Centro Universitario de Ciencias Exactas e Ingenierías

Licenciatura en Ingeniería en Computación

Materia: Seminario de Solución de Problemas de Inteligencia Artificial II. Clave: I7041.

Profesor: Valdés López Julio Esteban

Estudiante: Silva Moya José Alejandro. Código: 213546894.

Práctica VII: MLP para aproximar funciones

Problema a resolver: Programaremos y adaptaremos una Multi-Layer Perceptron Artificial Neural Network para que sea capaz de aproximar funciones matemáticas.

Desarrollo

Considerando que utilizaremos una MLP, pero solamente adaptaremos su arquitectura, los archivos MLP.py y Activations.py son los mismos que aquellos en la entrega anterior; lo único que cambiará será el main.

```
 6   def main():
 7       plt.figure(1)
 8
 9       print("----------Funciones----------")
10       print("1) Seno")
11       print("2) Coseno")
12       print("3) Tangente Hiperbolica")
13       print("4) Logaritmo Natural")
14       print("5) Valor Absoluto")
15       function = int(input("Opcion: "))
16
17       trainingPatternsFileName = "InputValues.csv"
18       x_funcFile = "X_Graph_FuncValues.csv"
```

Lo primero que haremos será crear un menú para elegir una de nuestras cuatro funciones para aproximar.

```
if function == 1:
    outputValuesFileName = "SinOutputValues.csv"
    y_funcFile = "Y_Graph_SinValues.csv"

elif function == 2:
    outputValuesFileName = "CosOutputValues.csv"
    y_funcFile = "Y_Graph_CosValues.csv"

elif function == 3:
    outputValuesFileName = "TanhOutputValues.csv"
    y_funcFile = "Y_Graph_TanhValues.csv"

elif function == 4:
    trainingPatternsFileName = "LogInputValues.csv"
    outputValuesFileName = "LogOutputValues.csv"
    x_funcFile = "Log_Graph_FuncValues.csv"
    y_funcFile = "Y_Graph_LogValues.csv"

elif function == 5:
    outputValuesFileName = "AbsOutputValues.csv"
    y_funcFile = "Y_Graph_AbsValues.csv"

else:
    raise ValueError('Funcion Desconocida')
# CSV documents selection.
```

A continuación, delimitaremos los archivos a leer y cargar dependiendo del caso de cada función.

```
epochs = 1500
learning_rate = 0.2
entries = 1  # of columns for the trainingPatternsFileName.
neurons_in_hidden_layer = 3
output_layer_neurons = 1

net = MLP((entries, neurons_in_hidden_layer, output_layer_neurons), ('tanh', 'linear'))
# First parenthesis:
    # First position: Number of entrances X to the NN.
    # Second to n-1 position: Number of neurons in hidden layers.
    # Last position: Number of neurons in the output layer.
# Second parenthesis: Activation functions for the hidden and output layers.

#############################################################################
X = []
y = []

x_func = []
y_func = []

X.append(np.array(np.loadtxt(trainingPatternsFileName, delimiter=',', usecols=0)))
y.append(np.array(np.loadtxt(outputValuesFileName, delimiter=',', usecols=0)))
#Obtaining training patterns in X and output values in y.

x_func.append(np.array(np.loadtxt(x_funcFile, delimiter=',', usecols=0)))
y_func.append(np.array(np.loadtxt(y_funcFile, delimiter=',', usecols=0)))
#Obtaining graphing patterns in X and output values in y.

#############################################################################
```

Posteriormente cargaremos los datos e inicializaremos nuestra red neuronal. Para este caso en específico tendremos una arquitectura de una neurona en la capa de entrada, generalmente de 2 a 3 neuronas en la capa oculta – cuya función de activación será tangente hiperbólica – y una neurona en la capa de salida, con la función de activación lineal. Tendremos una carga de datos para la gráfica de la función original (n cantidad de puntos), y otros que serán nuestros datasets, que serán una cantidad de puntos mucho menor a n. Hemos delimitado 1500 como el número máximo permitido de épocas de entrenamiento.

```
# Training section.

plt.axhline(y=0, color='k')
plt.axvline(x=0, color='k')
plt.xlim([-5, 5])
plt.ylim([-5, 10])

error_list = []

for i in range(epochs):
    error, pred = net.train(X, y, 1, learning_rate)
    error_list.append(error)
    print("Epoch:", i, "Error:", error)

    if i%10 == 0:
        plt.clf()
        plt.scatter(x_func, y_func, s=40, c='#1f77b4')
        plt.plot(X[0], pred[0], color='red', linewidth=3)
        plt.show()
        plt.pause(0.2)

        # if error < 0.03:
        if error < 0.0015: #Para tanh y log.
            break

plt.figure(2)
plt.plot(error_list, color='red', linewidth=3)

results = np.array(net.predict(X)).T
np.savetxt("Results.csv", results, delimiter=",", fmt='%.4f')
#############################################################################
```

Finalmente llegamos al entrenamiento. En cada época de entrenamiento obtendremos el error y nuestra predicción, para poder imprimir la línea de la predicción de la función de manera dinámica, y el error hasta el final.

