Ex-1: Multilayer Perceptron (MLP) for MNIST Handwritten Digit Classification

1. Introduction

A **Multilayer Perceptron (MLP)** is a type of **Artificial Neural Network (ANN)** consisting of multiple layers of neurons. It is used for various machine learning tasks, including **image classification, speech recognition, and natural language processing**. In this experiment, we will use an MLP to classify the **MNIST handwritten digits dataset**.

---

2. Algorithm: Multilayer Perceptron (MLP)

Steps of the Algorithm

1. **Load the Dataset** – Import the MNIST dataset, which contains **28x28 grayscale images** of digits (0-9).
2. **Preprocess the Data** – Normalize pixel values and convert labels to one-hot encoding.
3. **Build the MLP Model** – Create an MLP model using the **Keras Sequential API**.
4. **Compile the Model** – Define the loss function, optimizer, and evaluation metric.
5. **Train the Model** – Feed the dataset into the model and train it.
6. **Evaluate the Model** – Test the model's accuracy on unseen data.
7. **Make Predictions** – Use the trained model to classify handwritten digits.

---

3. Implementing MLP for MNIST in Python

We will use **Keras** (a high-level API of TensorFlow) to build and train our neural network.

Step 1: Import Required Libraries

```
import tensorflow as tf  # TensorFlow is the backend framework
from keras.models import Sequential  # Used to build a linear stack of layers
from keras.layers import Dense, Flatten  # Dense (Fully connected layer), Flatten (Convert 2D to 1D)
from keras.datasets import mnist  # Import the MNIST dataset
from keras.utils import to_categorical  # Converts labels to one-hot encoding

```

Explanation:

- tensorflow is the core deep learning framework.
- Sequential creates a **linear stack of layers** for our model.
- Dense represents a **fully connected (FC) layer** in the neural network.
- Flatten converts a **2D image** (28x28) into a **1D array** for the MLP.

- mnist.load_data() loads the **MNIST dataset** directly from Keras.
- to_categorical() converts integer labels into **one-hot encoding** format.

Step 2: Load and Preprocess the Data

```
# Load MNIST dataset
(train_images, train_labels), (test_images, test_labels) = mnist.load_data()

# Normalize image pixel values (Convert range from 0-255 to 0-1)
train_images, test_images = train_images / 255.0, test_images / 255.0

# Convert class labels to one-hot encoding
train_labels = to_categorical(train_labels, 10)
test_labels = to_categorical(test_labels, 10)
```

Explanation:

- mnist.load_data() returns two sets:
    - train_images, train_labels (Training set)
    - test_images, test_labels (Testing set)
- **Normalization:** Pixel values range from **0-255** (grayscale). We divide them by **255** to scale them between **0-1**, which helps in faster training.
- **One-Hot Encoding:**
    - The labels (digits 0-9) are converted into **categorical format**.
    - Example: If a label is 3, one-hot encoding converts it to [0,0,0,1,0,0,0,0,0,0].

Step 3: Build the MLP Model

```
# Define the MLP model
model = Sequential([
    Flatten(input_shape=(28, 28)),  # Flatten 28x28 images into a 1D array
    Dense(128, activation='relu'),  # Fully connected layer with 128 neurons
    Dense(10, activation='softmax')  # Output layer with 10 neurons (one for each digit)
])
```

Explanation:

- Flatten(input_shape=(28, 28)) – Converts each **28x28** image into a **1D vector of 784 values**.
- Dense(128, activation='relu') – A **hidden layer** with **128 neurons** using **ReLU (Rectified Linear Unit)** activation function.
- Dense(10, activation='softmax') – The **output layer** with **10 neurons** (one for each digit).
  - **Softmax Activation:** Converts output values into probabilities.
  - Example: [0.01, 0.02, 0.87, 0.05, 0.01, ...] (highest value is digit 2).

