Practica 4

Red neuronal unicapa

Seminario de solución de problemas de inteligencia artificial 2

Clave del curso: I7041

NRC: 124882

Calendario: 2023-A

Sección: D01

Martínez Sepúlveda Alan Jahir

216569127

INCO

18/03/2023

Introducción:

Para la realización de esta práctica se pide la simulación de una red neuron, en este caso implementando perceptrones, siendo posible un máximo de 4 neuronas, teniendo dos archivos csv, uno el cual se le dan las entradas y otro con las salidas esperadas. La red neuronal debe de ser capaz de graficar el error e ir iterando para que las salidas deseadas sean las correctas.

Desarrollo:

Para el desarrollo de esta práctica, y como punto principal se toman los valores que se quieren entrenar, teniéndolos estos en un archivo .csv llamados entradas y salidasDeseadas de la siguiente manera:

PRACTICAS > PRACTICA_4 > Entradas1.csv

```
0.96091,0.42613,0.49868,0.92964,0.91364
0.9567,0.36252,0.5036,0.94689,0.49552
0.5064,0.056677,0.090267,0.037416,0.53764
0.97755,0.36088,0.039305,0.94802,0.99409
0.49868,0.12837,0.048526,0.97335,0.081664
0.5036,0.03786,0.45743,0.05837,1
0.090267,0.49707,0.4375,0.41114,0.057048
0.039305,0.53293,0.07397,0.50981,0.027175
0.048526,0.5212,0.92964,0.14709,0.4809
0.45743,1,0.94689,0.041005,0.96545
0.4375,0.92458,0.037416,0.47241,0.5194
0.07397,0.49164,0.94802,0.38144,0.91522
0.088542,0.53168,0.97335,0.50018,0.46304
0.52819,0.0045916,0.05837,0.92253,0.51689
0.056342,0.51691,0.41114,0.11637,0.39358
0.45842,0.94118,0.50981,0.53557,0.52064
0.057257,0.48862,0.14709,0.42254,0.53508
0.46482,0.90426,0.041005,0.96623,0.91329
0.11876,0.49313,0.47241,0.39706,0.41108
0.067348,0.4664,0.44003,0.45978,0.076121
0.42951,0.91943,0.4809,0.39245,0.47188
0.49566,0.065059,0.48446,0.49918,0.90777
0.43715,0.95194,0.48739,0.011073,0.9475
0.45241,0.098947,0.4311,0.90549,0.96545
0.45705,0.93382,0.44697,0.50449,0.5194
0.4645,0.076433,0.49924,0.89907,0.91522
0.45479,0.90937,0.46196,0.49907,0.46304
0.037089,0.50753,0.42254,0.9012,0.47085
0.98552,0.3999,0.96623,0.40847,0.97335
0.45449,0.90914,0.39706,0,0.05837
0.94071,0.46973,0.45978,0.065956,0.41114
0.44003,0.97384,0.39245,0.42364,0.50981
0.4809,0.9557,0.49918,0.94433,0.14709
0.48446,0.9285,0.011073,0.14048,0.041005
```

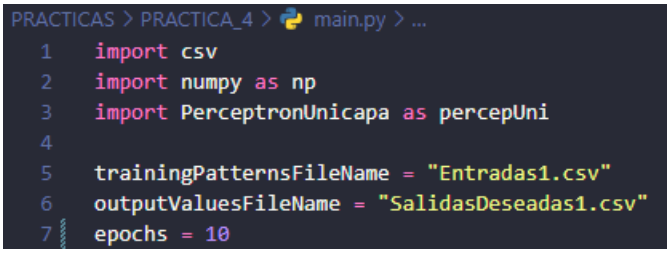
PRACTICAS > PRACTICA_4 > SalidasDeseadas1.csv

```
-1,1,-1,-1
-1,1,-1,-1
-1,-1,1,-1
-1,1,-1,-1
-1,-1,1,-1
-1,-1,1,-1
-1,-1,-1,1
-1,-1,-1,1
-1,-1,-1,1
1,-1,-1,-1
1,-1,-1,-1
-1,-1,-1,1
-1,-1,-1,1
-1,-1,1,-1
-1,-1,-1,1
1,-1,-1,-1
-1,-1,-1,1
1,-1,-1,-1
-1,-1,-1,1
-1,-1,-1,1
1,-1,-1,-1
-1,-1,1,-1
1,-1,-1,-1
-1,-1,1,-1
1,-1,-1,-1
-1,-1,-1,1
-1,1,-1,-1
1,-1,-1,-1
-1,1,-1,-1
1,-1,-1,-1
1,-1,-1,-1
1,-1,-1,-1
```

