Assignment 4: *Fully-Connected Neural Networks*

Date: *Mar 12, 2025*
Due : *Mar 25, 2025*

Acknowledgment This assignment has been adapted from the materials of ECE1513 by S. Emara.

Code of Honor Assignments are designed to enhance your understanding and advance your skills, constituting a significant portion of your final assessment. They must be completed individually, as engaging in any form of academic dishonesty violates the principles of the Code of Honor. If you encounter any challenges while solving the assignments, please contact the instructional team for guidance.

How to Submit This assignment has 3 questions. For each question, you need to upload one file on Crowdmark. All questions ask for your code. You need to copy and paste your code in the file that you submit or print your code and its output terminal as PDF.

Grading The grades add up to 100 and comprise roughly 10% of the final mark.

Deadline The deadline for your submission is on **Mar 25, 2025 at 11:59 PM**. Please note that this deadline is strict and **no late submission will be accepted.**

Questions

Question 1 [50 Points] **(Training a Basic NN)** In this question, we implement a basic fully-connected feedforward neural network (FNN) using PyTorch. To this end, we use the CIFAR 10 dataset, which has 60,000 color-images (RGB), each being 32×32-pixel. The images belong to ten classes, namely `plane`, `car`, `bird`, `cat`, `deer`, `dog`, `frog`, `horse`, `ship`, `truck`. For this questions, you need to import `torchvision` and `torch` libraries in addition to other ones you may need.

1. Use `torchvision.datasets.CIFAR10` to load the *train* and *test* sets of CIFAR 10. To use the dataset in PyTorch, we need to transform Python Image Library (PIL) images to tensors in the range $[0.0, 1.0]$, normalized to be in the range $[-1.0, 1.0]$ with mean and standard deviation be 0.5. This can be done using the transforms

   `torch.transforms.ToTensor()`

   and

   `torch.transforms.Normalize((0.5, 0.5, 0.5), (0.5, 0.5, 0.5))`

   when you load. You will end with a train set of size 50,000 and a test set of size 10,000.

2. Split the train set into two sets: a *training dataset* of size 40,000 and a validation set of size 10,000. For this task, you find function `random_split` in `torch.utils.data` useful.

3. Use `torch.utils.data.DataLoader` to load the training, validation and test sets into mini-batches of size $B = 4$. Pick a random mini-batch from the training set, and plot the images in it with their respective class labels.

4. Implement an FNN with three fully connected layers, i.e., two hidden layers and an output layer. The first hidden layer has 92 neurons and the second one has 46. All neurons in the hidden layers are activated by ReLU. You should figure out the input and output layers based on the problem. You can implement these layers directly using the classes in `torch.nn` module. In your implementation, include the `forward` function for forward pass.

5. Use cross entropy loss and stochastic gradient descent with learning rate 0.001 to train the model for 20 epochs. Plot the *training accuracy* vs. number of epoch. The loss function and optimizer are available in models `torch.nn` and `torch.optim`, respectively.

QUESTION 2 [30 Points] **(Hyperparameter Tuning)** For the implementation in Question 1, we now tune the following *hyperparameters*: batch-size and learning rate.

1. Write a validation function, which validates the accuracy and loss of your trained model on the validation set.

2. Keep the number of epochs to 20. Run the training loop for batch-sizes $B = 4$ and $B = 64$, and learning rates $\eta = 0.01$ and $\eta = 0.001$. This make 4 different combinations in total. Print the end training and validation accuracy and loss for each combination.

3. How do you pick the best batch size and learning rate combination?

QUESTION 3 [20 Points] **(Early Stop)** For the implementation in Questions 1 and 2, we now implement the *early stopping* algorithm. To this end, set the batch-size and learning rate that returned best validation in Question 2. In this algorithm, we do not train for a fix number of epochs. Instead, we validate our model (using the validation set) after every training epoch and stop training when the validation loss stops decreasing (or decreasing is less than a threshold). Implement this algorithm.