```
import numpy as np

# Generate synthetic data
np.random.seed(42)  # for reproducibility

# Number of samples and features
num_samples = 1000
num_features = 10

# Generate random feature vectors
X_train = np.random.randn(num_samples, num_features)

# Generate corresponding labels (binary classification)
y_train = np.random.randint(2, size=(num_samples, 1))

# Print shapes for verification
print("X_train shape:", X_train.shape)
print("y_train shape:", y_train.shape)
```

```
X_train shape: (1000, 10)
y_train shape: (1000, 1)
```

```
# Define neural network architecture
layer_sizes = [num_features, 2, 4, 8, 1]  # Including input and output layers
num_layers = len(layer_sizes) - 1  # Excluding input layer

# Initialize parameters
parameters = {}
for l in range(1, num_layers + 1):
    parameters['W' + str(l)] = np.random.randn(layer_sizes[l], layer_sizes[l-1]) * 0.01
    parameters['b' + str(l)] = np.zeros((layer_sizes[l], 1))

# Sigmoid activation function
def sigmoid(z):
    return 1 / (1 + np.exp(-z))



# Forward propagation
def forward_propagation(X, parameters):
    cache = {'A0': X.T}

    for l in range(1, num_layers + 1):
        cache['Z' + str(l)] = np.dot(parameters['W' + str(l)], cache['A' + str(l-1)]) + parameters['b' + str(l)]
        cache['A' + str(l)] = sigmoid(cache['Z' + str(l)])

    return cache

# Binary cross-entropy loss
def binary_cross_entropy_loss(A, Y):
    m = Y.shape[0]
    loss = -1/m * np.sum(Y * np.log(A) + (1 - Y) * np.log(1 - A))
    return loss

# Backpropagation
def backward_propagation(cache, parameters, X, Y):
    m = Y.shape[0]
    grads = {}
    dZ = cache['A' + str(num_layers)] - Y.T

    for l in range(num_layers, 0, -1):
        grads['dW' + str(l)] = 1/m * np.dot(dZ, cache['A' + str(l-1)].T)
        grads['db' + str(l)] = 1/m * np.sum(dZ, axis=1, keepdims=True)
        dZ = np.dot(parameters['W' + str(l)].T, dZ) * cache['A' + str(l-1)] * (1 - cache['A' + str(l-1)])

    return grads

# Update parameters using gradient descent
def update_parameters(parameters, grads, learning_rate):
    for l in range(1, num_layers + 1):
        parameters['W' + str(l)] -= learning_rate * grads['dW' + str(l)]
        parameters['b' + str(l)] -= learning_rate * grads['db' + str(l)]
```

```python
# Training hyperparameters
learning_rate = 0.01
num_epochs = 1000

# Training loop
for epoch in range(num_epochs):
    # Forward propagation
    cache = forward_propagation(X_train, parameters)

    # Calculate loss
    loss = binary_cross_entropy_loss(cache['A' + str(num_layers)].T, y_train)

    # Backpropagation
    grads = backward_propagation(cache, parameters, X_train, y_train)

    # Update parameters
    update_parameters(parameters, grads, learning_rate)

    # Print loss every 100 epochs
    if epoch % 100 == 0:
        print(f"Epoch {epoch}, Cost: {loss}")
```

```
Epoch 0, Cost: 0.6931516932333045
Epoch 100, Cost: 0.6931341691822668
Epoch 200, Cost: 0.6931302859331474
Epoch 300, Cost: 0.6931294254084869
Epoch 400, Cost: 0.6931292347147695
Epoch 500, Cost: 0.6931291924564577
Epoch 600, Cost: 0.6931291830918574
Epoch 700, Cost: 0.6931291810166236
Epoch 800, Cost: 0.693129180556743
Epoch 900, Cost: 0.6931291804548314
```

```python
# Training loop with outputs
for epoch in range(num_epochs):
    # Forward propagation
    cache = forward_propagation(X_train, parameters)

    # Calculate loss
    loss = binary_cross_entropy_loss(cache['A' + str(num_layers)].T, y_train)

    # Backpropagation
    grads = backward_propagation(cache, parameters, X_train, y_train)

    # Update parameters
    update_parameters(parameters, grads, learning_rate)

    # Print outputs every 100 epochs
    if epoch % 100 == 0:
        print(f"Epoch {epoch}, Loss: {loss}")
        print("Forward Propagation Outputs:")
        for key, value in cache.items():
            print(key, ":", value)
        print("Backward Propagation Gradients:")
        for key, value in grads.items():
            print(key, ":", value)
        print("\n")
```

```
  0.5029996  0.5029996  0.5029996  0.5029996  0.5029996  0.5029996
  0.5029996  0.5029996  0.5029996  0.5029996  0.5029996  0.5029996
  0.5029996  0.5029996  0.5029996  0.5029996  0.5029996  0.5029996
  0.5029996  0.5029996  0.5029996  0.5029996  0.5029996  0.5029996
  0.50299961 0.5029996  0.5029996  0.5029996  0.5029996  0.5029996
  0.5029996  0.5029996  0.5029996  0.5029996  0.5029996  0.5029996
  0.5029996  0.5029996  0.5029996  0.5029996  0.5029996  0.5029996
  0.5029996  0.5029996  0.5029996  0.5029996  0.5029996  0.5029996
  0.5029996  0.5029996  0.5029996  0.5029996  0.5029996  0.5029996
  0.5029996  0.5029996  0.5029996  0.5029996  0.5029996  0.5029996
  0.5029996  0.5029996  0.5029996  0.5029996  0.5029996  0.5029996
  0.5029996  0.5029996  0.5029996  0.5029996  0.5029996  0.5029996
  0.5029996  0.5029996  0.5029996  0.5029996  0.5029996  0.5029996
  0.5029996  0.5029996  0.5029996  0.5029996  0.5029996  0.5029996
  0.5029996  0.5029996  0.5029996  0.5029996  0.5029996  0.5029996
  0.5029996  0.5029996  0.5029996  0.5029996  0.5029996  0.5029996
  0.5029996  0.5029996  0.5029996  0.5029996  0.5029996  0.5029996
  0.5029996  0.5029996  0.5029996  0.5029996  0.5029996  0.5029996
  0.5029996  0.5029996  0.5029996  0.5029996  0.5029996  0.5029996
  0.5029996  0.5029996  0.5029996  0.5029996  0.5029996  0.5029996
  0.5029996  0.5029996  0.5029996  0.5029996  0.5029996  0.5029996
  0.5029996  0.5029996  0.5029996  0.5029996  0.5029996  0.5029996
  0.5029996  0.5029996  0.5029996  0.5029996  0.5029996  0.5029996
  0.5029996  0.5029996  0.5029996  0.5029996  0.5029996  0.5029996
  0.5029996  0.5029996  0.5029996  0.5029996  0.5029996  0.5029996
  0.5029996  0.5029996  0.5029996  0.5029996  0.5029996  0.5029996
  0.5029996  0.5029996  0.5029996  0.5029996  0.5029996  0.5029996
  0.5029996  0.5029996  0.5029996  0.5029996  0.5029996  0.5029996
  0.5029996  0.5029996  0.5029996  0.5029996  0.5029996  0.5029996
  0.5029996  0.5029996  0.5029996  0.5029996  0.5029996  0.5029996
  0.5029996  0.5029996  0.5029996  0.5029996  0.5029996  0.5029996
  0.5029996  0.5029996  0.5029996  0.5029996 ]]
Backward Propagation Gradients:
```