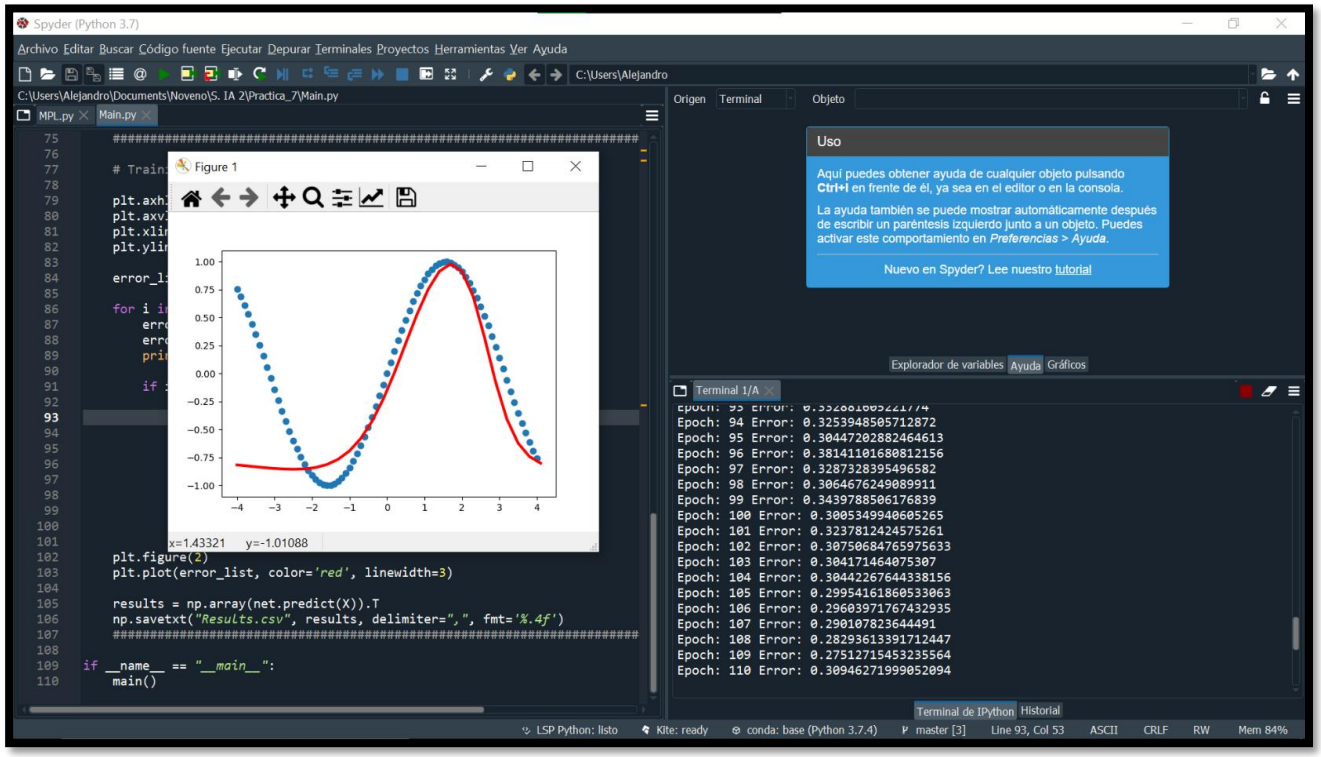
Imprimiremos el resultado de la red neuronal por cada 10 épocas, para asegurarnos de que no sea un proceso de graficación demasiado lento.

