Trabajo Práctico Especial 2

Redes Neuronales

Grupo 8

Colloca, Tomás

López, Noelia Belén

Suárez Bodean, Joaquín

Vera, Juan Sebastián

Índice

Introducción

Se desarrolló una red neuronal en la que se permite configurar la cantidad de capas ocultas y de neuronas por capa, para que, dado un conjunto de entrenamiento y otro de testeo, permite aprender el primero y luego generalizar el segundo utilizando el algoritmo *backpropagation.*

Para ello, se alimenta a la red con los distintos patrones del conjunto de entrenamiento con el algoritmo *forward* en forma *batch* sucesivas veces llamadas épocas. Cada cierta cantidad de épocas, configurable también, se calcula el error de entrenamiento y de testeo y en función de estos se decide si la red ha aprendido o está sobre-aprendiendo, en cuyo caso se pierde la capacidad de generalizar los patrones del conjunto de testeo.

Modelo

Se creó una clase que representa la red neuronal. El estado de la misma está conformado por las matrices de pesos de las distintas capas, el *learning rate*, los patrones a usar para entrenamiento y testeo, y otros valores utilizados para llevar a cabo los algoritmos forward y back propagation.

La red neuronal conoce dos funciones:

- tanh: $tanh(\beta x)$
- sigmoid: $\frac{1}{1+e^{-2\beta x}}$

Y las respectivas derivadas escritas en función de la imagen de dichas funciones.

La red define a partir de los patrones de aprendizaje una función para normalizar los patrones de entrada, y otra para normalizar las salidas esperadas de dichos patrones. La función que normaliza los patrones de entrada se aplica a los patrones de entrenamiento y testeo, y la que normaliza la salida se aplica a los ambos tipos de patrones para calcular el error. Por otro lado, para poder visualizar los valores de salida y graficarlos se usa la inversa de la función que normaliza las salidas.

La función de normalización en ambos casos es una función lineal que mapea el mínimo y el máximo valor de entrada de los patrones de aprendizaje a -1 y 1 respectivamente, el mínimo y máximo valor de la salida a 0.1 y 0.9 para la función exponencial, y a -0.8 y 0.8 para tanh. Al elegir estos valores no se está agregando una cota a la salida.

Para aprender, la red corre un conjunto de patrones de entrenamiento ordenados aleatoriamente para cada época, y calcula cuánto cambiar cada peso cada uno de estos de la red. Al final de la época se aplican dichos cambios a cada peso (es decir, el entrenamiento es *batch*). A su vez, en función de los parámetros de configuración, decide si modificar el *learning rate*, η, o si aplicar *momentum*, de la misma forma que se vio en clase.

El delta de error utilizado para ajustar el *learning rate* es el error cuadrático medio actual menos el error cuadrático medio de la época anterior; ambos normalizados por la cantidad de patrones de aprendizaje. Para el cálculo del error de testeo se normalizó dividiendo por la cantidad de patrones de testeo.

Para la variación donde el *learning rate* se ajusta a medida que se progresa con el aprendizaje, se agregó una variable *p* que define la probabilidad de volver hacia atrás antes de cambiar el valor de η si es que el delta de error dio mayor a cero. Por otro lado, en el caso de combinar dicha variación con *momentum*, si el delta de error da mayor a cero, no se aplicará esta segunda variación temporalmente.

El aprendizaje de la red se lleva a cabo con los algoritmos *forward* y *back propagation* vistos en clase, y son ejecutados de forma matricial/vectorial.

Resultados

Todos los resultados se corrieron con los mismos conjunto de entrenamiento y testeo, ambos generados dividiendo el archivo con los datos del terreno por la mitad aleatoriamente.

Arquitecturas

1. Ninguna capa oculta.
2. Una capa oculta de 10 neuronas.
3. Una capa oculta de 30 neuronas.
4. Dos capas ocultas de 10 neuronas cada una.
5. Dos capas ocultas de 30 neuronas cada una.
6. Tres capas ocultas de 20 neuronas cada una.
7. Cinco capas ocultas de 10 neuronas cada una.

Los siguientes gráficos de error se corrieron con los siguientes parámetros:

$$g(h): \ tanh(\beta h), \ con \ \beta = 1$$
$$\eta_i = 0,0001$$

Se utilizaron la siguientes variaciones:

- Variación que suma 0.1 a la derivada.
- Etha adaptativo con a = 5e-7, b = 0.7, *p* (probabilidad de deshacer) = 0.75 y actualizando etha cada 2 épocas.
- Momentum con $\alpha = 0.9$.

Además, todos se encuentran con escala logarítmica base 10.

