Forward and Backward Propagation Assignment Questions

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

Forward propagation is the process by which input data flows through the neural network to generate a prediction or output. It is the first step in training or using a neural network.

Here's how it works:

- Input data (e.g., feature vector) is fed into the input layer.
- Each neuron in the hidden and output layers performs two operations:
  - **Weighted sum** of the inputs (i.e., input × weight + bias).
  - Passes the result through an **activation function** to introduce non-linearity.
- The output from one layer becomes the input for the next layer.
- Finally, the network produces an output prediction (e.g., a class label or a value).

This process enables the network to **map inputs to outputs** based on the current set of weights and biases.

2. What is the purpose of the activation function in forward propagation?

The **activation function** introduces **non-linearity** into the network, allowing it to model complex relationships and patterns in the data.

Why it's essential:

- Without it, the network would simply be a stack of linear equations — no matter how many layers you add, it would still behave like a linear model.
- With activation functions (like ReLU, Sigmoid, Tanh), the network can learn to represent complex decision boundaries and abstract features.

In short: Activation functions are what give neural networks the power to learn from nonlinear data.

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

Backpropagation is the algorithm used to train neural networks by adjusting weights to minimize the error (loss).

Steps involved:

1. **Forward Pass:** Calculate the output of the network using the current weights.
2. **Compute Loss:** Compare predicted output with the actual target using a loss function (e.g., MSE, cross-entropy).
3. **Backward Pass (Backpropagation):**
    a. Compute the **gradient** of the loss with respect to each weight using the **chain rule**.
    b. Gradients flow from output to input layer.
4. **Update Weights:** Adjust weights and biases using an optimization algorithm (like Gradient Descent) to reduce the error.

This process is repeated for many iterations (epochs) until the model converges to a low-loss solution.

4. What is the purpose of the chain rule in backpropagation?

The **chain rule of calculus** is essential in backpropagation because it allows us to compute how changes in the loss affect each parameter (weight) in the network — even through multiple layers.

In a neural network, outputs are **nested functions** of inputs and weights. The chain rule helps us compute the derivative of the loss function with respect to weights and biases in each layer.

Example:

If L is the loss, z is the weighted sum, and a is the activation,
then:

DL/DW = DL/DA*DA/DZ*DZ/DW

This enables **efficient gradient computation**, which is critical for training deep networks.

5. Implement the forward propagation process for a simple neural network with one hidden layer using NumPy.

Here's a basic implementation in Python using NumPy:

```
import numpy as np


# Activation function (ReLU and Sigmoid)
def relu(x):
    return np.maximum(0, x)


def sigmoid(x):
    return 1 / (1 + np.exp(-x))


# Input vector (1 sample with 3 features)
X = np.array([[0.5, 0.2, 0.1]])


# Weights and biases for input → hidden layer (3 inputs → 4 hidden units)
W1 = np.random.randn(3, 4)
b1 = np.random.randn(1, 4)


# Weights and biases for hidden → output layer (4 hidden → 1 output)
W2 = np.random.randn(4, 1)
```

```python
b2 = np.random.randn(1, 1)

# Forward Propagation
Z1 = np.dot(X, W1) + b1     # Linear combination (input to hidden)
A1 = relu(Z1)               # Activation function (ReLU)
Z2 = np.dot(A1, W2) + b2    # Hidden to output
A2 = sigmoid(Z2)            # Output activation (Sigmoid for binary classification)

print("Final output prediction:", A2)
```