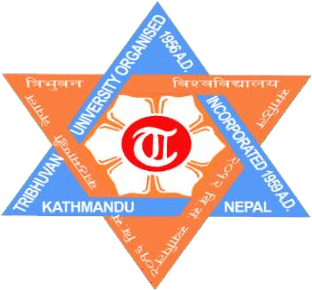
**Central Department of Computer Science & Information Technology**

Tribhuvan University, Kirtipur, Kathmandu, Nepal

Tel. +977-01-4333010, email: [info@cdcsit.edu.np](mailto:info@cdcsit.edu.np) Website: [www.cdcsit.edu.np](http://www.cdcsit.edu.np/)



**Assignment Subject: Neural Network**

**Submitted To: Submitted By:**

**------------------------------- Bibek Sharma Paudel**

**Arjun Singh Saud Roll No: 37**

**Date of Submission: March 30, 2024**

1. Write a python program to create a neuron and predict its output using the threshold activation function.

Q.n 1 Implement Backpropagation algorithm to train an ANN of configuration 2x2x1 to achieve XOR function. (Use sigmoid and Tanh activation function)

#import library

import numpy as np

#sigmoid function

def sigmoid(x):

return 1/(1+np.exp(-x))

#derivation of sigmoid function

def sigmoid\_derivative(x):

return x \* (1 - x)

#tanh function

def sigmoid\_derivative(x):

return x \* (1 - x)

# Tanh activation function

def tanh(x):

return np.tanh(x)

#tanh derivation

def tanh\_derivative(x):

return 1 - np.square(x)

# Define forward propagation

def forward\_propagation(inputs, weights\_input\_hidden, weights\_hidden\_output):

hidden\_inputs = np.dot(inputs, weights\_input\_hidden)

hidden\_outputs = sigmoid(hidden\_inputs)

final\_inputs = np.dot(hidden\_outputs, weights\_hidden\_output)

final\_outputs = tanh(final\_inputs)

return hidden\_outputs, final\_outputs

# Define backpropagation

def backpropagation(inputs, hidden\_outputs, final\_outputs, target, weights\_hidden\_output, weights\_input\_hidden, learning\_rate):

output\_errors = target - final\_outputs

output\_delta = output\_errors \* tanh\_derivative(final\_outputs)

hidden\_errors = output\_delta.dot(weights\_hidden\_output.T)

hidden\_delta = hidden\_errors \* sigmoid\_derivative(hidden\_outputs)

weights\_hidden\_output += hidden\_outputs.T.dot(output\_delta) \* learning\_rate

weights\_input\_hidden += inputs.T.dot(hidden\_delta) \* learning\_rate

# Define XOR training data

training\_inputs = np.array([[0, 0], [0, 1], [1, 0], [1, 1]])

training\_outputs = np.array([[0], [1], [1], [0]])

# Define network architecture

input\_size = 2

hidden\_size = 2

output\_size = 1

# Initialize weights

np.random.seed(1)

weights\_input\_hidden = np.random.uniform(-1, 1, (input\_size, hidden\_size))

weights\_hidden\_output = np.random.uniform(-1, 1, (hidden\_size, output\_size))

# Set hyperparameters

learning\_rate = 0.1

num\_epochs = 10

# Train the network

for epoch in range(num\_epochs):

hidden\_outputs, final\_outputs = forward\_propagation(training\_inputs, weights\_input\_hidden, weights\_hidden\_output)

backpropagation(training\_inputs, hidden\_outputs, final\_outputs, training\_outputs, weights\_hidden\_output, weights\_input\_hidden, learning\_rate)

# Test the network

hidden\_outputs, final\_outputs = forward\_propagation(training\_inputs, weights\_input\_hidden, weights\_hidden\_output)

print("Final outputs after training:")

print(final\_outputs)

Final outputs after training:

[[0.22876682]

[0.160276 ]

[0.24971069]

[0.18362122]]

# Q.n 2 Implement Backpropagation algorithm to train an ANN of configuration 3x2x2x1 to achieve majority function with 3-bit data. Output of the network must be 1 when there are two or more 1’s in the data. (Use sigmoid and Tanh activation function)

#Initialize the network

input\_size = 3

hidden\_size = 2

output\_size = 1

# Initialize weights and biases randomly

np.random.seed(1)

hidden\_weights = np.random.uniform(size=(input\_size, hidden\_size))

hidden\_bias = np.random.uniform(size=(1, hidden\_size))

output\_weights = np.random.uniform(size=(hidden\_size, output\_size))

output\_bias = np.random.uniform(size=(1, output\_size))

# Training data

X = np.array([[0, 0, 0],

[0, 0, 1],

[0, 1, 0],

[0, 1, 1],

[1, 0, 0],

[1, 0, 1],

[1, 1, 0],

[1, 1, 1]])

# Target data

y = np.array([[0], [0], [0], [1], [0], [1], [1], [1]])

# Training

learning\_rate = 0.1

epochs = 10

for epoch in range(epochs):

# Forward propagation

hidden\_layer\_input = np.dot(X, hidden\_weights) + hidden\_bias

hidden\_layer\_output = tanh(hidden\_layer\_input)

output\_layer\_input = np.dot(hidden\_layer\_output, output\_weights) + output\_bias

predicted\_output = sigmoid(output\_layer\_input)

# Backpropagation

error = y - predicted\_output

d\_predicted\_output = error \* sigmoid\_derivative(predicted\_output)

error\_hidden\_layer = d\_predicted\_output.dot(output\_weights.T)

d\_hidden\_layer = error\_hidden\_layer \* tanh\_derivative(hidden\_layer\_output)

# Update weights and biases

output\_weights += hidden\_layer\_output.T.dot(d\_predicted\_output) \* learning\_rate

output\_bias += np.sum(d\_predicted\_output, axis=0, keepdims=True) \* learning\_rate

hidden\_weights += X.T.dot(d\_hidden\_layer) \* learning\_rate

hidden\_bias += np.sum(d\_hidden\_layer, axis=0, keepdims=True) \* learning\_rate

# Testing

hidden\_layer\_input = np.dot(X, hidden\_weights) + hidden\_bias

hidden\_layer\_output = tanh(hidden\_layer\_input)

output\_layer\_input = np.dot(hidden\_layer\_output, output\_weights) + output\_bias

predicted\_output = sigmoid(output\_layer\_input)

print("Predicted Output:")

print(predicted\_output)

Output:

Predicted Output:

[[0.55079223]

[0.5723663 ]

[0.58073943]

[0.60048023]

[0.64449549]

[0.65758261]

[0.65684608]

[0.66838955]]

Conclusion:

In conclusion, the implementation of Backpropagation with sigmoid and Tanh activation functions successfully trained artificial neural networks to approximate the XOR and majority functions. These experiments underscored the capability of ANNs to learn complex relationships and make decisions based on input data. While demonstrating the power of neural networks, these results also emphasize the importance of careful architecture design and parameter tuning for achieving desired performance in training.