Figura 1:

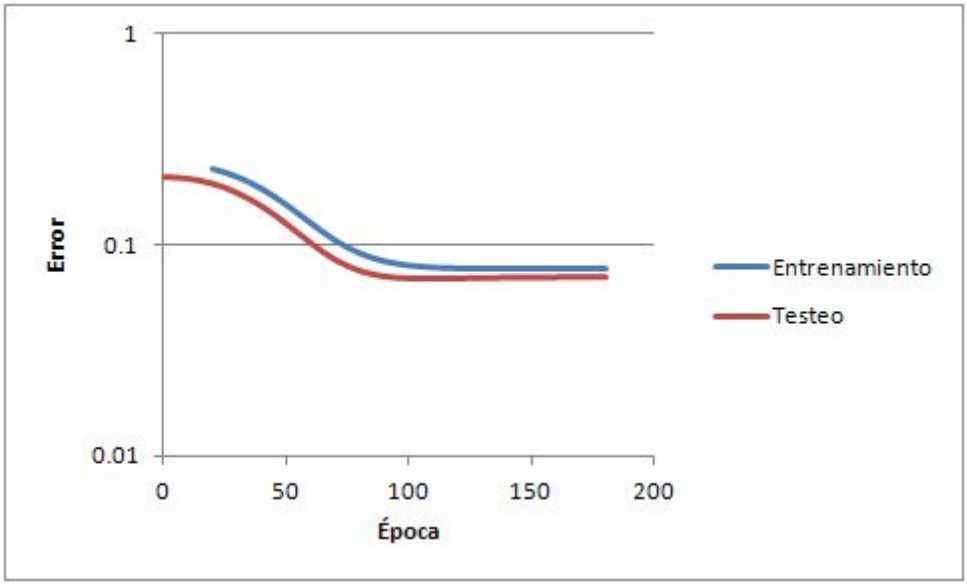

Figura 1: Error de entrenamiento y de testeo en función de las épocas de entrenamiento de la red para la arquitectura 1.

Figura 2:

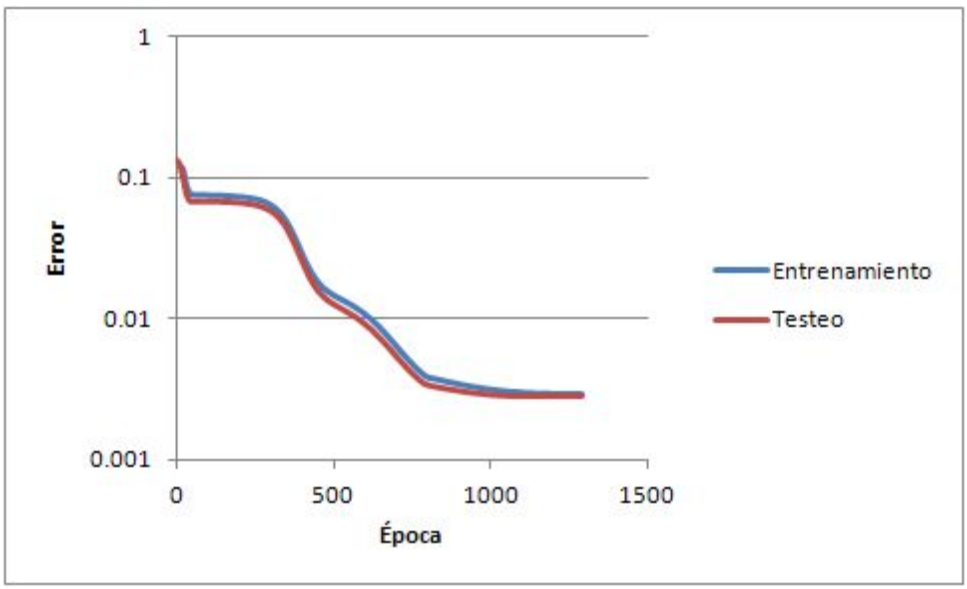

Figura 2: Error de entrenamiento y de testeo en función de las épocas de entrenamiento de la red para la arquitectura 2.

Figura 3:

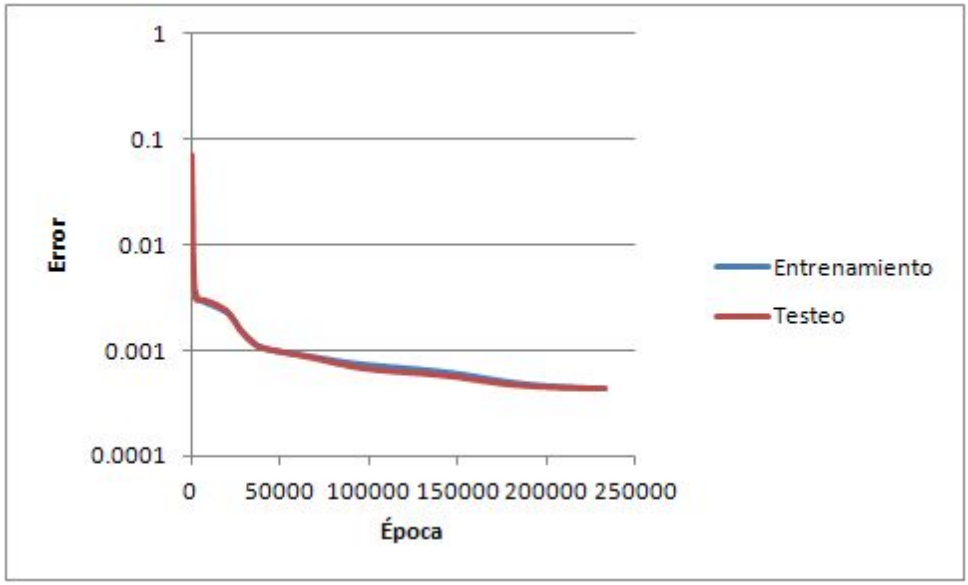

Figura 3: Error de entrenamiento y de testeo en función de las épocas de entrenamiento de la red para la arquitectura 3.

