Seminario de solución de problemas de inteligencia artificial II

I7041

Práctica 3.Multiples Perceptrón.

Sección: D02.

Alumno: Ramirez Sarabia Cesar Eduardo.

Código: 212572565

Índice

Introducción

En esta práctica número 3, se implementará el algoritmo completo del perceptrón, al igual que se implemento en la practica pasada. Sin embargo, la variación en esta práctica, es que se generaran mas de 1 Perceptrón para poder separar mas de 2 clases a la vez. De esta manera cada perceptrón creado simultáneamente tendrá sus pesos y sus bias con los cuales ajustará dichos pesos y de esta manera lograr separar distintas (más de 2 clases).

Objetivo

Implementar los multiple perceptrones para separar mas de 2 clases en un solo programa.

Metodología

Para probar el perceptrón realizado que se muestra mas adelante, probare con los datos de las compuertas lógicas AND y OR, los cuales apreciamos en la practica anterior son linealmente separables, pero antes de pasar a ver el resultado y el código que implemente para entrenar al perceptrón, primero vamos a ver los pasos que se deben tomar para implementar dicha neurona.

Pasos para implementar el perceptrón:

Paso 0:

Inicializar los pesos sinápticos con números aleatorios del intervalo [-1,1]. Ir al paso 1 con k=1

Paso 1:(k-ésima iteración)

Calcular $y(k) = \text{sgn}\left(\sum_{j=1}^{n+1} w_j x_j(k)\right)$

Paso 2:

Corrección de los pesos sinápticos. Si z(k)≠y(k) modificar los pesos sinápticos según la expresión:

Paso 3:

Si no se han modificado los pesos en las últimas p iteraciones, es decir,

$w_j(r) = w_j(k)$, j = 1,2…, n+1,r = k+ 1,…k + p, parar.

La red se ha estabilizado. En otro caso, ir al Paso 1 con k=k+1.

Función de activación F: Se encarga de mantener el conjunto de valores de salida en un rango determinado, normalmente (0,1) o (-1,1)

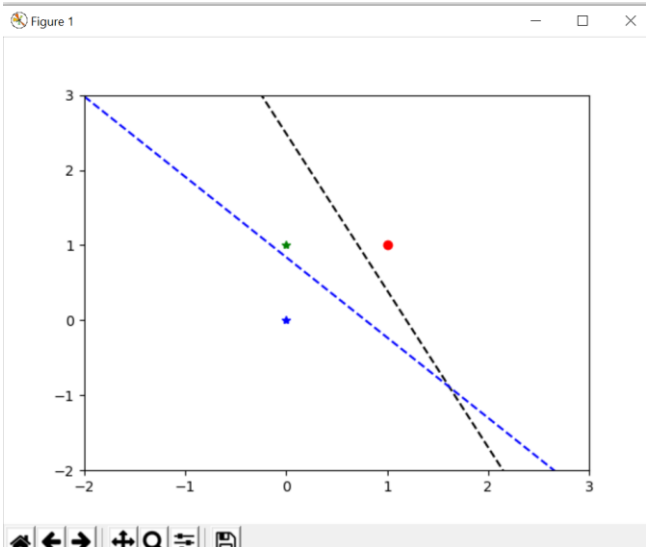
```
#Ramirez Sarabia César Eduardo
#Perceptron. Práctica 3
import numpy as np
import matplotlib
from pylab import xlim
from pylab import ylim
import matplotlib.pyplot as plt
import time
import numpy as np


def grafica_puntos():
    colores = ['*b','*g','pm','or','+k']
    aux_arr = []
    xlim([-2,3])
    ylim([-2,3])
    last_plot=-1
    for cont,this_element in enumerate(salidas_deseadas):
        str_salida = '-'.join(str(e) for e in this_element)
        if str_salida not in aux_arr:
            last_plot+=1
            aux_arr.append(str_salida)
            x_1 = datos_entradas[cont]
            ax.plot(x_1[0],x_1[1], colores[cont])
    return last_plot



def grafica_lineas(this_w,last_plot,is_last_plot):
    clases = ['--k','--b','--r','--y','--g']
    if not is_last_plot:
        index_plot = last_plot
        for c,item in enumerate(this_w):
            ax.plot([-2,3],[(-item[0]/item[1]) *
            (-2)-(b[c]/item[1]), (-item[0]/item[1])*(3)-
(b[c]/item[1])],clases[c])
            plt.pause(0.0000000000000000000000000000001)
            index_plot += 1
        for i in range(index_plot,last_plot,-1):
            ax.lines.pop(i)
    else:
```

```
        plt.cla()
        grafica_puntos()
        for c,item in enumerate(this_w):
            ax.plot([-2,3],[(-item[0]/item[1]) *
            (-2)-(b[c]/item[1]), (-item[0]/item[1])*(3)-
(b[c]/item[1])],clases[c])

        #plt.close('all')
    #for c in range(1,num_col_salidas):
    #   ax.lines.pop(last_plot + c)

def funcion_escalon(x):
    if x<0:
        return 0
    else:
        return 1


def obtiene_Salida(r,c):
    posicion = salidas_deseadas[r]
    if c == 0:
        return posicion[0]
    else:
        return posicion[c+1]


#training set contiene entradas y salidas
training_set = []

#plt.ion()
with open('entradas.txt') as f_1:
    datos_entradas = f_1.read().splitlines()

with open('salidas_deseadas_2.txt') as f_2:
    salidas_deseadas = f_2.read().splitlines()

#Convierte entrada a lista de enteros
for i, element in enumerate(datos_entradas):
    x =  [int(k) for k in element.split()]
    datos_entradas[i] = x

#Convierte salidas deseadas a lista de enteros
for i, element in enumerate(salidas_deseadas):
    x =  [int(k) for k in element.split()]
    salidas_deseadas[i] = x
```

