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 III: Perceptrón entrenado capaz de recibir más de dos entradas.

Problema a resolver: En la presente práctica programaremos un perceptrón capaz de recibir patrones de aprendizaje con más de 2 entradas, de manera que comenzaremos a dimensionar de R^3 en adelante.

También, ahora tendremos la entrada de datos por parte de dos archivos CSV: uno contendrá los patrones de entrada, y el otro tendrá las salidas deseadas. Desde ahí cargaremos todo al programa.

Finalmente, generaremos un archivo de salida con los resultados obtenidos del entrenamiento del perceptrón, que entonces deberá coincidir en su totalidad con el archivo de salidas deseadas. También reemplazaremos la gráfica de la línea de la neurona por la gráfica del error calculado en cada época.

Desarrollo

```
5   trainingPatternsFileName = "InputValues2.csv"
6   outputValuesFileName = "OutputValues2.csv"
7   epochs = 20
8
9   file = open(trainingPatternsFileName)
10  rows = len(file.readlines())
11  file.close()
12  #To obtain the number of rows from the CSV file
13
14  file = open(trainingPatternsFileName,'r')
15  reader = csv.reader(file,delimiter=',')
16  columns = len(next(reader))
17  file.close()
18  #To obtain the number of columns from the CSV file
19
20  net = pwe.neurona(columns, 0.1)
21  #Perceptron initialization.
```

Lo primero que haremos será recibir nuestros datos de entrada y salida a partir de dos archivos CSV, y delimitaremos la cantidad de épocas en las que nuestro perceptrón va a aprender.

Después leeremos la cantidad de filas y columnas que vamos a procesar, para poder inicializar correctamente nuestra neurona.

Finalmente inicializaremos nuestra neurona con el número adecuado de columnas para el vector de los pesos sinápticos, y delimitaremos un factor de aprendizaje.

```
24  patterns = []
25
26  for i in range(columns):
27      x = np.array(np.loadtxt(trainingPatternsFileName, delimiter=',', usecols=i))
28      patterns.append(x)
29  X = np.array(patterns)
30
31  y = np.array(np.loadtxt(outputValuesFileName, delimiter=',', usecols=0))
32  #Obtaining training patterns in X and output values in y.
```

A continuación, cargaremos nuestro dataset. Para esto, en un array X obtendremos toda la información de las entradas, que pueden ser desde 2 hasta n columnas.

De la misma manera, en el array Y obtendremos las salidas deseadas.

```
net.train(X, y, epochs)

results = []
for i in range(rows):
    prediction = net.predict(X[:, i])
    results.append(prediction)

np.savetxt("Results.csv", results, delimiter=",", fmt='%.0f')
```

Finalmente entrenaremos a la neurona y guardaremos los resultados de la misma en un nuevo archivo CSV, que deberá corresponder con nuestro archivo de salidas deseadas para indicar que el entrenamiento del perceptrón ha sido exitoso. El formato (fmt) es para delimitar la cantida de decimales después del punto, que en este caso no los necesitamos, ya que nuestros resultados son sencillamente 1's o -1's.

```
def train(self, X, y, epochs):  #x = matriz de entrenamiento, y = vector con
    n, m = X.shape
    #n = 2. m = 4 en el ejemplo de la compuerta AND, OR y XOR.
    average = 0
    for i in range(epochs):
        for j in range(m):
            y_pred = self.predict(X[:, j])
            #what that line did is sliced the array,
            #taking all rows (:) but keeping the column (j)
            if y_pred != y[j]:  #Si nuestro estimado es diferente a nuestro
                self.w += self.eta*(y[j] - y_pred) * X[:, j].reshape(-1, 1)
                self.b += self.eta*(y[j] - y_pred)
            average += (y[j] - y_pred)**2
        average /= m
        graphError(i, average)
        x_axis.append(i)
        y_axis.append(average)
        average = 0

    plt.plot(y_axis, 'c')
```

