

Centro Universitario de Ciencias Exactas e Ingenierías

Licenciatura en Ingeniería en Computación

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

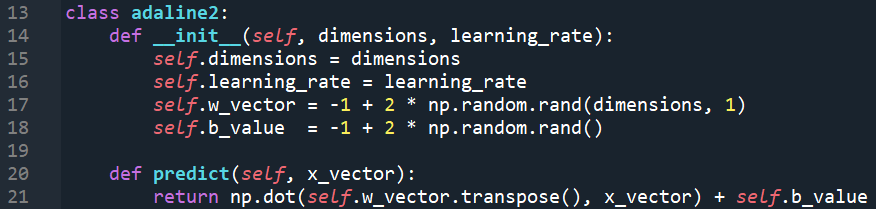
Profesor: Valdés López Julio Esteban

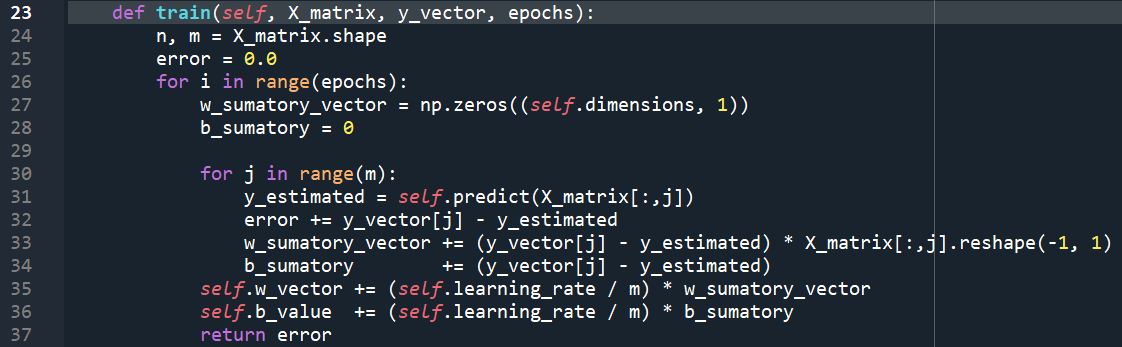
Estudiante: Silva Moya José Alejandro. Código: 213546894.

**Práctica V: ADALINE Batch Gradient Descent**

**Problema a resolver:** Programar un ADALINE BGD para poder resolver un problema de machine learning de una manera más eficiente al proceso realizado con un único perceptrón, o con perceptrones en unicapa.

**Desarrollo**

En el proceso de desarrollo y aprendizaje de una neurona en el modelo del ADALINE, comparado con el perceptrón, los principales aspectos que cambian son la función de predicción y la manera en que se entrena a la neurona. Observemos los puntos importantes.

Como podemos observar, se confirma que los aspectos de inicialización de la neurona se mantienen iguales que en el perceptrón; sin embargo, el cambio importante viene en la función de predicción, en donde podemos ver que se ha eliminado la función de activación, y en cada ocasión retornaremos directamente el producto punto de los vectores, junto con la suma del bias.

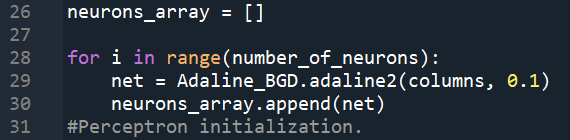
En la imagen anterior podemos ver la función de entrenamiento. El primer cambio importante se encuentra dentro del ciclo de las épocas, en donde generamos un vector de pesos sinápticos inicializado en ceros, que tendrá las mismas dimensiones que el vector de pesos original de la neurona. Además de esto, también tendremos un bias temporal o auxiliar.

A continuación, obtenemos la estimación de la neurona y el error para graficar, e iremos acumulando en nuestras variables temporales:

* El error multiplicado por el vector de entradas correspondiente a la iteración, en W temporal.
* El error, en el bias temporal.

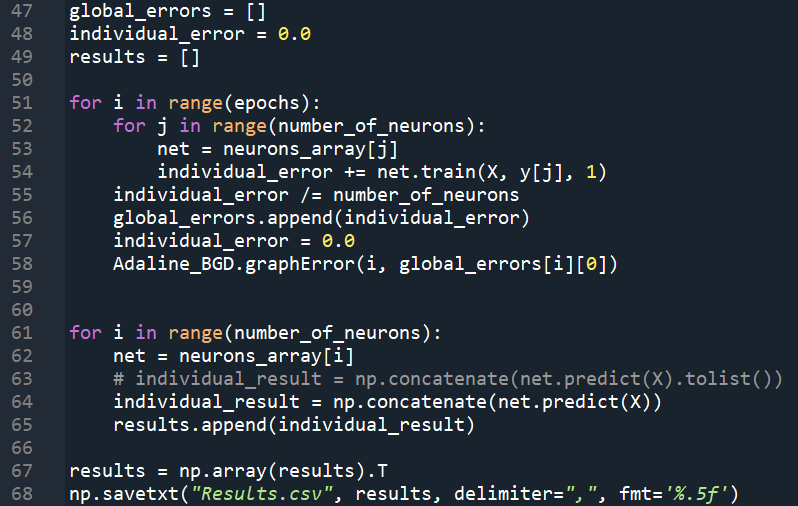
Al terminar la iteración de los patrones de entrenamiento – justo antes de comenzar una nueva época – actualizaremos los valores oficiales de nuestro vector de pesos sinápticos y de nuestro bias.