Figura 4:

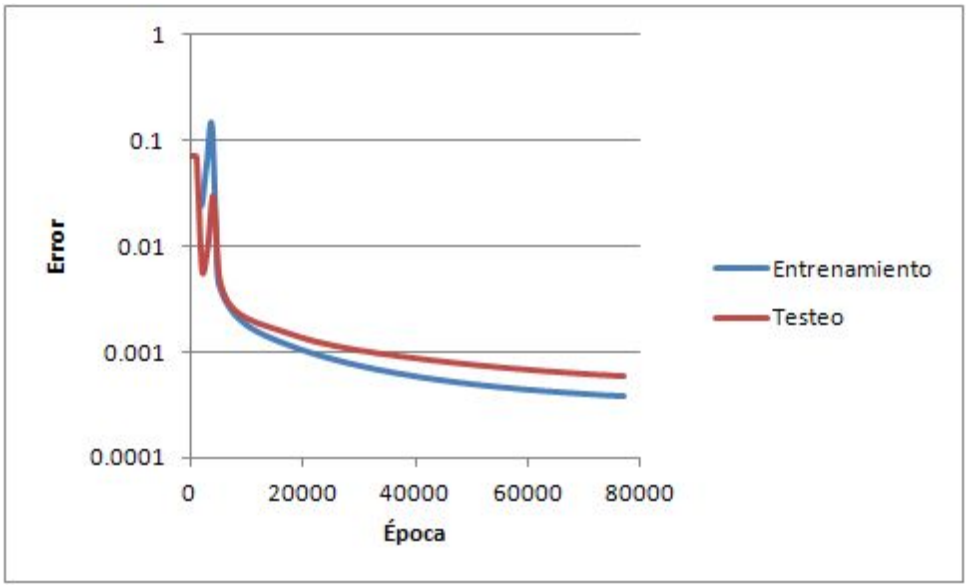

Figura 4: Error de entrenamiento y de testeo en función de las épocas de entrenamiento de la red para la arquitectura 4.

Figura 5:

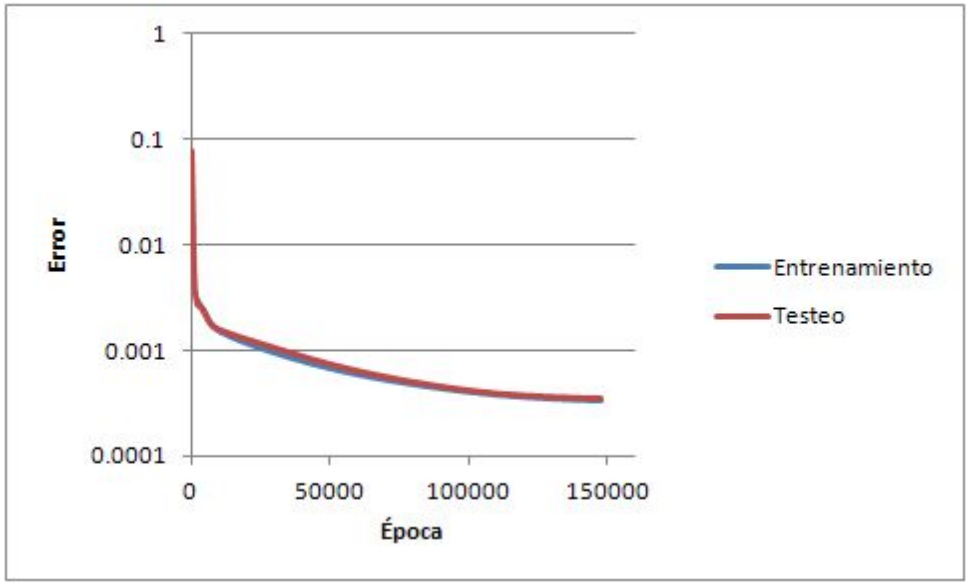

Figura 5: Error de entrenamiento y de testeo en función de las épocas de entrenamiento de la red para la arquitectura 5.

