**Neural Network Implementation for MNIST Classification**

This repository contains a Python script (nnScript.py) that implements a neural network from scratch for the classification of handwritten digits using the MNIST dataset.

**Overview**

The script is divided into several key functions, each handling specific aspects of the data preprocessing, model training, and evaluation workflow. Below is a description of the main functions used in the script.

**Functions**

**1. preprocess()**

**Description**:  
Handles the loading and preprocessing of the MNIST dataset. This function performs normalization, data splitting (train, validation, test), and feature selection.

**Key Features**:

* Loads mnist\_all.mat file.
* Splits data into training, validation, and test sets.
* Normalizes pixel values to [0, 1].
* Removes duplicate features to optimize model performance.

**2. initializeWeights(input\_size, output\_size)**

**Description**:  
Initializes the weights and biases for the neural network using a random uniform distribution.

**Parameters**:

* input\_size: Number of input features.
* output\_size: Number of output nodes.

**Returns**:

* A dictionary containing initialized weight and bias matrices.

**3. sigmoid(z)**

**Description**:  
Computes the sigmoid activation function.

**Parameters**:

* z: Input value or matrix.

**Returns**:

* Sigmoid-transformed value.

**4. nnObjFunction(params, \*args)**

**Description**:  
Defines the objective function for the neural network. This function calculates the forward pass, backpropagation, and the cost function including regularization.

**Parameters**:

* params: Flattened weight and bias values.
* \*args: Contains training data, labels, and other hyperparameters.

**Returns**:

* The computed cost and gradients for optimization.

**5. nnPredict(W1, W2, data)**

**Description**:  
Predicts the class labels for a given dataset using the trained neural network.

**Parameters**:

* W1, W2: Weights of the neural network.
* data: Input data.

**Returns**:

* Predicted labels.

**6. train()**

**Description**:  
Handles the training process of the neural network by optimizing the weights using a minimization function (scipy.optimize.minimize).

**Key Features**:

* Calls nnObjFunction for gradient computation.
* Optimizes weights using L-BFGS-B method.

**7. evaluateModel(W1, W2, data, labels)**

**Description**:  
Evaluates the trained model by calculating its accuracy on a given dataset.

**Parameters**:

* W1, W2: Trained weights.
* data: Dataset to evaluate.
* labels: True labels for the dataset.

**Returns**:

* Accuracy of the model on the dataset.

**How to Run**

1. **Preprocess Data**:  
   The preprocess() function automatically prepares the data for training and testing.
2. **Train the Model**:  
   Use the train() function to train the neural network.
3. **Evaluate the Model**:  
   Evaluate model performance using the evaluateModel() function.

**Dependencies**

Ensure the following Python libraries are installed:

* numpy
* scipy
* matplotlib