Resolución TP Nº3

Perceptrón Multicapa con Back-propagation

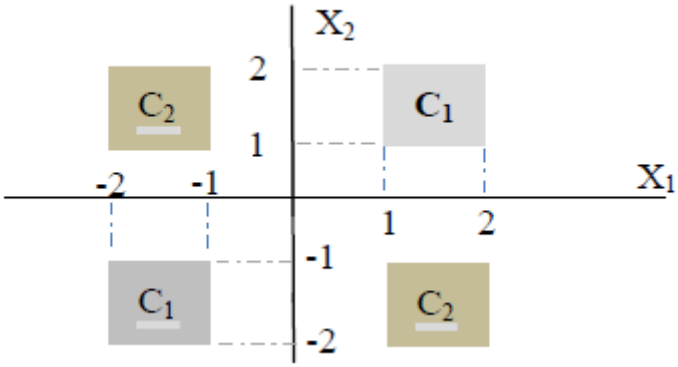
Alumno: Salomón Nicolás

Curso: Inteligencia Artificial aplicada a la Identificación y Control

Profesor: Dr. Ing. H. Daniel Patiño

Año: 2022

1. **Aprendizaje Supervisado de funciones booleanas no separables linealmente. Implementar un programa de aprendizaje supervisado usando el algoritmo de back-propagation (o sus variantes) para clasificar dos clases mostradas en la figura. Emplear una $RN^2_{2,2,1}$.**



Considerar un conjunto de entrenamiento infinito y finito. Evaluar el sobre aprendizaje.

El código de resolución se encuentra disponible en el archivo adjunto (TP3 - Perceptrón Multicapa.ipynb - Jupyter Notebook) a este informe.

Con el objetivo de tener una primera aproximación a los datos y observar los Clusters a clasificar se procedió a generar un pequeño set de datos y graficar el mismo. El resultado de esta prueba se puede apreciar en la Figura 1.

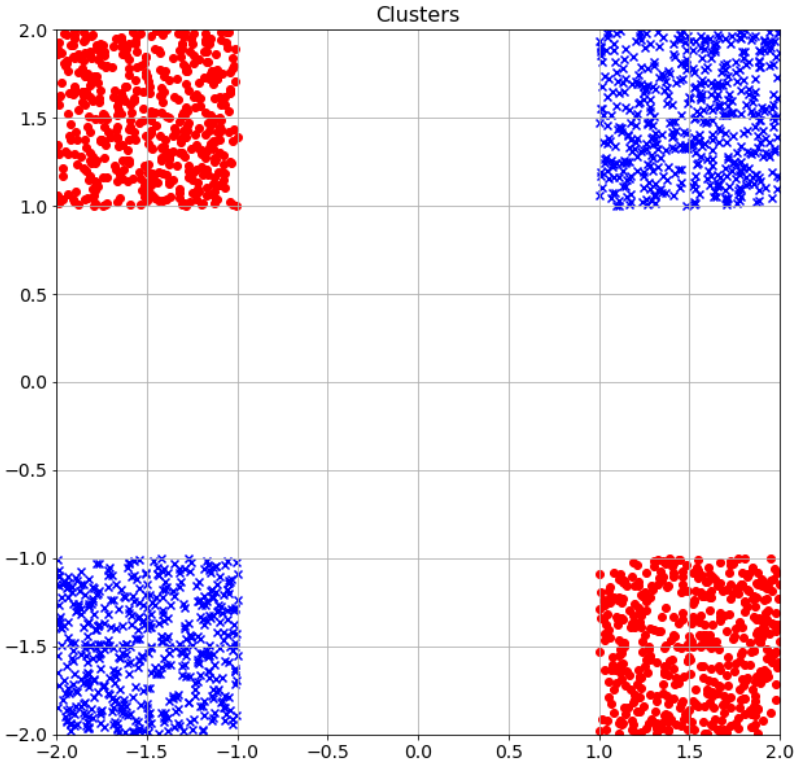


Figura 1. Clusters a clasificar.

Una vez visualizado el caso de análisis, se construyó la Red Neuronal propuesta en el enunciado, con 2 entradas (x_1 y x_2) correspondientes a las coordenadas de cada punto en el plano, 2 neuronas en la capa oculta y una neurona en la capa de salida. La representación gráfica de esta estructura puede apreciarse en la Figura 2.

