Central Department of Computer Science & Information Technology

Tribhuvan University, Kirtipur, Kathmandu, Nepal

Tel. +977-01-4333010, email: info@cdcsit.edu.np Website: www.cdcsit.edu.np

Assignment

Subject: Neural Network

Submitted To:

Arjun Singh Saud
Date of Submission: March 30, 2024

Submitted By:

Sagar Timalsena
Roll No: 34

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.