Esto representa los aspectos y cambios más importantes del programa principal de la presente entrega. Ahora podemos proceder a observar la función main y entender cómo se están procesando los datos.



Comenzamos por generar un arreglo de neuronas de acuerdo a la cantidad de mismas que necesitaremos para resolver el problema en cuestión.

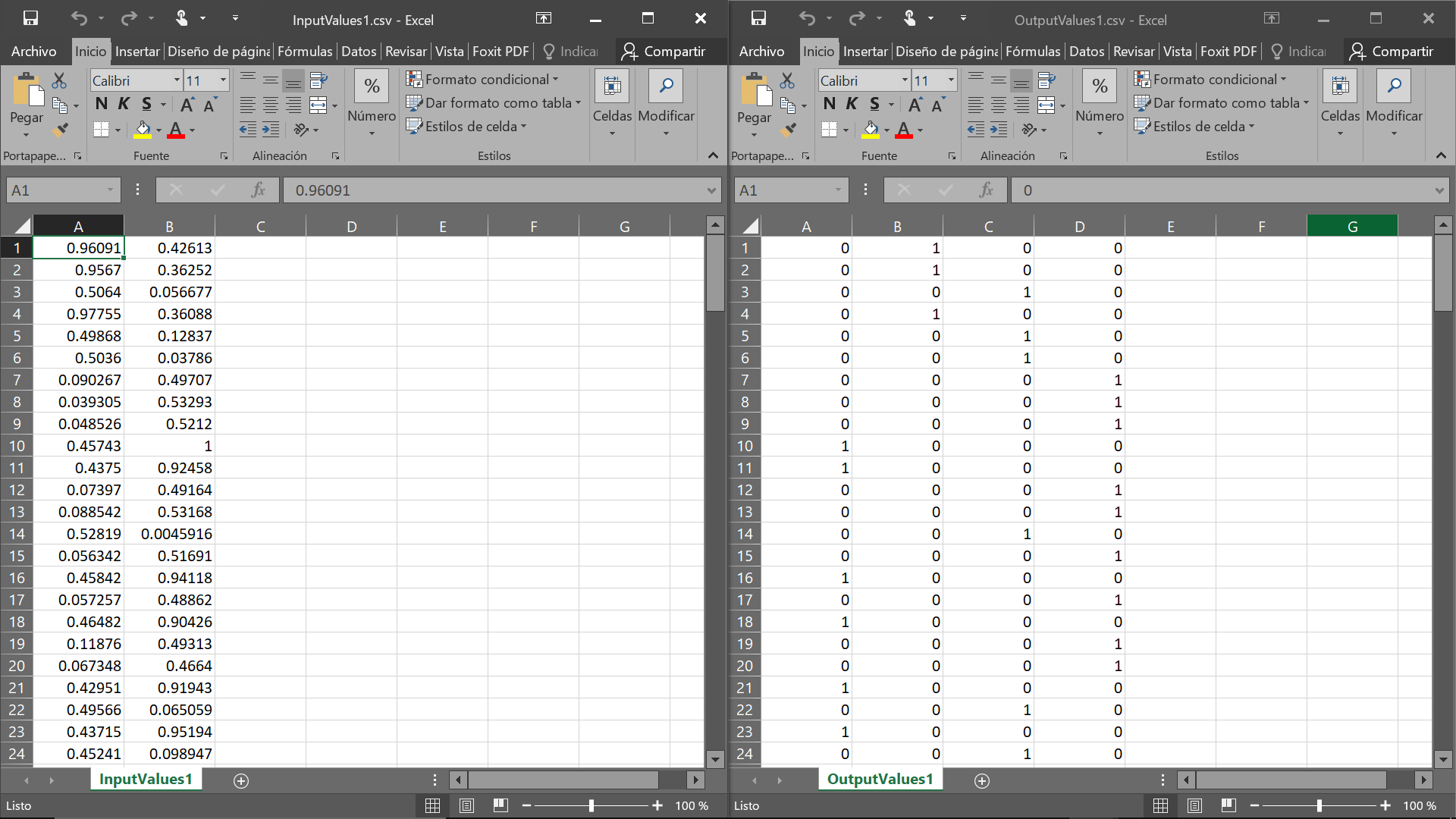
La carga de datos de patrones de aprendizaje y salidas deseadas se mantiene funcionando de la misma manera que lo hacía con anterioridad.