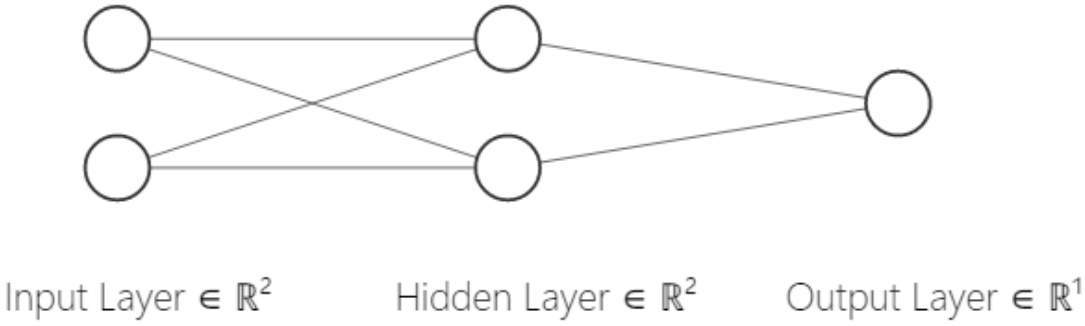


Figura 2. Red Neuronal propuesta.

Dado que nuestro caso de análisis no es separable linealmente, debido a la disposición de los clusters, el empleo de un perceptrón con una sola neurona no será suficiente, por ello debe incluirse una segunda neurona, para asegurar la correcta clasificación.

Por otro lado, a la salida de cada neurona se empleó como función de activación una función sigmoide de la forma:

$$y = \frac{1}{1+e^{-x}} \qquad y' = \frac{e^{-x}}{(1+e^{-x})^2} = y * (1-y)$$

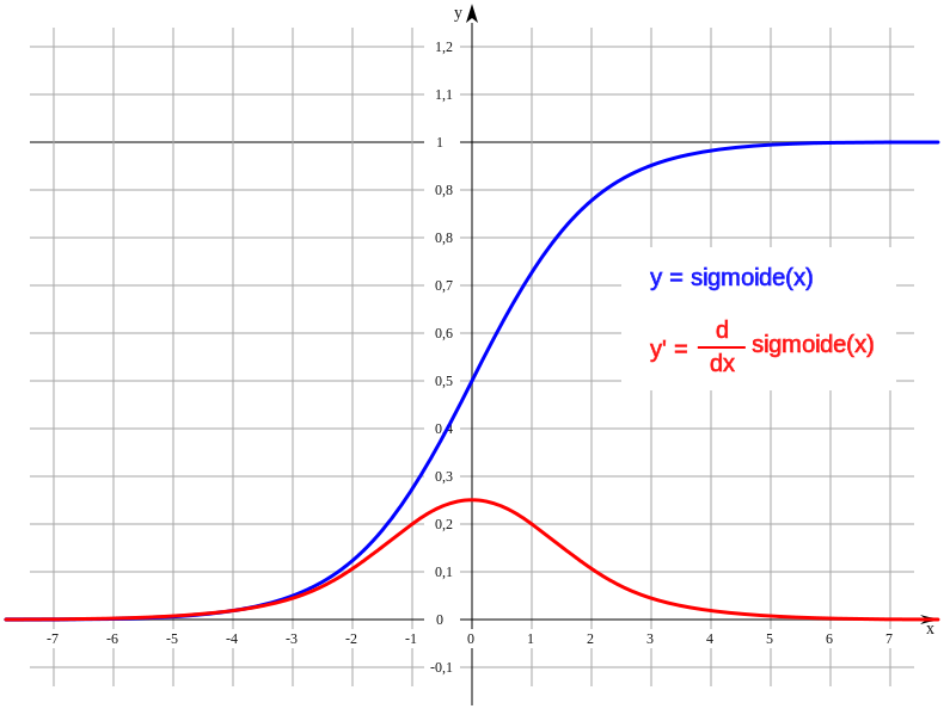


Figura 3. Función Sigmoide.

Conjunto de entrenamiento infinito

Se procedió a entrenar nuestra red neuronal empleando un conjunto de entrenamiento infinito (en cada iteración se genera un nuevo dato de entrada y salida), en conjunto con el algoritmo de back-propagation visto en clases y un coeficiente de aprendizaje de 0.05. Además, se fijó un umbral de 0.005 para detener el entrenamiento al compararlo con el valor de pérdida o error calculado en cada iteración.