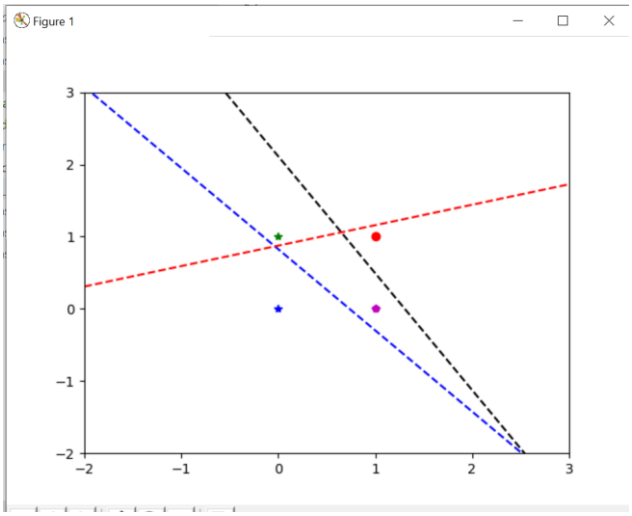
```
num_col_salidas = len(salidas_deseadas[0]) - salidas_deseadas[0].count(' ')

# Inicializacion de parametros ( pesos, eta , b)
################## c,r
w = np.random.rand(num_col_salidas,2)
errors = []
eta = .5
epoch = 8
b = np.zeros(num_col_salidas)

#crea el training set
for i in range(num_col_salidas):
    row = 0
    training_set_aux = []
    for j in datos_entradas:
        aux = salidas_deseadas[row]
        a = ((int(j[0]),
                int(j[1])),
                aux[i])
        training_set_aux.append(a)
        row += 1
    training_set.append(training_set_aux)

fig, ax = plt.subplots()

lst_i_plot = grafica_puntos()

for i in range(epoch):
    for count,fila_training_item in enumerate(training_set):
        for x,y in fila_training_item:
            u = sum(x*w[count]) + b[count]
            error = y - funcion_escalon(u)
            for index, value in enumerate(x):
                w[count][index] += eta * error * value
                b[count] += eta*error
                grafica_lineas(w,lst_i_plot,False)

print("PERCEPTRON ENTRENADOS")
grafica_lineas(w,lst_i_plot,True)
plt.show()
#grafica_lineas(w,lst_i_plot,True)


#plt.show()
#print('Done')
```

Resultados:

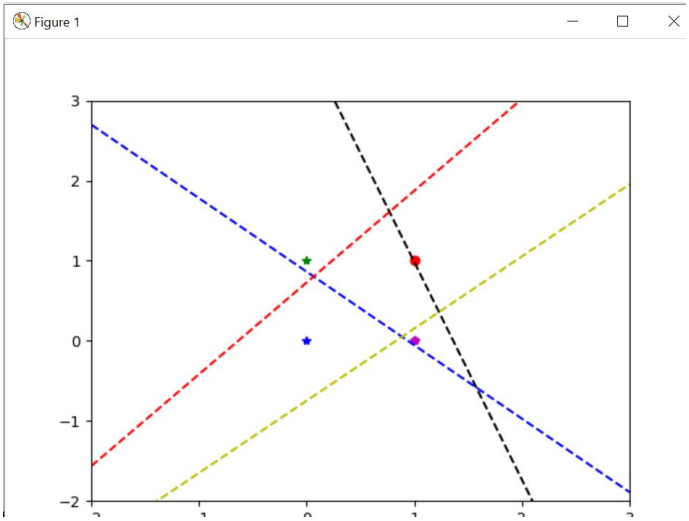
Salidas Deseadas




```
0 0
0 1
0 1
1 1
```




```
0 0 0
0 1 1
0 1 0
1 1 0
```




```
0 0 0 0
0 1 1 0
0 1 0 1
1 1 0 0
```

Conclusión

Al implementar el perceptrón simple, pude observar porque este es uno de los modelos básicos o de entrada en la inteligencia artificial ya que es un modelo de red neuronal básico y muy limitado que solo puede resolver problemas con conjuntos que sean linealmente separables de caso contrario. Sin embargo con esta implementación de usar múltiples perceptrones , podemos resolver múltiples clases y separarlas aunque con la limitante que deben seguir siendo linealmente separables

Además, pude apreciar la relación que tiene con la representación de una neurona biológica y como es que el modelo fue diseñado para comprender de una manera general y más práctica el funcionamiento de estas neuronas artificiales.

Bibliografía

[1] Munt, A. M. (2018). Introducción a los modelos de redes neuronales artificiales el perceptrón simple y multicapa. *Universidad de Zaragoza, España*.

[2] Stephen, I. (1990). Perceptron-based learning algorithms. *IEEE Transactions on neural networks*, *50*(2), 179.