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 IV: Perceptrones en unicapa

Problema a resolver: Adaptar la arquitectura del perceptrón de la entrega anterior para lograr que, además de resolver problemas en múltiples dimensiones, sea capaz de procesar y resolver problemas en donde se necesita más de un perceptrón a la vez.

Desarrollo

Las modificaciones a este programa son realmente mínimas, y el 99% de las mismas ocurrieron en el main, mas no en el código del perceptrón.

```
20  file = open(outputValuesFileName,'r')
21  reader = csv.reader(file,delimiter=',')
22  number_of_neurons = len(next(reader))
23  file.close()
24  #To obtain the number of neurons for the
25
26  neurons_array = []
27
28  for i in range(number_of_neurons):
29      net = pwe.neurona(columns, 0.1)
30      neurons_array.append(net)
31  #Perceptron initialization.
```

Lo primero que realizamos es contar la cantidad de columnas en el archivo de salidas esperadas, ya que matemáticamente cada columna corresponde al resultado de los patrones de aprendizaje de un único perceptrón o neurona, de manera que haciendo este conteo podemos determinar cuántas neuronas necesitaremos para resolver el problema presentado. Posterior a eso modificaremos la inicialización de la – antes – única neurona por una inicialización de n neuronas, determinadas por el código anterior.

```
42  for i in range(number_of_neurons):
43      y.append(np.array(np.loadtxt(outputValuesFileName, delimiter=',', usecols=i)))
44  #Obtaining training patterns in X and output values in y.
```

Por la misma naturaleza de la multiplicidad de neuronas, tendremos que cargar más de un solo patrón de salidas al entrenamiento, así que procederemos a cargar cada uno como conjuntos distintos en una misma lista, e iteraremos sobre ellos según lo necesitemos.

