Practica 2

Neurona perceptrón con entrenamiento

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

26/02/2023

Introducción:

Para la realización de esta práctica se pide la simulación de una neurona perceptrón con entrenamiento, en la cual se generan los valores de w1, w2 y el bias, se grafica la salida correspondiente y dependiendo de la salida esperada, se entrena la neurona.

Desarrollo:

Como punto de entrada, se reanuda la practica 1 ya realizada, en la cual se crea una neurona perceptrón pero sin entrenamiento, la cual esta es modificada agregándole una función de entrenamiento para que, según la salida esperada correspondiente, esta cambie los valores para su clasificación.

El modelo de la neurona quedaría de la siguiente manera, en la que se le dan dos valores de entrada, como lo son las dimensiones que esta va a tener y el coeficiente de aprendizaje (que tan rápido va a aprender) de nuestra neurona.

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

La función train es utilizada para entrenar una neurona perceptrón. Toma como entrada los datos de entrenamiento X, los resultados esperados y y el número de épocas de entrenamiento epocas. Durante el entrenamiento, la función itera sobre las épocas y luego itera sobre los datos de entrenamiento. Para cada dato, se utiliza la función predict para hacer una predicción y se compara con el resultado esperado. Si la predicción es incorrecta, se actualizan los pesos y el sesgo de la

neurona utilizando la regla de aprendizaje del perceptrón. Además, se llama a la función graph para graficar los datos y el límite de decisión de la neurona después de cada actualización de pesos y sesgo, y se guarda la figura en un archivo. La función devuelve la neurona entrenada.

```
# TODO x -> matriz, y ->vector de resultados esperados, epocas
def train(self, X, y, epocas):
    n, m = X.shape
    #n = 2. m = 4 en el ejemplo de la compuerta AND, OR y XOR.
    for i in range(epocas):
        for j in range(m):
            y_pred = self.predict(X[:, j])
            # Cortar la matriz, tomando las filas (:) pero manteniendo las columnas
            if y_pred != y[j]:  #Si nuestro estimado es diferente a nuestro esperado, entrenamos.
                self.w += self.aprendizaje*(y[j] - y_pred) * X[:, j].reshape(-1, 1)
                self.b += self.aprendizaje*(y[j] - y_pred)
                graph(self.w)
                plt.savefig(str(i) +'grafica')
```

Abrimos el archivo CSV "entradas.csv" y utilizan la función "readlines()" para obtener una lista de todas las líneas en el archivo. Luego, se usa la función "len()" para contar cuántas líneas hay en el archivo. Esto nos da el número de filas en la entrada del archivo

```
# Obtener el numero de filas en la entrada del archivo
with open("entradas.csv") as file:
    rows = len(file.readlines())
```

Abrimos el archivo CSV "entradas.csv" nuevamente, pero esta vez utilizan la función "csv.reader()" para leer el archivo y la función "next()" para obtener la primera fila (que contiene los encabezados de las columnas). Luego, se utiliza la función "len()" para contar cuántos elementos hay en la primera fila (que nos da el número de columnas en el archivo).

```
# Obtener el numero de columnas en la entrada del archivo
with open("entradas.csv") as f:
    reader = csv.reader(f,delimiter=',')
    columns = len(next(reader))
```

La instancia se inicializa con el número de columnas menos uno (ya que la última columna es la columna de salida), y una tasa de aprendizaje de 0.1.

```
percep = neurona(columns-1, 0.1)
arreglo = []
```

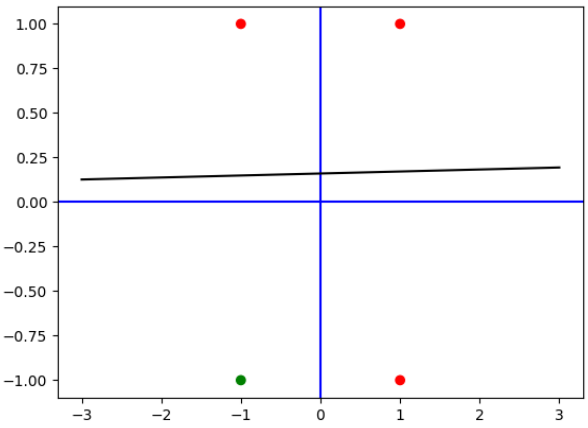
Las siguientes líneas se utilizan para preparar los datos de entrada (X) y de salida (y) para el entrenamiento. Primero, se inicializa una lista vacía llamada "arreglo". Luego, se utiliza un bucle "for" para recorrer todas las columnas del archivo (excepto la última, que es la columna de salida). En cada iteración del bucle, se utiliza la función "np.loadtxt()" para cargar los datos de la columna en un arreglo NumPy y se agrega el arreglo a la lista "arreglo". Finalmente, se convierte la lista "arreglo" en un arreglo NumPy bidimensional llamado "X", que se utilizará como entrada para el entrenamiento. La última línea carga los datos de la última columna (la columna de salida) en un arreglo NumPy llamado "y".

