1 Explain the concept of forward propagation in a neural network

Forward propagation in a neural network is the process where input data is passed through the network layers to compute the output. It involves the following steps:

1. **Input Layer**: The input features are fed into the network.
2. **Weighted Sum**: Each input is multiplied by a weight, and a bias is added.
3. **Activation Function**: The weighted sum is passed through an activation function to introduce non-linearity.
4. **Hidden Layers**: The process repeats in hidden layers until the final output is computed.
5. **Output Layer**: The result is generated, which is the network's prediction.

This process helps in making predictions by utilizing learned weights and biases.

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

The activation function introduces non-linearity into the network. Without it, the network would only be able to learn linear relationships, limiting its ability to model complex patterns. It enables the network to approximate complex functions by transforming the weighted sum of inputs at each layer, allowing it to learn from data more effectively.

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

Backpropagation is the process used to train a neural network by minimizing the error between predicted and actual outputs. The steps involved are:

1. **Calculate the Loss**: Compute the error (loss) between the network's prediction and the actual output using a loss function.
2. **Compute Gradients**: Calculate the gradients (partial derivatives) of the loss with respect to each weight and bias in the network by applying the chain rule.
3. **Backpropagate the Error**: Start from the output layer and propagate the error backward through the hidden layers, adjusting weights and biases at each layer.
4. **Update Weights**: Use an optimization algorithm like gradient descent to adjust the weights and biases to minimize the loss.
5. **Repeat**: Iterate through the training process for multiple epochs until the model converges.

This process enables the network to learn from its mistakes and improve its predictions over time.

4 What is the purpose of the chain rule in backpropagation

The chain rule in backpropagation allows us to compute the gradient of the loss function with respect to each weight and bias in the network by breaking down the complex derivatives into simpler, manageable parts. It enables the error to be propagated backward through the network layer by layer, allowing the gradients to be calculated efficiently and ensuring that the weights are updated correctly in the direction that reduces the error.

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

```
                import numpy as np


# Define the sigmoid activation function and its derivative

def sigmoid(x):

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


# Define the sigmoid derivative for backpropagation

def sigmoid_derivative(x):

    return x * (1 - x)


# Initialize input data (example: 4 samples, 2 features)

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

            [0, 1],

            [1, 0],

            [1, 1]])


# Initialize the output labels (example: 4 samples, 1 output)

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


# Initialize weights and biases
```

```
input_layer_neurons = 2   # Number of input features
hidden_layer_neurons = 2  # Number of neurons in hidden layer
output_layer_neurons = 1  # Number of output neurons

# Random weights and biases initialization
np.random.seed(42)
weights_input_hidden = np.random.rand(input_layer_neurons, hidden_layer_neurons)
bias_hidden = np.random.rand(1, hidden_layer_neurons)
weights_hidden_output = np.random.rand(hidden_layer_neurons, output_layer_neurons)
bias_output = np.random.rand(1, output_layer_neurons)

# Forward propagation process
# Step 1: Compute the hidden layer's input (weighted sum)
hidden_layer_input = np.dot(X, weights_input_hidden) + bias_hidden

# Step 2: Apply the activation function to the hidden layer
hidden_layer_output = sigmoid(hidden_layer_input)

# Step 3: Compute the output layer's input (weighted sum)
output_layer_input = np.dot(hidden_layer_output, weights_hidden_output) + bias_output

# Step 4: Apply the activation function to the output layer
output_layer_output = sigmoid(output_layer_input)

# Print the output of forward propagation
print("Output from forward propagation:\n", output_layer_output)
```