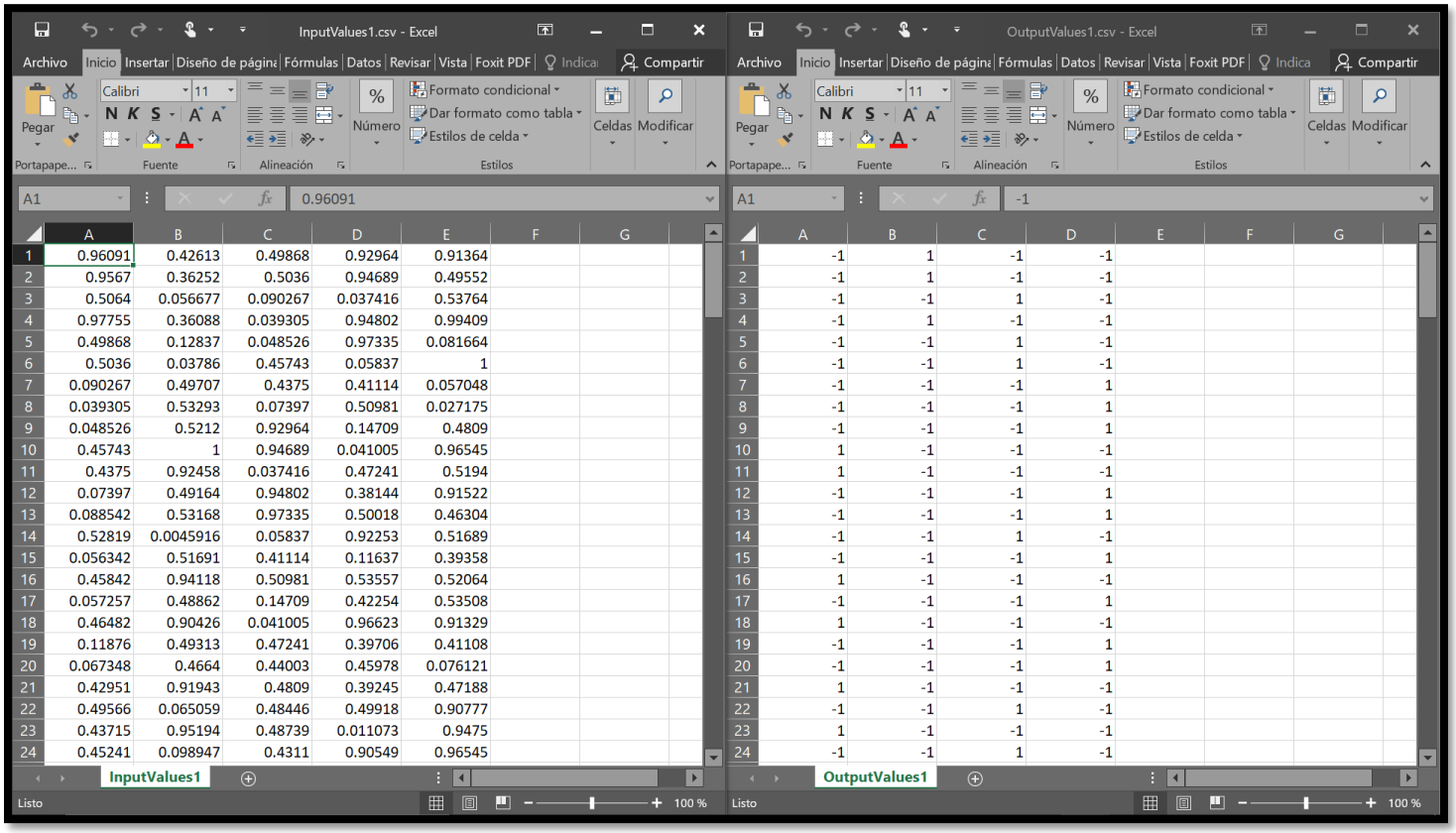
```
47  global_errors = []
48  for i in range(number_of_neurons):
49      net.train(X, y[i], epochs)
50
51      individual_error = []
52      for i in range(rows):
53          prediction = net.predict(X[:, i])
54          individual_error.append(prediction)
55      global_errors.append(individual_error)
56
57  global_errors = np.array(global_errors).T
58
59  np.savetxt("Results.csv", global_errors, delimiter=",", fmt='%.0f')
```

Finalmente tendremos que graficar los errores de cada una de las neuronas por individual y escribirlas en un archivo csv de resultados, por lo que según el entrenamiento de cada perceptrón obtendremos sus errores, y los añadiremos a una lista de errores globales, y los transpondremos solo para que aparezcan como columnas y no como renglones. Finalmente escribimos los datos en un archivo csv y terminamos con el programa.

La única añadidura que se realizó al código del peceptrón fue una limpieza de datos después de cada graficación de error individual, para evitar que en cada ocasión nueva agregara datos viejos, en lugar de solo graficar lo de cada neurona.

```
y_axis.clear()
```