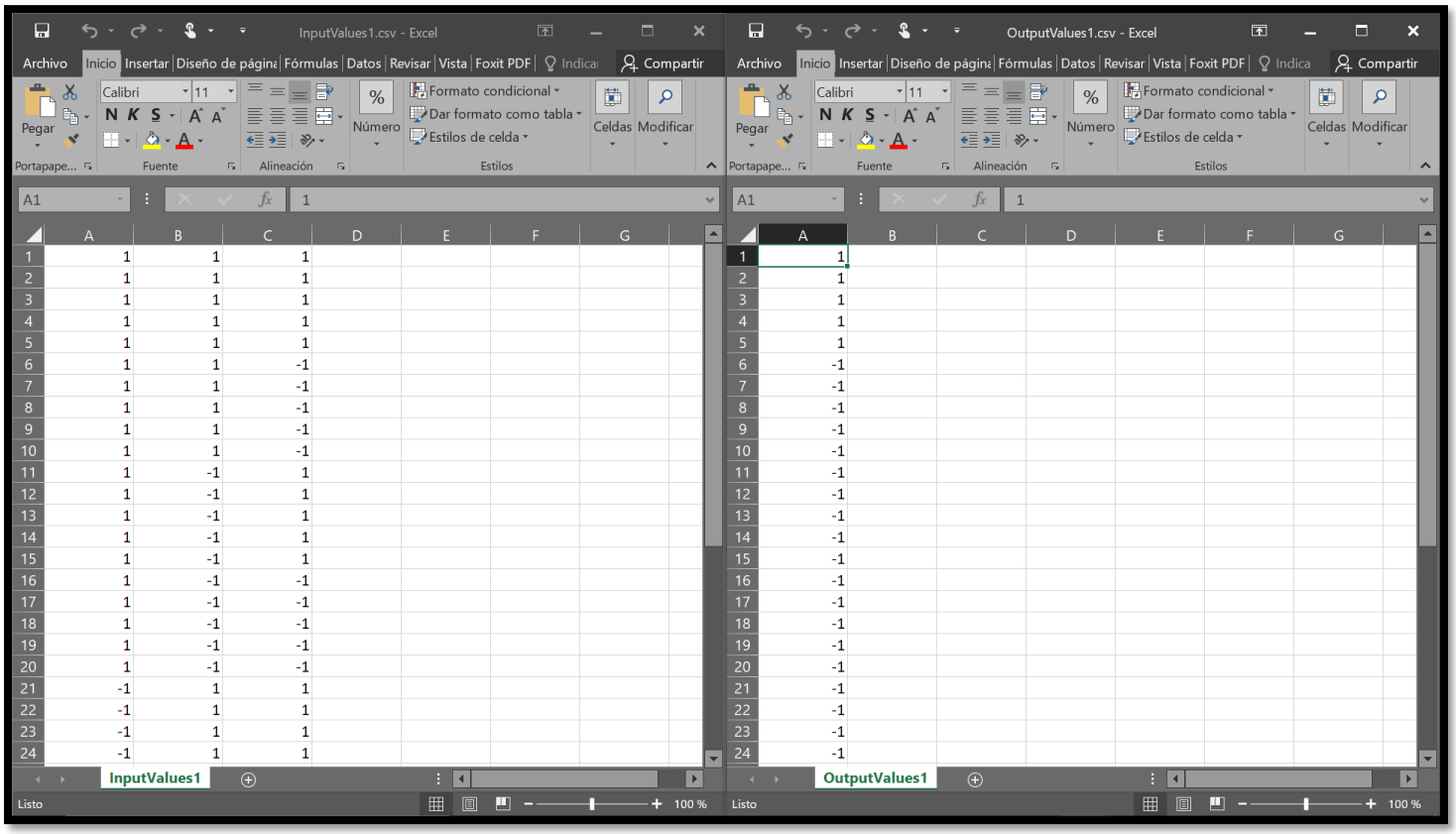
Ahora bien; para poder obtener el error final es necesario calcular el error de cada entrada y elevarlo al cuadrado (para evitar problemas futuros si obtenemos números negativo), para después sumarlo en conjunto con el resto de la época. Una vez terminada la época obtenemos el promedio, lo mandamos a graficar con coordenadas X y Y, y guardamos los datos en listas extra para poder graficar la línea final, ya que por el momento solo graficaremos puntos. Por último, reseteamos el valor del promedio, y una vez graficado el conjunto de puntos graficamos la línea aparte.

```
4   x_axis = []
5   y_axis = []
6
7   plt.ylabel('Error', fontsize=30)
8   plt.xlabel('Epocas', fontsize=30)
9   plt.title('Graficacion del error del perceptron', fontsize=40)
10
11  def graphError(x_coordinate, y_coordinate):
12      plt.scatter(x_coordinate, y_coordinate)
13      plt.plot(x_coordinate, y_coordinate, '*', markersize=14)
14      plt.pause(0.3)
```

