Forward and Backward Propagation Assignment

Q1. Explain the concept of forward propagation in a neural network.

ANS:

Forward propagation is the process through which input data passes through the neural network layer by layer to produce an output. In this process, the network computes the weighted sum of inputs at each layer and applies an activation function to generate outputs for the next layer.

The steps involved in forward propagation are:

Input Layer: The input data is fed into the neural network. Weight Multiplication: The input is multiplied by the weights associated with each connection between neurons. Bias Addition: A bias term is added to the weighted input. Activation Function: The result of the weighted input plus bias is passed through an activation function, which introduces non-linearity to the model. Repeat for Each Layer: These steps are repeated for every hidden layer. Output Layer: In the final layer, the network computes the output using the final set of weights and activation functions.

Q2. What is the purpose of the activation function in forward propagation

ANS:

The activation function introduces non-linearity to the model, which is crucial for the neural network to learn and model complex patterns. Without activation functions, the network would simply be a linear transformation of the input, which limits its capacity to solve non-linear problems.

Linear Activation: No non-linearity, meaning the output is simply a weighted sum of inputs. Non-Linear Activation: Functions like ReLU, Sigmoid, or Tanh add non-linearity, allowing the model to approximate complex functions and capture intricate patterns in the data.

Q3. Describe the steps involved in the backward propagation (backpropagation) algorithm'

ANS:

Backpropagation is the process of updating the weights of a neural network based on the error of the output. It computes the gradient of the loss function with respect to each weight by applying the chain rule of calculus.

The key steps in backpropagation are:

Compute the Loss: Calculate the loss (or error) between the predicted output and the actual target using a loss function (e.g., Mean Squared Error, Cross-Entropy).

Calculate the Gradient of the Loss: For each layer, calculate the gradient of the loss function with respect to the weights and biases. This involves:

Output Layer: Compute the gradient of the loss with respect to the output of the layer and then backpropagate through the activation function. Hidden Layers: For each hidden layer, backpropagate the gradient to the previous layer by calculating the partial derivative of the loss with respect to the layer's weights. Weight Update: Use the computed gradients to update the weights by adjusting them in the direction that minimizes the loss. Typically, a learning rate is used to control the size of the weight update.

Repeat: Repeat this process for each batch of data until the loss converges to a low value.

Q4. What is the purpose of the chain rule in backpropagation

ANS:

The chain rule is essential in backpropagation because it allows us to compute the gradient of the loss function with respect to each weight in the network. Since the output of each neuron depends on the weights of the previous layer, we use the chain rule to break down the derivative of the loss function into simpler parts. This enables us to efficiently propagate the error from the output layer back to the input layer, adjusting the weights at each step.

Q5. Implement the forward propagation process for a simple neural network with one hidden layer usingNumPy.

```
import numpy as np

# Define the activation function (Sigmoid in this case)
def sigmoid(x):
    return 1 / (1 + np.exp(-x))

# Define the input, weights, and biases for a simple network
# Assume input layer has 3 neurons, hidden layer has 4 neurons, and
output layer has 1 neuron

# Input (batch size = 2, features = 3)
X = np.array([[0.1, 0.2, 0.3],
              [0.4, 0.5, 0.6]])

# Weights for the hidden layer (3 input neurons, 4 hidden neurons)
W_hidden = np.random.randn(3, 4)

# Biases for the hidden layer (4 neurons in the hidden layer)
b_hidden = np.random.randn(4)

# Weights for the output layer (4 hidden neurons, 1 output neuron)
W_output = np.random.randn(4, 1)
```

```python
# Bias for the output layer (1 output neuron)
b_output = np.random.randn(1)

# Forward Propagation
# Step 1: Calculate the weighted sum for the hidden layer
Z_hidden = np.dot(X, W_hidden) + b_hidden

# Step 2: Apply activation function to the hidden layer (using
Sigmoid)
A_hidden = sigmoid(Z_hidden)

# Step 3: Calculate the weighted sum for the output layer
Z_output = np.dot(A_hidden, W_output) + b_output

# Step 4: Apply activation function to the output (Sigmoid for a
binary classification task)
A_output = sigmoid(Z_output)

# Print the output
print("Output of the network:")
print(A_output)
```

```
Output of the network:
[[0.55563549]
 [0.48364651]]
```