Se crea un archivo **main.py** para mantener el flujo de los archivos, así como las llamadas a las funciones:

Este código se encarga de entrenar un perceptrón multicapa utilizando el algoritmo de retropropagación del error. Se utiliza la librería numpy para realizar operaciones matemáticas y la librería csv para leer archivos csv con los patrones de entrenamiento y las salidas deseadas.

En primer lugar, se definen dos variables para los nombres de los archivos csv que contienen los patrones de entrenamiento y las salidas deseadas. También se define la cantidad de épocas que se utilizarán para entrenar la red neuronal.



A continuación, se abre el archivo csv que contiene los patrones de entrenamiento y se cuentan la cantidad de filas que contiene. Luego, se vuelve a abrir el archivo y se utiliza la función csv.reader para obtener la cantidad de columnas. Esto es necesario para crear la cantidad adecuada de neuronas en la capa de entrada de la red neuronal.

```
file = open(trainingPatternsFileName)
rows = len(file.readlines())
file.close()

file = open(trainingPatternsFileName,'r')
reader = csv.reader(file,delimiter=',')
columns = len(next(reader))
file.close()

file = open(outputValuesFileName,'r')
reader = csv.reader(file,delimiter=',')
number_of_neurons = len(next(reader))
file.close()
```

Luego, se abre el archivo csv que contiene las salidas deseadas y se utiliza la función csv.reader para obtener la cantidad de neuronas en la capa de salida de la red neuronal.

Se crea un arreglo vacío llamado neurons_array que será utilizado para almacenar las neuronas de la capa oculta de la red neuronal. Se utiliza un bucle for para crear la cantidad adecuada de neuronas y se agrega cada neurona al arreglo.

```
neurons_array = []

for i in range(number_of_neurons):
    net = percepUni.neurona(columns, 0.1)
    neurons_array.append(net)
```

Se crea una lista vacía llamada patterns y otra llamada y. Se utiliza un bucle for para obtener cada patrón de entrenamiento y cada salida deseada de los archivos csv y agregarlos a las listas correspondientes.

Se crea un arreglo numpy llamado X que contiene los patrones de entrenamiento.

```
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)))
```

Se utiliza otro bucle for para entrenar cada neurona de la capa oculta de la red neuronal utilizando la función train del archivo PerceptronUnicapa.py, que se importa al principio del código. Se utiliza otro bucle for para predecir la salida de la red neuronal para cada patrón de entrenamiento y se almacena el error individual en una lista llamada individual_error. Luego, se agrega la lista de errores individuales a una lista global llamada global_errors.

```
for i in range(number_of_neurons):
    y.append(np.array(np.loadtxt(outputValuesFileName, delimiter=',', usecols=i)))

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
```

Finalmente, se convierte la lista global_errors en un arreglo numpy y se guarda en un archivo csv llamado Results.csv utilizando la función savetxt de numpy.

```
global_errors = np.array(global_errors).T

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

Archivo PerceptronUnicapa.py:

El código comienza importando los paquetes numpy y matplotlib. Luego, se define una lista vacía y_axis.

A continuación, se definen dos etiquetas para los ejes x e y del gráfico a través de las funciones plt.ylabel() y plt.xlabel(). También se establece el título del gráfico mediante la función plt.title().

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

y_axis = []

plt.ylabel('Error', fontsize=30)
plt.xlabel('Epocas', fontsize=30)
plt.title('Perceptrón unicapa', fontsize=40)
```