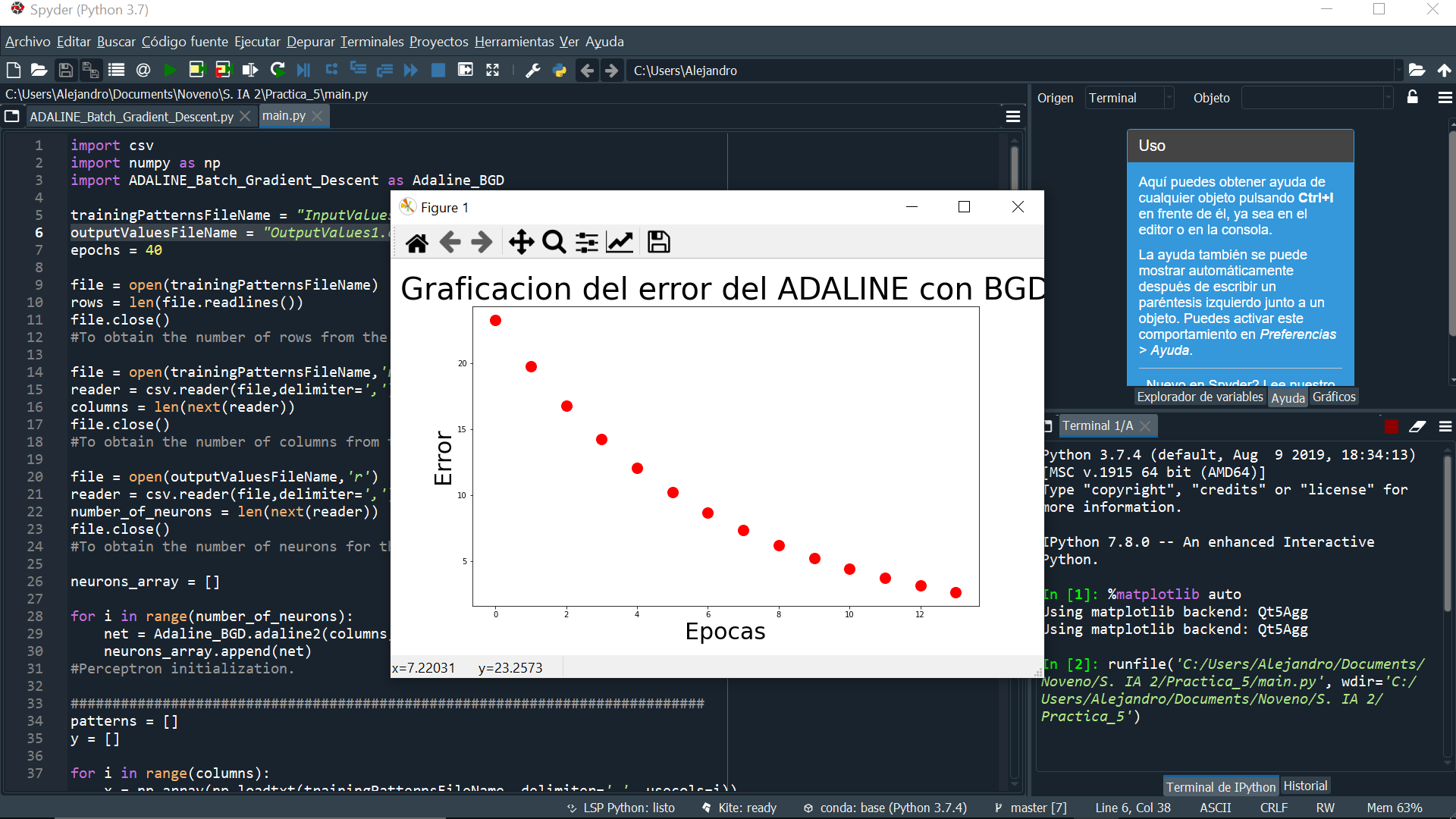
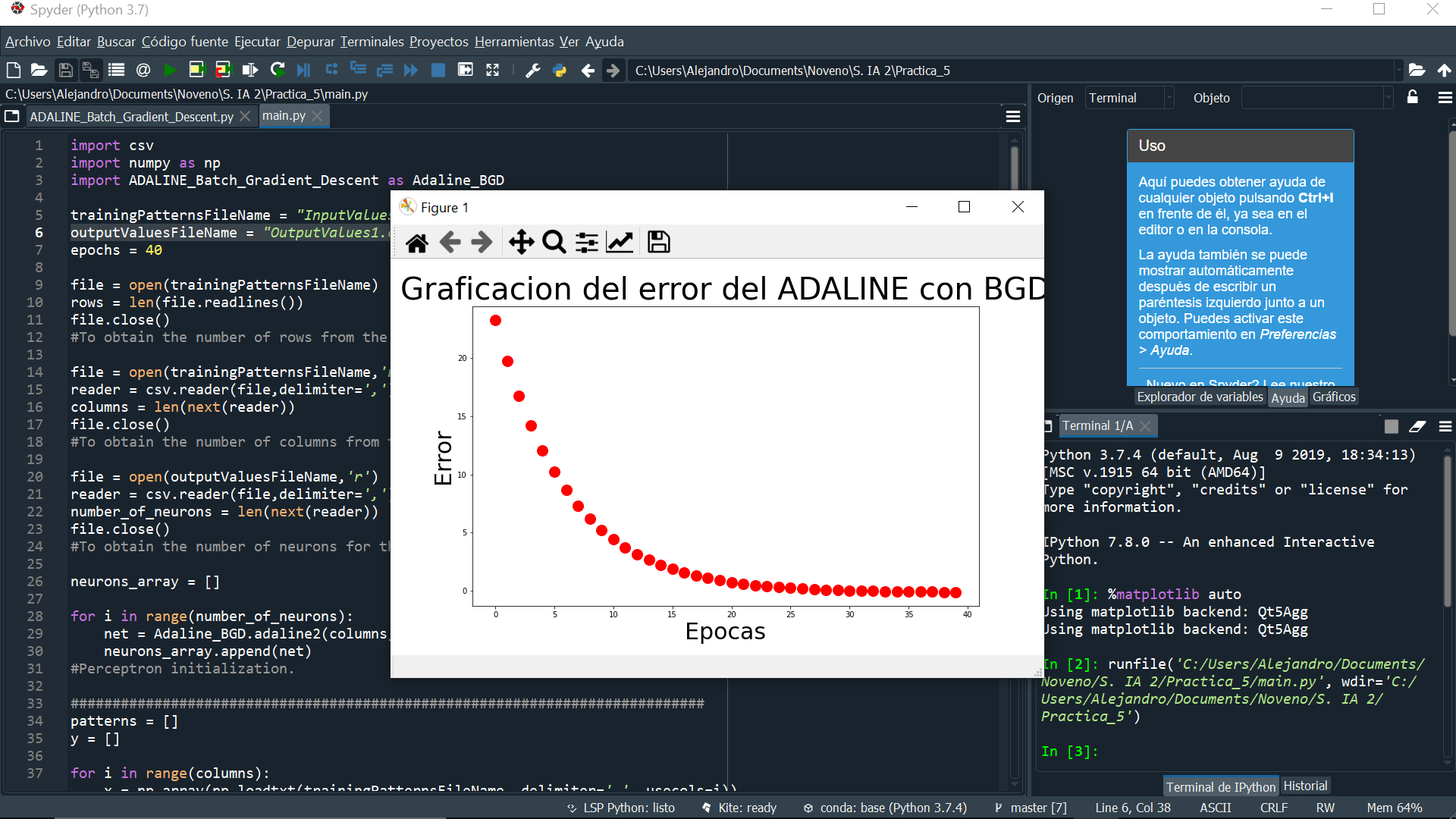
El proceso de entrenamiento sí cambia de manera sustancial. Observemos en qué consiste.

