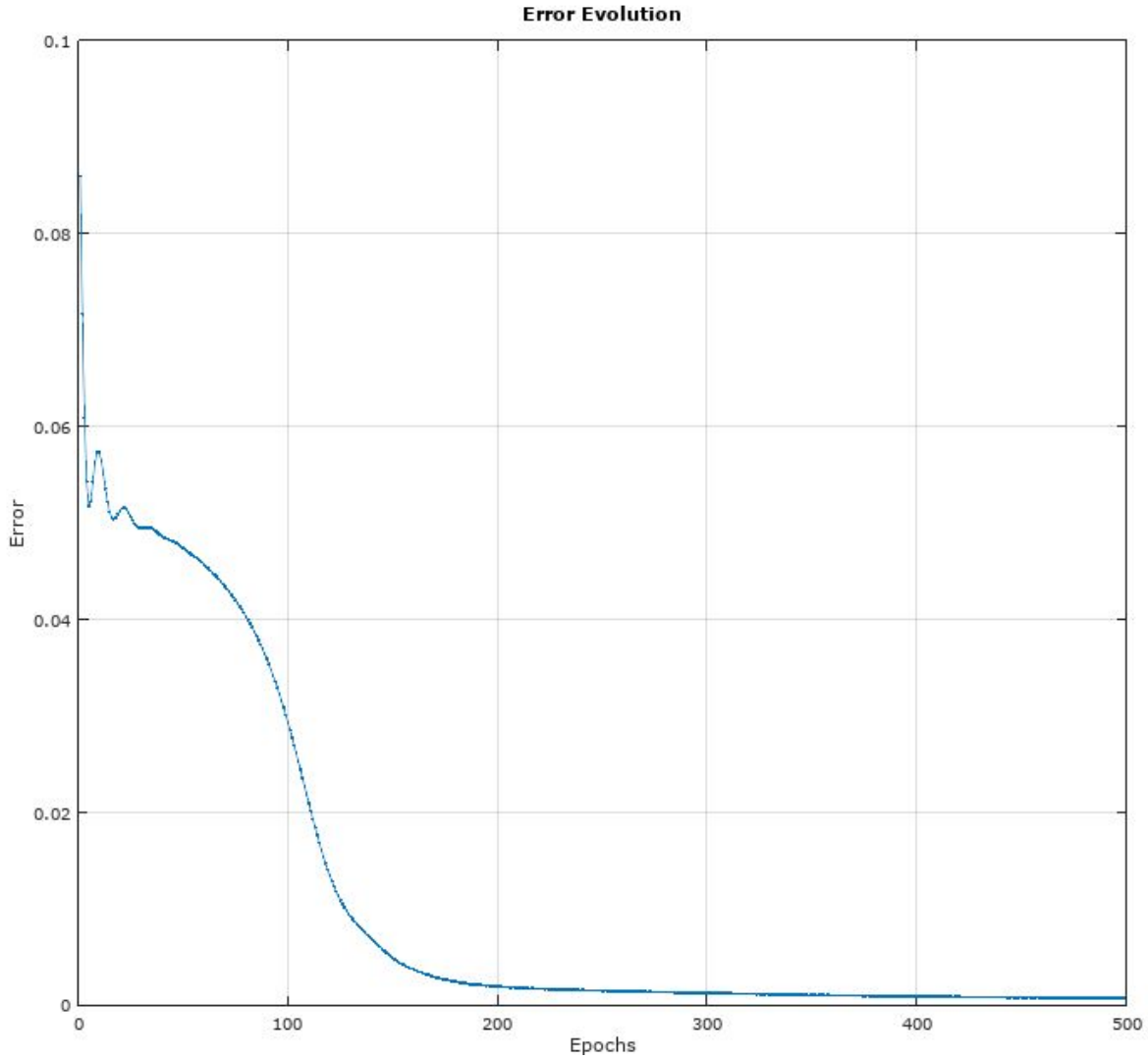


Figura 2: Gráfico de error a través de épocas para dos capas ocultas de 10 neuronas cada una (batch, eta 0.01, exponencial, ADAM)

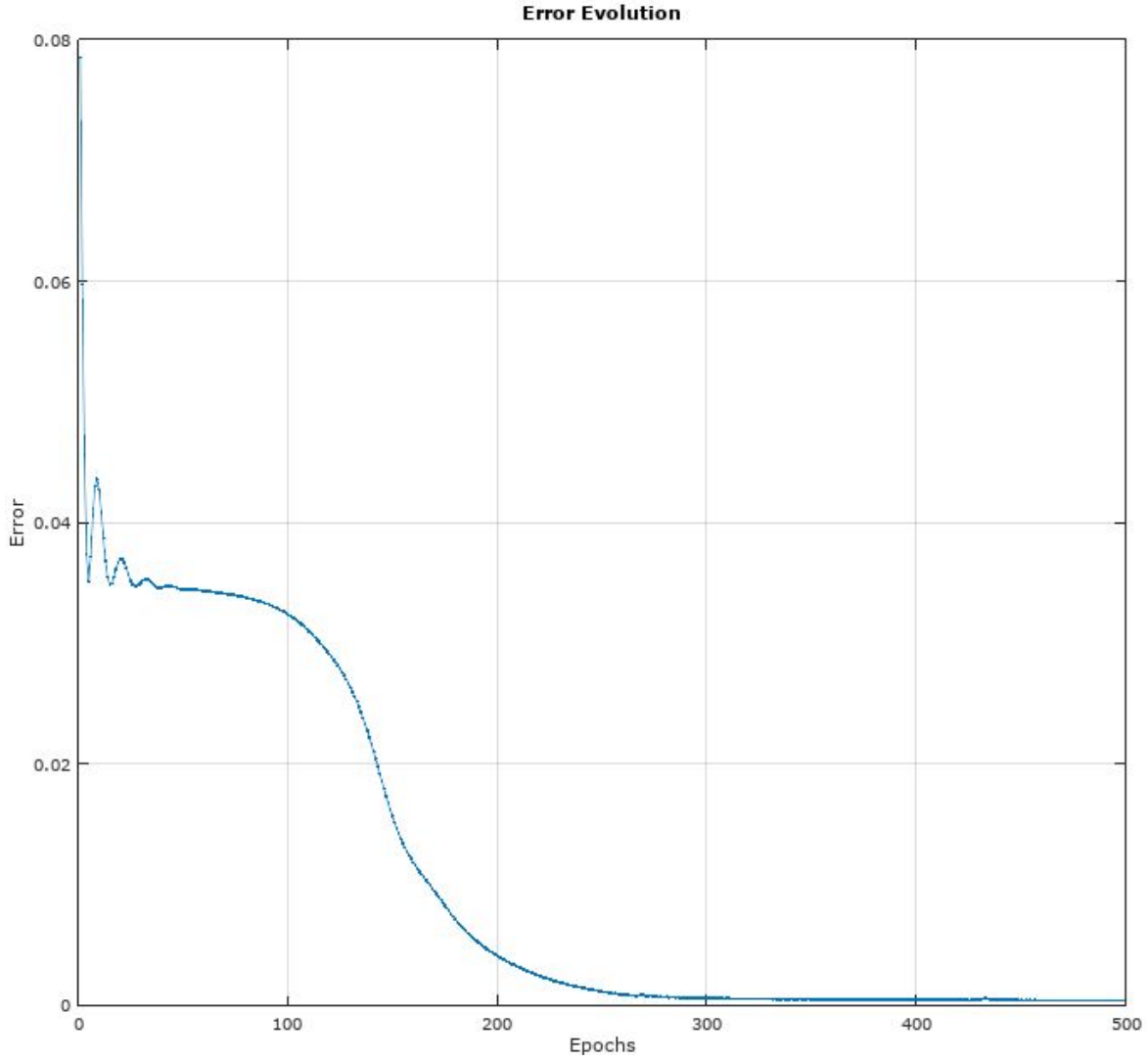


Figura 3: Gráfico de error a través de épocas para tres capas ocultas de 10 neuronas cada una (batch, eta 0.01, tangente hiperbólica, ADAM)