El entrenamiento total duró 3,94 [s] y 16.031 iteraciones para alcanzar el umbral de pérdida descripto anteriormente. Por otro lado, la variación de los pesos durante el entrenamiento puede apreciarse en la Figura 4.

Figura 4. Evolución de pesos y bias durante el entrenamiento.

Se observa como los pesos y bias tienden a un valor final gradualmente, teniendo una variación muy leve hacia el final, lo cual nos indica que el entrenamiento fue correcto. Esto se traduce en un error continuamente decreciente que tiende a estabilizarse hacia el final del entrenamiento, como puede apreciarse en la Figura 5.

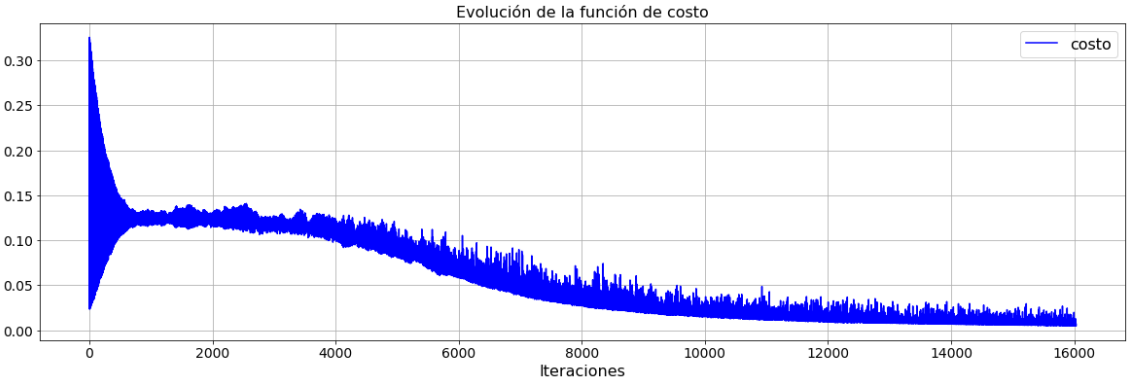


Figura 5. Variación del error durante el entrenamiento.

Por último, para evaluar el desempeño de la red, se generaron 10.000 puntos aleatorios y se clasificaron los mismos a través de la red entrenada anteriormente. El resultado de dicha clasificación se observa en la Figura 6, observando la región de decisión para cada cluster.

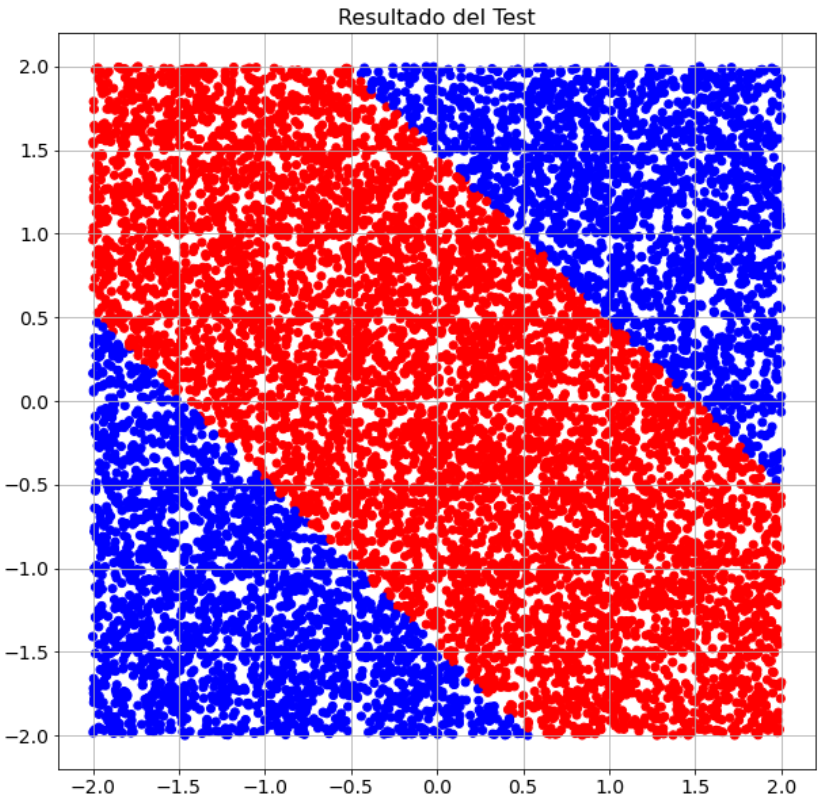


Figura 6. Resultado sobre el dataset de prueba.