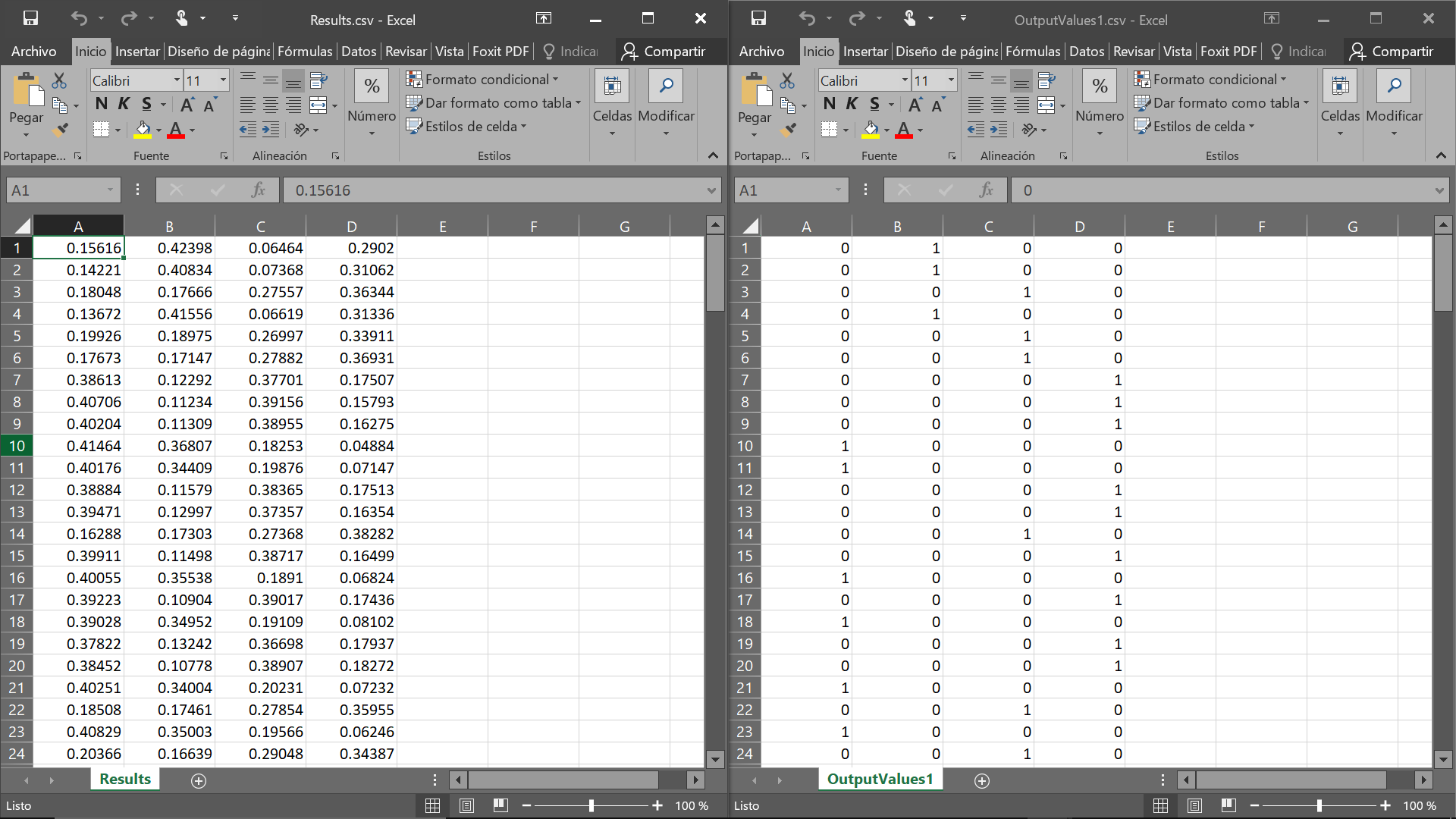
Comenzamos entrenando las neuronas de acuerdo a la cantidad de épocas que hayamos delimitado al inicio del código; de esta manera podemos simular paralelismo en el entrenamiento de las neuronas del ADALINE sin tener que recurrir a la implementación de hilos. Una vez que entrenemos todas las neuronas por una época completa obtendremos el error de entrenamiento, su promedio, y lo agregaremos a una lista de datos para poder graficar correctamente una línea además de los puntos.