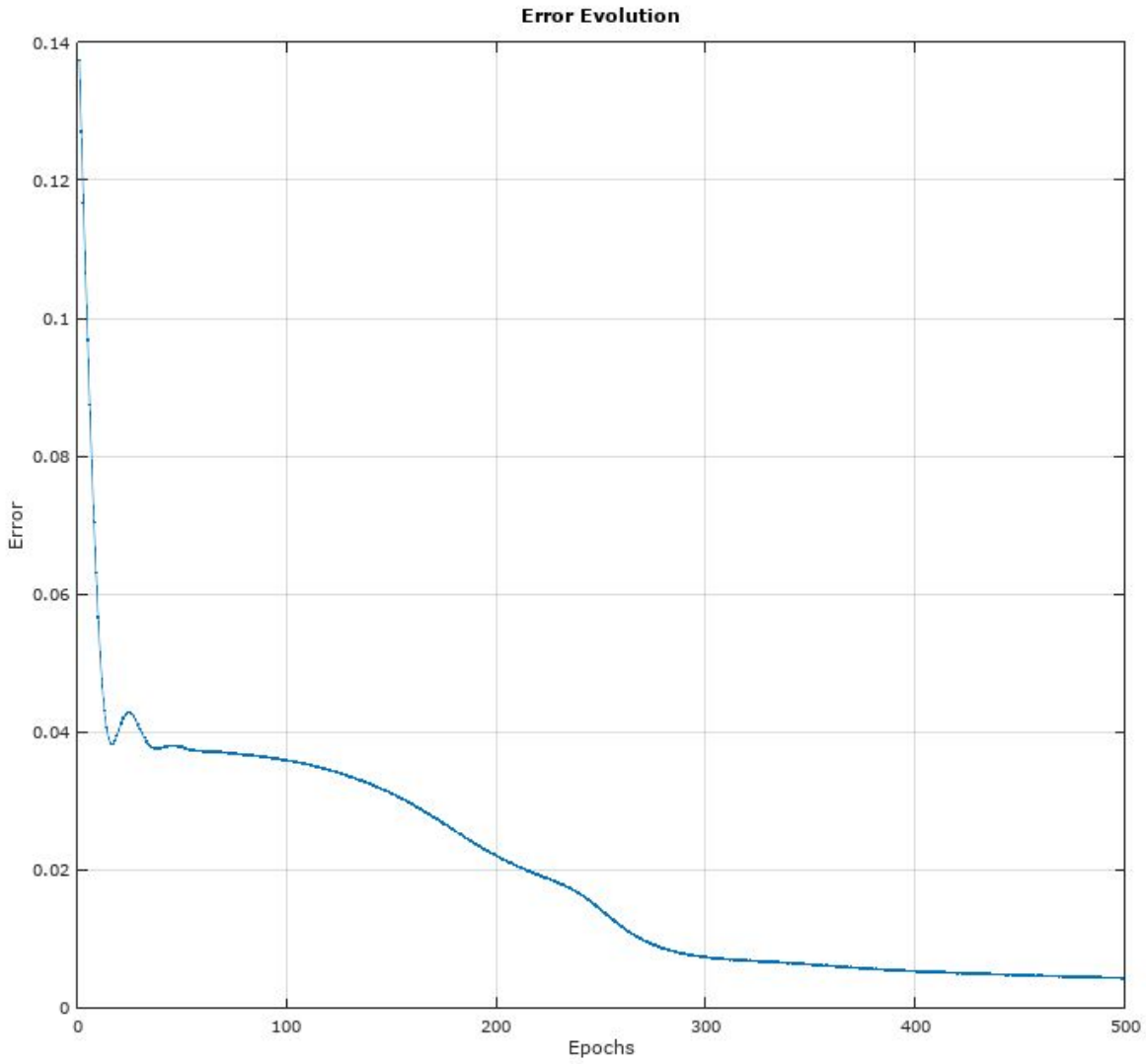


Figura 4: Gráfico de error a través de épocas para dos capas ocultas de 5 neuronas cada una (eta 0.01, tangente hiperbólica, ADAM)

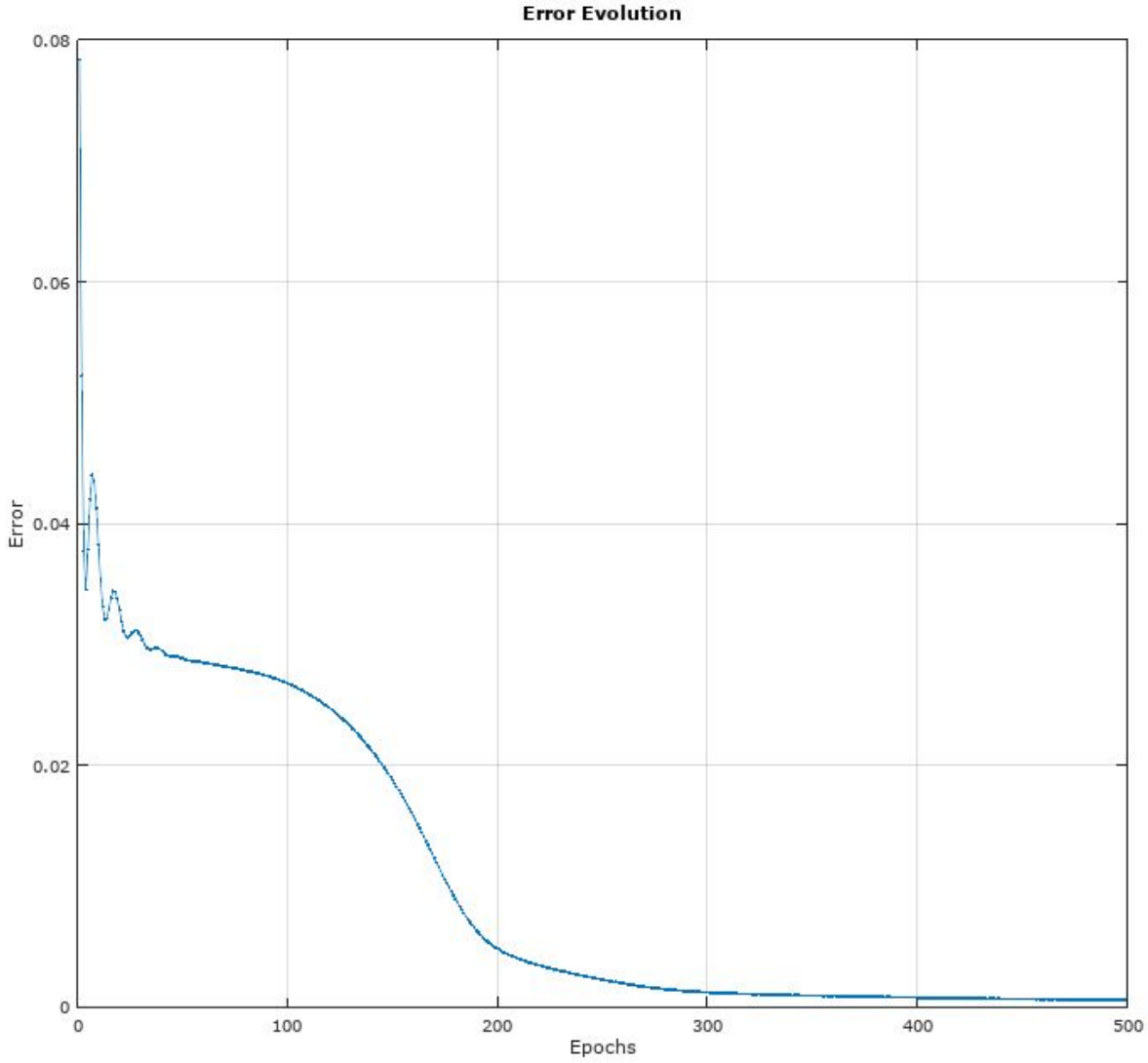


Figura 5: Gráfico de error a través de épocas para dos capas ocultas de 15 neuronas cada una (eta 0.01, tangente hiperbólica, ADAM)