Resultados



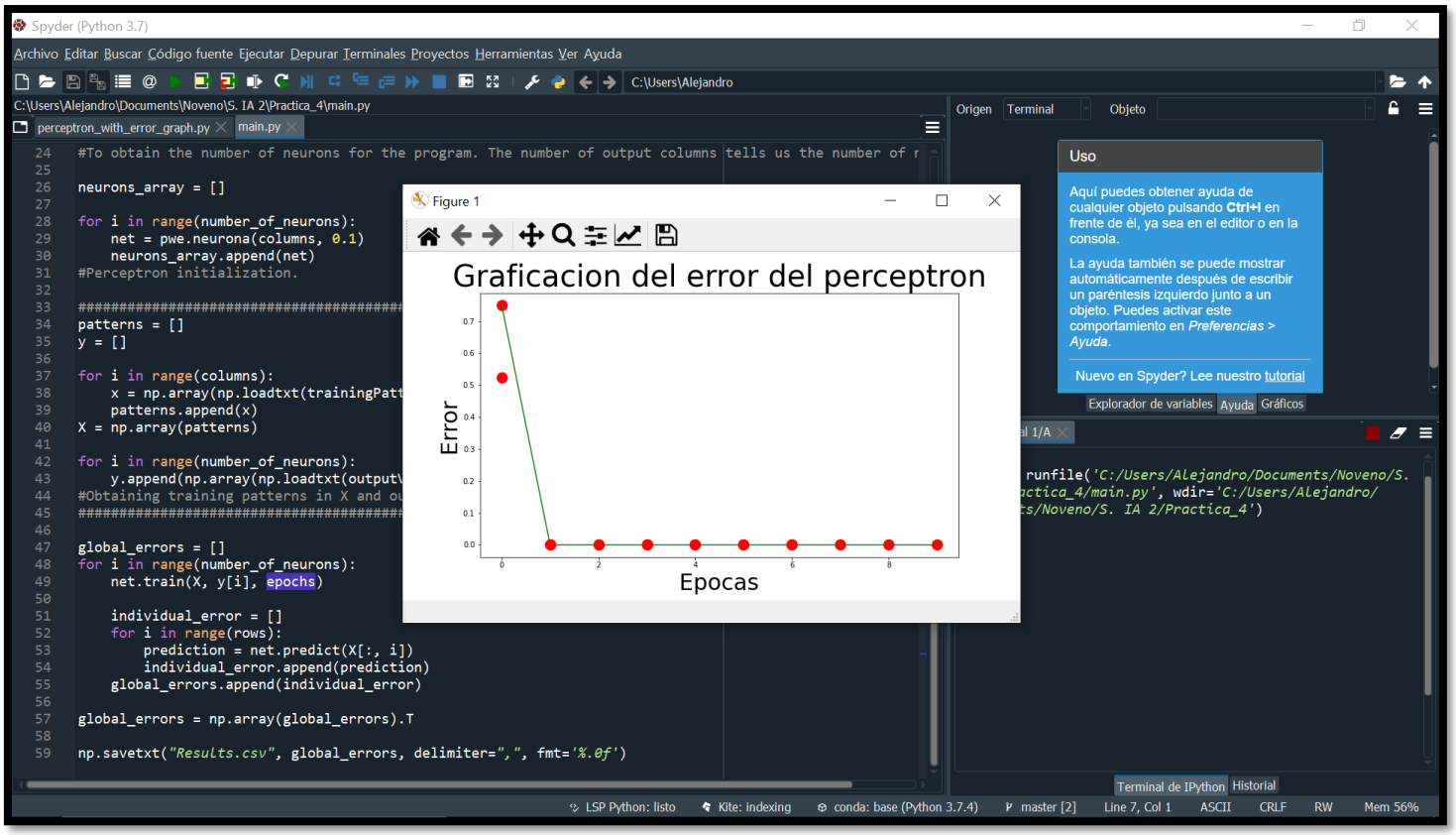
InputValues1.csv - Excel

	A	B	C	D	E	F	G
1	0.96091	0.42613	0.49868	0.92964	0.91364		
2	0.9567	0.36252	0.5036	0.94689	0.49552		
3	0.5064	0.056677	0.090267	0.037416	0.53764		
4	0.97755	0.36088	0.039305	0.94802	0.99409		
5	0.49868	0.12837	0.048526	0.97335	0.081664		
6	0.5036	0.03786	0.45743	0.05837	1		
7	0.090267	0.49707	0.4375	0.41114	0.057048		
8	0.039305	0.53293	0.07397	0.50981	0.027175		
9	0.048526	0.5212	0.92964	0.14709	0.4809		
10	0.45743	1	0.94689	0.041005	0.96545		
11	0.4375	0.92458	0.037416	0.47241	0.5194		
12	0.07397	0.49164	0.94802	0.38144	0.91522		
13	0.088542	0.53168	0.97335	0.50018	0.46304		
14	0.52819	0.0045916	0.05837	0.92253	0.51689		
15	0.056342	0.51691	0.41114	0.11637	0.39358		
16	0.45842	0.94118	0.50981	0.53557	0.52064		
17	0.057257	0.48862	0.14709	0.42254	0.53508		
18	0.46482	0.90426	0.041005	0.96623	0.91329		
19	0.11876	0.49313	0.47241	0.39706	0.41108		
20	0.067348	0.4664	0.44003	0.45978	0.076121		
21	0.42951	0.91943	0.4809	0.39245	0.47188		
22	0.49566	0.065059	0.48446	0.49918	0.90777		
23	0.43715	0.95194	0.48739	0.011073	0.9475		
24	0.45241	0.098947	0.4311	0.90549	0.96545		