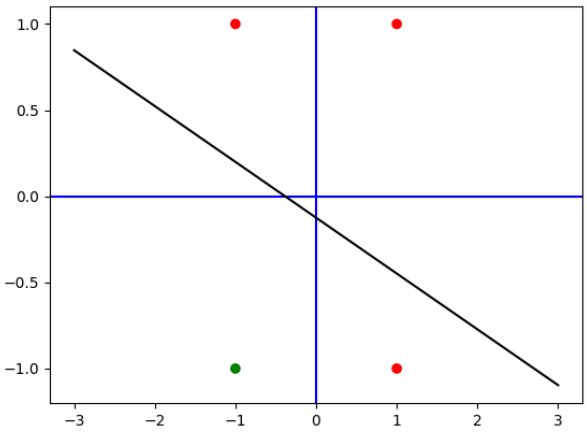
```
for i in range(columns-1):
    x = np.array(np.loadtxt("entradas.csv", delimiter=',', usecols=i))
    arreglo.append(x)
X = np.array(arreglo)
# Obtener el entrenamiento en el arreglo en x y la salida en y.
y = np.array(np.loadtxt("entradas.csv", delimiter=',', usecols=columns-1))
```

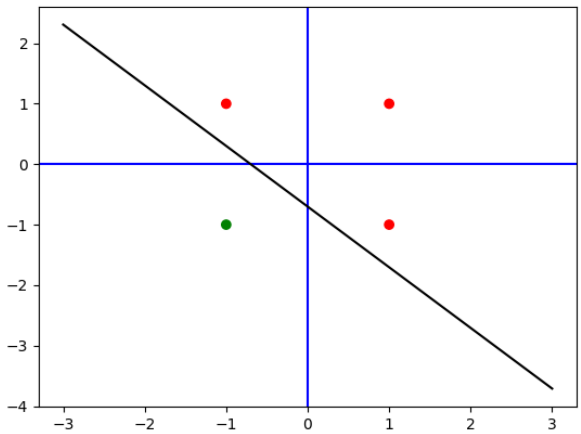
La última línea utiliza el objeto "percep" (que es una instancia de la clase "neurona" que se creó anteriormente) para entrenar la red neuronal durante 10 épocas. La función "train()" toma los datos de entrada (X), los datos de salida (y) y el número de épocas como argumentos. Durante cada época, la red neuronal se entrena utilizando los datos de entrada y de salida y se ajustan los pesos y el sesgo de la red neuronal.

```
percep.train(X, y, 10) # 10 -> Numero de epocas
```

Graficas generadas:







Conclusión:

El código implementa una red neuronal de una sola neurona para la clasificación binaria utilizando el algoritmo de entrenamiento de perceptrón. La neurona tiene una función de activación lineal, donde si la suma ponderada de las entradas más el sesgo es mayor o igual que cero, la salida es 1, de lo contrario es -1.

La función "graph" se utiliza para visualizar el proceso de entrenamiento en una gráfica, donde los puntos de entrada se representan en verde si la etiqueta de salida es 1 y en rojo si es 0. También se muestra la línea que separa las dos clases calculada por la neurona.

El código lee los datos de entrada desde un archivo csv y utiliza la primera fila para determinar el número de columnas (variables de entrada) y el número de filas (muestras). Luego crea una instancia de la clase neurona con un coeficiente de aprendizaje de 0.1 y entrena la red neuronal durante 10 épocas. Por último, se realiza una predicción para cada muestra de entrada y se imprimen los resultados.**Código:**

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

def graph(w_values):
    plt.clf()
    #Colorear entradas, si es 1 verde si es 0 rojo
    colors = np.where(y == 1, 'green', 'red')
    plt.scatter(X[0], X[1], color=colors)

    plt.axhline(color="blue")
    plt.axvline(color="blue")
    x_values = [-3,3]
    y_values = [-(percep.w[0][0]/percep.w[1][0])*(-3) - (percep.b /
percep.w[1][0]),
```

```
                -(percep.w[0][0]/percep.w[1][0])*(3) - (percep.b /
percep.w[1][0])]
    plt.plot(x_values, y_values, color="black")
    plt.pause(1)

# TODO dim-> Dimensiones, aprendizaje -> Coeficiente de aprendizaje

class neurona:
    def __init__(self, dim, aprendizaje):
        self.n = dim
        self.aprendizaje = aprendizaje
        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

    # TODO x -> matriz, y ->vector de resultados esperados, epocas
    def train(self, X, y, epocas):
        n, m = X.shape
        #n = 2. m = 4 en el ejemplo de la compuerta AND, OR y XOR.
        for i in range(epocas):
            for j in range(m):
                y_pred = self.predict(X[:, j])
                # Cortar la matriz, tomando las filas (:) pero manteniendo
las columnas
                if y_pred != y[j]:  #Si nuestro estimado es diferente a
nuestro esperado, entrenamos.
                    self.w += self.aprendizaje*(y[j] - y_pred) * X[:,
j].reshape(-1, 1)
                    self.b += self.aprendizaje*(y[j] - y_pred)
                    graph(self.w)
                    plt.savefig(str(i) +'grafica')


# Obtener el numero de filas en la entrada del archivo
with open("entradas.csv") as file:
    rows = len(file.readlines())

# Obtener el numero de columnas en la entrada del archivo
```

```
with open("entradas.csv") as f:
    reader = csv.reader(f,delimiter=',')
    columns = len(next(reader))

percep = neurona(columns-1, 0.1)
arreglo = []

for i in range(columns-1):
    x = np.array(np.loadtxt("entradas.csv", delimiter=',', usecols=i))
    arreglo.append(x)
X = np.array(arreglo)
# Obtener el entrenamiento en el arreglo en x y la salida en y.
y = np.array(np.loadtxt("entradas.csv", delimiter=',', usecols=columns-1))

percep.train(X, y, 10) # 10 -> Numero de epocas

for i in range(rows):
    print(percep.predict(X[:, i]))
```