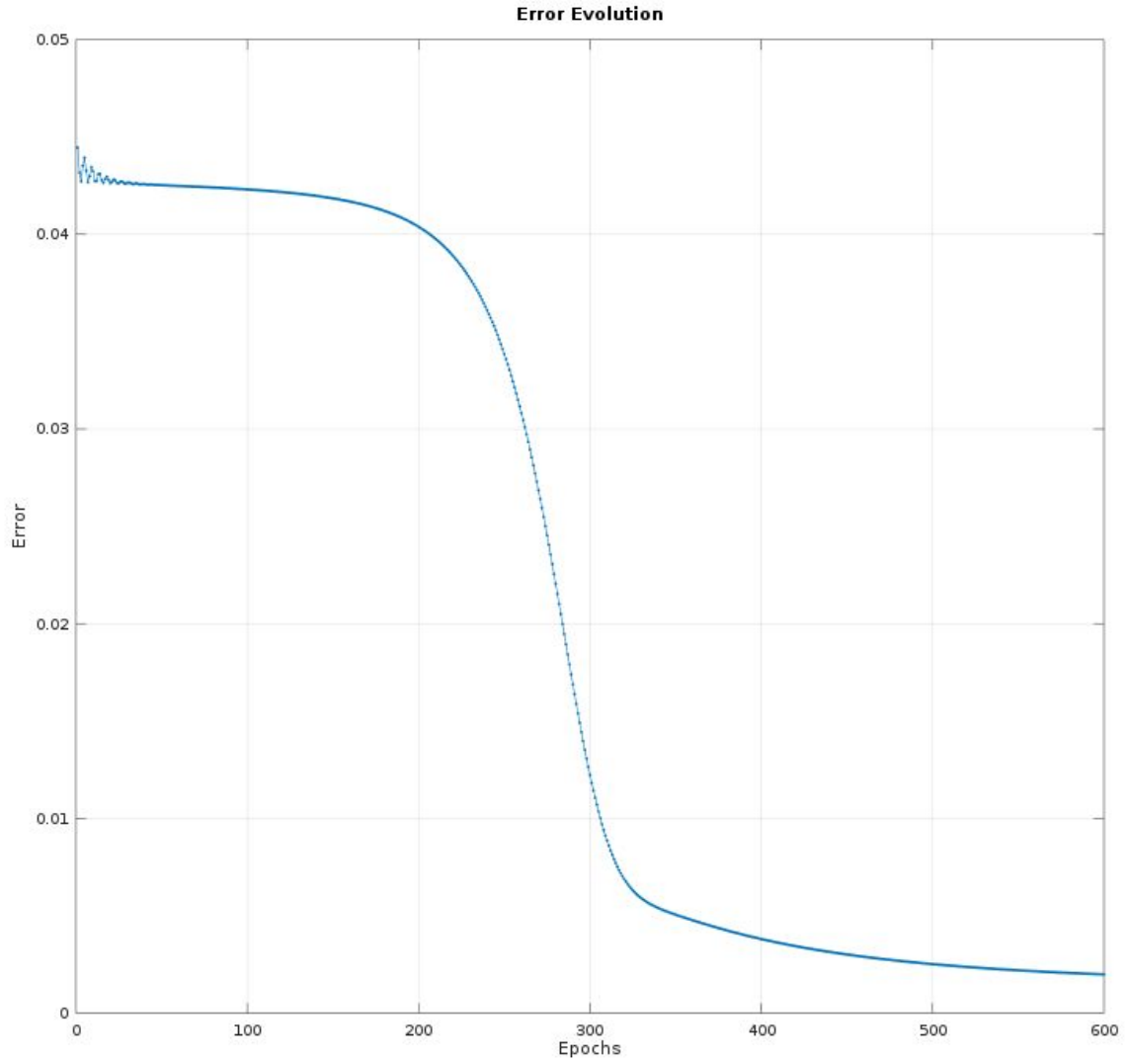


Figura 6: Gráfico de error a través de épocas para dos capas ocultas de 10 neuronas cada una (eta 0.01, exponencial, momentum_alpha = 0.9, sin eta adaptativo) (Batch)

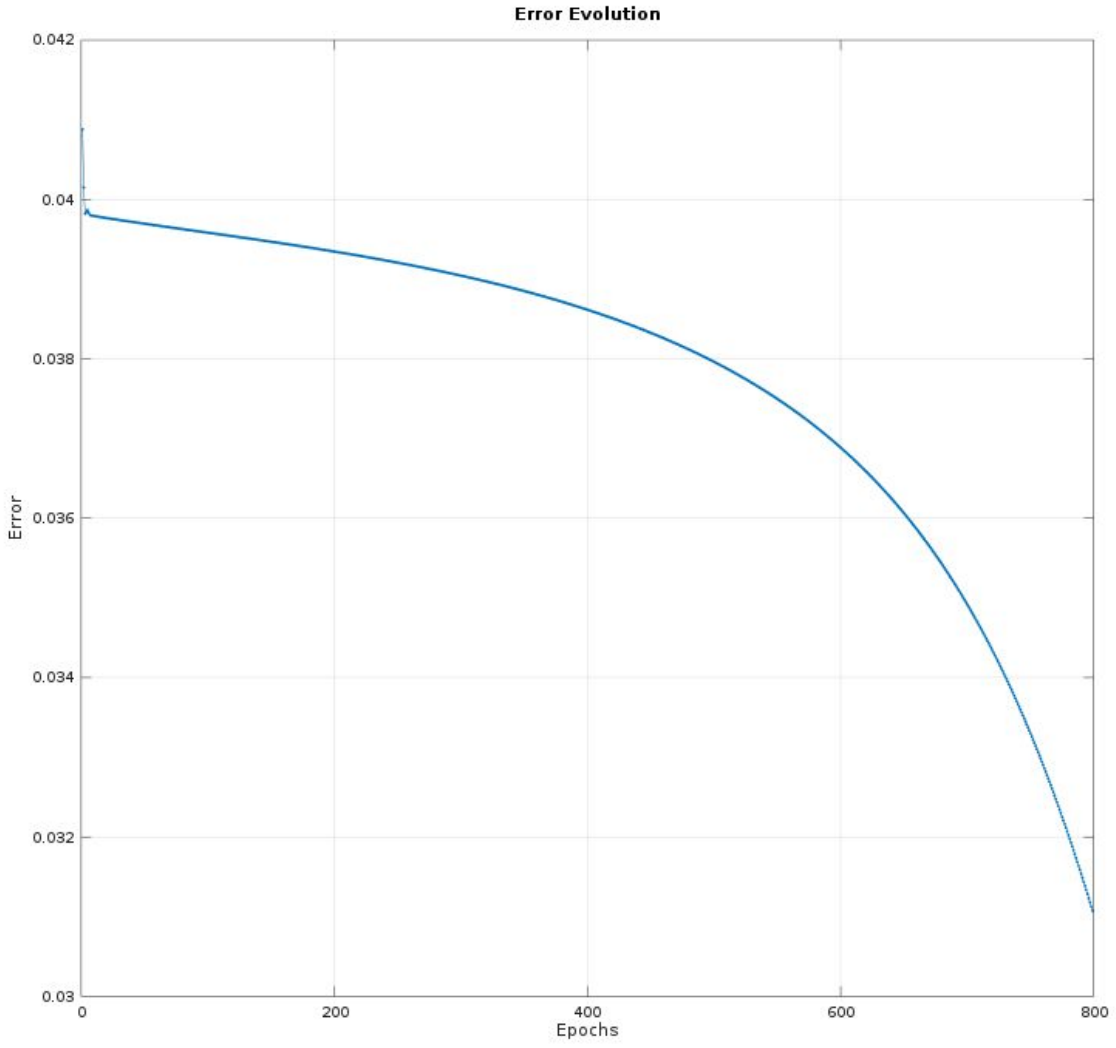


Figura 7: Gráfico de error a través de épocas para dos capas ocultas de 10 neuronas cada una (eta 0.01, exponencial, momentum_alpha = 0.5, sin eta adaptativo) (Batch)