Conjunto de entrenamiento finito

Para evaluar el efecto del sobreentrenamiento, se generó un dataset de 200 muestras (160 para entrenamiento y 40 para prueba) y se introdujo en la red neuronal descripta anteriormente, empleando un coeficiente de aprendizaje de 0.05 y 20.000 iteraciones.

Se obtuvo como resultado del entrenamiento una error que, a partir de la iteración 1600 aproximadamente comenzó a incrementarse, como se observa en la Figura 7.

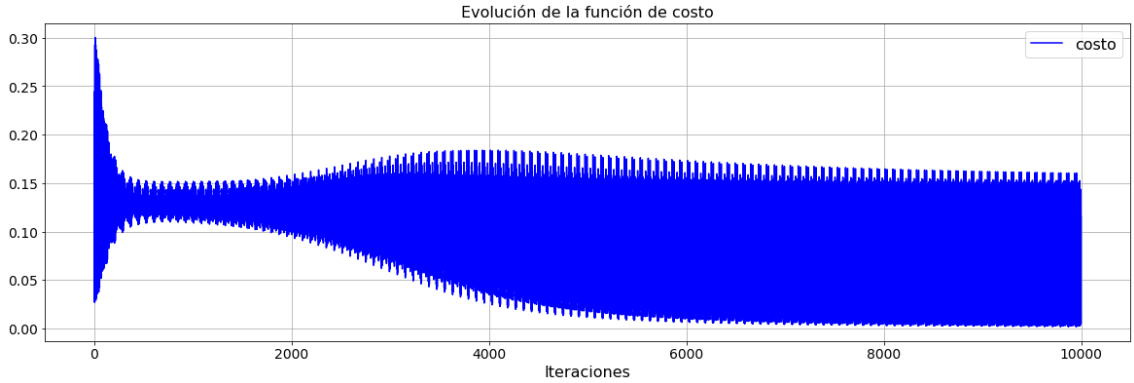


Figura 7. Variación de la función de costo durante el entrenamiento.

Mientras que las pruebas sobre el set de pruebas arrojaron el siguiente resultado (ver Figura 8):

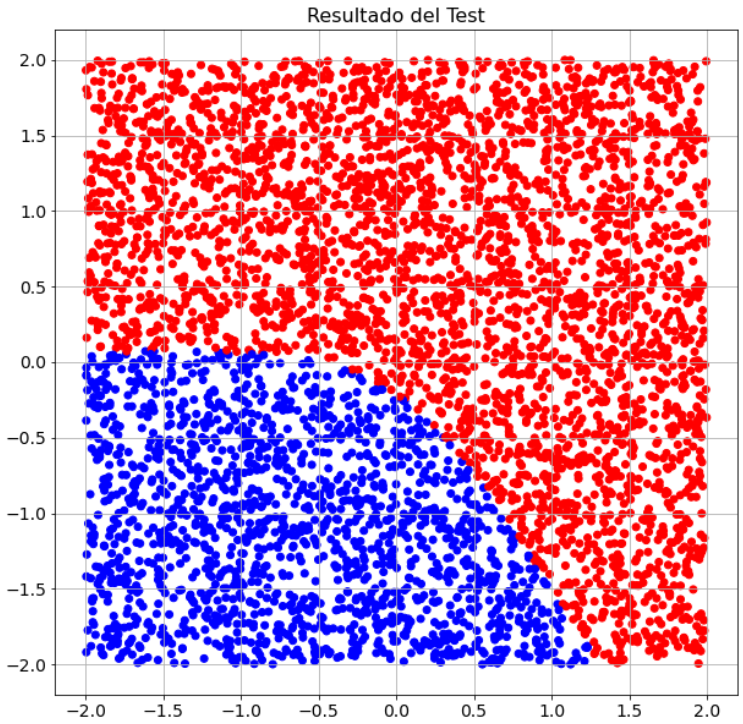


Figura 8. Resultado del entrenamiento.

