INSTITUTO POLITÉCNICO NACIONAL

ESCUELA SUPERIOR DE CÓMPUTO

Introducción a las Redes Neuronales Artificiales

Dr. Juan Humberto Sossa Azuela

Reporte Práctica No.3

Acosta Rosales Jair Sebastián

Máquinas de aprendizaje extremo

En esta práctica se han implementado las máquinas de aprendizaje extremo, las cuáles son un tipo de redes neuronales de una sola capa oculta que permiten reducir el tiempo de entrenamiento resolviendo problemas de datasets como lo son MNIST, IRIS.

Estas máquinas de aprendizaje extremo no implementan la técnica de backpropagation, ya que su fundamento matemático está basado en la fómula que permite obtener la salida de la red, sin embargo, en este punto no se busca hallar la salida de la red sino los pesos de la capa de salida que van a permitir hacer la clasificación, de modo que podemos mantener los pesos aleatorios de la primera capa oculta.

La fórmula para obtener los pesos es:

$$\hat{\boldsymbol{\beta}} = (X^T X)^{-1} X^T \boldsymbol{y}.$$

Donde:

B = Pesos de la capa de salida.

X = matriz de feedforward hasta la capa oculta.

Xt = matriz transpuesta de feedforward hasta la capa oculta.

y = Etiquetas del datasaset.

Para la implementación de esta técnica se usó el lenguaje Python, el cual contiene librerías optimizadas para las operaciones con matrices que permiten hacer un proceso de operaciones mucho más rápido, en este caso se usará la librería **numpy** el objetivo será resolver la compuerta XOR**.**

Definiendo el dataset

Se define el dataset para la compuerta XOR, el cual esta compuesto de 4 patrones con dos características cada uno y las respectivas etiquetas de cada uno.

```
In [101]:
x_train = np.array([[0,0],[0,1],[1,0],[1,1]]) #Dataset de entrenamiento
y_train = np.array([0,1,1,0]) #Etiquetas del dataset
print(x_train)
print(x_train.shape)
print()
print(y_train)
print(y_train.shape)
```

```
[[0 0]
 [0 1]
 [1 0]
 [1 1]]
(4, 2)

[0.1 0.9 0.9 0.1]
(4,)
```

También es posible usar un dataset con cierto ruido, tanto en los patrones como en las etiquetas.

Definiendo la arquitectura de la red

La arquitectura de la red neuronal consta de una capa oculta y una capa de salida, con sus respectivos bias, para ello se definió una función que permite ingresar el número deseado de neuronas en la capa oculta, aunque para el uso de este algoritmo se recomienda que el número de neuronas sea igual o menor que el número de patrones que se tienen para el entrenamiento.

```
In [88]:
def Neural_network_model(n_neurons_hl,x_train):
    weights = {}
    biases = {}

    #Initializing weights
    weights['W_hl1'] = np.random.normal(size=(n_neurons_hl,x_train.shape[1]))#matriz de pesos de la capa oculta
    weights['W_out'] = np.random.normal(size=(1,n_neurons_hl))#matriz de pesos de la capa de salida

    #Initializing biases
    biases['b_hl1'] = np.random.normal(size=(n_neurons_hl))#bias de la capa oculta
    biases['b_out'] = np.random.normal(size=(1))#bias de la capa de salida

    return [weights,biases]
```

Funciones de apoyo para el algoritmo

En esta parte se han definido funciones que apoyan el proceso de entrenamiento, en concreto han sido cuatro funciones definidas, las cuales son:

Feedforward

La función que va a propagar hacia adelante los resultados de cada patrón propagado a través de la red, que en este tipo de redes solo es la matriz resultante del producto punto del patrón de entrada por la matriz de pesos de la capa oculta, la función queda de la siguiente manera:

```
def feedforward(x_train,ANN):
    output = np.add(np.dot(x_train,np.transpose(ANN[0]['W_hl1'])),ANN[1]['b_hl1'])#propagación hacia adelante del dataset en
    #output = np.matmul(x_train,np.transpose(ANN[0]['W_hl1']))
    activate_output = relu(output)#activación de la salida
    #activate_output = sigmoid(output)
    return output
```

Funciones de activación

Se definieron dos funciones de activación, la función sigmoide y la función Relu de la siguiente manera.

```
def relu(z):
    return np.maximum(0,z)#función de activación relu, (que valores se van a comparar, umbral del cual se va a partir)

def sigmoid(z):
    sig = 1 / (1 + np.exp(-z))
    return(sig)
```

Función de entrenamiento

Está función involucra el calculo de la matriz inversa de Penrose, la cual gracias al módulo linalg de la misma librería numpy se puede calcular, de modo que la ecuación antes mencionada para encontrar los pesos de la capa de salida va a ser implementada de la siguiente forma:

```
def Train_neural_network(x_train,y_train,ANN):
    H_matrix = feedforward(x_train,ANN) #matriz de propagación hacia adelante
    Ht_matrix = np.transpose(H_matrix) #matriz inversa
    B = np.dot(np.linalg.inv(np.dot(Ht_matrix, H_matrix)), np.dot(Ht_matrix, y_train)) #calculo de los pesos de salida
    return B
```

Probando el accuracy de la red neuronal

Para esta parte se implementaron los mismos pasos de feedforward, pero con la diferencia que la última está involucrada para finalmente darnos una salida, de este modo la función de training queda de la siguiente forma.

```
def Testing_neural_network(x_train,y_train,ANN):
    #Feedforward
    #out_hl1 = sigmoid(np.add(np.dot(x_train,np.transpose(ANN[0]['W_hl1'])),ANN[1]['b_hl1']))
    out_hl1 = relu(np.matmul(x_train,np.transpose(ANN[0]['W_hl1'])))#propagación hacia adelante
    output = np.add(np.dot(out_hl1,np.transpose(ANN[0]['W_out'])),ANN[1]['b_out'])

    Final_results = []
    pos = 0
    for x in x_train: #por cada vector del dataset
        Final_results.append([x,y_train[pos],output[pos]])
        pos += 1

    print(tabulate(Final_results,headers=['Input','Target','Output NN'],tablefmt='fancy_grid'))
```

En la primer parte podemos observar, el proceso de feedforward de la red para obtener una salida y en la parte final se adjunta cada resultado con su respectiva entrada y target deseado para hacer la comparativa de los resultados.

A continuación vemos dos tablas de resultados:

Sin ruido

Input	Target	Output NN
[0 0]	0	0.530932
[0 1]	1	0.916245
[1 0]	1	0.56114
[1 1]	0	0.488313

Con ruido

Input	Target	Output NN
[0 0]	0.011	0.795783
[0 1]	0.999	1.78378
[1 0]	0.999	1.78378
[1 1]	0.011	0.795783

En ambas es posible observar que los resultados donde la salida de la red debe ser igual a 1 los valores de salida de la red son mayores que en donde debe ser cero, además que la red generaliza de una mejor forma cuando los datos contienen cierto nivel de ruido en ellos.