La función graphError() se define para generar un gráfico en tiempo real que muestra la evolución del error durante el entrenamiento de la neurona. La función utiliza una secuencia de colores para cada punto y pausa la ejecución durante 0.3 segundos después de cada actualización.

```
def graphError(x_coordinate, y_coordinate):
    # generar una secuencia de 32 colores distintos
    colors = plt.cm.tab20(np.linspace(0, 1, 32))
    # utilizar un color diferente para cada punto
    plt.scatter(x_coordinate, y_coordinate, c=colors[x_coordinate % 32])
    plt.pause(0.3)
```

Después se define una clase neurona que implementa el modelo de Perceptrón Unicapa. La clase tiene un constructor que toma como entrada la dimensión de entrada y el coeficiente de aprendizaje. En el constructor, se inicializan los pesos y el sesgo con valores aleatorios en el rango [-1, 1].

La clase neurona también tiene dos métodos: predict() y train(). El método predict() calcula la salida de la neurona utilizando los pesos y el sesgo actuales. Si la salida es mayor o igual a cero, la función retorna 1; de lo contrario, retorna -1.

```
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)
        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
```

El método train() es utilizado para entrenar la neurona utilizando un conjunto de datos de entrenamiento X y una serie de resultados esperados y. El parámetro epochs indica el número de veces que se debe iterar sobre el conjunto de entrenamiento. Dentro del ciclo de entrenamiento, la neurona ajusta sus pesos y sesgo utilizando la regla de aprendizaje del Perceptrón Unicapa.

```
def train(self, X, y, epochs):  #x = matriz de entrenamiento, y = vector con
    n, m = X.shape
    average = 0
    for i in range(epochs):
        for j in range(m):
            y_pred = self.predict(X[:, 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)
        y_axis.append(average)
        average = 0

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

Conclusión:

El primer código utiliza la librería CSV para cargar los datos de entrenamiento y salida deseada desde dos archivos CSV diferentes. Luego, utiliza la librería Numpy para convertir

los datos en matrices y entrenar el modelo utilizando el algoritmo de aprendizaje del Perceptrón Unicapa. Finalmente, el código guarda los resultados en un archivo CSV.

El segundo código utiliza la librería Matplotlib para graficar la evolución del error durante el entrenamiento del modelo. Además, define una clase de neurona que implementa el algoritmo del Perceptrón Unicapa y contiene los métodos de predicción y entrenamiento.

En resumen, ambos códigos trabajan juntos para entrenar y evaluar un modelo de Perceptrón Unicapa utilizando datos de entrada y salida deseada y graficando la evolución del error durante el entrenamiento.

Código:

Main.py:

```
import csv
import numpy as np
import PerceptronUnicapa as percepUni

trainingPatternsFileName = "Entradas1.csv"
outputValuesFileName = "SalidasDeseadas1.csv"
epochs = 10

file = open(trainingPatternsFileName)
rows = len(file.readlines())
file.close()

file = open(trainingPatternsFileName,'r')
reader = csv.reader(file,delimiter=',')
columns = len(next(reader))
file.close()

file = open(outputValuesFileName,'r')
reader = csv.reader(file,delimiter=',')
number_of_neurons = len(next(reader))
file.close()

neurons_array = []

for i in range(number_of_neurons):
    net = percepUni.neurona(columns, 0.1)
    neurons_array.append(net)
```

```
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)))

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')
```

PerceptronUnicapa.py

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

y_axis = []

plt.ylabel('Error', fontsize=30)
plt.xlabel('Epocas', fontsize=30)
plt.title('Perceptrón unicapa', fontsize=40)

def graphError(x_coordinate, y_coordinate):
    # generar una secuencia de 32 colores distintos
    colors = plt.cm.tab20(np.linspace(0, 1, 32))
    # utilizar un color diferente para cada punto
    plt.scatter(x_coordinate, y_coordinate, c=colors[x_coordinate % 32])
    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)
        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
        average = 0
        for i in range(epochs):
            for j in range(m):
                y_pred = self.predict(X[:, 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()
```