Se observa un resultado erroneo debido a los pocos datos de entrenamiento con que se disponían, mostrando en conjunto con el aumento del error, un claro ejemplo de sobreajuste.

2. Aproximación de Funciones Continuas

Implementar una máquina de aprendizaje para aproximar una función f(x) usando el algoritmo de back-propagation (o sus variantes) mediante una red neuronal $RN^{3}_{1,50,25,1}$. Considere:

$$f(x) = 2 * \sin(20\pi x) + 3 * \cos(10\pi x) + sin\left(30\pi + \frac{\pi}{2}\right), x\epsilon[0, 2\pi]$$

- **Emplear un conjunto infinito de entrenamiento de entrada-salida.**
- **Evaluar la aproximación de la función graficando f(x) para una señal x que varíe linealmente de 0 a 2π.**

El código de resolución de este punto se encuentra adjunto a este informe (Aproximacion_funcion.m – Código en Matlab).

El primer paso realizado fue graficar la función, para tener una idea más clara del problema de aproximación. Dicha función puede apreciarse en la Figura 9.

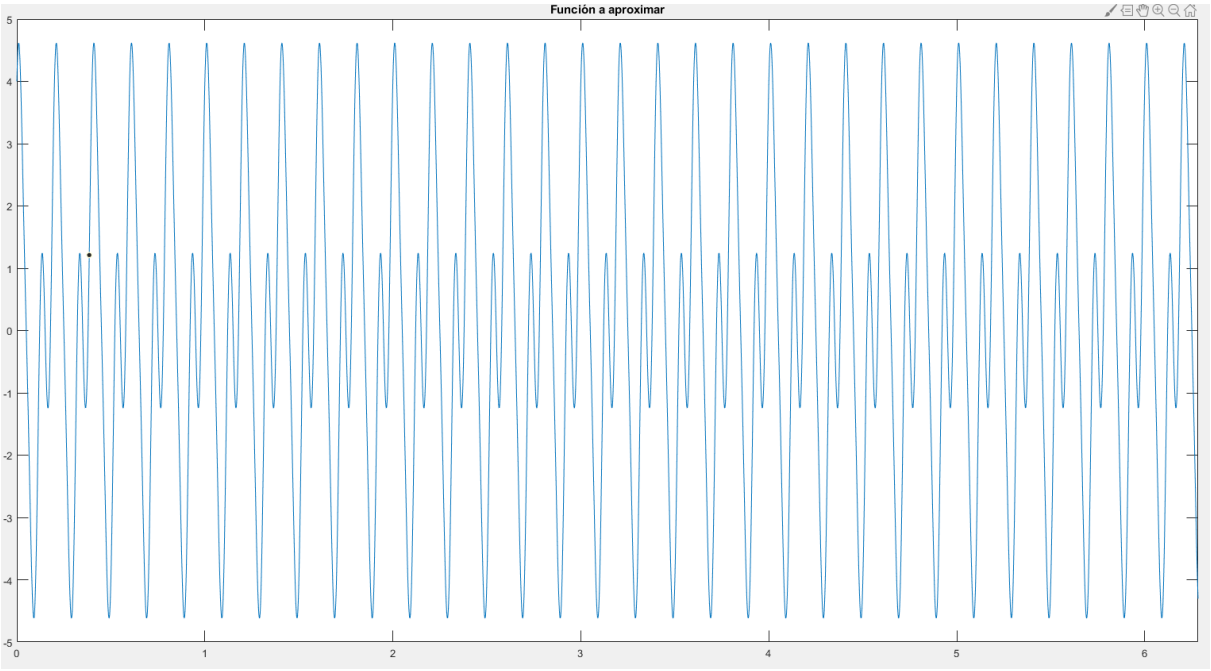


Figura 9. Función a aproximar.

Se procedió a crear la Red Neuronal según lo descripto en el enunciado, cuyo resultado puede observarse en la Figura 10.

