Tarea 12 Perceptron

Escrito por Jesús Omar Cuenca Espino

A01378844

9/04/2021

In [1]:
```
import numpy as np
LEARNING_RATE = .1
```

Funcion de activacion

In [2]:
```
def activation(a):
    threshold = .5
    if(a >= threshold):
        return 1
    return 0
```

Funcion de perceptron

In [3]:
```
def perceptron(data,w,evaluate=False):
    global LEARNING_RATE
    if(evaluate):
        return np.array(list(map(activation,np.dot(data,w))))
    else:
        nW = w
        epoch = 0
        inp     = data[:,:-1]
        results = data[:,-1]
        while(epoch < 100):
            epoch += 1
            purePrediction = np.dot(inp,nW)
            pred = np.array(list(map(activation,purePrediction)))
            error = (results - pred)
            nW   = nW + LEARNING_RATE * np.dot(error,inp)
        return nW
```

Training the Not Perceptron

In [4]:
```
notTable = [
    [1,1,0],
    [0,1,1]
]
notTable = np.array(notTable)
notWeights = np.ones(2)
```

In [5]:
```
notWeights = perceptron(notTable,notWeights)
print(notWeights)
```

```
[-0.1  0.5]
```

In [6]:
```
perceptron(notTable[:,:-1],notWeights,True) == notTable[:,-1]
```

Out[6]:
```
array([ True,  True])
```

Training for AND Perceptron

In [7]:
```
andTable = [
    [0,0,1,0],
    [0,1,1,0],
    [1,0,1,0],
    [1,1,1,1]
]
andTable = np.array(andTable)
andWeights = np.ones(3)
```

In [8]:
```
andWeights = perceptron(andTable,andWeights)
print(andWeights)
```

```
[ 0.5  0.5 -0.2]
```

In [9]:
```
perceptron(andTable[:,:-1],andWeights,True) == andTable[:,-1]
```

Out[9]:
```
array([ True,  True,  True,  True])
```

Training the OR Perceptron

In [10]:
```
orTable = [
    [0,0,1,0],
    [0,1,1,1],
    [1,0,1,1],
    [1,1,1,1]
]
orTable = np.array(orTable)
orWeights = np.ones(3)
```

In [11]:
```
orWeights = perceptron(orTable,orWeights)
print(orWeights)
```

```
[1.  1.  0.4]
```

In [12]:
```
perceptron(orTable[:,:-1],orWeights,True) == orTable[:,-1]
```

Out[12]:
```
array([ True,  True,  True,  True])
```

Training the NOR Perceptron

In [13]:
```
norTable = [
    [0,0,1,1],
    [0,1,1,0],
    [1,0,1,0],
    [1,1,1,0]
]
norTable = np.array(norTable)
norWeights = np.ones(3)
```

In [14]:
```
norWeights = perceptron(norTable,norWeights)
print(norWeights)
```

```
[-0.1 -0.1  0.6]
```

In [15]:
```
perceptron(norTable[:,:-1],norWeights,True) == norTable[:,-1]
```

Out[15]:
```
array([ True,  True,  True,  True])
```

Training for the XNOR Percpetron

In [16]:
```
xnorTable = [
    [0,0,1,1],
    [0,1,1,0],
    [1,0,1,0],
    [1,1,1,1]
]
xnorTable = np.array(xnorTable)
xnorWeights = np.ones(3)
```

In [17]:
```
xnorWeights = perceptron(xnorTable,xnorWeights)
print(xnorWeights)
```

```
[0.1 0.1 0.5]
```

In [18]:
```
perceptron(xnorTable[:,:-1],xnorWeights,True) == xnorTable[:,-1]
```

Out[18]:
```
array([ True, False, False,  True])
```

Debido a que no es posible resolver el problema del XNOR o del XOR con 1 solo perceptron se tiene que recurrir a conectar varios perceptrones en una red para resolver problema

In [19]:
```
def andPerceptron(inputs):
    return perceptron(inputs,andWeights,True)
def orPerceptron(inputs):
    return perceptron(inputs,orWeights,True)
def notPerceptron(inputs):
    return perceptron(inputs,notWeights,True)
```

la compuerta XNOR se puede descomponer en ~ (~AB + ~BA)

definiremos C = ~AB

definiremos D = ~BA

por lo tanto XNOR = ~(C+D)

Y XNOR = ~XOR

por lo que empezaremos por la compuerta XOR

Compuerta XOR

In [20]:
```
xorTable = [
    [0,0,1,0],
    [0,1,1,1],
    [1,0,1,1],
    [1,1,1,0]
]
xorTable = np.array(xorTable)
```

In [21]:
```
def xorNN(inputs):
    A = inputs[:,0] ## Sacar columna A
    B = inputs[:,1] ## Sacar columna B
    ones = np.ones(inputs.shape[0]) ## Columna de 1
    ## Agregamos Bias a las columnas
    A = np.c_[A,ones]
    B = np.c_[B,ones]
    ## Usando el perceptron de NOT negamos los valores
    notA = notPerceptron(A)
    notB = notPerceptron(B)
    ## Sacamos los valores de C y D
    temp = np.c_[notA,B]
    C = andPerceptron(temp)
    temp = np.c_[notB,A]
    D = andPerceptron(temp)
    ## Agregamos una columna de bias para el perceptron que sigue
    temp = np.c_[C,D,ones]
    return orPerceptron(temp) ## Regresamos el resultado de el perceptron OR
```

In [22]:
```
xorNN(xorTable[:,:-1]) == xorTable[:,-1]
```

Out[22]:
```
array([ True,  True,  True,  True])
```

NN XNOR

In [23]:
```
def xnorNN(inputs):
    return notPerceptron(np.c_[xorNN(inputs),np.ones(inputs.shape[0])])
```

In [24]:
```
xnorNN(xnorTable[:,:-1]) == xnorTable[:,-1]
```

Out[24]:
```
array([ True,  True,  True,  True])
```