OutputValues1.csv - Excel

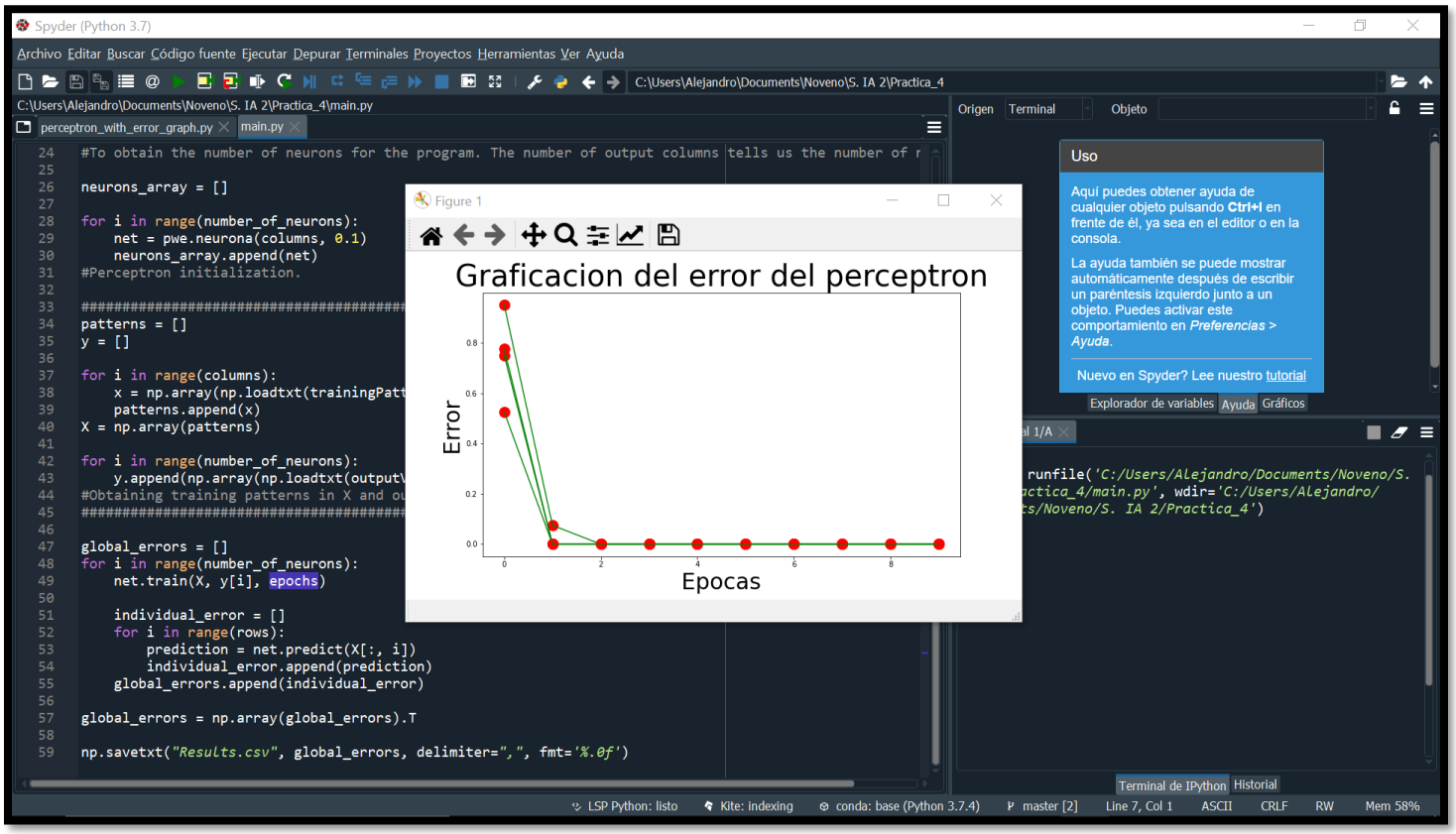
	A	B	C	D	E	F	G
1	-1	1	-1	-1			
2	-1	1	-1	-1			
3	-1	-1	1	-1			
4	-1	1	-1	-1			
5	-1	-1	1	-1			
6	-1	-1	1	-1			
7	-1	-1	-1	1			
8	-1	-1	-1	1			
9	-1	-1	-1	1			
10	1	-1	-1	-1			
11	1	-1	-1	-1			
12	-1	-1	-1	1			
13	-1	-1	-1	1			
14	-1	-1	1	-1			
15	-1	-1	-1	1			
16	1	-1	-1	-1			
17	-1	-1	-1	1			
18	1	-1	-1	-1			
19	-1	-1	-1	1			
20	-1	-1	-1	1			
21	1	-1	-1	-1			
22	-1	-1	1	-1			
23	1	-1	-1	-1			
24	-1	-1	1	-1			