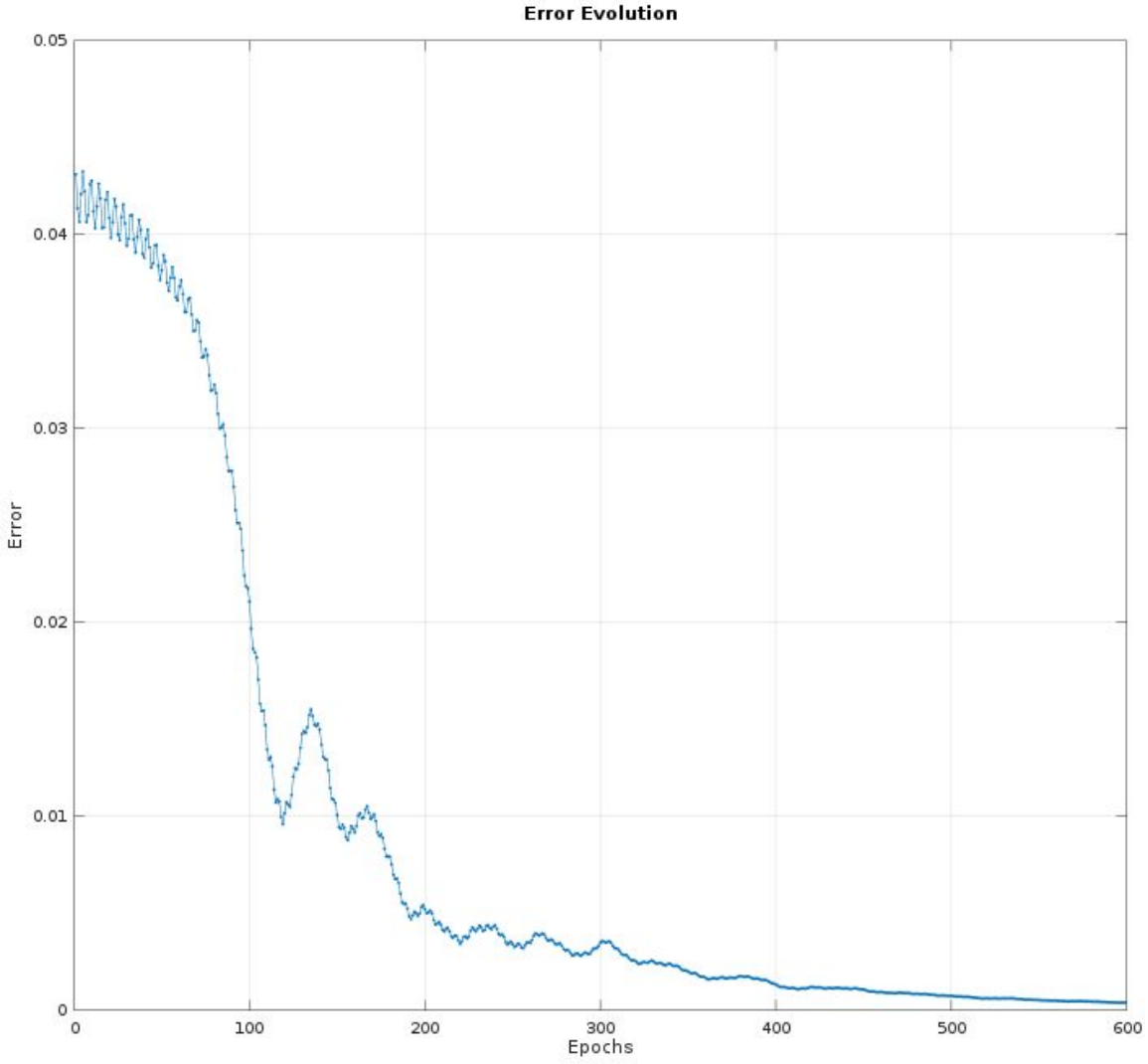


Figura 8: Gráfico de error a través de épocas para dos capas ocultas de 10 neuronas cada una (eta 0.01, exponencial, momentum_alpha = 0.99, sin eta adaptativo) (Batch)

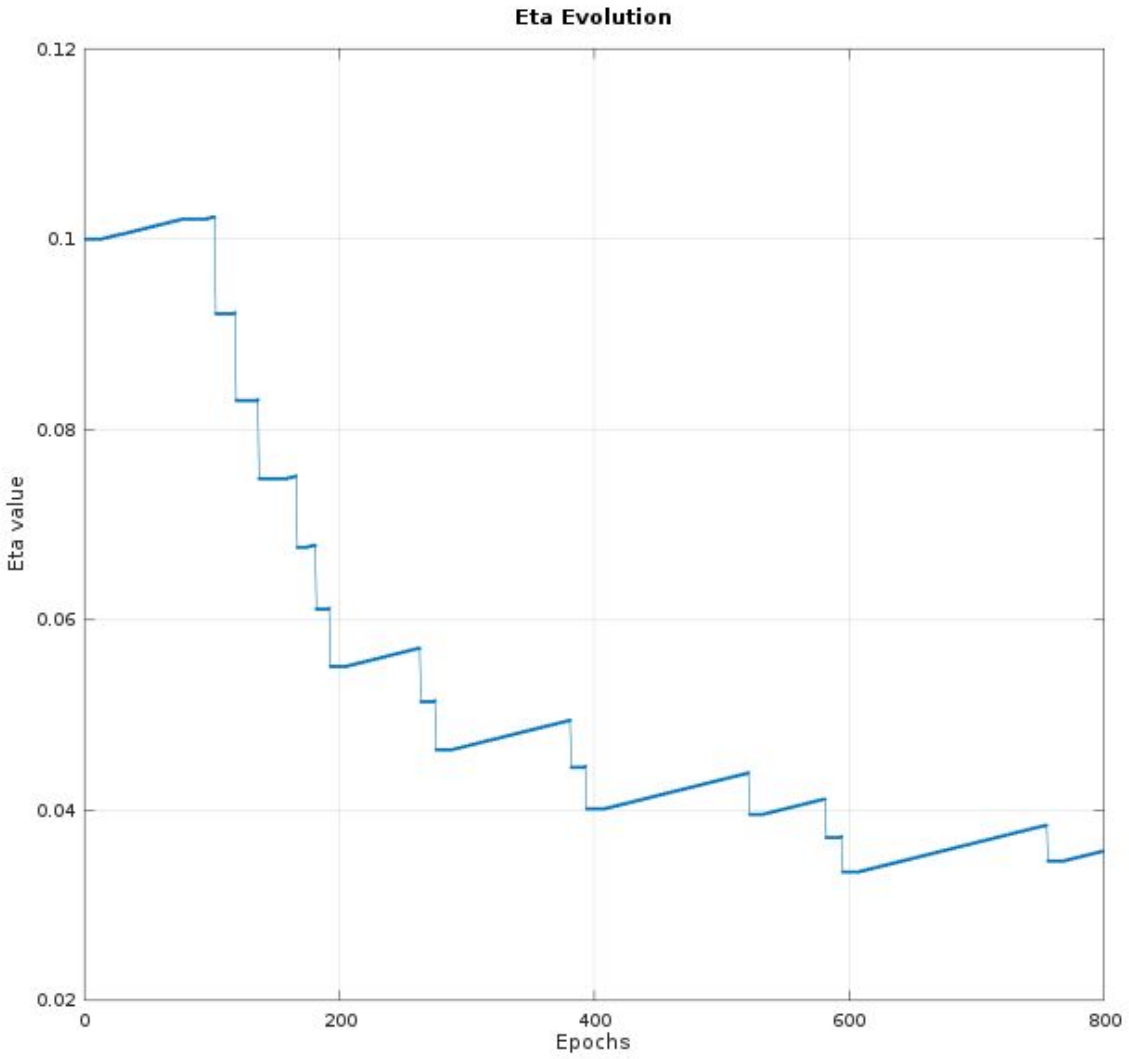


Figura 9: Gráfico de eta a través de épocas para dos capas ocultas de 10 neuronas cada una (eta 0.1, exponencial, momentum_alpha = 0.9, eta adaptativo con a = 0.0001 y b = 0.1, y epoch_min_reduction_steps = 3) (Batch)