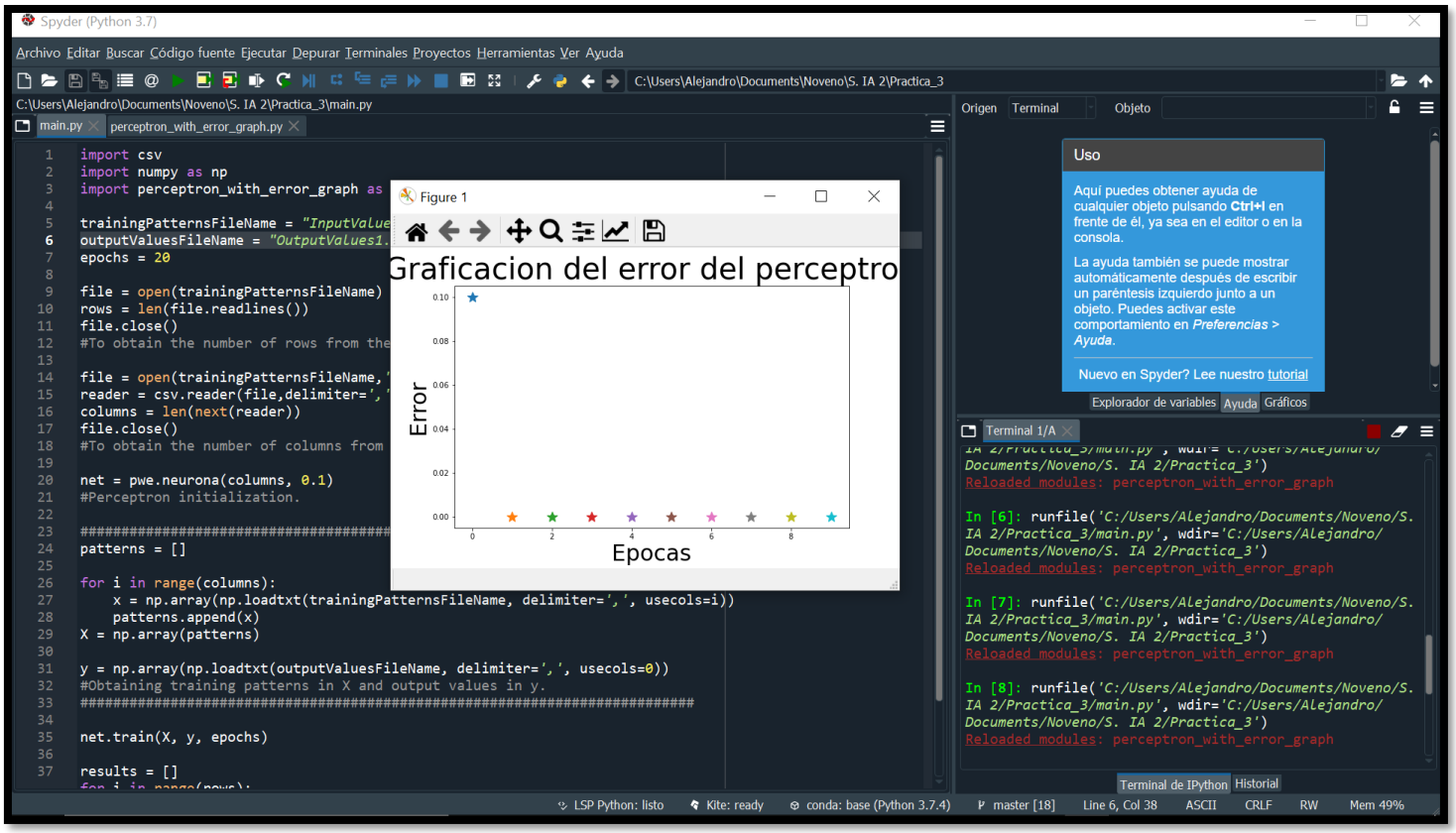
Nuestra función para graficar se puede observar en la última imagen, en donde personalizamos las fuentes, graficamos los puntos, modificamos colores y dimensiones, y damos una pausa entre cada punto para hacer más dinámico el proceso.