Vamos a entrenar una red unicapa de perceptrones para resolver un problema en cinco dimensiones con 4 clasificaciones. Aquí podemos observar los patrones de entrada y las salidas esperadas.

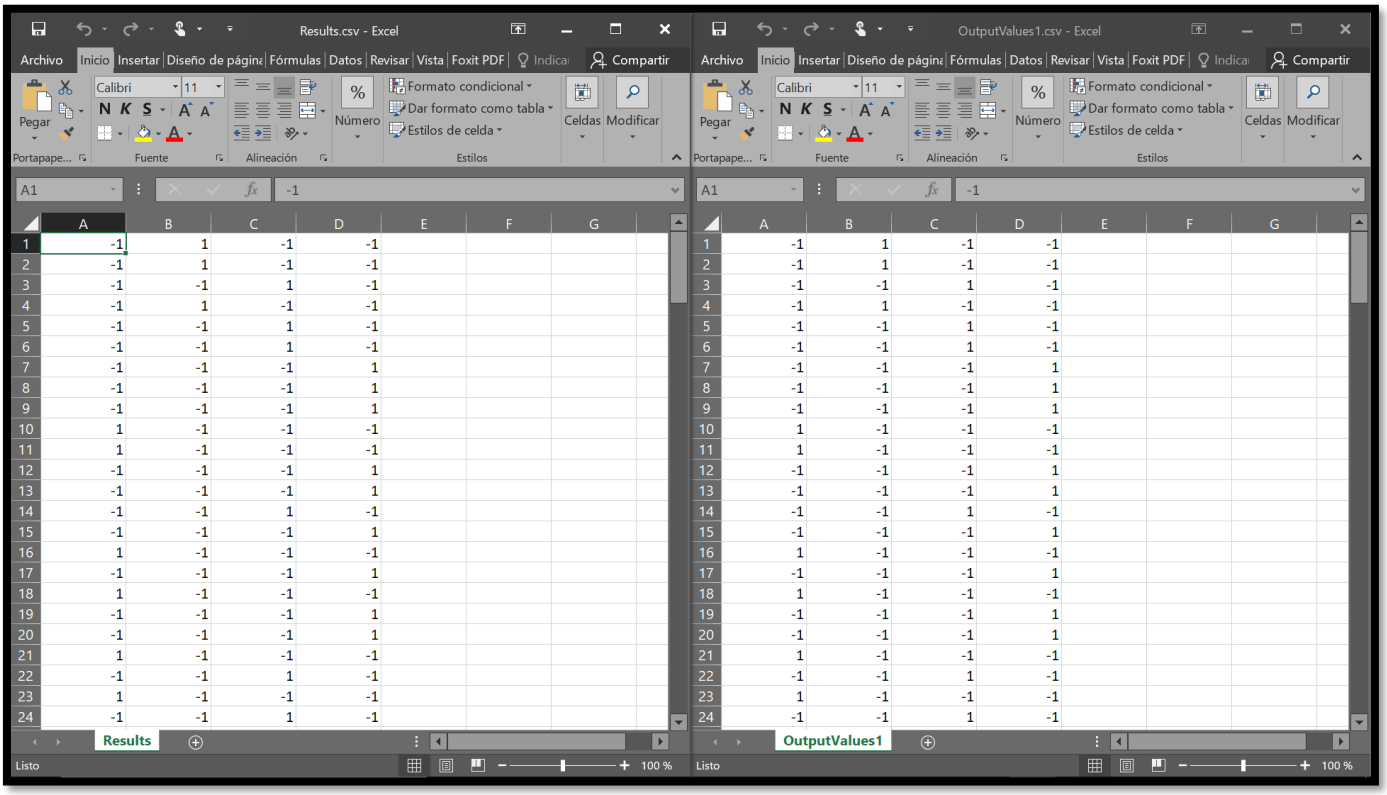



```
#To obtain the number of neurons for the program. The number of output columns tells us the number of r

neurons_array = []

for i in range(number_of_neurons):
    net = pwe.neurona(columns, 0.1)
    neurons_array.append(net)
#Perceptron initialization.

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

for i in range(columns):
    x = np.array(np.loadtxt(trainingPatt
    patterns.append(x)
X = np.array(patterns)

for i in range(number_of_neurons):
    y.append(np.array(np.loadtxt(output\
#Obtaining training patterns in X and ou
########################################

global_errors = []
for i in range(number_of_neurons):
    net.train(X, y[i], epochs)

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

global_errors = np.array(global_errors).T

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

En la primera figura podemos observar una neurona que ha finalizado con éxito su entrenamiento, y una segunda neurona que está comenzando.



En esta segunda imagen podemos observar en entrenamiento completado de las 4 neuronas, y cómo todas lograron llegar al cero en su error.



Finalmente podemos observar y comparar los datos de las salidas deseadas con los datos de las salidas predichas por las neuronas al término de su entrenamiento, y ver que son exactamente iguales, por lo que podemos concluir que el entrenamiento ha sido existoso.

Conclusión