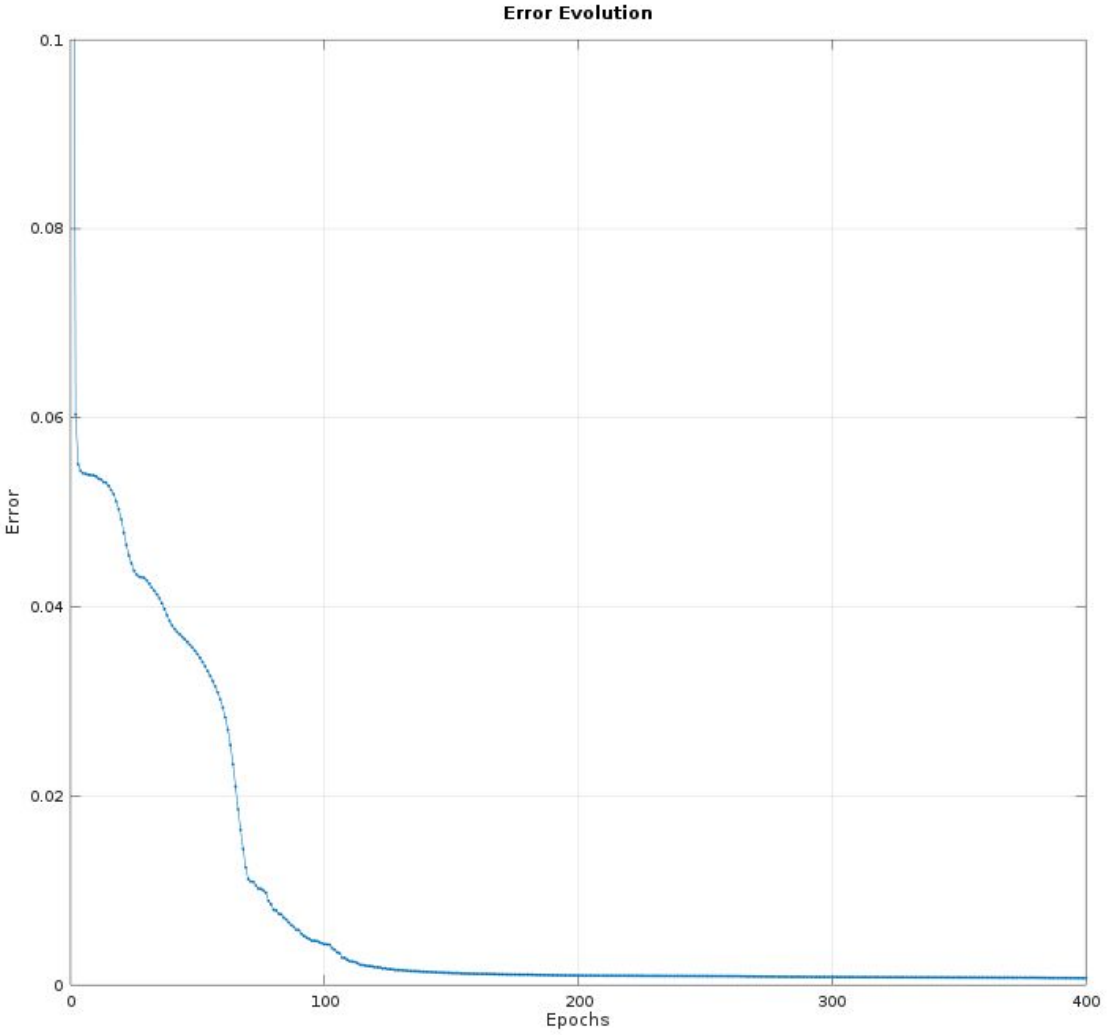


Figura 10: Gráfico de error a través de épocas para dos capas ocultas de 10 neuronas cada una (eta 0.1, exponencial, momentum_alpha = 0.9, eta adaptativo con a = 0.0001 y b = 0.1, y epoch_min_reduction_steps = 3) (pertenece a la ejecución de la figura 9) (Batch)

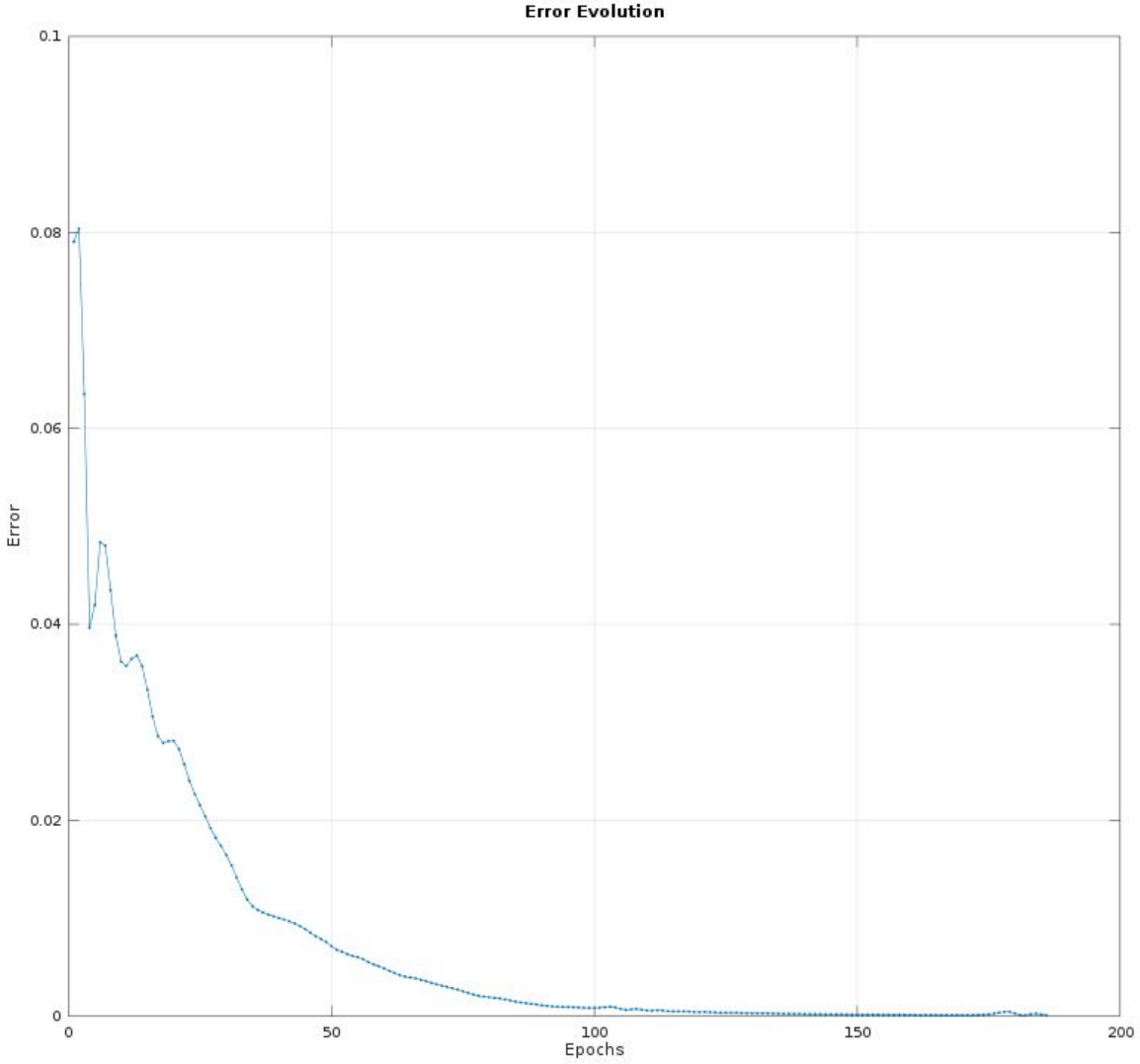


Figura 11: Gráfico de eta a través de épocas para dos capas ocultas de 10 neuronas cada una (eta 0.1, exponencial, sin eta adaptativo, adam com beta1 = 0.9, beta2 = 0.999, adam_epsilon = 1e-8) (Batch)

