Due: 14th April 2021 2:00 PM Total points: **150**

This project has you classifying the MNIST dataset of hand-written digits using neural networks. In part 1 of this project you will implement your own multi-layer perceptron network, write code to train the network using back-propagation, and train the network to perform classification. You will be graded on the accuracy of your classifier.
In part 2, you will use Pytorch to create and train a simple convolutional neural network to perform the same task.
As always, you will document your work and show your results in a pdf report. You have been given (just under) 3 weeks for this project, which is *worth 50% more* than Projects 1 and 2. It will likely take at least 50% more effort. Start early!

Note: Training your MLP over the full dataset could take a significant amount of time (twenty minutes on a high-end CPU and well over an hour on slower laptops). Be sure to give yourself ample time to complete this project. Also, be sure to read the full assignment before starting. Parts of the assignment (e.g., plotting intermediate accuracy) require modifying the code associated with earlier sections.

1 Restricted Functions and Installation

For Part 1, you may import numpy, matplotlib, and time. For Part 2, you may also import torch. Any and all functions in these packages are fair game. Please do not import additional packages.
The easiest way to install pytorch is using Anaconda: https://pytorch.org/. You don't need GPU support for this project.

2 Multi-layer Perceptrons

100 points total

Your first task is to create, train, and test a fully connected neural network (a.k.a. MLP). Your MLP will have 64 neurons in its hidden layer and 10 neurons in its output layer. Each neuron should have a bias variable. The hidden layers will use a sigmoid activation function. The output layer will apply a soft-max function across the 10 neurons. See Figure 1.

The included `template.py` file provides a MLP class definition that you will be completing.

2.1 Define the network

10 points

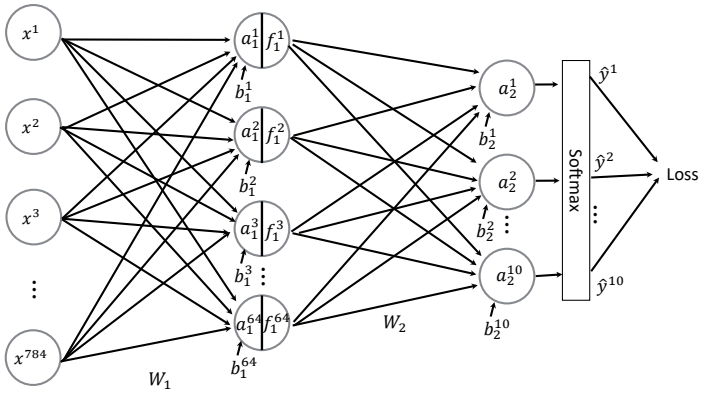

Figure 1: Network Architecture

Complete the `forward` definition in the code. It should take in an input image x and output an estimate $\hat{y}$, representing the probability the image belongs to each of the 10 classes. In matrix form, the network can be described by

$$\hat{y} = \text{softmax}(\mathbf{W}_2(\sigma(\mathbf{W}_1 x + b_1) + b_2).$$

You may find it very useful later on to store all intermediate results within the MLP object. E.g., `self.f1=sigmoid(self.a1)`. This will allow you to access these results during back-propagation without having to recompute them.

2.2 Initialize the network

5 points

Initialize the network weights and biases. The network weights should be initialized with random Gaussian variables with some standard deviation 0.1. The biases should be initialized with zeros.

2.3 Back-propagation

2.3.1 Derive the Jacobians

15 points

The Jacobian of a multivariate vector-valued function $f : \mathbb{R}^n \to \mathbb{R}^m$, is a matrix

$$\mathbf{J} = \begin{bmatrix} \frac{\partial f_1}{\partial x_1} & \cdots & \frac{\partial f_1}{\partial x_n} \\ \vdots & & \vdots \\ \frac{\partial f_m}{\partial x_1} & \cdots & \frac{\partial f_m}{\partial x_n} \end{bmatrix}.$$

With some slight abuse of notation, let's call this Jacobian $\frac{\partial f}{\partial x}$. Note, each row of the Jacobian is a transposed Gradient.

Derive the following Jacobian's matrices and include them in your report: $\frac{\partial a_2}{\partial b_2}$, $\frac{\partial a_2}{\partial \mathbf{W}_2}$, $\frac{\partial a_2}{\partial f_1}$, $\frac{\partial f_1}{\partial a_1}$, $\frac{\partial a_1}{\partial b_1}$, and $\frac{\partial a_1}{\partial \mathbf{W}_1}$.
The Jacobian $\frac{\partial L}{\partial a_2}$ is a 1×10 matrix with elements

$$\frac{\partial L}{\partial a_2} = \left[\hat{y}^1 - y^1, \quad \hat{y}^2 - y^2, \quad \ldots \quad \hat{y}^{10} - y^{10}\right],$$

This expression will help with the next section. (The term y^i denotes the i^{th} element of y, not y to the i^{th} power.)

2.3.2 Compute the gradients

15 points

Complete the `update_grad` definition in the code. Your code will first compute and back-propagate the loss. The back-propagated gradients will then be added to estimates of the gradients over the entire batch: `self.W2_grad`, `self.b2_grad`, `self.W1_grad`, and `self.b1_grad`. The expressions for the Jacobians you just computed will be useful here.