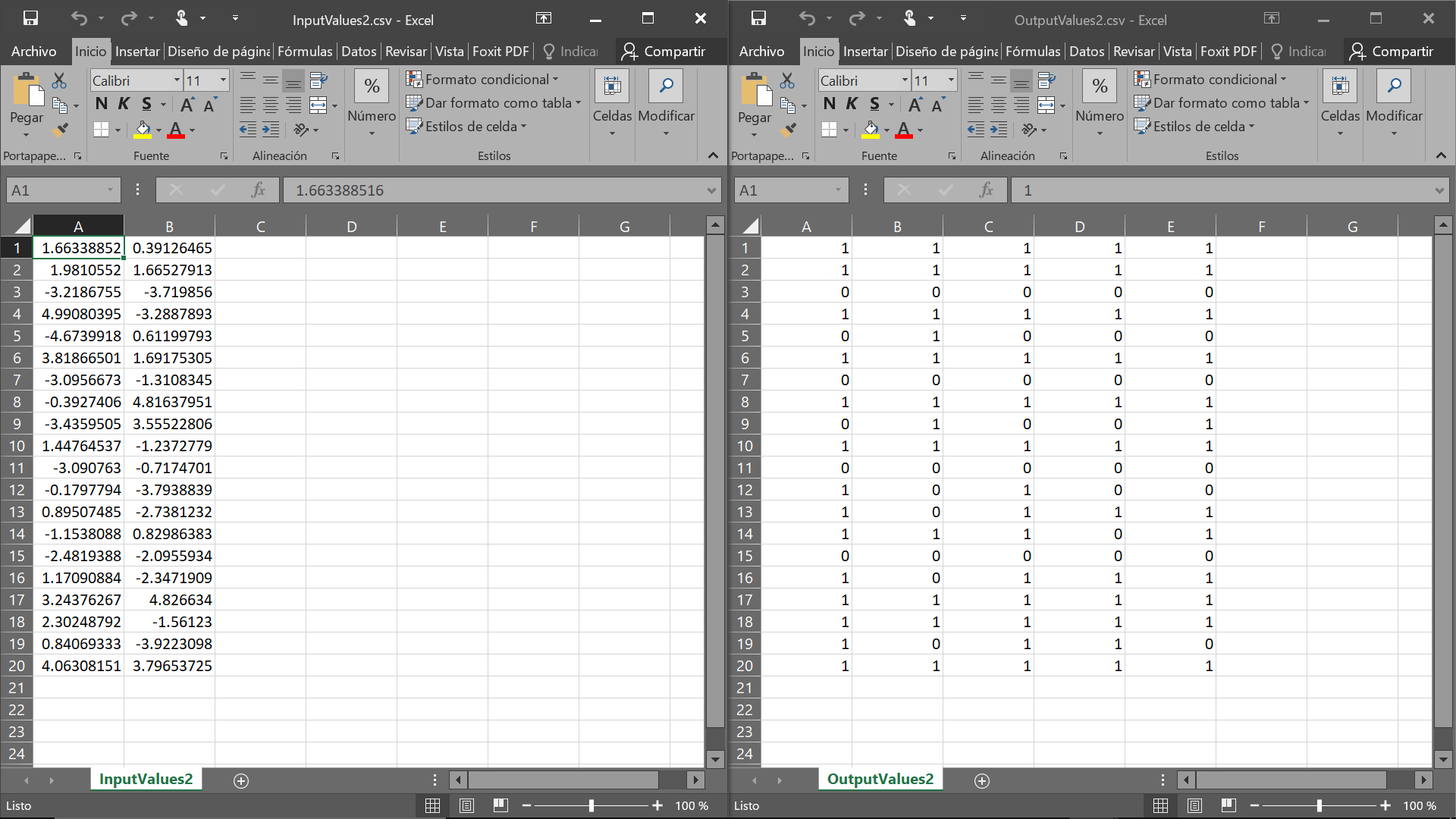
Finalmente, para cada neurona en el modelo obtendremos la predicción de los valores de entrenamiento y los agregaremos a un arreglo para poder mandarlos al csv de resultados finales, que podremos comparar con el documento de salidas esperadas. Evidentemente no serán iguales, debido a que las salidas esperadas son binarias, mientras que las salidas predichas por el modelo son discretas; sin embargo, resulta sencillo observar y deducir si el modelo ha sido entrenado de manera correcta, además de tener la gráfica del error como un dato visual extra.