Figura 6:

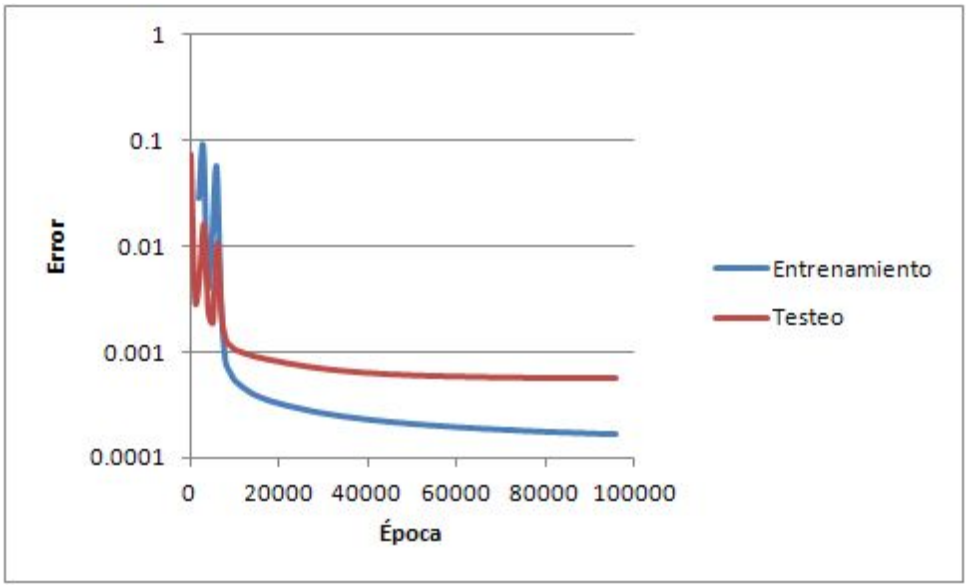

Figura 6: Error de entrenamiento y de testeo en función de las épocas de entrenamiento de la red para la arquitectura 6.

Figura 7:

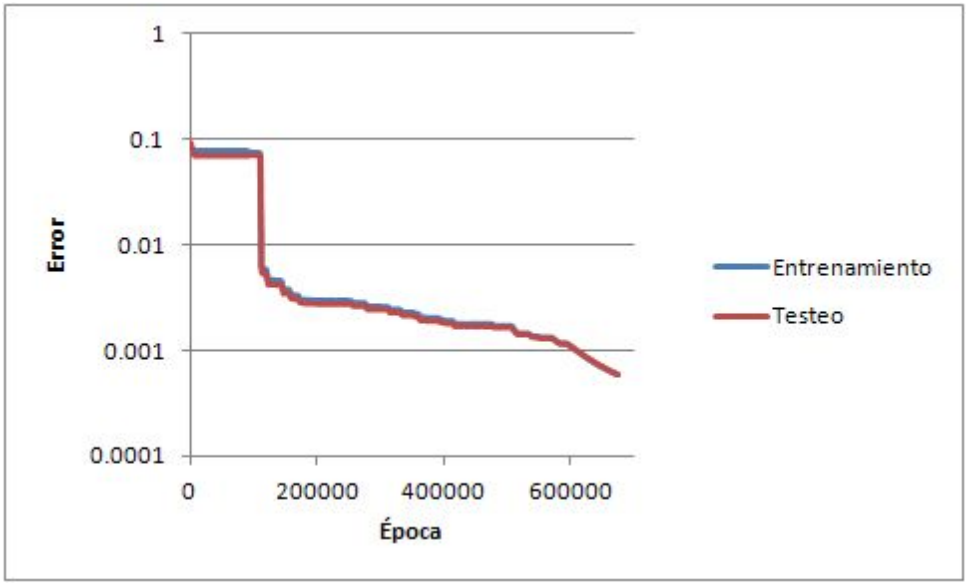

Figura 7: Error de entrenamiento y de testeo en función de las épocas de entrenamiento de la red para la arquitectura 7.

Tabla 1:

Arquitectura	Mín. error de entrenamiento alcanzado	Mín. error de testeo alcanzado
1 - [0]	0.07628695	0.06989342
2 - [10]	0.00293081	0.00287611
3 - [30]	0.00044182	0.0004365
4 - [10, 10]	0.00038096	0.00058133
5 - [30, 30]	0.00033776	0.00034388
6 - [20, 20, 20]	0.00017006	0.00057585
7 - [10, 10, 10, 10, 10]	0.00059725	0.00058852

Tabla 1: Tabla con los valores mínimos del error de entrenamiento y testeo alcanzados.

Los resultados a continuación se obtuvieron a partir de la arquitectura 4.

Figura 8:

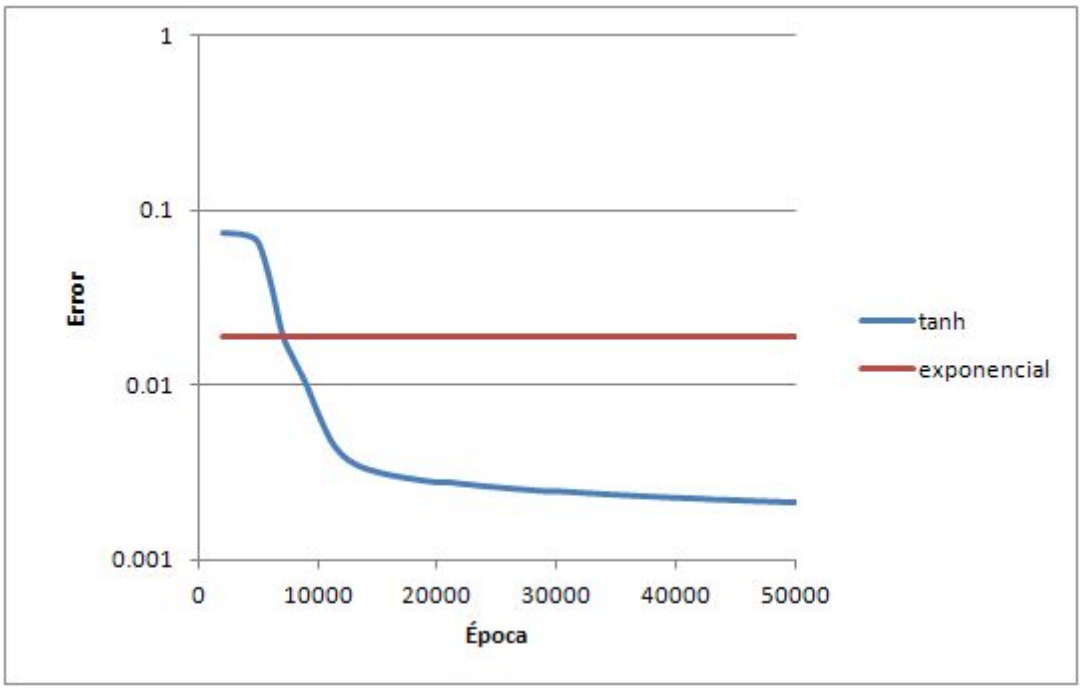

Figura 8: Error de entrenamiento con las funciones de activación exponencial y tanh y utilizando solamente la variación de backpropagation *que suma 0.1 a las derivadas.*

Figura 9:

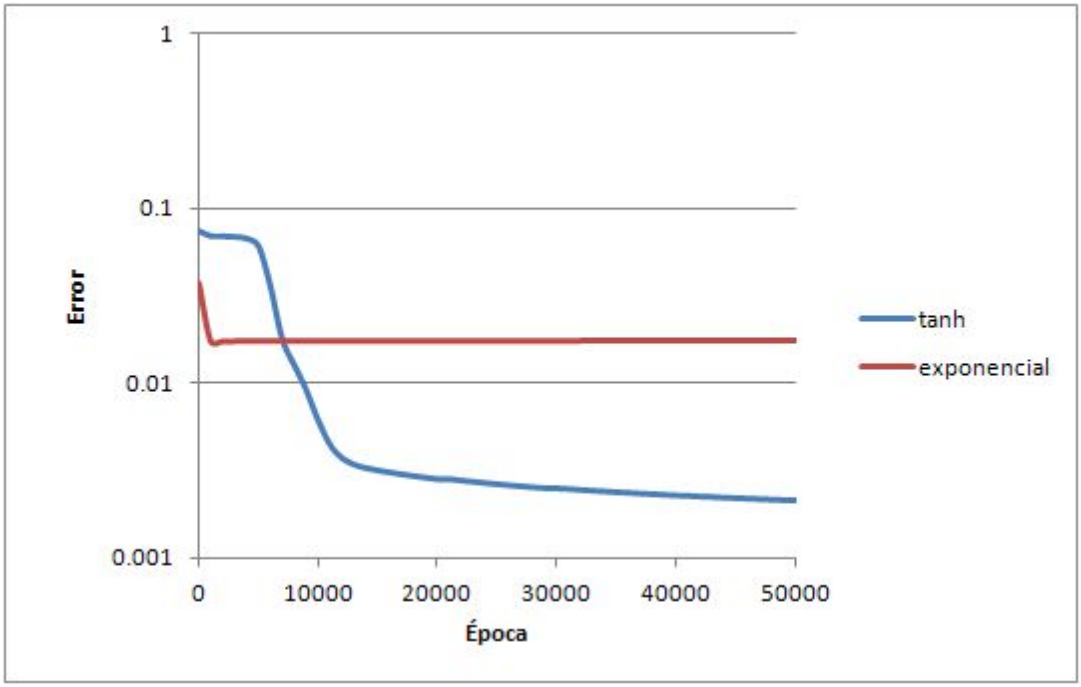

Figura 9: Error de testeo con las funciones de activación exponencial y tanh y utilizando solamente la variación de backpropagation *que suma 0.1 a las derivadas.*

Figura 10:

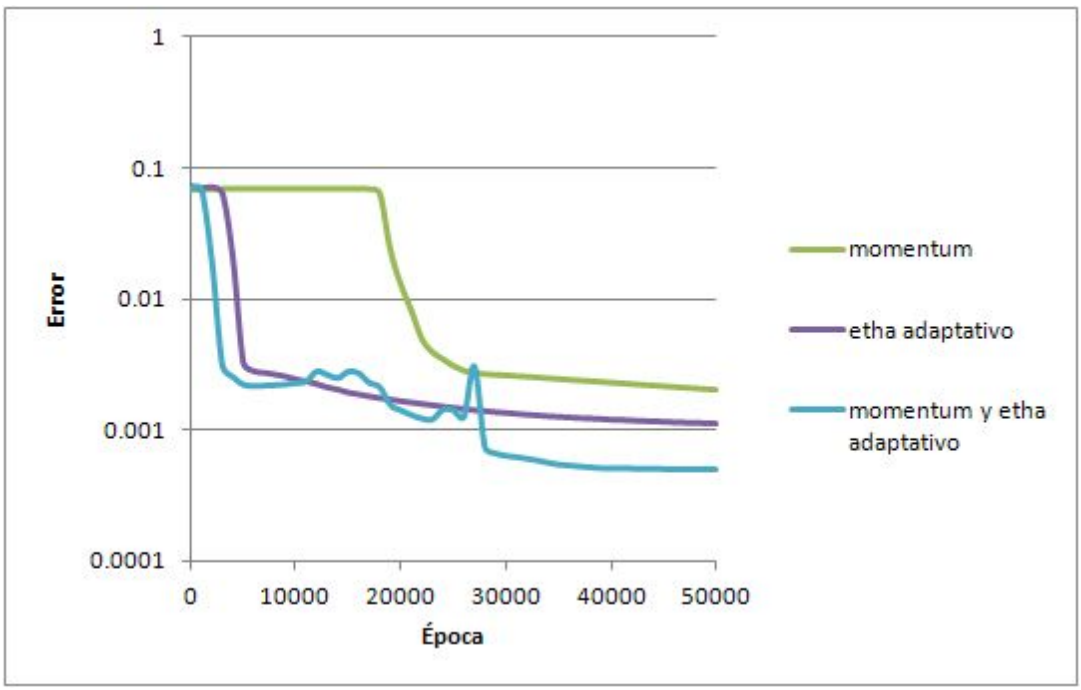

Figura 10: Error de entrenamiento con distintas variaciones del algoritmo backpropagation *utilizando tanh como función de activación.*

Figura 11:

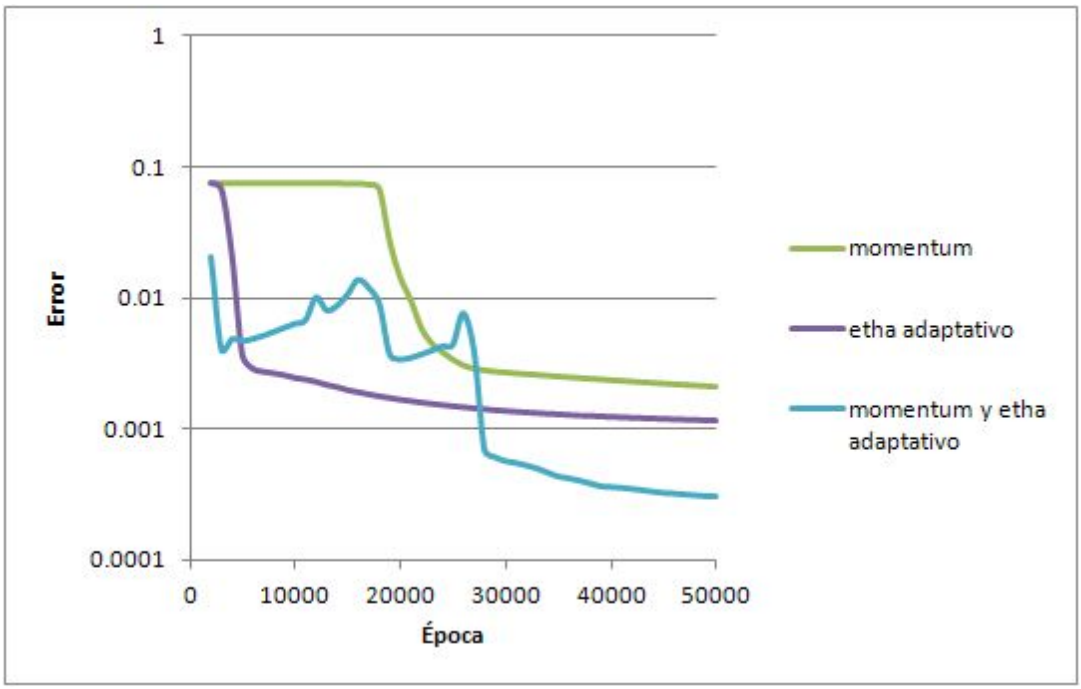

Figura 11: Error de testeo con distintas variaciones del algoritmo backpropagation *utilizando tanh como función de activación.*

Figura 12:

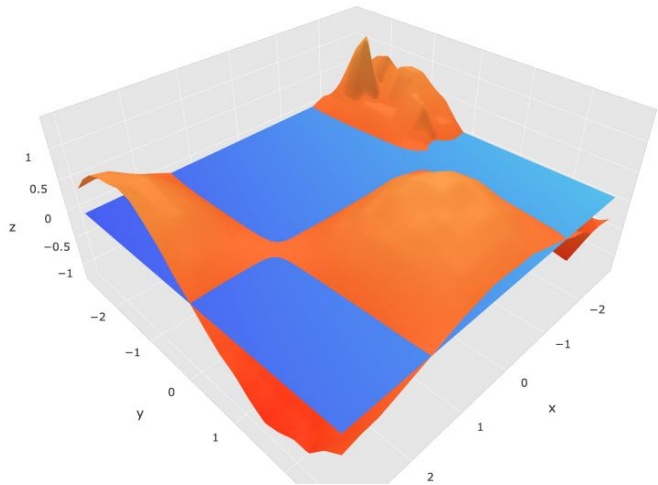

Figura 12: Gráfico del terreno basados en los datos conocidos en naranja, y en azul a partir de los obtenidos con la red para la arquitectura 1.