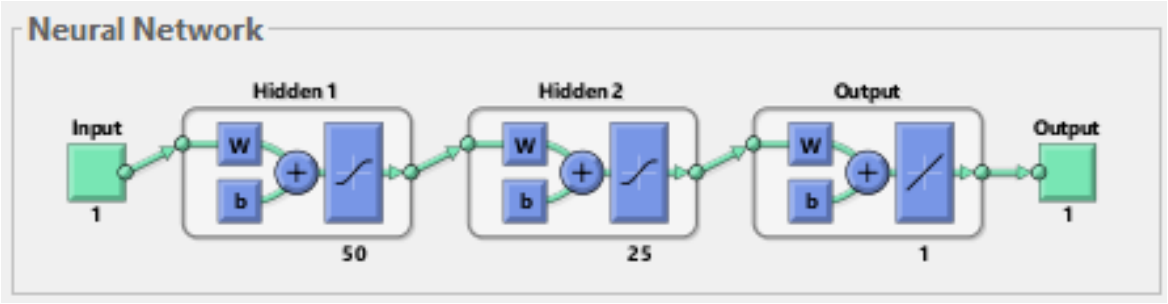


Figura 10. Red Neuronal a emplear.

Para el entrenamiento de la red neuronal se empleó el algoritmo de Levenberg-Marquardt debido a que el mismo presentó una gran rapidez y excelentes resultados en la aproximación frente a otros algoritmos, por ejemplo, el algoritmo del Gradiente Descendente.

Para el mismo se emplearon 50, 100 y 500 épocas, cuyos resultados pueden apreciarse en la Figuras 11, 12 y 13 respectivamente.

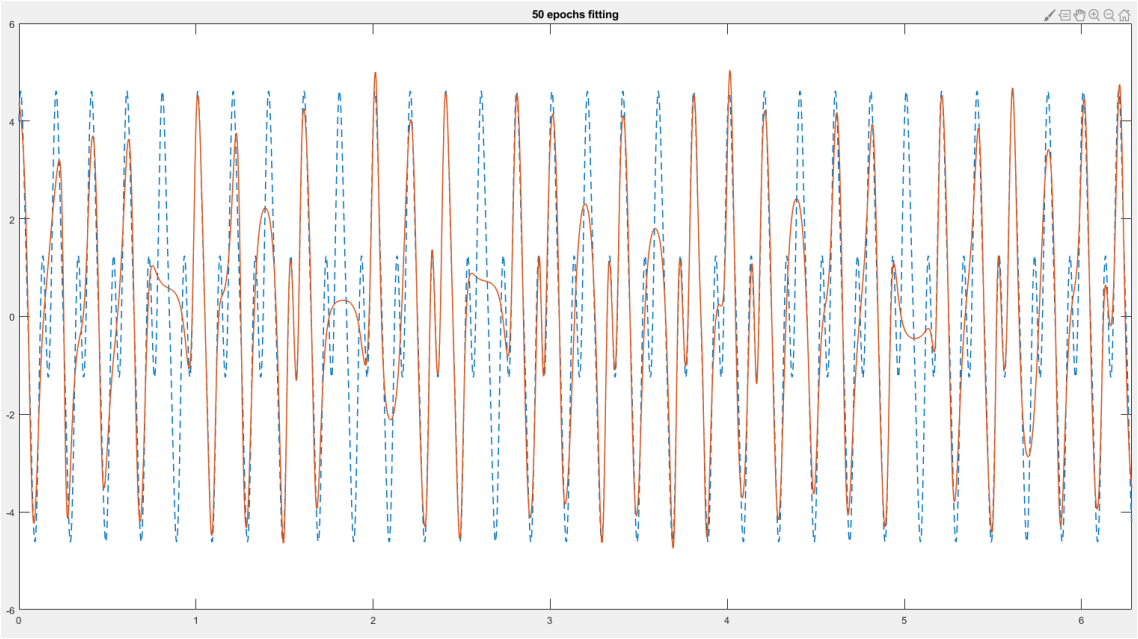


Figura 11. Resultado del entrenamiento con 50 épocas.