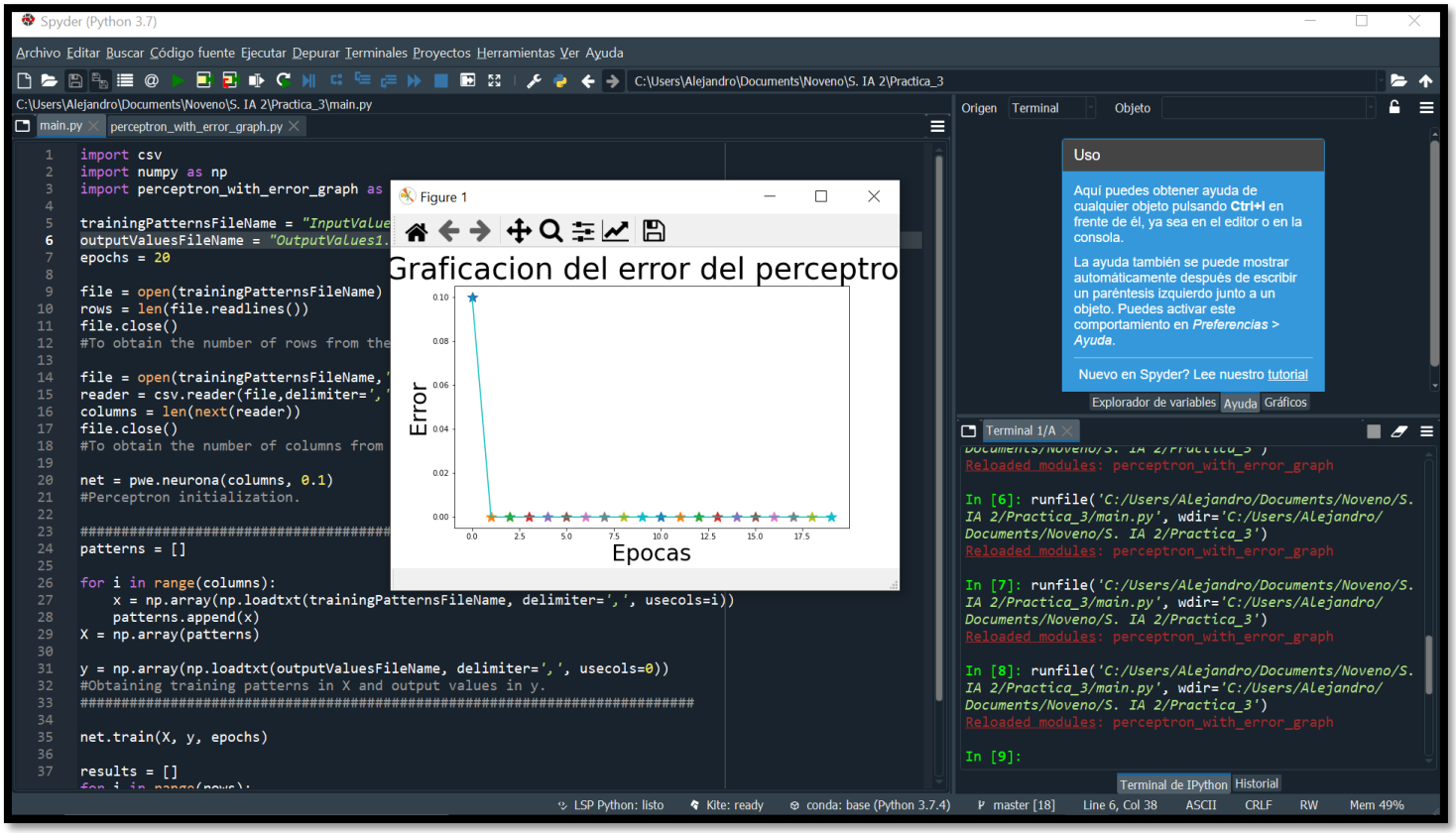
Resultados

A continuación, veremos el entrenamiento y resultados del perceptrón con la lógica de una compuerta AND en 3 dimensiones (3 entradas) y con la compuerta OR en 5 dimensiones (o 5 entradas).



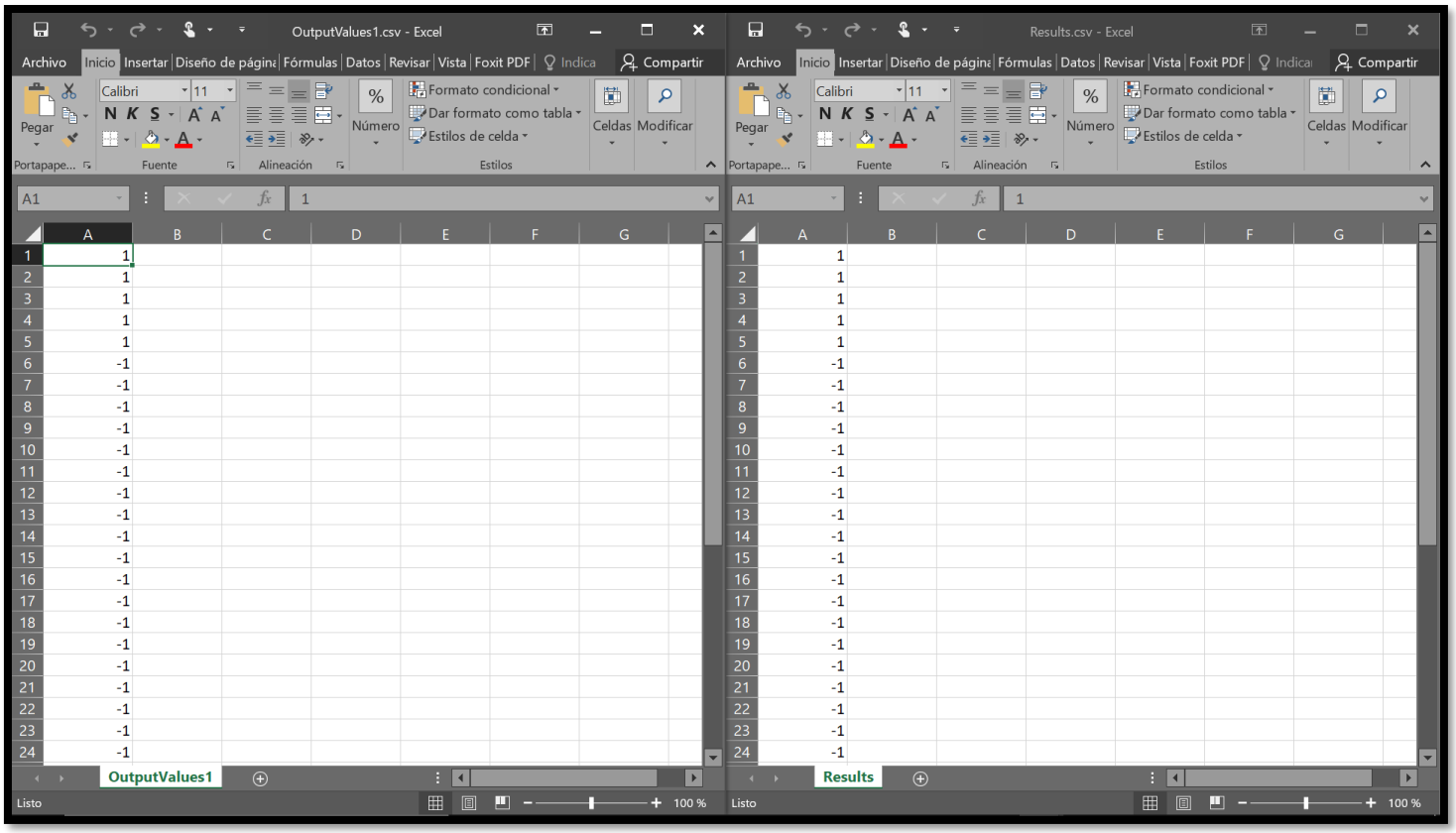
En la imagen anterior podemos observar una parte de los patrones de aprendizaje para el perceptrón con la compuerta AND en 3 dimensiones. Ahora observemos los resultados obtenidos.





