```python
import numpy as np
import matplotlib.pyplot as plt
from sklearn.datasets import fetch_openml
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import OneHotEncoder

# Load Fashion-MNIST dataset
def load_fashion_mnist():
    fashion_mnist = fetch_openml("Fashion-MNIST")
    X = fashion_mnist.data / 255.0  # normalize pixel values
    y = fashion_mnist.target.astype(int)
    return X, y

# One-hot encoding for labels
def one_hot_encoding(y, num_classes):
    encoder = OneHotEncoder(sparse_output=False, categories='auto')
    y_one_hot = encoder.fit_transform(y.to_numpy().reshape(-1, 1))
    return y_one_hot

# ReLU activation
def relu(x):
    return np.maximum(0, x)

# Derivative of ReLU
def relu_derivative(x):
    return np.where(x > 0, 1, 0)

# Softmax function
def softmax(x):
    exp_x = np.exp(x - np.max(x, axis=1, keepdims=True))
    return exp_x / np.sum(exp_x, axis=1, keepdims=True)

# Cross-entropy loss
def cross_entropy_loss(y_true, y_pred):
    return -np.mean(np.sum(y_true * np.log(y_pred + 1e-8), axis=1))

# Compute accuracy
def accuracy(y_true, y_pred):
    return np.mean(np.argmax(y_true, axis=1) == np.argmax(y_pred, axis=1))

# # Initialize weights and biases
# def initialize_parameters(input_size, hidden_size, output_size):
#     np.random.seed(0)
#     W1 = np.random.randn(input_size, hidden_size) * 0.01
#     b1 = np.zeros((1, hidden_size))
#     W2 = np.random.randn(hidden_size, output_size) * 0.01
#     b2 = np.zeros((1, output_size))
#     return W1, b1, W2, b2
```

```
# Dropout layer function
def apply_dropout(A, dropout_rate):
    mask = np.random.rand(*A.shape) > dropout_rate  # Create mask for dropout
    A_dropout = mask * A / (1 - dropout_rate)  # Scale activations and apply mask
    return A_dropout, mask

# Forward pass
def forward_pass(X, W1, b1, W2, b2, W3, b3, dropout_rate, training):
    Z1 = np.dot(X, W1) + b1
    A1 = relu(Z1)

    if training:  # Apply dropout only during training
        A1, mask1 = apply_dropout(A1, dropout_rate)
    else:
        mask1 = None

    Z2 = np.dot(A1, W2) + b2
    A2 = relu(Z2)

    if training:  # Apply dropout only during training
        A2, mask2 = apply_dropout(A2, dropout_rate)
    else:
        mask2 = None

    Z3 = np.dot(A2, W3) + b3
    A3 = softmax(Z3)

    return Z1, A1, Z2, A2, Z3, A3, mask1, mask2

# Backward pass
def backward_pass(X, y_true, Z1, Z2, A1, A2, A3, W2, W3, mask1, mask2, dropout_rate):
    m = X.shape[0]

    # Output layer gradients
    dZ3 = A3 - y_true
    dW3 = np.dot(A2.T, dZ3) / m
    db3 = np.sum(dZ3, axis=0, keepdims=True) / m

    # Second hidden layer gradients
    dA2 = np.dot(dZ3, W3.T)
    if mask2 is not None:
        dA2 = dA2 * mask2 / (1 - dropout_rate)  # Apply mask during backprop

    dZ2 = dA2 * relu_derivative(Z2)
    dW2 = np.dot(A1.T, dZ2) / m
    db2 = np.sum(dZ2, axis=0, keepdims=True) / m

    # First hidden layer gradients
```

```
    dA1 = np.dot(dZ2, W2.T)
    if mask1 is not None:
        dA1 = dA1 * mask1 / (1 - dropout_rate)  # Apply mask during backprop

    dZ1 = dA1 * relu_derivative(Z1)
    dW1 = np.dot(X.T, dZ1) / m
    db1 = np.sum(dZ1, axis=0, keepdims=True) / m

    return dW1, db1, dW2, db2, dW3, db3

# Mini-batch SGD
def sgd_update(W1, b1, W2, b2, W3, b3, dW1, db1, dW2, db2, dW3, db3, learning_rate):