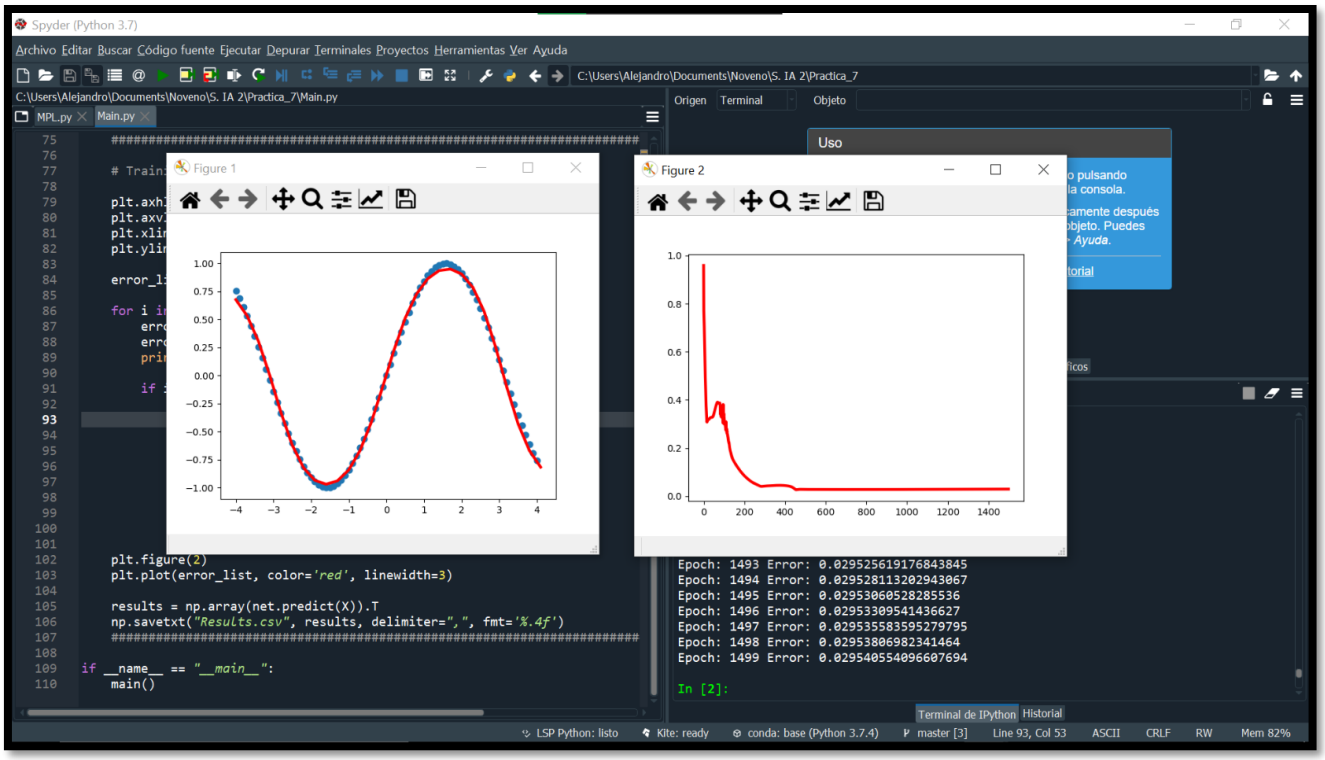
Finalmente, interrumpiremos el proceso de aprendizaje en caso de que el error sea menor que un umbral determinado previamente.

Añadido a esto, graficaremos el error una vez terminado el entrenamiento, y guardaremos nuestras aproximaciones en un csv extra, con 4 decimales de precisión.