Step 4: Compile the Model

```
# Compile the model
model.compile(optimizer='adam', loss='categorical_crossentropy', metrics=['accuracy'])

```

Explanation:

- **Optimizer:**
  - adam (Adaptive Moment Estimation) is used for **fast and efficient training**.
- **Loss Function:**
  - categorical_crossentropy is used for **multi-class classification**.
- **Metric:**
  - accuracy measures the percentage of **correct predictions**.

Step 5: Train the Model

```
# Train the model
model.fit(train_images, train_labels, epochs=5, batch_size=32, validation_split=0.2)

```

Explanation:

- **epochs=5** – The model will iterate 5 times over the dataset.
- **batch_size=32** – The model will process 32 samples before updating weights.

- **validation_split=0.2** – 20% of training data is used for **validation**.

Step 6: Evaluate the Model

```python
# Evaluate model performance
test_loss, test_acc = model.evaluate(test_images, test_labels)
print(f'Test Accuracy: {test_acc}')

```

Explanation:

- The model evaluates on the **test set** to measure real-world performance.
- test_acc represents **accuracy on unseen data**.

Step 7: Make Predictions

```python
import numpy as np
import matplotlib.pyplot as plt

# Get predictions for the first 5 test images
predictions = model.predict(test_images[:5])

# Plot the images with predicted labels
for i in range(5):
    plt.imshow(test_images[i], cmap='gray')
    plt.title(f'Predicted: {np.argmax(predictions[i])}')
    plt.show()

```

Explanation:

- model.predict() generates predictions for input images.
- np.argmax(predictions[i]) retrieves the digit with the **highest probability**.
- **plt.imshow()** displays the image along with the predicted label.

---

4. Expected Output

- **Train Accuracy:** ~98% after 5 epochs.
- **Test Accuracy:** ~97% (depends on hyperparameters).
- **Prediction Example:**
  - Model correctly classifies a handwritten 5 as 5.
  - Model correctly classifies a handwritten 7 as 7.

---

5. Advantages of Using MLP for MNIST

✅ **Simple and Efficient** – Fast training on small datasets.
✅ **Good Generalization** – Performs well on unseen data.
✅ **Easily Extendable** – Can be modified with more layers for better accuracy.

---

6. Result

We implemented a **Multilayer Perceptron (MLP)** to classify MNIST handwritten digits. The model was trained using **Keras** and achieved **high accuracy**. We evaluated the model on **test data** and made **real-time predictions**.

```python
# Import Required Libraries
import tensorflow as tf  # TensorFlow is the backend framework
from keras.models import Sequential  # Used to build a linear stack of layers
from keras.layers import Dense, Flatten  # Dense (Fully connected layer), Flatten (Convert 2D to 1D)
from keras.datasets import mnist  # Import the MNIST dataset
from keras.utils import to_categorical  # Converts labels to one-hot encoding
import numpy as np
import matplotlib.pyplot as plt

# Load and Preprocess the Data
(train_images, train_labels), (test_images, test_labels) = mnist.load_data()
train_images, test_images = train_images / 255.0, test_images / 255.0
train_labels = to_categorical(train_labels, 10)
test_labels = to_categorical(test_labels, 10)

# Build the MLP Model
model = Sequential([
    Flatten(input_shape=(28, 28)),  # Flatten 28x28 images into a 1D array
    Dense(128, activation='relu'),  # Fully connected layer with 128 neurons
    Dense(10, activation='softmax')  # Output layer with 10 neurons (one for each digit)
])


# Compile the Model
model.compile(optimizer='adam', loss='categorical_crossentropy', metrics=['accuracy'])

# Train the Model
model.fit(train_images, train_labels, epochs=5, batch_size=32, validation_split=0.2)

# Evaluate the Model
test_loss, test_acc = model.evaluate(test_images, test_labels)
print(f'Test Accuracy: {test_acc}')

# Make Predictions
predictions = model.predict(test_images[:5])

# Plot the images with predicted labels
for i in range(5):
    plt.imshow(test_images[i], cmap='gray')
    plt.title(f'Predicted: {np.argmax(predictions[i])}')
    plt.show()
```