Como podemos observar, el entrenamiento de más de una neurona del perceptrón es un proceso relativamente sencillo, ya que hemos resuelto el problema sencillamente con arreglos de neuronas e iteraciones de entrenamiento. También seguimos observando que este tipo de red neuronal presenta excelentes resultados en dimensiones múltiples, aunque ello signifique que el problema se complique. Finalmente, los resultados no solo son favorables, sino que se llega a ellos de manera tan rápida – por lo menos por ahora – que son hasta casi instantáneos, logrando la mayoría de las veces una solución en un máximo de 3 épocas, aunque normalmente se alcanza en 2. Será interesante observar el desempeño con datos que pudieran tener una naturaleza más compleja y ver los resultados obtenidos.

Código

main.py

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

trainingPatternsFileName = "InputValues1.csv"
outputValuesFileName = "OutputValues1.csv"
epochs = 10

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 = pwe.neurona(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 = []
for i in range(number_of_neurons):
    net.train(X, y[i], epochs)

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

global_errors = np.array(global_errors).T

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

perceptron_with_error_graph.py

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

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, 'ro', 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)
            y_axis.append(average)
            average = 0

        plt.plot(y_axis, 'g')
        y_axis.clear()
```

Link al repositorio

- https://github.com/TheGenesisX/S_IA_2/tree/master/Practica_4