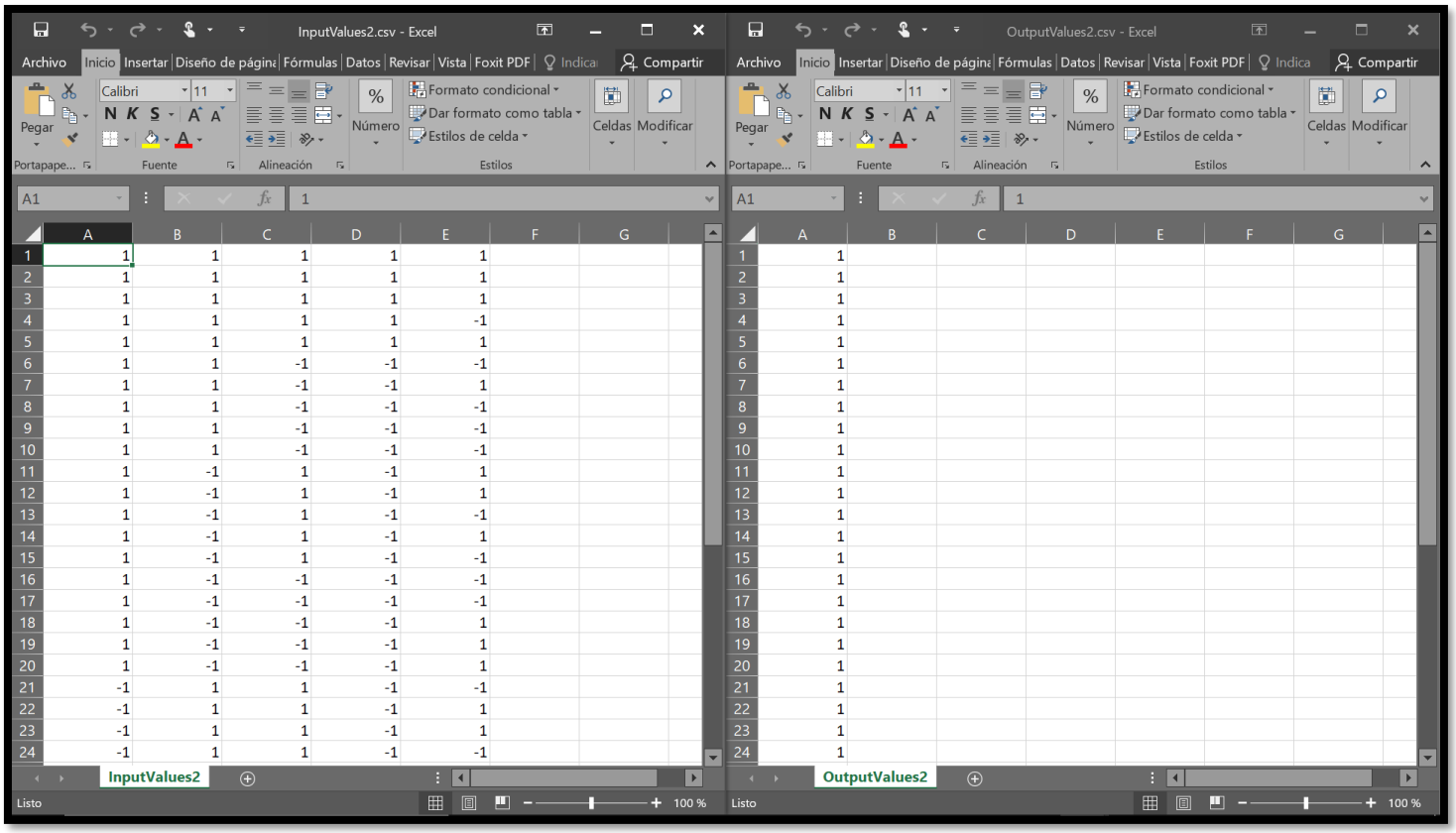
Como podemos observar, los resultados han sido satisfactorios y bastante rápidos, ya que se le otorgaron 20 épocas para aprender, y en la segunda ya logró resolver el problema.

Ahora haremos una comparación entre el archivo de salidas deseadas y el de salidas predichas, para confirmar el correcto entrenamiento.

