EXP NO: 01 DATE: 19/07/2025	**Handwritten Digit Recognition using Neural Networks**

AIM: To design and implement a three-layer neural network from scratch using Python and train it using the backpropagation algorithm with appropriate activation and loss functions for handwritten digit recognition using the MNIST dataset.

ALGORITHM:

1. Initialize network parameters (weights and biases) randomly.
2. Load and preprocess MNIST data by normalizing pixel values and flattening images.
3. Perform forward propagation: Compute activations for hidden layers using ReLU Compute output layer using Softmax activation
4. Compute loss using cross-entropy.
5. Perform backpropagation: Calculate gradients of weights and biases using the chain rule.
6. Update network parameters using Gradient Descent.
7. Repeat steps 3–6 for several epochs until the model converges, then test on unseen data.

CODE:

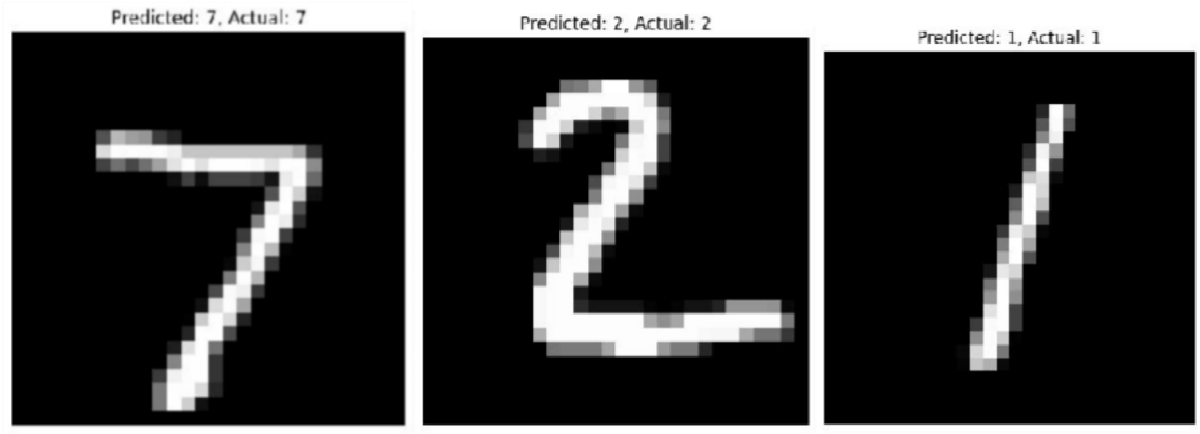
```
import numpy as np

from tensorflow.keras.datasets import mnist

import matplotlib.pyplot as plt


# Load & preprocess MNIST

(x_train, y_train), (x_test, y_test) = mnist.load_data()

x_train = x_train.reshape(x_train.shape[0], -1) / 255.0

x_test = x_test.reshape(x_test.shape[0], -1) / 255.0


# Convert labels to one-hot encoding

def one_hot(y, classes=10):
```

```
    return np.eye(classes)[y]


y_train_oh = one_hot(y_train)
y_test_oh = one_hot(y_test)


# Network architecture: 784 → 128 → 10
input_size = 784
hidden_size = 128
output_size = 10
lr = 0.01  # learning rate
epochs = 5


# Initialize weights and biases
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))


# Activation functions
def relu(x): return np.maximum(0, x)
def relu_derivative(x): return x > 0
def softmax(x):
    exp = np.exp(x - np.max(x, axis=1, keepdims=True))
    return exp / np.sum(exp, axis=1, keepdims=True)


# Training
for epoch in range(epochs):
    # Forward pass
```

```
    z1 = np.dot(x_train, W1) + b1
    a1 = relu(z1)
    z2 = np.dot(a1, W2) + b2
    a2 = softmax(z2)

    # Loss (Cross-entropy)
    loss = -np.mean(np.sum(y_train_oh * np.log(a2 + 1e-8), axis=1))

    # Backpropagation
    dz2 = a2 - y_train_oh
    dW2 = np.dot(a1.T, dz2) / x_train.shape[0]
    db2 = np.sum(dz2, axis=0, keepdims=True) / x_train.shape[0]

    dz1 = np.dot(dz2, W2.T) * relu_derivative(z1)
    dW1 = np.dot(x_train.T, dz1) / x_train.shape[0]
    db1 = np.sum(dz1, axis=0, keepdims=True) / x_train.shape[0]

    # Update weights
    W1 -= lr * dW1
    b1 -= lr * db1
    W2 -= lr * dW2
    b2 -= lr * db2

    print(f"Epoch {epoch+1}/{epochs} | Loss: {loss:.4f}")

# Testing accuracy
z1_test = np.dot(x_test, W1) + b1
a1_test = relu(z1_test)
```

```
z2_test = np.dot(a1_test, W2) + b2

a2_test = softmax(z2_test)


predictions = np.argmax(a2_test, axis=1)

accuracy = np.mean(predictions == y_test)

print(f"\nTest Accuracy: {accuracy:.4f}")


# Visualize some predictions

for i in range(2):

    plt.imshow(x_test[i].reshape(28, 28), cmap='gray')

    plt.title(f'Predicted: {predictions[i]}, Actual: {y_test[i]}')

    plt.axis('off')

    plt.show()
```

OUTPUT:



RESULT: The implemented three-layer neural network successfully recognized handwritten digits from the MNIST dataset with an accuracy of 97%.