Deep Learning Assignment - 1

Instructions.

- This coding assignment revolves around developing a basic Multi-Layer Feedforward network and manually training it. Additional instructions for each question are provided accordingly.
- You have to submit the **GoogleColab** file named as **rollno-A1.ipynb**. Files submitted without following naming convention, .py files and any other **will not be evaluated.**
  For example if your roll number is 1234567 / MT34567 file name must be 1234567-A1.ipynb / MT34567-A1.ipynb
- No extensions will be granted for this assignment under any circumstances.
- You can refer to the code available in the notebook shared on Google Classroom for guidance.

40 marks (10 marks for each Question.)

You have to write the code for calculating execution time from scratch, not doing so will fetch 0 marks in respective question.

1. Download the MNIST dataset (provide code to download in google colab) and create a custom dataloader using **torch.utils.data.Dataset, DataLoader**. Write another dataloader completely from scratch and compare the loading performance of your scratch implemented data loader with the one written with PyTorch classes across different batch sizes (128, 256, 512, 1024). Plot a graph illustrating the relationship between batch size and loading time.
2. Implement a Feed-Forward neural network architecture using **torh.nn.Linear** featuring four hidden layers, each comprising minimum 32 neurons (excluding input and output layers). Train the model using the most effective data loader identified in the previous question with **ReLU** activation function. Employ the **Cross-Entropy loss** function and opt for the **Stochastic Gradient Descent (SGD)** optimizer with default parameters, setting the learning rate to 0.0003. Plot graphs depicting the loss and accuracy during training, validation and testing for a total of 60 epochs. For this question you can use whatever PyTorch has to offer.
3. Implement everything defined in Question 2 completely from scratch. Also implement the back-propagation algorithm from scratch using only PyTorch tensor operations (calling PyTorch backward function will fetch you 0 marks). Train the scratch implemented model with the same settings (epochs, learning rate) as in Question 2 and generate graphs depicting the loss and accuracy for training, validation, and testing. Compare the performance of your custom back-propagation algorithm with PyTorch's implementation and provide a report on the results.
4. Execute the tasks outlined in questions 2 and 3, but this time, use a sigmoid activation function while keeping all other parameters and configurations unchanged.

Bonus Marks (2 marks for each Question.)

1. Train a neural network with the same variables and training settings as described in Question 2, but this time, vary the number of hidden layers (n), ensuring that each hidden layer consists of a minimum of 400 neurons. The students achieving the highest value of n will be rewarded bonus marks.
2. Employ your custom back-propagation method to train the network specified in Question 2. The students achieving the shortest training time will be rewarded bonus marks.