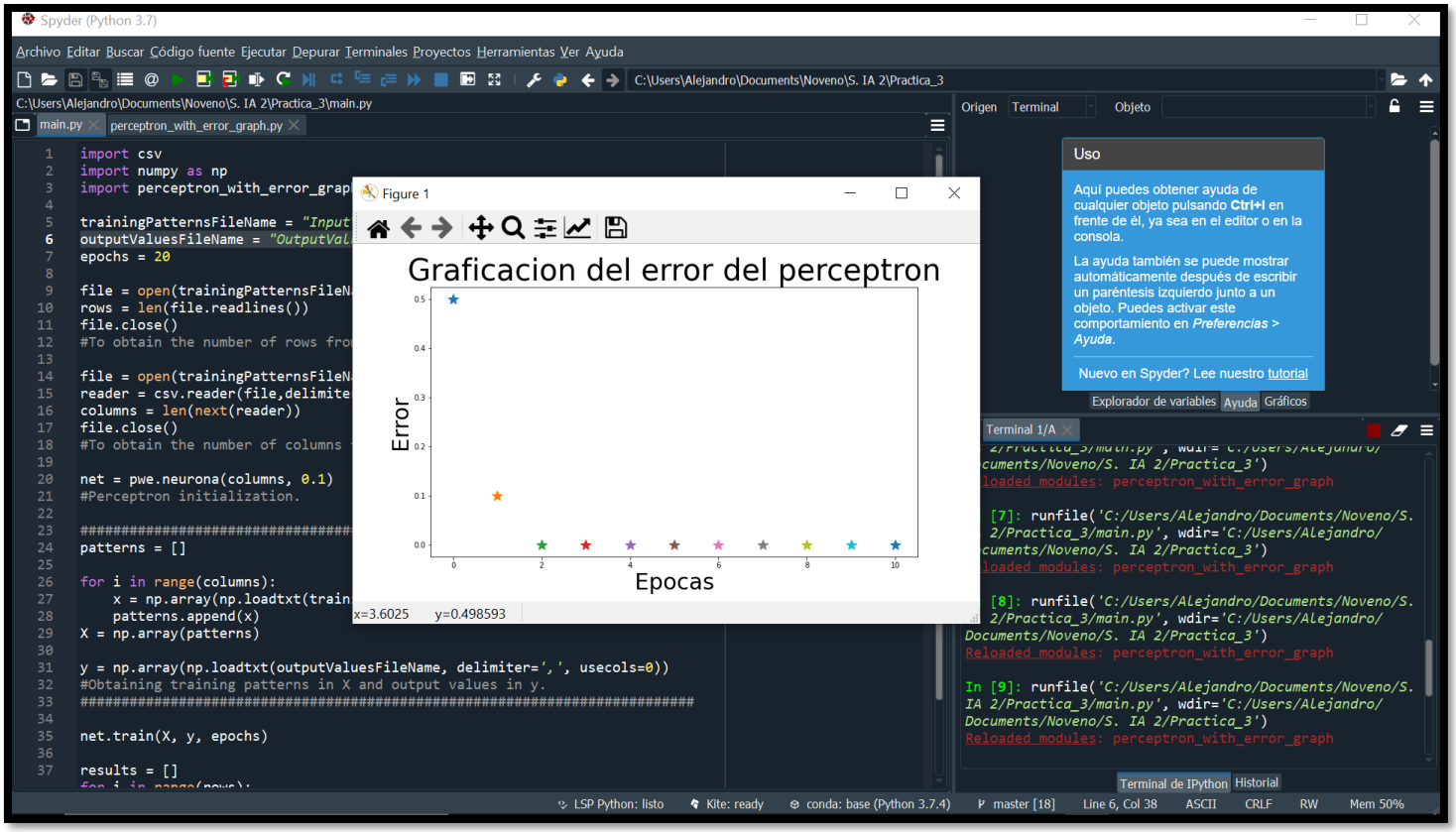


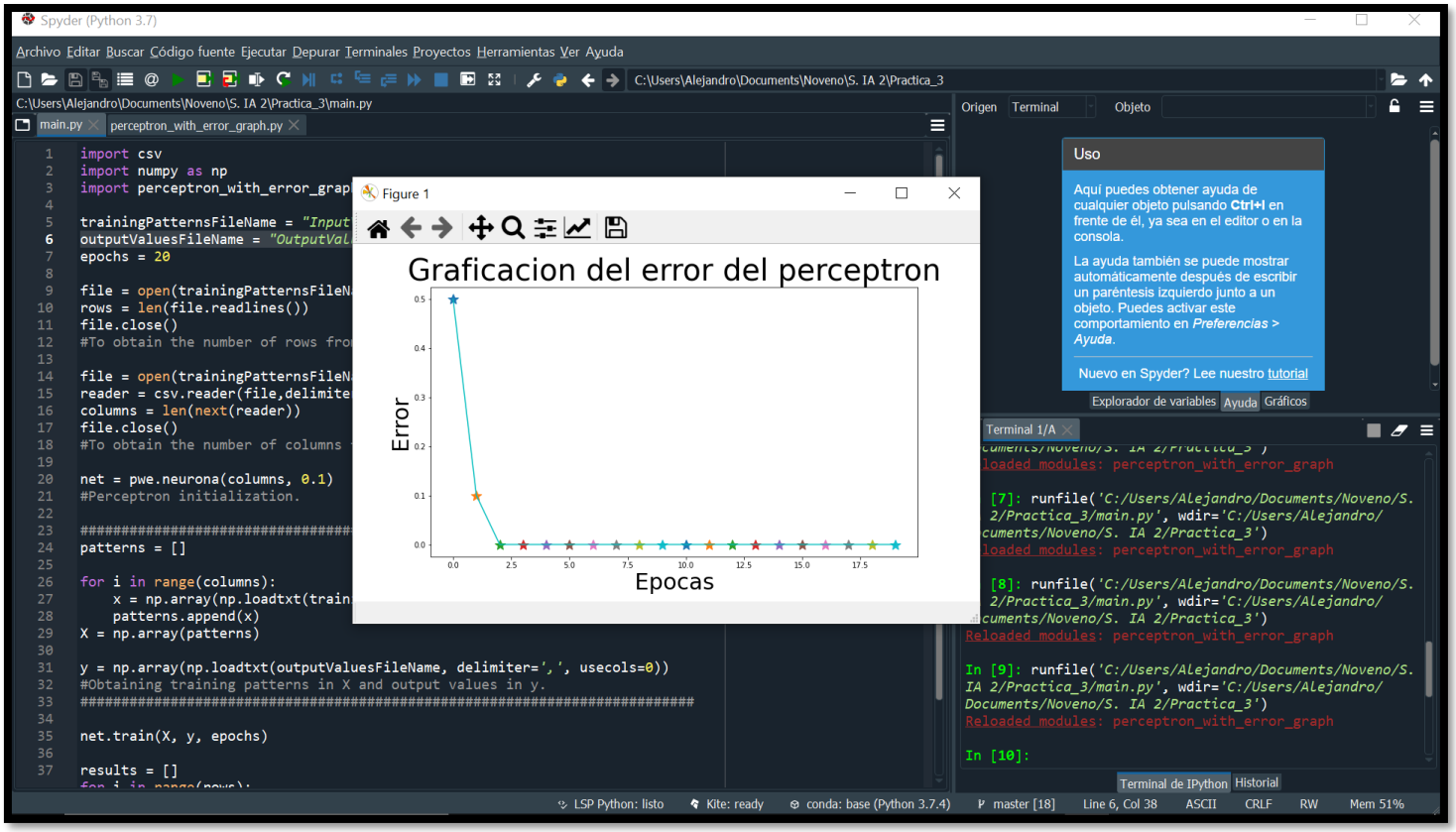
Como podemos observar en la vista previa, los resultados son correctos.