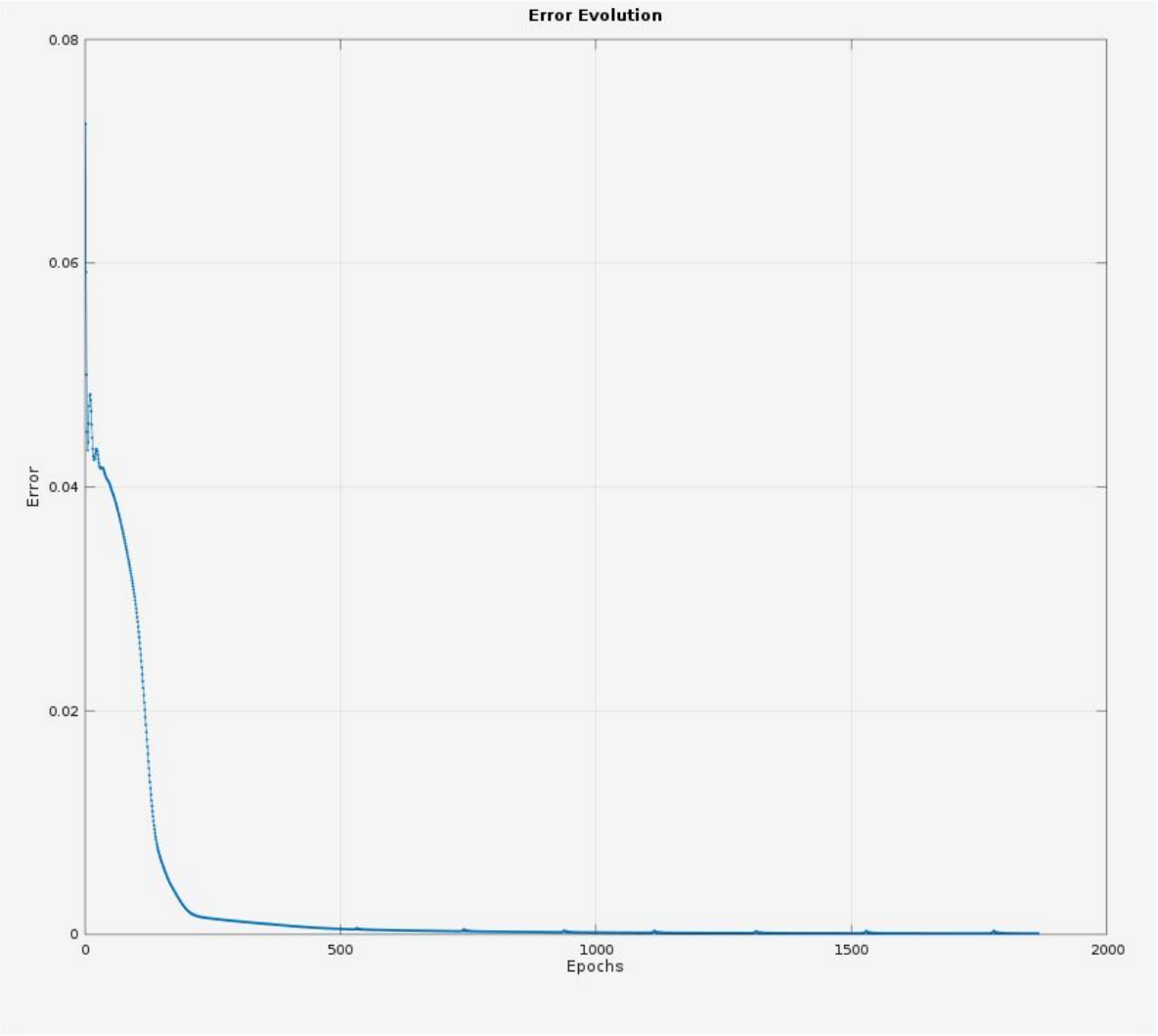


Figura 12: Gráfico de eta a través de épocas para dos capas ocultas de 10 neuronas cada una (eta 0.01, exponencial, sin eta adaptativo, adam com beta1 = 0.9, beta2 = 0.999, adam_epsilon = 1e-8) (Batch) → 1867 epocas

