MNIST handwritten digit recognition

Overview

The **MNIST** data-set is a widely used collection of handwritten digit images, often employed as a benchmark for machine learning algorithms, especially in computer vision. It consists of 70,000 gray-scale images of handwritten digits (0-9), each 28x28 pixels in size. The data-set is split into 60,000 training images and 10,000 test images.
In this task, we are required a create and train a model that can classify or predict each image to it's real class. To achieve the goal, we have used fully connected layers and output layer in our model to classify images.

Techniques and Model Used

To build our neural network based model, we have used fully connected layers and output layer for simple classification. Fully connected layers learn high-level features that can be used when combining and transformation of features are needed. They can handle various input structures without requiring specific assumptions about the input data. We have used four layers: first three layers are fully connected layers and last layer is output layer consisting of 10 units. Let's breakdown the layers unit-wise:

Layer Name	Input	Output	Units
Fully connected 1	784	128	128
Fully connected 2	128	64	64
Fully connected 3	64	32	32
Output	32	10	10

This approach needs input image to be **flattened** first, so that we can deal with 1D input that eliminates the use of convolution layers to keep things simple. For this multi-class classification task, we have used **categorical cross-entropy loss** and **Adam optimizer** as an optimization function to update weights. To train till convergence, we used validation data-set with early stopping method to avoid model over-fitting provided good results. In addition to convergence technique, **scheduling learning rate** is also beneficial.

Preprocessing Steps

Before feeding data into the model, some preprocessing has to be done because raw data often contains inconsistencies, noise, and irrelevant information that can hinder the model's performance and accuracy. In the process, firstly images are converted to **tensor** then **normalization** is applied on the tensor values. Normalization ensures that all features contribute equally during the learning process, preventing larger-magnitude features from dominating smaller ones. Then, we load and split data-set into training and validation, and test data-set.

Challenges

While training, model often over-fits due to finding unnecessary relationships that is not required. Learning parameters on every data-point leads to poor performance on unseen data points. Hence, we introduced validation data-set to ensure that the learned weights shouldn't increase validation loss after each successive epoch. By applying **patience counter** (may vary according to requirement), we keep tracking increasing val. loss and stopped training after the number of successive epochs crossed the patience counter. Then, we load the best model from the checkpoint (for future purpose).

Accuracy and Observations

On test data-set, running the best model achieves a accuracy of 97.86% which is closer to the state-of-the art models.
The loss curves shows that our model works well as expected. From the loss curve, we can see that after epoch 12, validation loss started increasing till epoch 17 and then training stopped to avoid over-fitting.