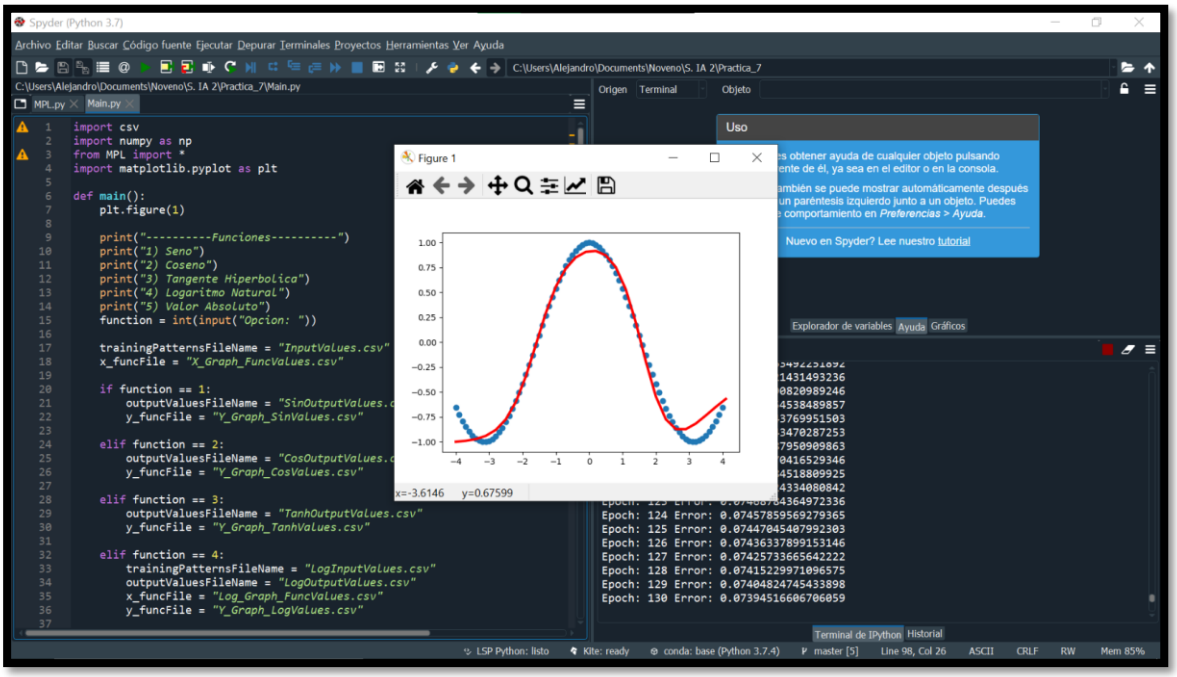
Resultados

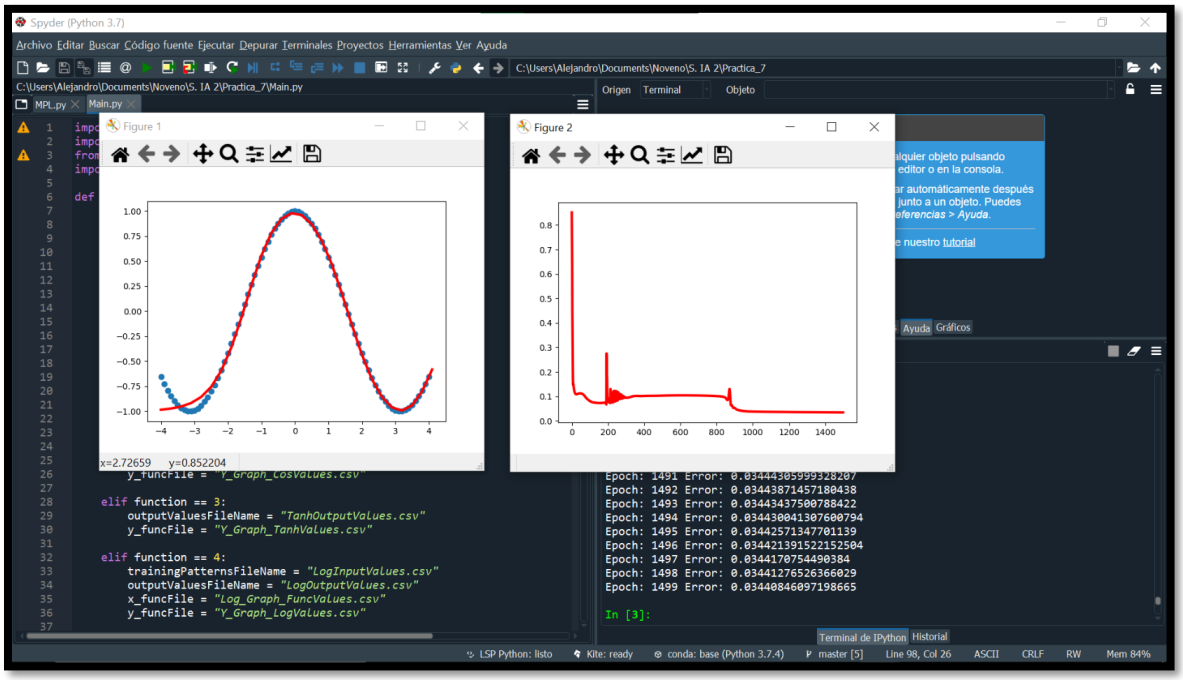
Aproximación de la función Seno.