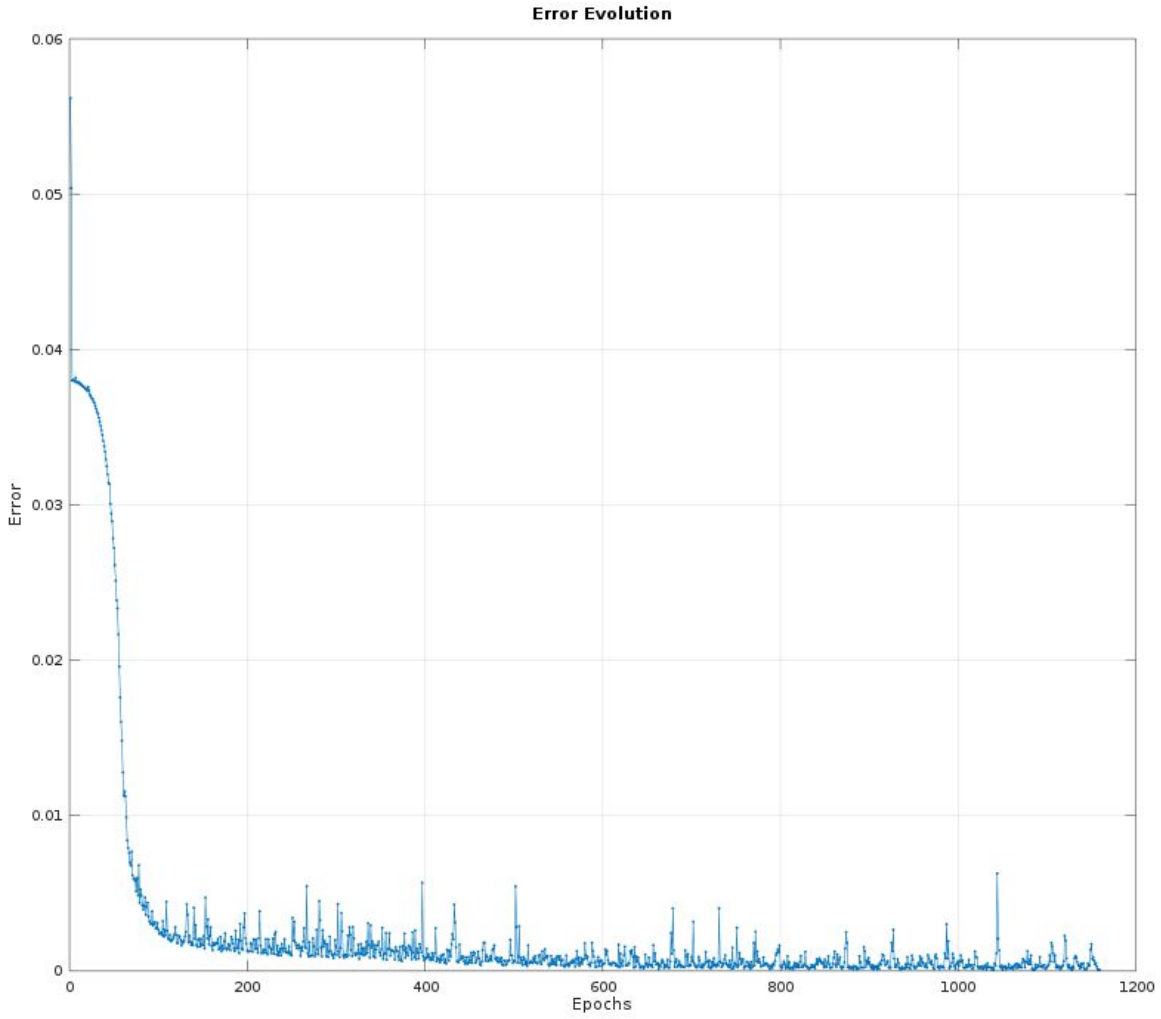


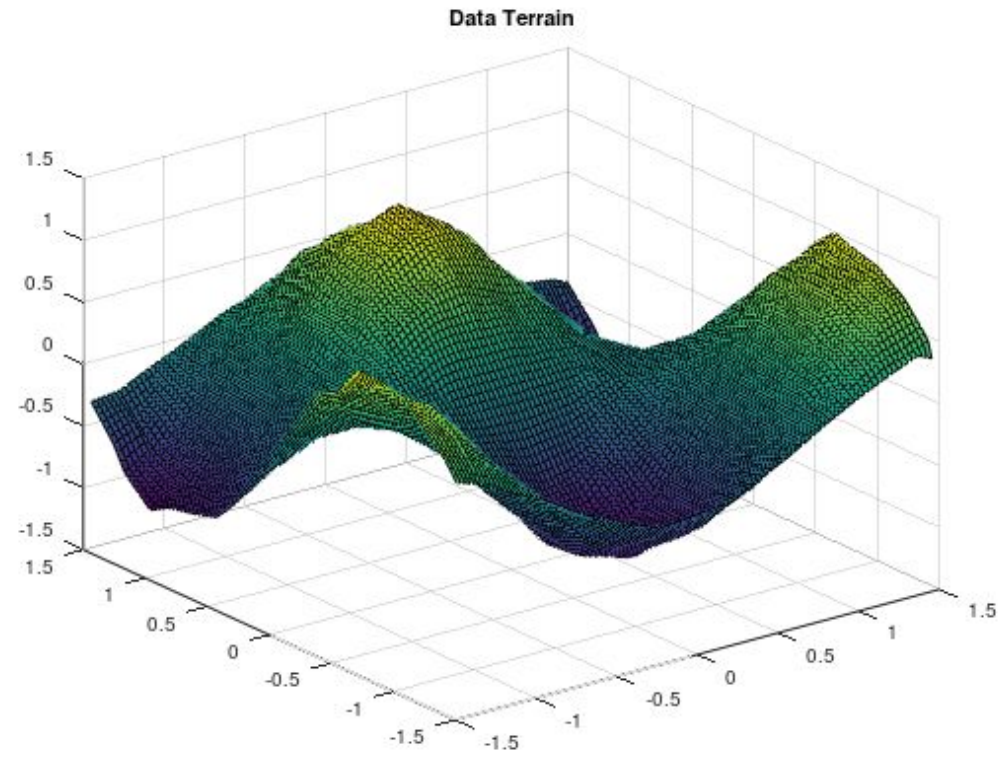
Figura 13: Gráfico de eta a través de épocas para dos capas ocultas de 10 neuronas cada una (eta 0.01, exponencial, sin eta adaptativo, adam com beta1 = 0.9, beta2 = 0.999, adam_epsilon = 1e-8) (Incremental) → 1160 epocas

	Promedio de épocas	
Número de capas ocultas	**Función exponencial**	**Función tangente hiperbólica**
1	2186	> 40k
2	756	1231
3	1543	1704

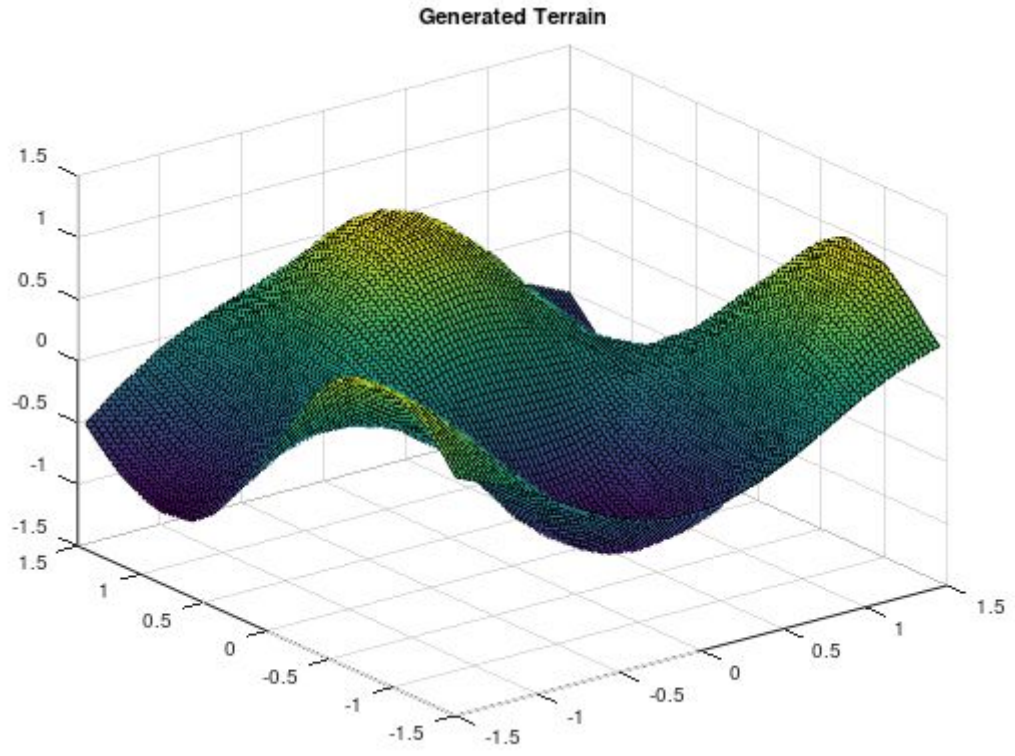
Figura 14: Comparación del promedio de épocas para distintas funciones de activación y distinto número de capas ocultas

% Generalización (utilizando 0.05 de epsilon para considerar correctitud de salidas)		Épocas		
		500	**2000**	**5000**
Porcentaje de datos usados para entrenamiento	**0.1**	34.92%	40.14%	45.45%
	0.25	51.92%	71.65%	77.10%
	0.4	57.14%	74.38%	87.07%

Figura 15: Parámetros: Batch training, ETA = 0.01 con Adam y función exponencial, 2 capas de 12.



Figuras 16 y 17: Terreno descrito por el set de datos vs generado (5000 épocas, 87.07% con 40% de los datos totales para entrenar)



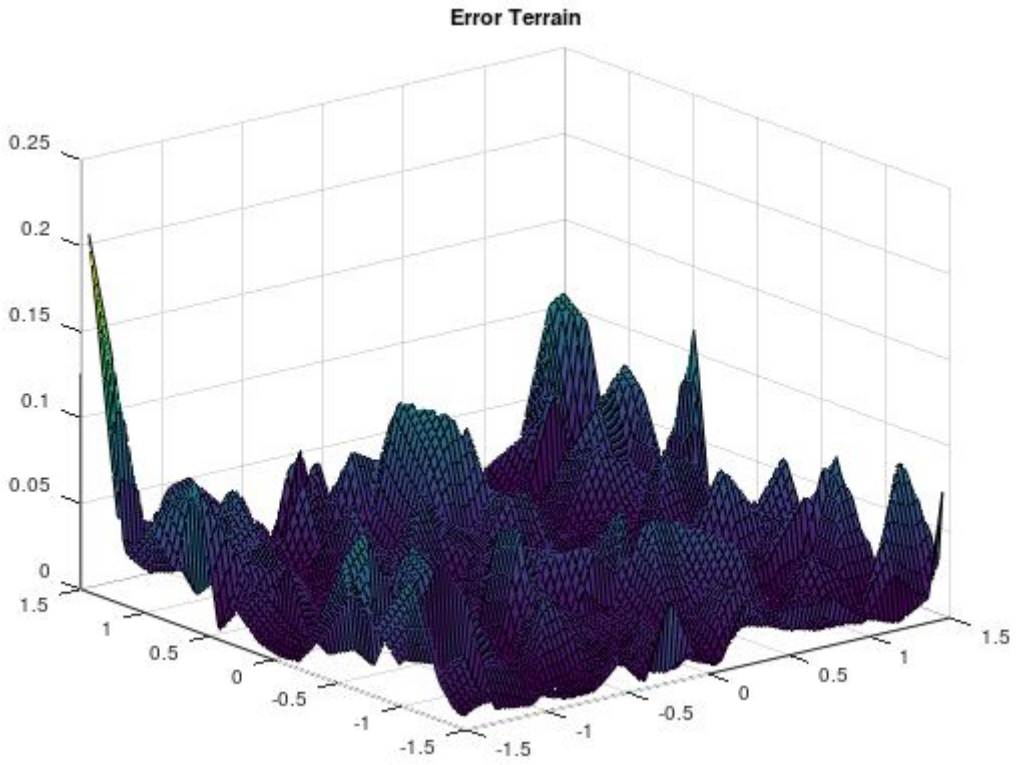
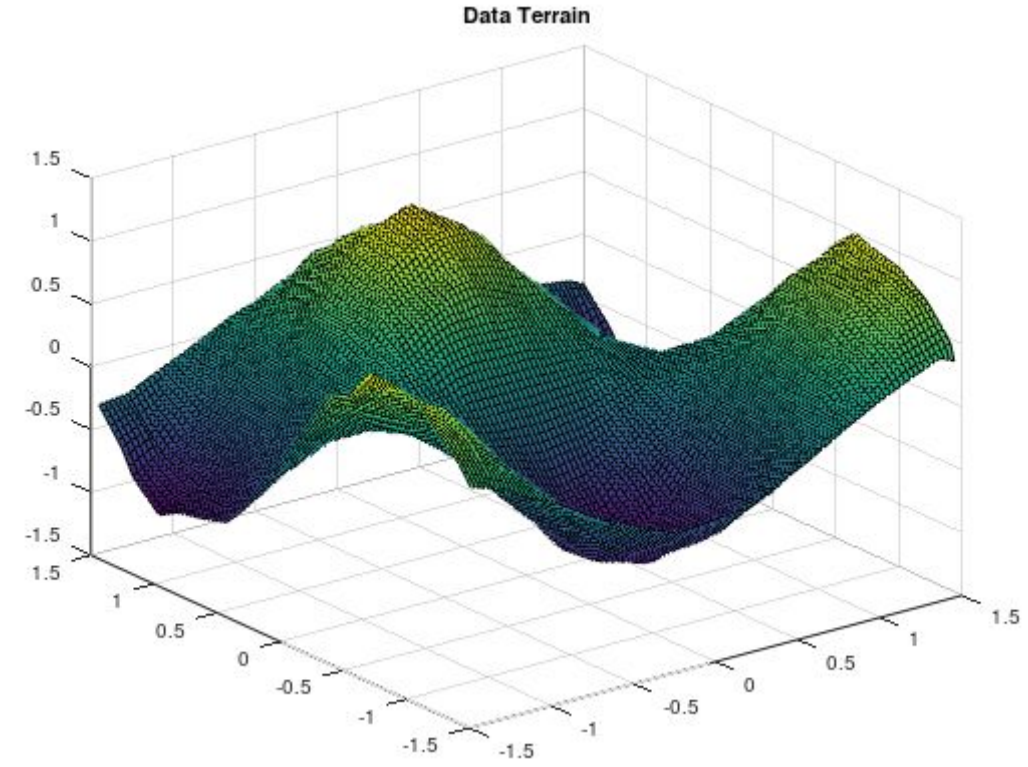


