```python
#FORWARD AND BACKWARD PROPAGATION.


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

# Forward propagation is the process by which input data flows through a neural network from the input layer to the output layer to gener

# The input data is fed into the input layer neurons
# Each neuron calculates a weighted sum of its inputs (including a bias term)
# An activation function is applied to this weighted sum to introduce non-linearity
# The output from one layer becomes the input to the next layer
# This process continues until the output layer produces the final prediction

# Key characteristics:

# Data flows in one direction (forward)
# No learning occurs during forward propagation
# The network's current weights determine the output


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

# The activation function serves several important purposes:

# Introduces non-linearity: Without activation functions, neural networks would only be able to learn linear relationships, no matter how
# Determines neuron output: It transforms the weighted sum of inputs into an output value that gets passed to the next layer.
# Enables learning complex patterns: Different activation functions (ReLU, sigmoid, tanh, etc.) allow networks to learn different types o
# Controls output range: Some functions (like sigmoid) bound outputs between 0-1, which is useful for probability outputs.
# Helps with gradient flow: Certain activation functions help mitigate the vanishing gradient problem during backpropagation.


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

# Backpropagation is the algorithm used to train neural networks by adjusting weights based on the error in predictions. The steps are:
# Forward pass: Perform forward propagation to get the network's output.
# Calculate loss: Compute the error between predicted output and true labels using a loss function.
# Initialize gradient: Start at the output layer by calculating the gradient of the loss with respect to the output.
# Backward pass:
# Calculate gradient of loss with respect to weights in current layer
# Use chain rule to propagate error backward through the network
# Compute how much each weight contributed to the error
# Update weights:
# Adjust weights in proportion to their contribution to the error
# Use optimization algorithm (typically gradient descent) to update weights
# Learning rate determines size of weight updates
# Repeat: Iterate through these steps for multiple epochs until the model converges.


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

# The chain rule from calculus is fundamental to backpropagation because:
# Error attribution: It allows us to decompose the overall error into contributions from each individual weight in the network.
# Efficient computation: It provides a systematic way to calculate derivatives of composite functions (which neural networks are) by bre
# Layer-by-layer propagation: The chain rule enables us to compute gradients for earlier layers by multiplying gradients from later laye
# Partial derivatives: It helps compute how much a small change in each weight affects the final output error.
# Recursive calculation: The chain rule allows gradients to be computed recursively from the output layer back to the input layer.


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

import numpy as np

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

def forward_propagation(X, W1, b1, W2, b2):
    """
    X: input data (n_features x n_samples)
    W1: weights of hidden layer (n_hidden x n_features)
    b1: biases of hidden layer (n_hidden x 1)
    W2: weights of output layer (n_output x n_hidden)
    b2: biases of output layer (n_output x 1)
    """

    # Hidden layer calculations
    Z1 = np.dot(W1, X) + b1  # Weighted sum
    A1 = sigmoid(Z1)         # Activation

    # Output layer calculations
    Z2 = np.dot(W2, A1) + b2  # Weighted sum
    A2 = sigmoid(Z2)          # Activation (output)
```

```python
    return A2, (Z1, A1, Z2, A2)  # Return output and cache for backprop

# Example usage
n_features = 3
n_hidden = 4
n_output = 1
n_samples = 5

# Initialize random weights and biases
W1 = np.random.randn(n_hidden, n_features)
b1 = np.zeros((n_hidden, 1))
W2 = np.random.randn(n_output, n_hidden)
b2 = np.zeros((n_output, 1))

# Create random input data
X = np.random.randn(n_features, n_samples)

# Perform forward propagation
output, cache = forward_propagation(X, W1, b1, W2, b2)
print("Network output shape:", output.shape)
print("Sample output values:", output[:, :3])

# This implementation:
# Defines a sigmoid activation function
# Implements forward propagation for a network with one hidden layer
# Calculates weighted sums and applies activation functions at each layer
# Returns both the final output and intermediate values (for potential use in backpropagation)
# Includes example initialization and usage
```

```
Network output shape: (1, 5)
Sample output values: [[0.44975765 0.29147325 0.17069313]]
```