Neural Network Assignment

Mentor: Ankit Maurya and Sachin Awasthi

Objective

Your task is to build a simple neural network from scratch (without using any pre-built frameworks like TensorFlow or PyTorch for the model) and train it on the MNIST dataset of handwritten digits. The goal is to understand the underlying steps involved in creating a neural network, including forward propagation, loss computation, backward propagation, and parameter updates.

Instructions

1. **Get the MNIST dataset:**

   - Use the `keras.datasets` library to download the MNIST dataset.
   - Example code to load the dataset:

```
from keras.datasets import mnist
(X_train, y_train), (X_test, y_test) = mnist.load_data()
```

   - Alternatively, you can download the dataset from the official MNIST page at http://yann.lecun.com/exdb/mnist/.

2. **Preprocess the data:**

   - Normalize the pixel values of images to the range $[0, 1]$.
   - Flatten the 28×28 images into vectors of size 784 for input into the neural network.

3. **Define a neural network architecture:**

   - Input layer with 784 neurons (one for each pixel in the image).
   - Hidden layer with 128 neurons and ReLU activation function.
   - Output layer with 10 neurons (one for each digit) and softmax activation function.

4. **Implement the following steps in Python:** Follow the steps below to implement the neural network from scratch.

Steps of Neural Network Implementation

Step 1: Data Preprocessing

- Normalize the dataset:

$$X_{\text{normalized}} = \frac{X - \mu}{\sigma + \epsilon}$$

  where μ is the mean, σ is the standard deviation, and ϵ is a small constant to avoid division by zero(like 0.00001).

- Reshape the input data from a 3D tensor (e.g., $28 \times 28 \times \text{num\_samples}$) into a 2D array where each column represents a flattened image.

Step 2: Initialize Parameters

- Randomly initialize weights W_1 and W_2 with small values:

$$W_1 \in \mathbb{R}^{128\times 784}, \quad W_2 \in \mathbb{R}^{10\times 128}$$

- Set biases b_1 and b_2 to zeros:

$$b_1 \in \mathbb{R}^{128\times 1}, \quad b_2 \in \mathbb{R}^{10\times 1}$$

Step 3: Forward Propagation

- Compute the activations:

$$\begin{aligned}
Z_1 &= W_1 \cdot A_0 + b_1 && \text{(Linear transformation for hidden layer)} \\
A_1 &= \text{ReLU}(Z_1) && \text{(ReLU activation)} \\
Z_2 &= W_2 \cdot A_1 + b_2 && \text{(Linear transformation for output layer)} \\
A_2 &= \text{Softmax}(Z_2) && \text{(Softmax activation for probabilities)}
\end{aligned}$$

- Here:

$$\text{ReLU}(Z) = \max(0, Z), \quad \text{Softmax}(Z) = \frac{\exp(Z)}{\sum \exp(Z)}$$

Step 4: Loss Calculation (Cross-Entropy Loss)

- Compute the loss using the cross-entropy loss function:

$$L = -\frac{1}{m} \sum_{i=1}^{m} \sum_{j=1}^{10} y_j^{(i)} \log(A_{2j}^{(i)})$$

  where m is the number of samples, $y_j^{(i)}$ is the one-hot encoded true label, and $A_{2j}^{(i)}$ is the predicted probability for the correct class.

Step 5: Backward Propagation

- Compute gradients for the output layer:

$$dZ_2 = A_2 - \text{one\_hot\_y}$$
$$dW_2 = \frac{1}{m} dZ_2 \cdot A_1^T$$
$$db_2 = \frac{1}{m} \sum dZ_2$$

- Compute gradients for the hidden layer:

$$dZ_1 = W_2^T \cdot dZ_2 \cdot \text{ReLU}'(Z_1)$$
$$dW_1 = \frac{1}{m} dZ_1 \cdot A_0^T$$
$$db_1 = \frac{1}{m} \sum dZ_1$$

- Here, $\text{ReLU}'(Z) = Z > 0$ (1 if $Z > 0$, otherwise 0).

Step 6: Parameter Updates

- Update weights and biases using gradient descent:

$$W_1 = W_1 - \alpha \cdot dW_1$$
$$b_1 = b_1 - \alpha \cdot db_1$$
$$W_2 = W_2 - \alpha \cdot dW_2$$
$$b_2 = b_2 - \alpha \cdot db_2$$

  where α is the learning rate.

Step 7: Evaluate the Model

- Compute the accuracy of the model:

$$\text{Accuracy} = \frac{\text{Number of correct predictions}}{\text{Total number of samples}}$$

Deliverables

1. Submit your Python code implementing the neural network from scratch.
2. Include a report with:
   - Training and test accuracy.
   - Loss curve (loss vs. iterations).
   - Sample predictions with actual and predicted labels.