Forward and Backward Propagation Submission

---

1. Explain the Concept of Forward Propagation in a Neural Network

Definition:
Forward propagation refers to the process of feeding input data through the layers of a neural network to compute the output. Each layer applies a weighted sum of inputs, adds a bias, and passes the result through an activation function.

Steps:

1. **Input Layer:** The input features are fed into the network.
2. **Weighted Sum:** Each neuron computes the weighted sum of inputs: z=∑(w⋅x)+bz = \sum (w \cdot x) + b Where ww is the weight, xx is the input, and bb is the bias.
3. **Activation Function:** The computed sum is passed through an activation function to introduce non-linearity.
4. **Hidden and Output Layers:** The process continues through all hidden layers until the final prediction is made.

---

2. What is the Purpose of the Activation Function in Forward Propagation?

Purpose:

- Introduce non-linearity to the network, enabling it to learn complex patterns.
- Without activation functions, the neural network would behave as a linear model regardless of its depth.
- Activation functions also help control the output range, making models more stable during training.

---

3. Steps Involved in the Backward Propagation (Backpropagation) Algorithm

Step 1: Compute the loss using a loss function such as Mean Squared Error (MSE) or Cross-Entropy Loss.

Step 2: Compute the gradient of the loss with respect to the network's output (error signal).
Step 3: Apply the **chain rule** to calculate the gradient for each layer in the network backward from the output layer to the input layer.
Step 4: Update the weights and biases using gradient descent:

wi←wi−η⋅∂L∂wiw_i \leftarrow w_i - \eta \cdot \frac{\partial L}{\partial w_i}

Where η\eta is the learning rate, and ∂L∂wi\frac{\partial L}{\partial w_i} is the gradient.
Step 5: Repeat the process for multiple iterations until convergence.

---

4. What is the Purpose of the Chain Rule in Backpropagation?

Purpose:

- The chain rule allows the computation of gradients for all parameters in the network by systematically applying the rule to composite functions.
- It ensures that errors are propagated backward through the network so that weights and biases can be updated efficiently.
- Without the chain rule, training deep neural networks would be computationally infeasible.

---

5. Implementation of Forward Propagation in NumPy

```
import numpy as np



# Define the activation function (ReLU)

def relu(x):

    return np.maximum(0, x)



# Define the forward propagation function

def forward_propagation(X, weights1, bias1, weights2, bias2):

    # Hidden layer computation

    Z1 = np.dot(X, weights1) + bias1
```

```
    A1 = relu(Z1)


    # Output layer computation

    Z2 = np.dot(A1, weights2) + bias2

    return Z2


# Example input data

X = np.array([[1.0, 2.0]])  # 1 sample with 2 features


# Weights and biases for a network with 2 input features, 1 hidden layer with 3 neurons, and 1 output

weights1 = np.array([[0.5, -0.2, 0.3], [0.8, 0.5, -0.6]])

bias1 = np.array([0.1, 0.2, 0.1])

weights2 = np.array([[0.4], [0.3], [-0.5]])

bias2 = np.array([0.2])


# Perform forward propagation

output = forward_propagation(X, weights1, bias1, weights2, bias2)

print("Network Output:", output)
```

Explanation:

- The example computes the forward propagation for a simple neural network with one hidden layer.
- **ReLU** is used as the activation function.
- The computed **network output** is displayed as a result.