Figura 13:

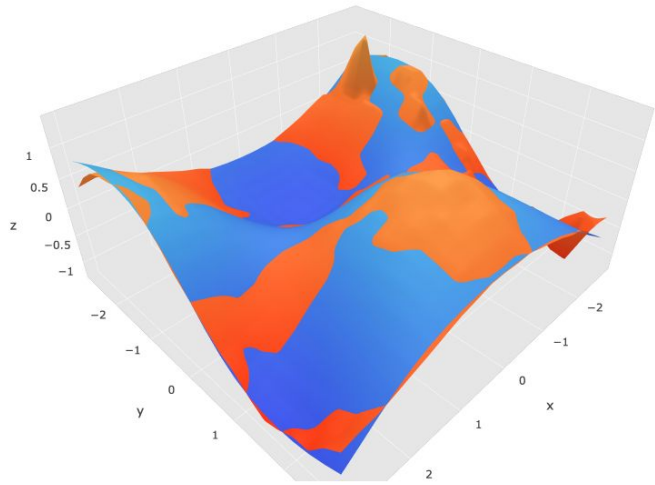

Figura 13: Gráfico del terreno basados en los datos conocidos en naranja, y en azul a partir de los obtenidos con la red para la arquitectura 2.

Figura 14:

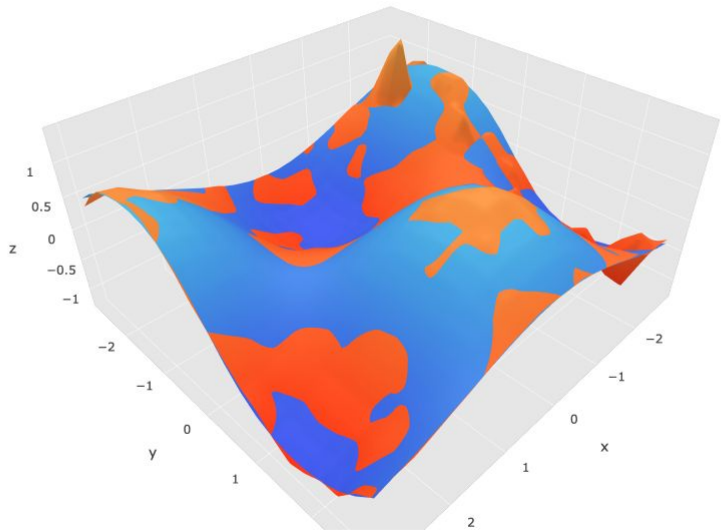

Figura 14: Gráfico del terreno basados en los datos conocidos en naranja, y en azul a partir de los obtenidos con la red para la arquitectura 3.

Figura 15:

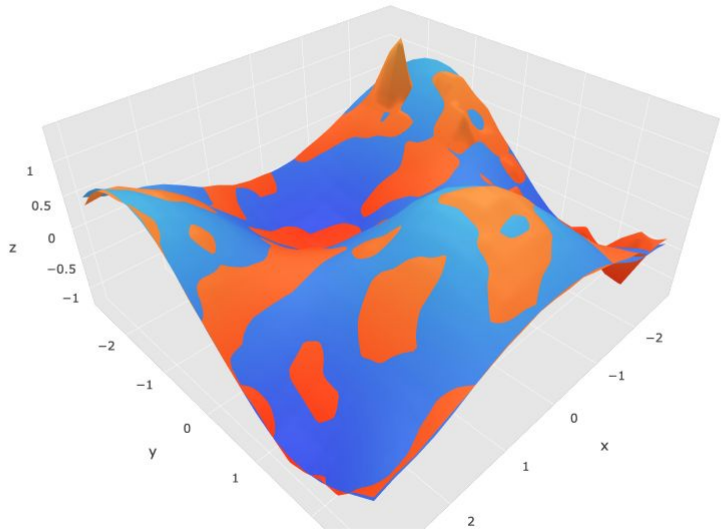

Figura 15: Gráfico del terreno basados en los datos conocidos en naranja, y en azul a partir de los obtenidos con la red para la arquitectura 4.