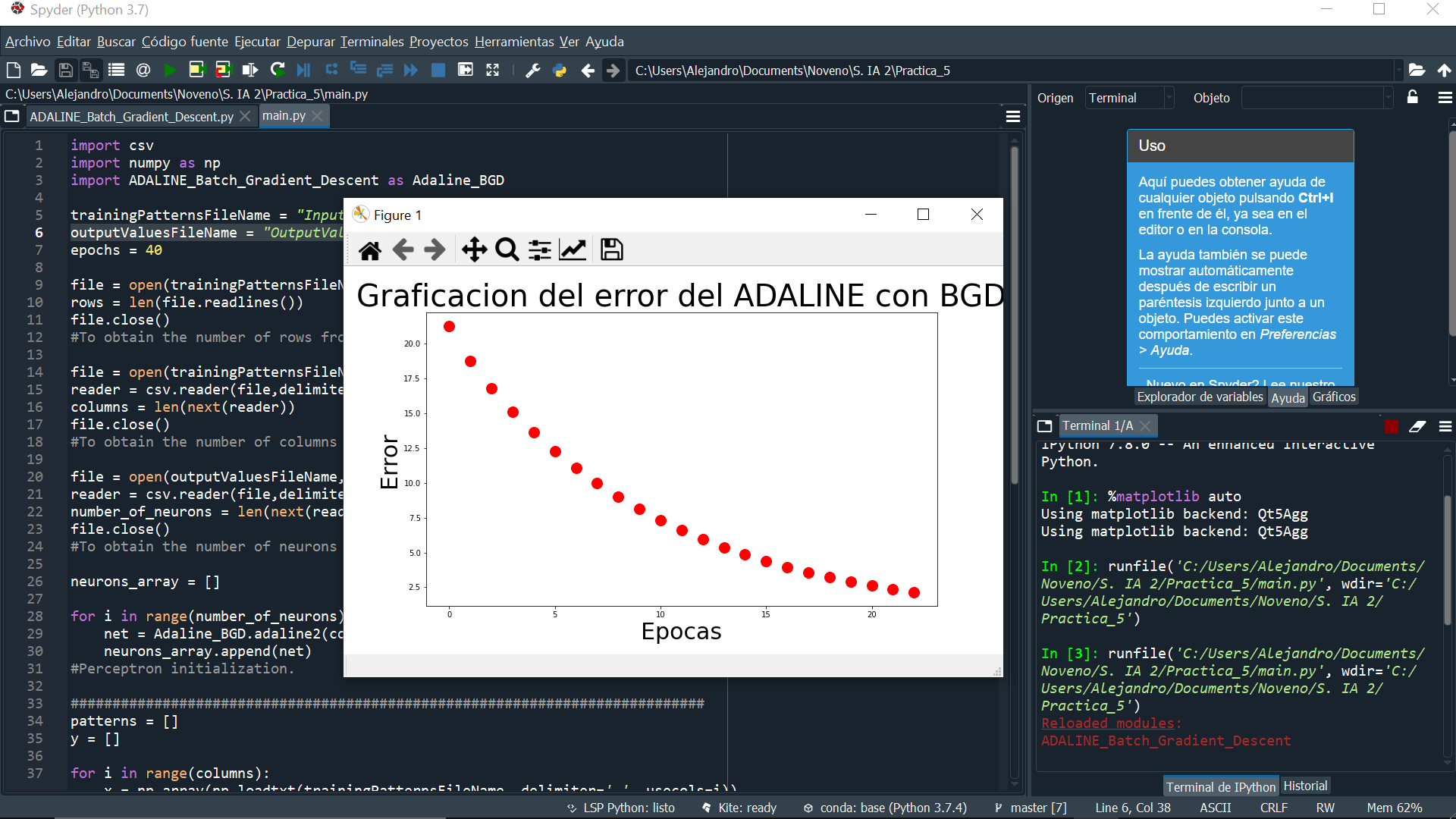
**Resultados**

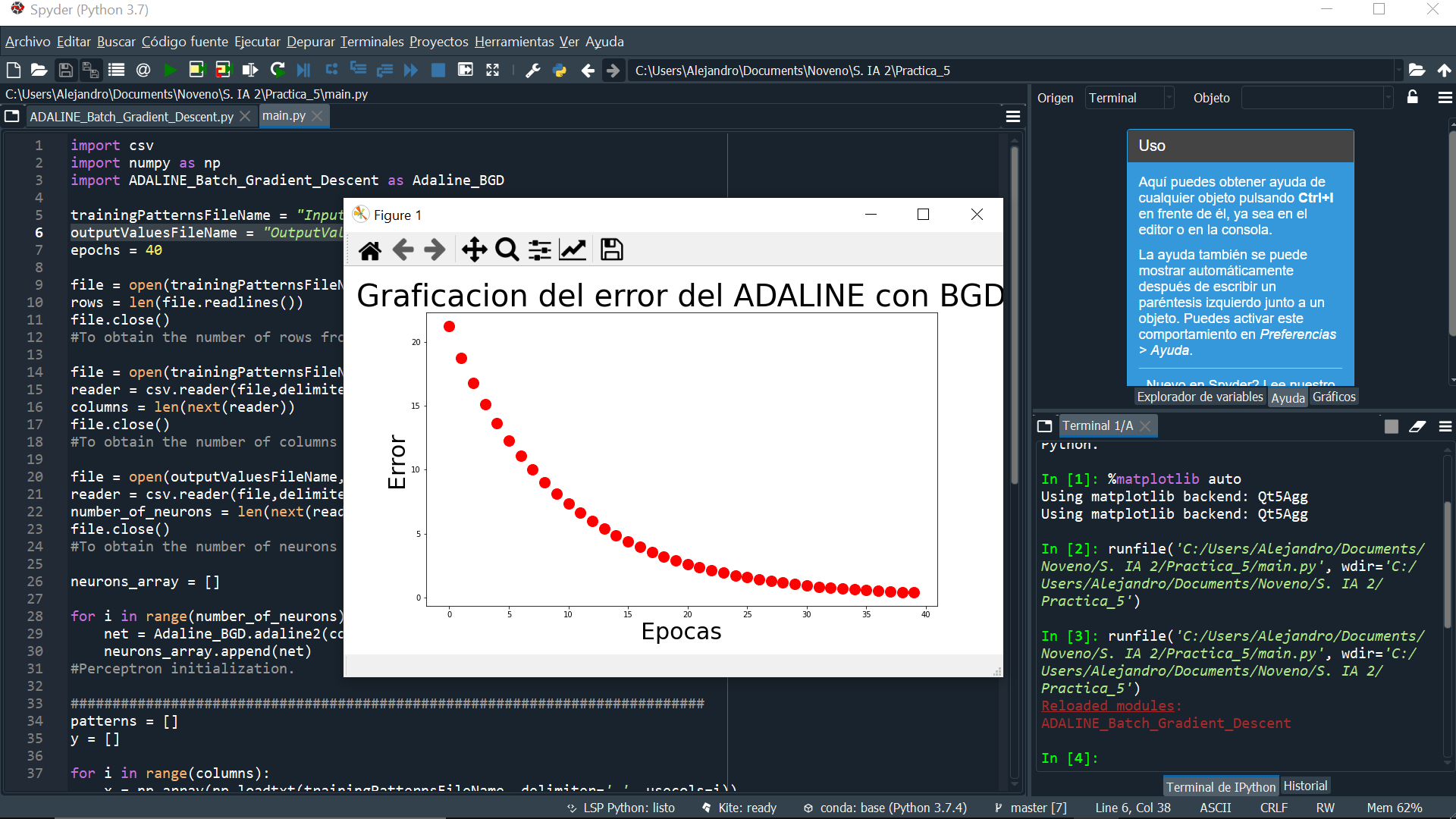
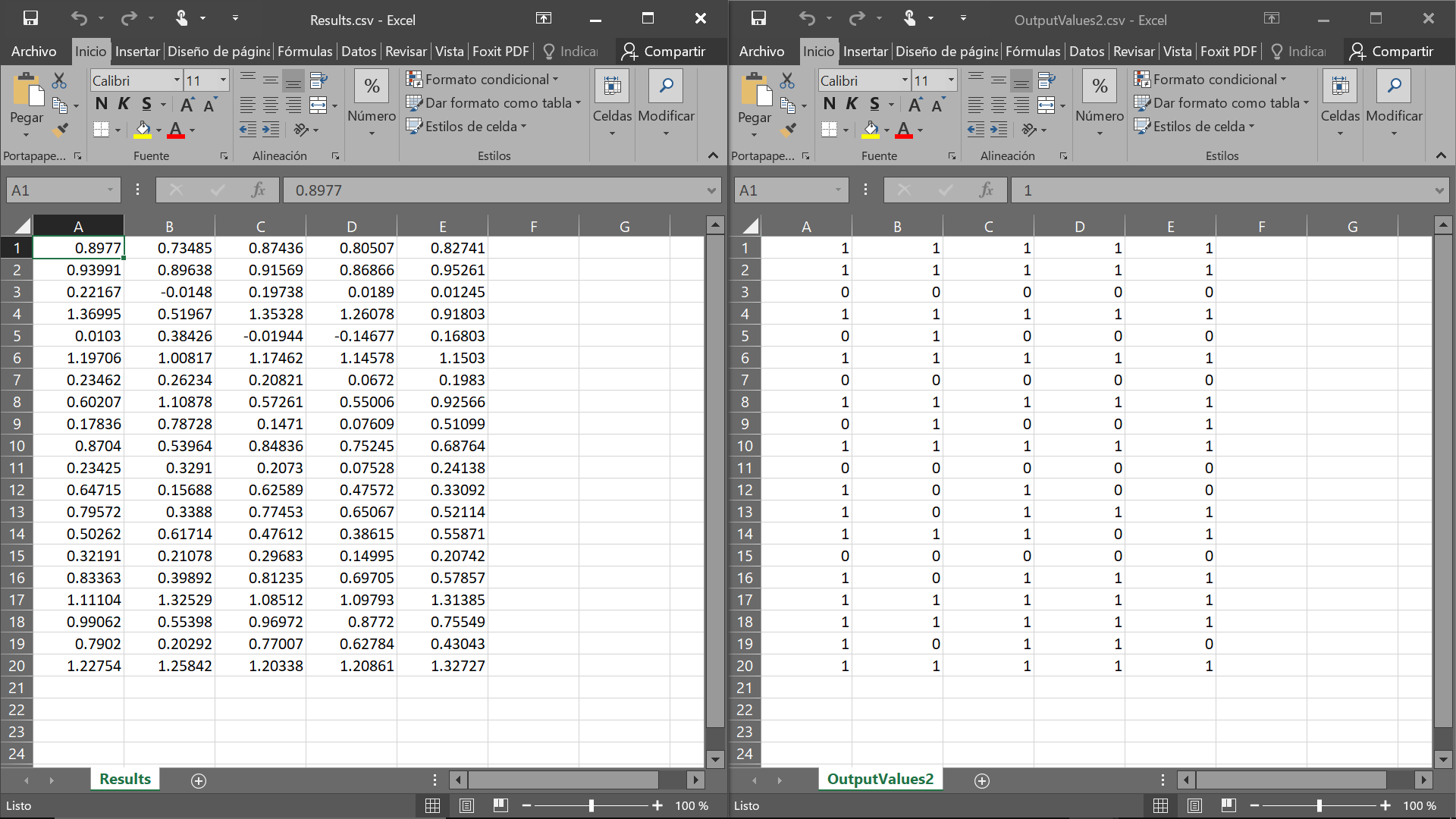
Aquí podemos observar el primero de los dos datasets que hemos delimitado para entrenar a modelo. Es un problema en dos dimensiones que se resuelve con 4 neuronas. Observemos el desempeño y sus resultados.