    W1 -= learning_rate * dW1
    b1 -= learning_rate * db1
    W2 -= learning_rate * dW2
    b2 -= learning_rate * db2
    W3 -= learning_rate * dW3
    b3 -= learning_rate * db3
    return W1, b1, W2, b2, W3, b3

# Train MLP
def train_mlp(X_train, y_train, X_test, y_test, hidden_size, batch_size, epochs, learning_rate,
dropout_rate):
    input_size = X_train.shape[1]
    output_size = y_train.shape[1]

    # Initialize weights and biases
    np.random.seed(0)
    W1 = np.random.randn(input_size, hidden_size) * 0.01
    b1 = np.zeros((1, hidden_size))
    # W2 = np.random.randn(hidden_size, output_size) * 0.01
    # b2 = np.zeros((1, output_size))
    W2 = np.random.randn(hidden_size,hidden_size) * 0.01
    b2 = b1 = np.zeros((1, hidden_size))
    W3 = np.random.randn(hidden_size, output_size) * 0.01
    b3 = np.zeros((1, output_size))

    training_acc = []
    testing_acc = []

    for epoch in range(epochs):
        # Shuffle training data
        indices = np.random.permutation(X_train.shape[0])
        X_train_shuffled = X_train.iloc[indices]
        y_train_shuffled = y_train[indices]

        # Mini-batch training
        for i in range(0, X_train.shape[0], batch_size):
            X_batch = X_train_shuffled[i:i+batch_size]
```

```
            y_batch = y_train_shuffled[i:i+batch_size]

            # Forward pass
            Z1, A1, Z2, A2, Z3, A3, mask1, mask2 = forward_pass(X_batch, W1, b1, W2, b2, W3, b3, dropout_rate, training=True)

            # Backward pass
            dW1, db1, dW2, db2, dW3, db3 = backward_pass(X_batch, y_batch, Z1, Z2, A1, A2, A3, W2, W3, mask1, mask2, dropout_rate)

            # Update weights
            W1, b1, W2, b2, W3, b3 = sgd_update(W1, b1, W2, b2, W3, b3, dW1, db1, dW2, db2, dW3, db3, learning_rate)

        # Calculate accuracy for training and test sets
        _, _, _, _, _, train_pred, _, _ = forward_pass(X_train, W1, b1, W2, b2, W3, b3, dropout_rate, training=False)
        _, _, _, _, _, test_pred, _, _ = forward_pass(X_test, W1, b1, W2, b2, W3, b3, dropout_rate, training=False)

        train_acc = accuracy(y_train, train_pred)
        test_acc = accuracy(y_test, test_pred)

        training_acc.append(train_acc)
        testing_acc.append(test_acc)

        print(f'Epoch {epoch+1}/{epochs} - Training Accuracy: {train_acc:.4f}, Test Accuracy: {test_acc:.4f}')

    return training_acc, testing_acc


if __name__ == '__main__':
    X, y = load_fashion_mnist()

    # Split into train and test sets
    X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)

    # One-hot encode labels
    num_classes = 10
    y_train_one_hot = one_hot_encoding(y_train, num_classes)
    y_test_one_hot = one_hot_encoding(y_test, num_classes)

    # Train MLP
    hidden_size=256
    batch_size=256
    epochs=10
    learning_rate=0.1
    dropout_rate=0.5
```

```
    training_acc, testing_acc = train_mlp(X_train, y_train_one_hot, X_test, y_test_one_hot,
hidden_size, batch_size, epochs, learning_rate, dropout_rate)

    # Plot accuracy curves
    plt.plot(training_acc, label='Training Accuracy')
    plt.plot(testing_acc, label='Test Accuracy')
    plt.xlabel('Epochs')
    plt.ylabel('Accuracy')
    plt.title('Accuracy vs. Epochs')
    plt.legend()
    plt.savefig('p4_d.jpg')
```