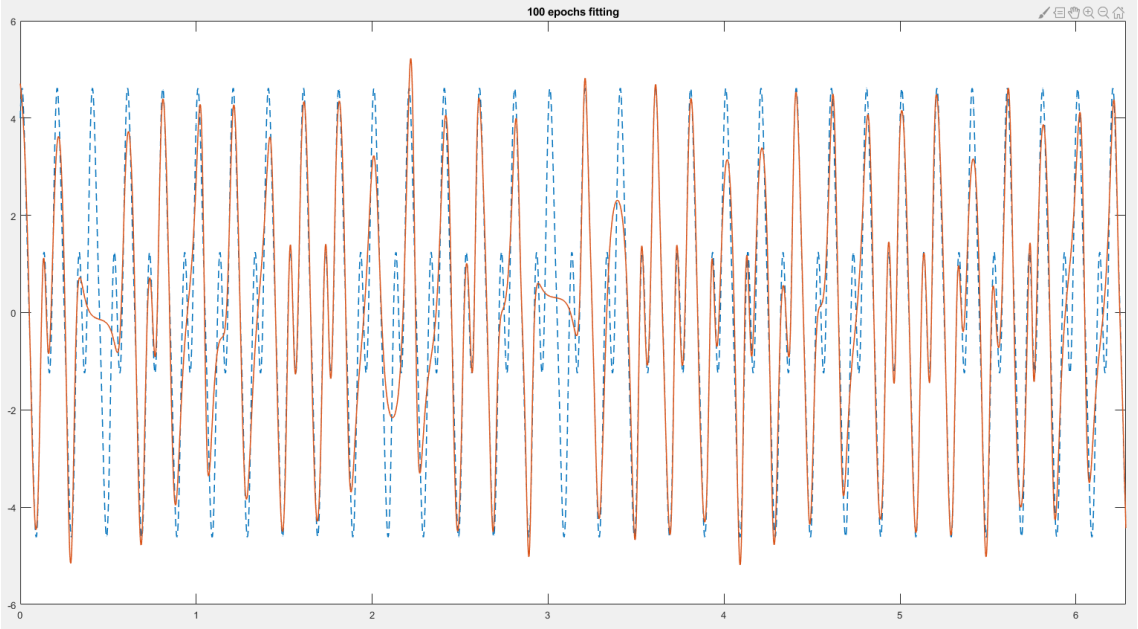


Figura 12. Resultado del entrenamiento con 100 épocas.

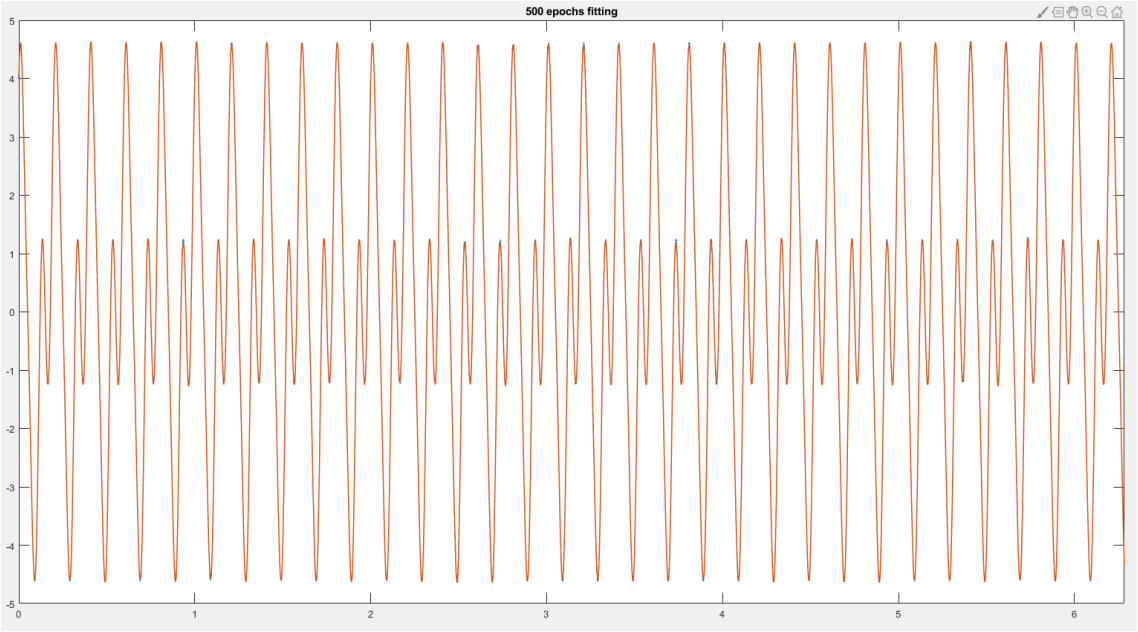


Figura 13. Resultado del entrenamiento con 100 épocas.

Se puede apreciar como con el aumento de las épocas de entrenamiento la red neuronal lograr aproximar casi exactamente la función propuesta, siendo el gráfico de líneas punteadas azules la función original, cumpliendo así con el enunciado propuesto.