Figura 16:

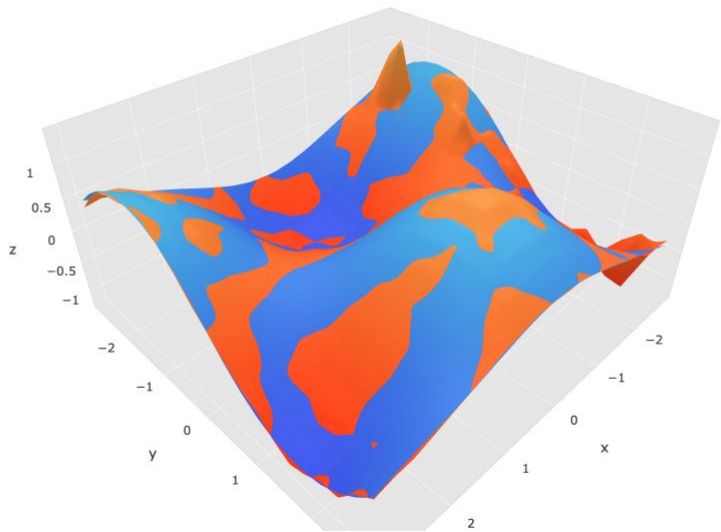

Figura 16: Gráfico del terreno basados en los datos conocidos en naranja, y en azul a partir de los obtenidos con la red para la arquitectura 5.

Figura 17:

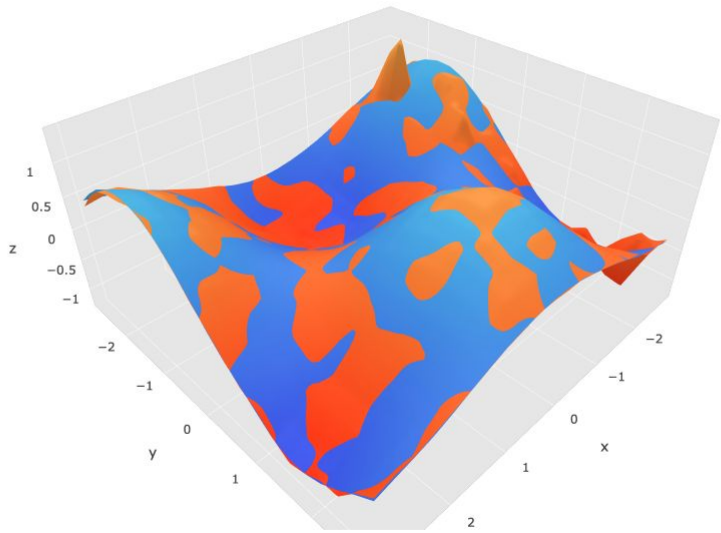

Figura 17: Gráfico del terreno basados en los datos conocidos en naranja, y en azul a partir de los obtenidos con la red para la arquitectura 6.

Figura 18:

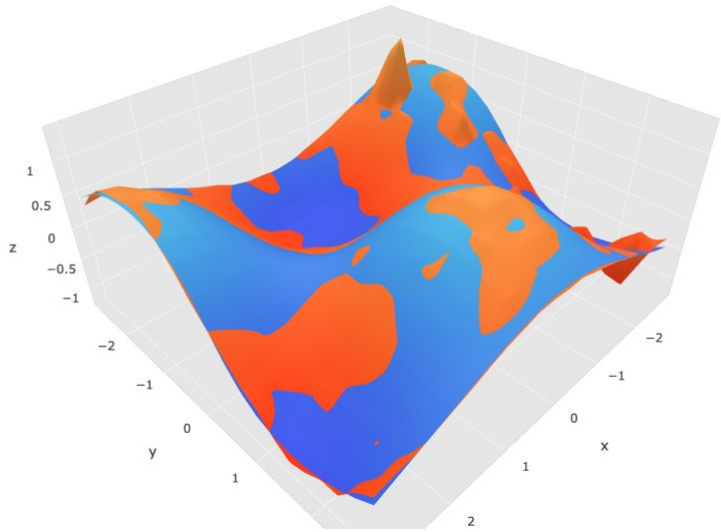

Figura 18: Gráfico del terreno basados en los datos conocidos en naranja, y en azul a partir de los obtenidos con la red para la arquitectura 7.

Conclusiones

Se puede ver a partir de los gráficos de los errores y de las superficies que la arquitectura que mejor aprende el problema es la número 6, es decir, la de tres capas de 20 neuronas cada una (Ver figuras 6 y 17). Esto demuestra que no necesariamente agregando capas se mejora, sino que también es importante la cantidad de neuronas en cada capa, puesto que la arquitectura 7 al tener menos neuronas que la 6 (por capa) da peores resultados. También es posible ver que en general agregar capas permite mejorar la capacidad de aprendizaje de la red, al igual que aumentar la cantidad de neuronas. Esto coincide con lo visto en la teórica, donde se puede pensar a un red neuronal de forma similar a una combinación de perceptrones simples, con la diferencia de que como función de activación no se utiliza la función escalón, puesto que esta no tiene derivada y se necesita de una función continua y diferenciable para poder realizar el algoritmo *backpropagation*.

Además, se concluyó que es conveniente usar tanh frente a la función exponencial como función de activación (Ver figuras 8 y 9), puesto que la primera permite un rango más amplio de valores de entrada sin que la red se sature, y por ende, los puntos son más distinguibles entre sí.

Por otro lado, se puede ver que el error decrece más rápidamente al utilizar etha adaptativo junto con momentum, aunque el mismo oscila más que utilizando dichas variantes individualmente (Ver figuras 10 y 11).