Ahora observemos el caso de la compuerta OR en 5 dimensiones.



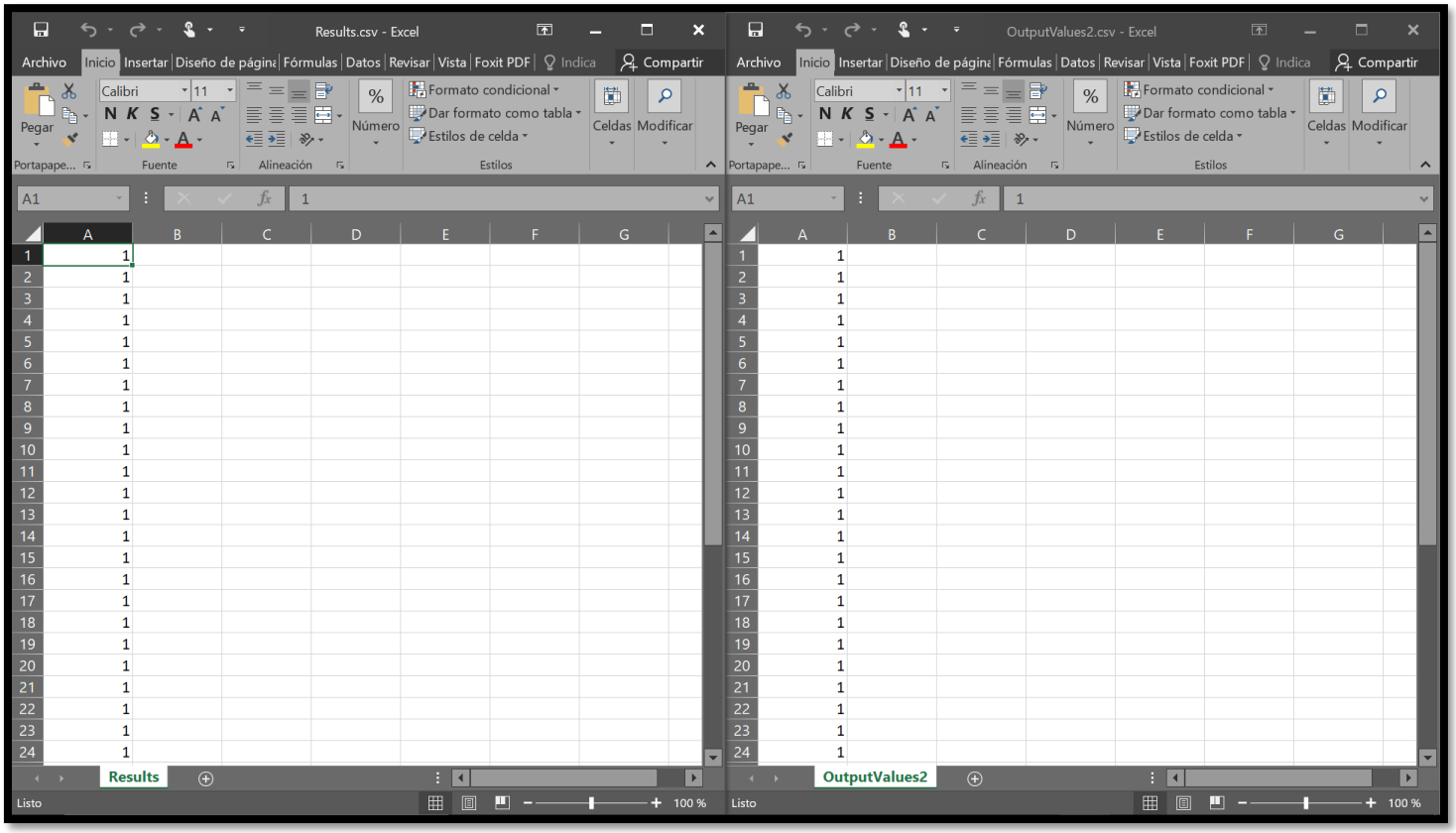
Aquí podemos ver la vista previa del dataset mencionado. Ahora observemos su entrenamiento.