2.4 Training

15 points

Write code to train the network using stochastic gradient descent with a batch size of 256, a cross-entropy loss function, and a learning rate of 0.001.Your code should iteratively call `myNet.reset_grad()`, `myNet.forward(x)`, `myNet.update_grad()`, and `myNet.update_params(lr)`, in that order.

2.5 Over-fitting?

10 points

Train your network using the first 2000 images and labels from the training set. During or at the end of each training epoch, compute the accuracy over the entire training set and over the entire validation set. Plot the accuracy vs iteration for both the training and validation set. Your plot should looking something like Figure 2.
Repeat this process and make an analogues plot using all 50000 images (this could take a while!). Comment on the differences between the two plots.

2.6 How well does it work?

20 points + E.C.

Test your trained network on the testing dataset and compute the average accuracy. You will receive an extra credit points for every percentage point the network does better than 90% and will lose a point for every percentage point it does less than 90% (up ot 70%).
You can improve the network's performance with any number of methods discussed in class including drop-out, augmentation, regularization, adaptive learning rates, etc.

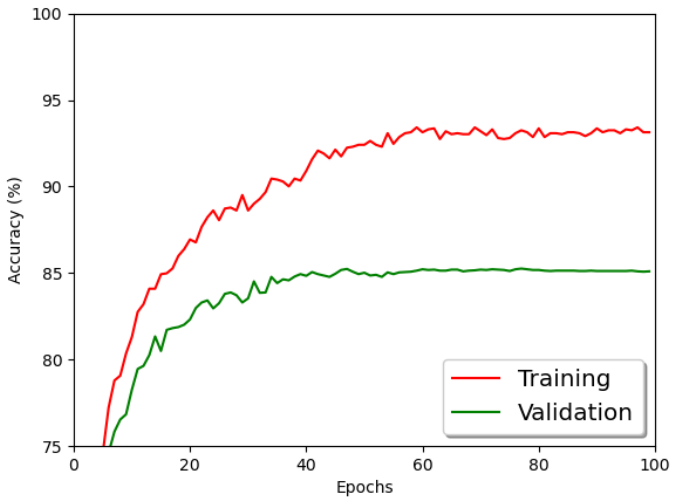

Figure 2: Example plot for Section 2.5

Important: Do-not train on the test set. Doing so will result in a loss of 20 points on this portion of the assignment.

2.7 Where does it make mistakes?

5 points

Compute the 10×10 confusion matrix. Which numbers are difficult to identify?

2.8 Visualize the weights

5 points

The first layer of the neural network is effectively computing inner products between the incoming image x and the rows of the weight matrix W_1. In effect, it is performing a basic form of template matching. Visualize and comment on these templates (rows of W_1).

3 Convolutional Neural Networks

50 points total

Read through the Pytorch classification tutorial on https://pytorch.org/tutorials/beginner/blitz/cifar10_tutorial.html. Much of the code to complete this portion of the assignment can be copied directly from the tutorial. Copying code from the tutorial is allowed, just make sure you specify which portions of code were copied.

3.1 Describe the network architecture

10 points

The template comes with code to define a simple convolutional neural network. Describe this network sufficiently that someone could re-implement it without looking at the code. E.g., how many channels per layer, stride size, padding, etc.?

3.2 Train CNN

10 points

Write code to train the provided convolutional neural network on the provided MNIST dataset. Again train with a batch size of 256, a learning-rate of 0.001, and a cross-entropy loss function. It may be easier to reuse code you wrote in the previous section to feed data, rather than using Pytorch's dataloader.

3.3 Over-fitting?

10 points

Train your network using the first 2000 images and labels from the training set. During or at the end of each training epoch, compute the accuracy over the entire training set and over the entire validation set. Plot the accuracy vs iteration for both the training and validation set. Your plot should looking something like Figure 2.
Repeat this process and make an analogues plot using all 50000 images. Comment on the differences between the two plots.

3.4 How well does it work?

20 points + E.C.

Test your trained network on the testing dataset and compute the average accuracy. You will receive an extra credit points for every percentage point the network does better than 98% and will lose a point for every percentage point it does less than 98% (up ot 78%).
You can improve the network's performance with any number of methods discussed in class including drop-out, augmentation, regularization, adaptive learning rates, etc.
Important: Do-not train on the test set. Doing so will result in a loss of 20 points on this portion of the assignment.

Submission Instructions

Your canvas submission should consist of a zip file named **YourDirectoryID_Project3.zip**, for example xyz123_Project3.zip. The file must contain the following:

- Project3.py, *not .ipynb*
- report.pdf

Do not include the datasets in your submission. Rather use relative pathing assuming the code and Project3_data directory are in the same parent directory.

Collaboration Policy

You are encouraged to discuss ideas with your peers. However, the code should be your own and should represent your understanding of the assignment. With the exception of the Pytorch tutorial mentioned in section 3, **code should not be shared or copied.** If you reference anyone else's code in writing your project, you must properly cite it in your code (in comments) and in your report.

Please list any individuals you collaborated with at the end of your report.

Plagiarism

Plagiarism of any form will not be tolerated. You are expected to credit all sources explicitly. If you have any doubts regarding what is and is not plagiarism, talk to me.

Credit

Thanks to Ashok Veeraraghavan, Ioannis Gkioulekas, and Mohammad Teli for sharing their course resources.