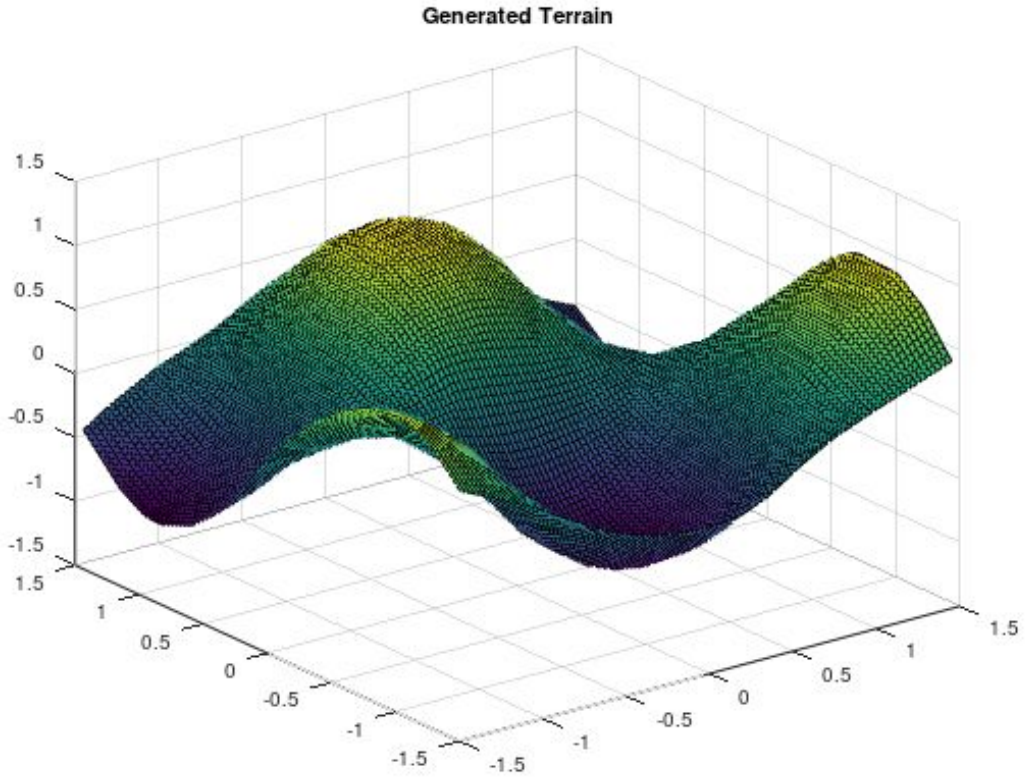
Figura 18: Representación tridimensional del error para cada coordenada (5000 épocas, 87.07% generalización con 40% de los datos totales para entrenar)

% Generalización (utilizando 0.05 de epsilon para considerar correctitud de salidas)		**Épocas**	
		500 (batch)	**500 (incremental)**
Porcentaje de datos usados para entrenamiento	**0.1**	34.92%	41.72%
	0.25	51.92%	64.62%
	0.4	57.14%	78.01%

Figura 19: Parámetros: ETA = 0.01 con Adam y función exponencial, 2 capas de 12.



Figuras 20 y 21: Terreno descrito por el set de datos vs generado (500 épocas, 78.01% con 40% de los datos totales para entrenar)



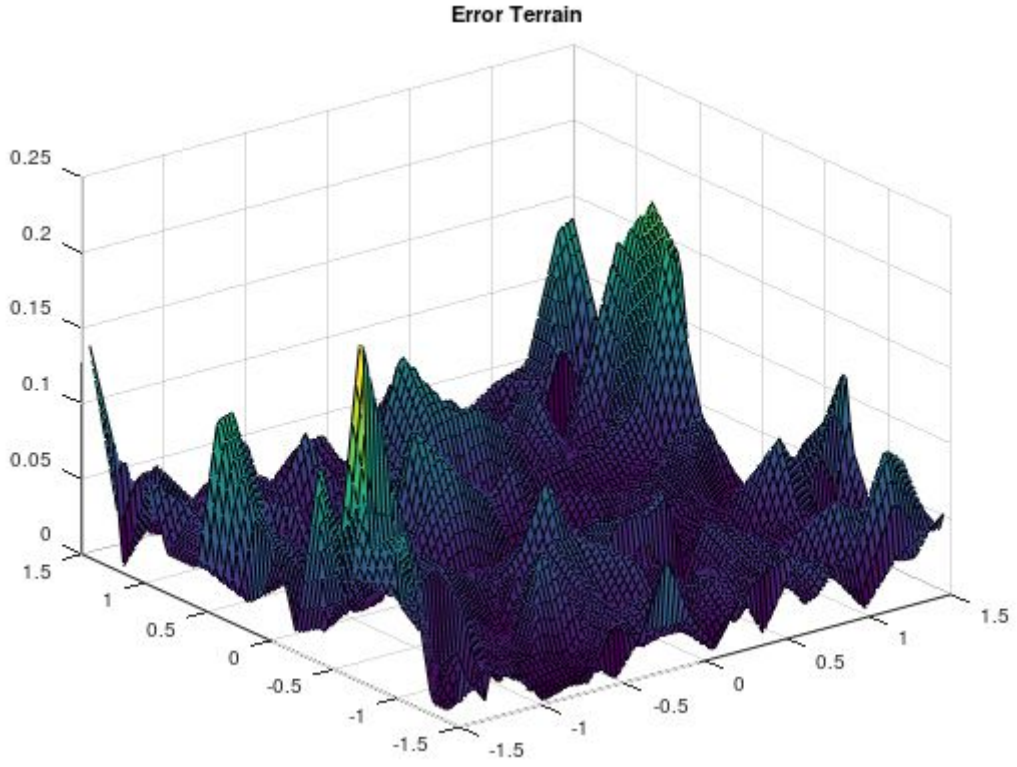


Figura 22: Representación tridimensional del error para cada coordenada (500 épocas, 78.01% generalización con 40% de los datos para entrenar)