Como podemos observar, el entrenamiento le ha costado una época más que en el caso anterior, lo cual es bastante impresionante considerando que se aumentaron dos dimensiones al problema.

Finalmente comparemos el archivo de salidas deseadas con el de resultados predichos por el perceptrón entrenado.



Una vez más los resultados son completamente correctos y satisfactorios.

Conclusión

Como podemos observar, resulta completamente posible entrenar un perceptrón en más de dos dimensiones, con datasets mucho más grandes y sin disminuir su funcionalidad general. Sin embargo, es posible que su desemepeño se vea afectado si generamos datos más complejos inclusive en menos dimensiones. Eso quedará como ejecución y prueba fuera de entrega de práctica para mí, para experimentar y entender más acerca de este funcionamiento más a fondo.

Código

Main.py:

```
import csv
import numpy as np
import perceptron_with_error_graph as pwe

trainingPatternsFileName = "InputValues2.csv"
outputValuesFileName = "OutputValues2.csv"
epochs = 20

file = open(trainingPatternsFileName)
rows = len(file.readlines())
file.close()
#To obtain the number of rows from the CSV file

file = open(trainingPatternsFileName,'r')
reader = csv.reader(file,delimiter=',')
columns = len(next(reader))
file.close()
#To obtain the number of columns from the CSV file

net = pwe.neurona(columns, 0.1)
#Perceptron initialization.

##############################################################################
patterns = []

for i in range(columns):
    x = np.array(np.loadtxt(trainingPatternsFileName, delimiter=',', usecols=i))
    patterns.append(x)
X = np.array(patterns)

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

net.train(X, y, epochs)

results = []
for i in range(rows):
    prediction = net.predict(X[:, i])
    results.append(prediction)

np.savetxt("Results.csv", results, delimiter=",", fmt='%.0f')
```

perceptron_with_error_graph.py:

```
import numpy as np
import matplotlib.pyplot as plt

x_axis = []
y_axis = []

plt.ylabel('Error', fontsize=30)
plt.xlabel('Epocas', fontsize=30)
plt.title('Graficacion del error del perceptron', fontsize=40)

def graphError(x_coordinate, y_coordinate):
    plt.scatter(x_coordinate, y_coordinate)
    plt.plot(x_coordinate, y_coordinate, '*', markersize=14)
    plt.pause(0.3)

class neurona:
    def __init__(self, dim, eta):   #Dimension y coeficiente de aprendizaje.
        self.n = dim
        self.eta = eta
        self.w = -1 + 2 * np.random.rand(dim, 1)  #x = min + (max - min)*rand()
        self.b = -1 + 2 * np.random.rand()

    def predict(self, x):
        y = np.dot(self.w.transpose(), x) + self.b
        if y >= 0:
            return 1
        else:
            return -1

    def train(self, X, y, epochs):  #x = matriz de entrenamiento, y = vector con resultado esperados, epochs = épocas.
        n, m = X.shape
        #n = 2. m = 4 en el ejemplo de la compuerta AND, OR y XOR.
        average = 0
        for i in range(epochs):
            for j in range(m):
                y_pred = self.predict(X[:, j])
                #what that line did is sliced the array,
                #taking all rows (:) but keeping the column (j)
                if y_pred != y[j]:  #Si nuestro estimado es diferente a nuestro esperado, entrenamos.
                    self.w += self.eta*(y[j] - y_pred) * X[:, j].reshape(-1, 1)
                    self.b += self.eta*(y[j] - y_pred)
                average += (y[j] - y_pred)**2
            average /= m
            graphError(i, average)
            x_axis.append(i)
            y_axis.append(average)
            average = 0

        plt.plot(y_axis, 'c')
```

Link al repositorio

- https://github.com/TheGenesisX/S_IA_2/tree/master/Practica_3