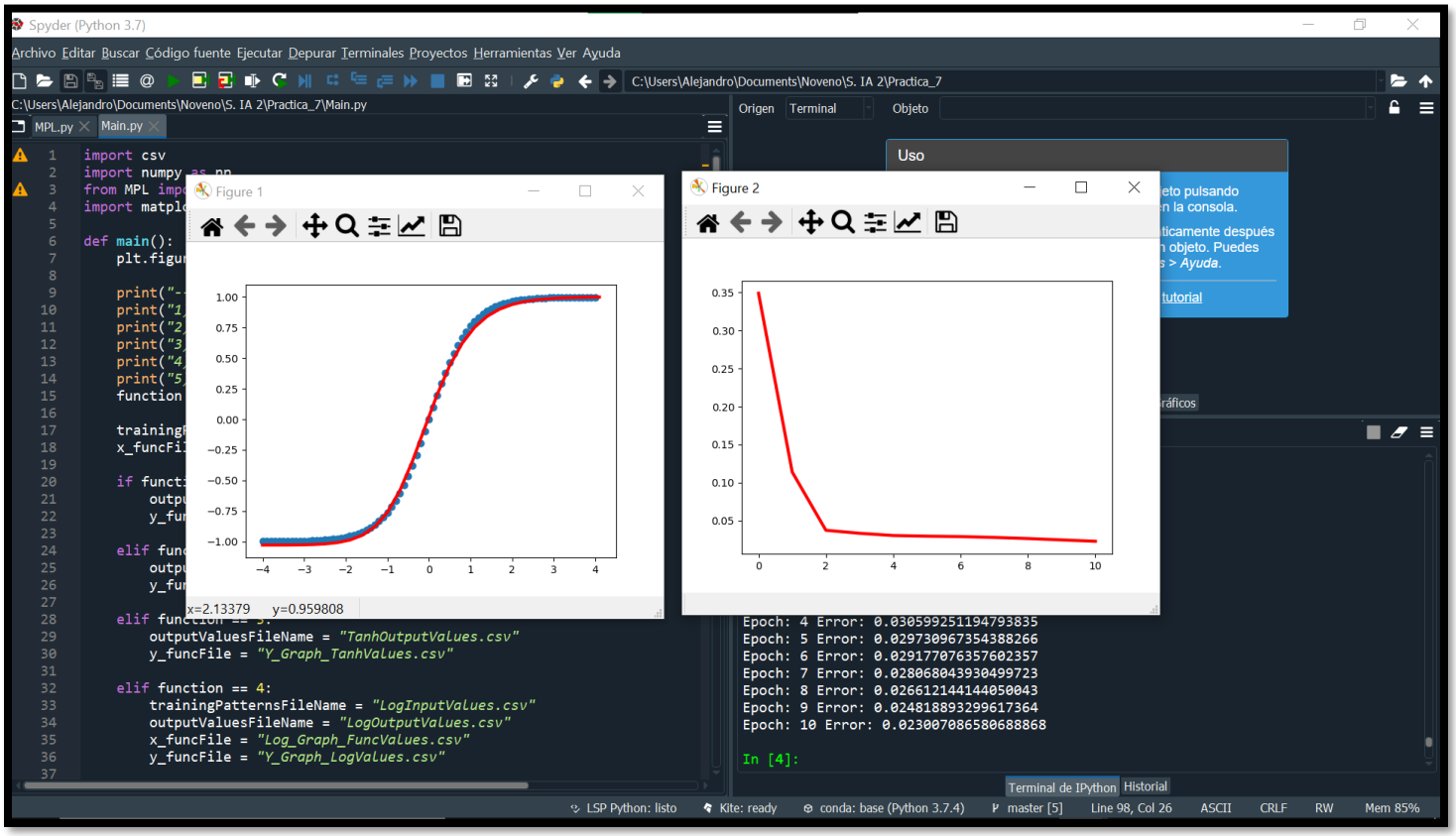


Aproximación de la función Coseno:

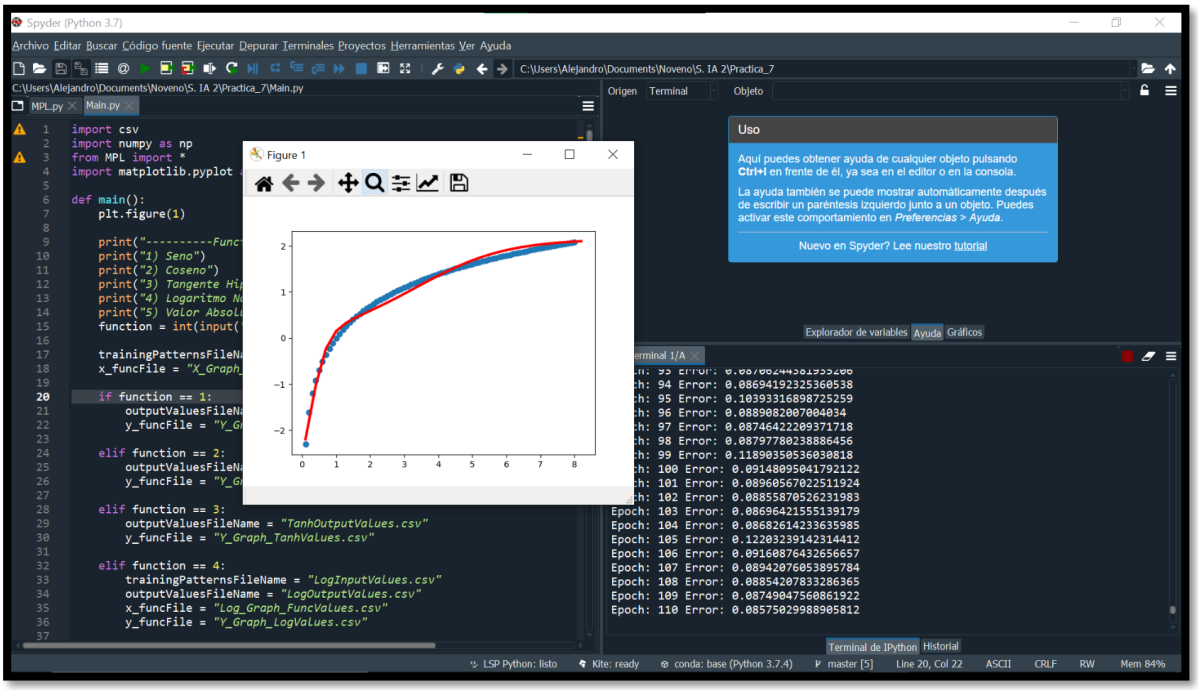


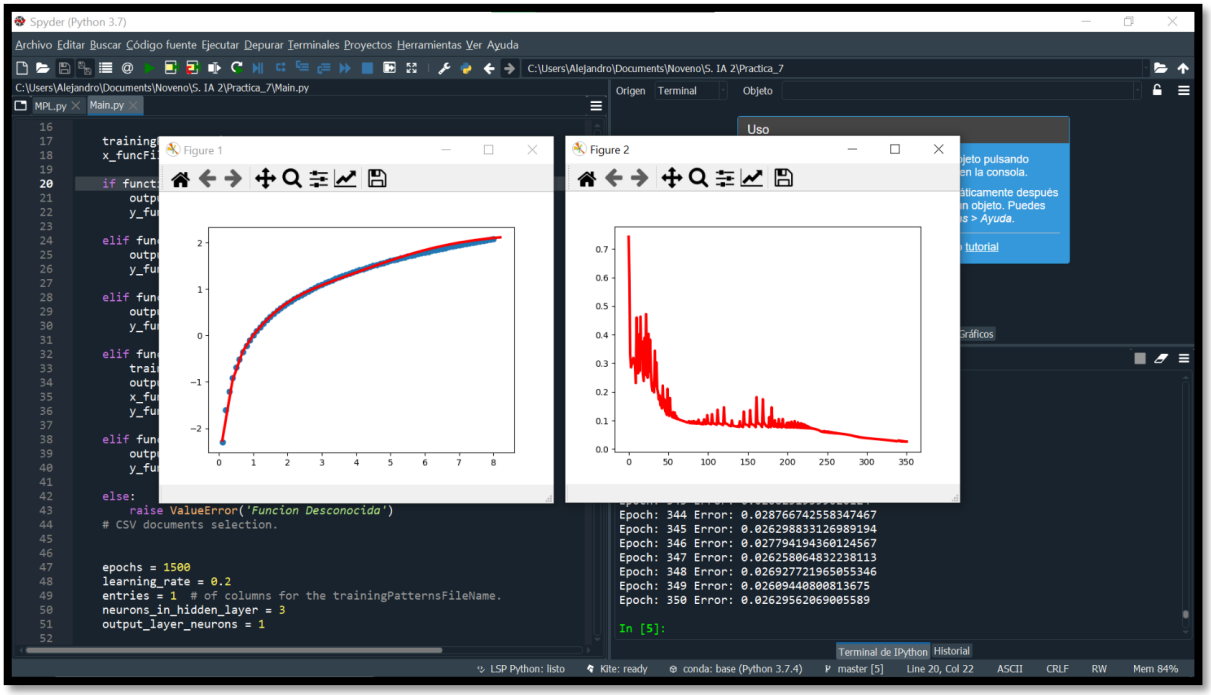


Aproximación de la función Tangente Hiperbólica:

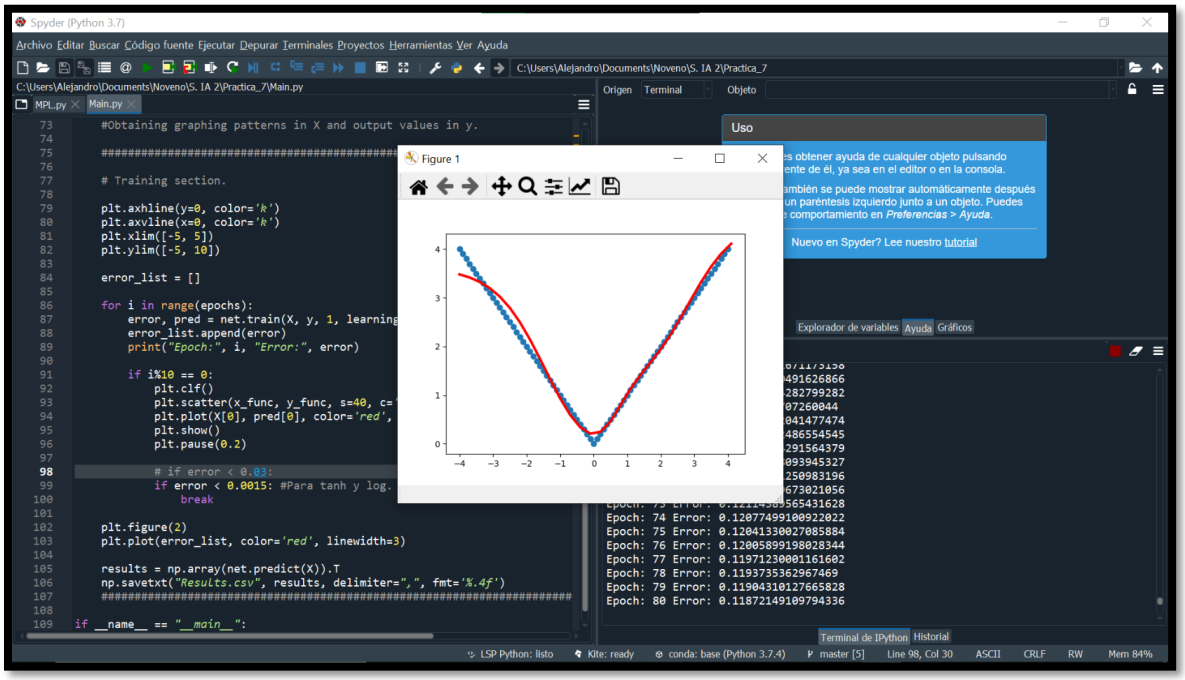


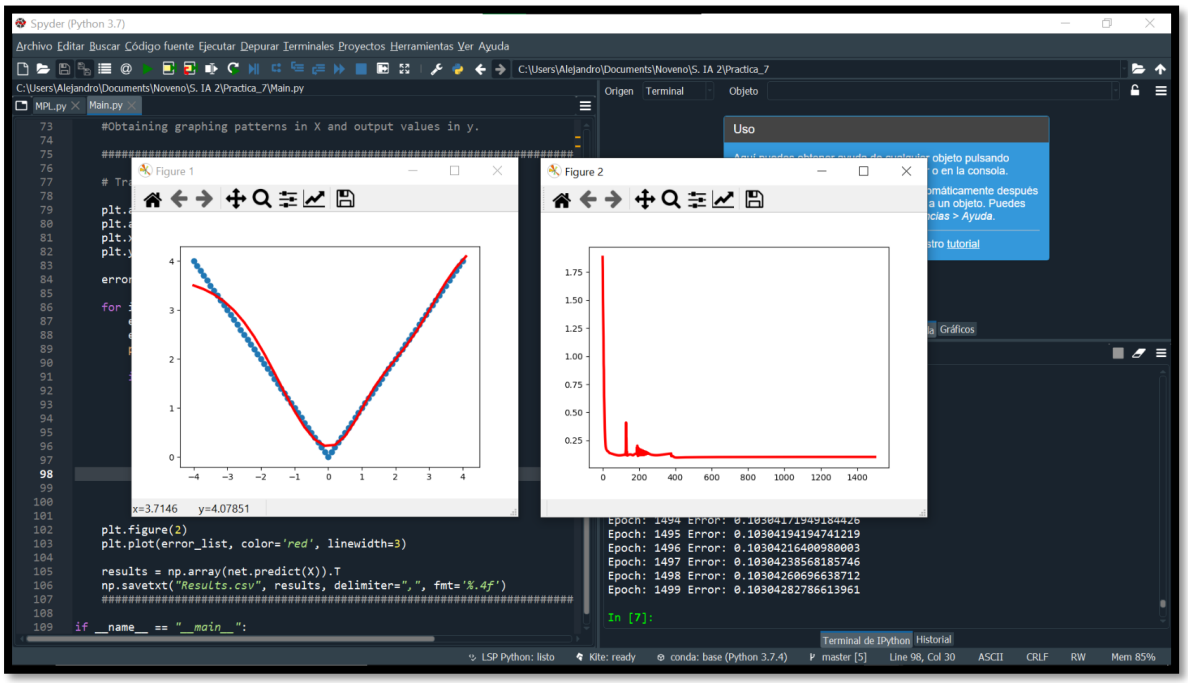
Aproximación de la función Logaritmo Natural:





Aproximación de la función Valor Absoluto:





Conclusión

Como podemos observar, obtenemos resultados considerablemente buenos en el entrenamiento de casi todas las funciones presentes, a excepción de coseno y el valor absoluto, para los cuáles en definitiva tiene problemas para dar una aproximación final convincente. Esto no mejora en el caso de alterar los parámetros o la arquitectura general de nuestra red neuronal, por lo que puedo concluir que, en definitiva, obtendríamos un resultado mayoritariamente satisfactorio en el caso de cambiar el solucionador de este problema por una Red Neuronal de Función de Base Radial.

Código

Main.py

```
import csv
import numpy as np
from MPL import *
import matplotlib.pyplot as plt

def main():
    plt.figure(1)

    print("----------Funciones----------")
    print("1) Seno")
    print("2) Coseno")
    print("3) Tangente Hiperbolica")
    print("4) Logaritmo Natural")
    print("5) Valor Absoluto")
    function = int(input("Opcion: "))

    trainingPatternsFileName = "InputValues.csv"
    x_funcFile = "X_Graph_FuncValues.csv"

    if function == 1:
        outputValuesFileName = "SinOutputValues.csv"
        y_funcFile = "Y_Graph_SinValues.csv"

    elif function == 2:
        outputValuesFileName = "CosOutputValues.csv"
        y_funcFile = "Y_Graph_CosValues.csv"

    elif function == 3:
        outputValuesFileName = "TanhOutputValues.csv"
        y_funcFile = "Y_Graph_TanhValues.csv"

    elif function == 4:
        trainingPatternsFileName = "LogInputValues.csv"
        outputValuesFileName = "LogOutputValues.csv"
        x_funcFile = "Log_Graph_FuncValues.csv"
        y_funcFile = "Y_Graph_LogValues.csv"

    elif function == 5:
```

```
    outputValuesFileName = "AbsOutputValues.csv"
    y_funcFile = "Y_Graph_AbsValues.csv"

else:
    raise ValueError('Funcion Desconocida')
# CSV documents selection.


epochs = 1500
learning_rate = 0.1
entries = 1  # of columns for the trainingPatternsFileName.
neurons_in_hidden_layer = 3
output_layer_neurons = 1

net = MLP((entries, neurons_in_hidden_layer, output_layer_neurons), ('tanh', 'linear'))
# First parenthesis:
    # First position: Number of entrances X to the NN.
    # Second to n-1 position: Number of neurons in hidden layers.
    # Last position: Number of neurons in the output layer.
# Second parenthesis: Activation functions for the hidden and output layers.

#############################################################################
X = []
y = []

x_func = []
y_func = []

X.append(np.array(np.loadtxt(trainingPatternsFileName, delimiter=',', usecols=0)))
y.append(np.array(np.loadtxt(outputValuesFileName, delimiter=',', usecols=0)))
#Obtaining training patterns in X and output values in y.

x_func.append(np.array(np.loadtxt(x_funcFile, delimiter=',', usecols=0)))
y_func.append(np.array(np.loadtxt(y_funcFile, delimiter=',', usecols=0)))
#Obtaining graphing patterns in X and output values in y.

#############################################################################

# Training section.

plt.axhline(y=0, color='k')
plt.axvline(x=0, color='k')
plt.xlim([-5, 5])
plt.ylim([-5, 10])

error_list = []

for i in range(epochs):
    error, pred = net.train(X, y, 1, learning_rate)
    error_list.append(error)
    print("Epoch:", i, "Error:", error)
```

```
        if i%10 == 0:
            plt.clf()
            plt.scatter(x_func, y_func, s=40, c='#1f77b4')
            plt.plot(X[0], pred[0], color='red', linewidth=3)
            plt.show()
            plt.pause(0.2)

            # if error < 0.03:
            if error < 0.0015: #Para tanh y log.
                break

    plt.figure(2)
    plt.plot(error_list, color='red', linewidth=3)

    results = np.array(net.predict(X)).T
    np.savetxt("Results.csv", results, delimiter=",", fmt='%.4f')
    ###############################################################################

if __name__ == "__main__":
    main()
```

Link al repositorio

- https://github.com/TheGenesisX/S_IA_2/tree/master/Practica_7