Como podemos observar, el error se encuentra considerablemente cercano a cero después de 40 épocas de entrenamiento. Observemos los resultados numéricos.

Como podemos observar, exitosamente los valores que debería ser 1 en el archivo de salidas deseadas son cercanos a 1 en el archivo de salidas predichas, mientras que aquellos que deberían ser ceros están bastante cercanos a dicho valor, por lo cual podemos concluir que el entrenamiento ha sido bastante favorable. Ahora observemos los resultados con el segundo dataset.

Este modelo igualmente es de dos dimensiones y tiene una cantidad mucho menor de datos de entrada; sin embargo, requiere de cinco neuronas para poder resolverse, por lo que se vuelve un poco más complejo. Observemos el entrenamiento y los resultados.



Una vez más podemos observar cómo en entrenamiento de las neuronas de este modelo ha sido exitoso, obteniendo resultados con error muy cercanos a cero. Veamos los CSV.

Resulta nuevamente apreciable que los valores, si bien no son binarios, son bastante cercanos a los correspondientes que deberían de ser. De esta manera podemos entender que le modelo está entrenando de una manera satisfactoria.

**Conclusión**

El modelo aplicado en esta entrega difiere ligeramente con los anteriores estudiados; sin embargo, demuestra tener una ejecución bastante más eficiente, entregando resultados iguales o incluso mejores, por lo que se convierte en una opción más viable en general gracias a que trabaja con una función de optimización, y de esa manera nos evitamos estar fluctuando entre puntos altos y bajos como lo llegaba a hacer en ciertos momentos el perceptrón, llegando en ocasiones al escenario en que era virtualmente incapaz de aprender. En este caso corregimos este error y logramos obtener resultados completamente satisfactorios.

**Código**

ADALINE\_Batch\_Gradient\_Descent.py

import numpy as np

import matplotlib.pyplot as plt

plt.ylabel('Error', fontsize=30)

plt.xlabel('Epocas', fontsize=30)

plt.title('Graficacion del error del ADALINE con BGD', fontsize=40)

def graphError(x\_coordinate, y\_coordinate):

plt.scatter(x\_coordinate, y\_coordinate)

plt.plot(x\_coordinate, y\_coordinate, 'ro', markersize=14)

plt.pause(0.3)

class adaline2:

def \_\_init\_\_(self, dimensions, learning\_rate):

self.dimensions = dimensions

self.learning\_rate = learning\_rate

self.w\_vector = -1 + 2 \* np.random.rand(dimensions, 1)

self.b\_value = -1 + 2 \* np.random.rand()

def predict(self, x\_vector):

return np.dot(self.w\_vector.transpose(), x\_vector) + self.b\_value

def train(self, X\_matrix, y\_vector, epochs):

n, m = X\_matrix.shape

error = 0.0

for i in range(epochs):

w\_sumatory\_vector = np.zeros((self.dimensions, 1))

b\_sumatory = 0

for j in range(m):

y\_estimated = self.predict(X\_matrix[:,j])

error += y\_vector[j] - y\_estimated

w\_sumatory\_vector += (y\_vector[j] - y\_estimated) \* X\_matrix[:,j].reshape(-1, 1)

b\_sumatory += (y\_vector[j] - y\_estimated)

self.w\_vector += (self.learning\_rate / m) \* w\_sumatory\_vector

self.b\_value += (self.learning\_rate / m) \* b\_sumatory

return error

main.py

import csv

import numpy as np

import ADALINE\_Batch\_Gradient\_Descent as Adaline\_BGD

trainingPatternsFileName = "InputValues2.csv"

outputValuesFileName = "OutputValues2.csv"

epochs = 40

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

file = open(outputValuesFileName,'r')

reader = csv.reader(file,delimiter=',')

number\_of\_neurons = len(next(reader))

file.close()

#To obtain the number of neurons for the program. The number of output columns tells us the number of neurons.

neurons\_array = []

for i in range(number\_of\_neurons):

net = Adaline\_BGD.adaline2(columns, 0.1)

neurons\_array.append(net)

#Perceptron initialization.

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

patterns = []

y = []

for i in range(columns):

x = np.array(np.loadtxt(trainingPatternsFileName, delimiter=',', usecols=i))

patterns.append(x)

X = np.array(patterns)

for i in range(number\_of\_neurons):

y.append(np.array(np.loadtxt(outputValuesFileName, delimiter=',', usecols=i)))

#Obtaining training patterns in X and output values in y.

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

global\_errors = []

individual\_error = 0.0

results = []

for i in range(epochs):

for j in range(number\_of\_neurons):

net = neurons\_array[j]

individual\_error += net.train(X, y[j], 1)

individual\_error /= number\_of\_neurons

global\_errors.append(individual\_error)

individual\_error = 0.0

Adaline\_BGD.graphError(i, global\_errors[i][0])

for i in range(number\_of\_neurons):

net = neurons\_array[i]

# individual\_result = np.concatenate(net.predict(X).tolist())

individual\_result = np.concatenate(net.predict(X))

results.append(individual\_result)

results = np.array(results).T

np.savetxt("Results.csv", results, delimiter=",", fmt='%.5f')

**Link al repositorio**

* <https://github.com/TheGenesisX/S_IA_2/tree/master/Practica_5>