FML - CS 725 - 2019 Assignment 2

(TAs - Sahil, Kamlesh)

Go through these instructions carefully to avoid any confusions.
Deadline : October 11, 2019 11:59 PM

1. The assignment is to be done individually. Any kind of plagiarism will be dealt with strict action resulting in zero marks for this assignment and/or decrement in one grade for this course and/or report to Disciplinary action committee.
2. The assignment is divided into two parts Part A and Part B with marks [30 + 20 Marks].
3. This assignment is entirely programming-based and has 2 components. The first part **Part A** is implementing all class functions and loss, activation functions (along with their gradient functions). The base code is given to you and you are required to fill the unimplemented functions or part of the function. In the second part **Part B** of the assignment, you have to achieve the desired accuracy over datasets (XOR and MNIST datasets) to get marks.
4. This time we will not judge your score based on the Kaggle leaderboard and your rank. But you will give you a contest hosted on Kaggle for MNIST dataset where you can submit your prediction file to know your accuracy over test output file. Which we will use to calculate your accuracy score for Part B of assignment. Note: You also have to submit the same file (with your highest accuracy score) in Moodle submission also.
5. For instructions on how to access Kaggle.
    a. Go to the Kaggle site and make a new account (If not already).
    b. **Make sure your display name is your ROLL NUMBER (ie 193050000). This is very important as we'll check your submission using your roll number. No other username will be evaluated**
6. Link for the contest will be posted on Moodle shortly after the release of this Assignment.
7. The files related to this assignment are present in **rollno_assignment2.tar.gz** Download and extract this to get started. The directory is organized as follows

```
rollno_assignment2/
     |- main.py (specify the model architecture here)
     |- nn.py (implement a fully connected layer and cross-entropy
     loss here)
     |- autograder.py
     |- data
          |- mnist
               |- train.pkl
               |- test.pkl
          |- xor
               |- train.pkl
               |- test.pkl
     |- testcases
          |- testcase_01.pkl
```

Understand the flow of the program before starting code. You can run this code from main.py with the command: `python3 main.py`

8. You are free to add any new function but do not update the current function as they will be automatically graded with autograder.

9. **Submission Details**
    a. You have to submit the following files **main.py, nn.py, predictionsMnist.csv, predictionsXor.csv** (The prediction you get from your model for test data for corresponding datasets).
    b. Create a directory with the name <YourRollNumber>_assignment2 and put the above files in it. Use this command to compress the directory

    ```
    tar -czvf <YourRollNumber>_assignment2.tar.gz path/to/directory
    ```

    c. On running main.py your program will generate **predictionsMnist.csv** and **predictionsXor.csv** files of which, predictionsMnist.csv should be uploaded on Kaggle. Ensure that both these csvs are included in your submission and that the csv uploaded on Kaggle (your final selection on Kaggle) is the same as the one being submitted. Also, ensure that the code and hyperparameters that generated the predictions.csv file are the **SAME** as the code and hyperparameters in your final submission as we will run your code to verify that the csv produced by your code is the same as the one submitted. For sample look sampleSubmission.csv file on Kaggle.
    d. We will be using vimdiff to compare these two files (that generated by running your code, and the one submitted on Kaggle/Moodle) if it does not match exactly you will get a **zero** for Problem B (7 + 13 Marks) depending on which dataset the files did not match for.
    e. Make sure that you use the __main__ function in main.py to generate the csv file, since the seed has been fixed to ensure that the same output will be produced every time. It is hence preferred that you use the random module from numpy for any random numbers you need to generate. If you are using some other library make sure that you set the seed for the same since the output should match the file submitted.
    f. Any type of mistake in the naming conventions of folder and files will lead to 50% penalty marks i.e. we will deduct 25 marks for this assignment. This is necessary since the entire assignment submission will be automatically graded.

10. **Queries** - In case of any queries regarding the assignment either email the TAs at cs725_iitb@googlegroups.com or post on **Moodle** (recommended).
11. The entire assignment is in python. You are encouraged to use **numpy** for faster matrix operations.
12. You are not allowed to use libraries like Tensorflow, PyTorch, Scipy, etc which has predefined implemented machine learning codes.

13. Your program shouldn’t take more than 10 minutes to generate the predictions{Mnist,Xor}.csv files (both files combines). Hence, make sure your implementation is efficient and should not take unnecessary time. This means that you should avoid writing loops as far as possible and use vectorized implementations (ideally, none but one of the functions for the forwardpass and backwardpass should require a for loop). Typically, 30s and 3 minutes should be more than enough time for models to be trained on the XOR and MNIST datasets, respectively

In this assignment, you will perform **Classification** by implementing **Feed Forward Neural Network** with loss (like cross entropy loss) using the method of gradient descent. We will use only simple Fully Connected Layers with different activation functions stacked one after the other. Training must be done using simple Stochastic Gradient Descent using mini batches. Since only classification tasks are considered, cross-entropy loss is used after applying the softmax function to the logits obtained from the network.
For more concepts, refer to the class notes/slides.

LIST OF TASKS

Here we will enlist the tasks that you need to complete, along with a brief description of each. Please also look at the comments provided in each function that you need to implement for further specification of the input and output formats and shapes.

The general structure of the codebase is as follows. There is a class called called FullyConnectedLayer which represents one fully connected linear layer followed by a non-linear activation function which could be a ReLU or softmax layer. The NeuralNetwork class consists of a series of FullConnectedLayers stacked one after the other, with the output of the last layer representing a probability distribution over the classes for the given input. For this reason, the activation of the last function should always be the softmax function. Both these files are defined in nn.py.

In main.py, there are 2 tasks - taskXor and taskMnist - corresponding to the 2 datasets. In this, you need to define neural networks by adding fully connected layers. The code for the XOR dataset trains the model and prints the test accuracy at the end, while the code for the MNIST dataset trains the model and then uses the trained model to make predictions on the test set. Note that the answers to the test set have not been provided for the MNIST dataset.

Task 1

You need to implement the following functions in the FullyConnectedLayer class:

a. __init__: Initialise the parameters (weights and biases) as needed. This is not graded, but necessary for the rest of the assignment
b. relu_of_X: Return ReLU(X) where X is the input

c. softmax_of_X: Return softmax(X) where X is the input. The output of this layer now represents a probability distribution over all the output classes
d. forwardpass: Compute the forward pass of a linear layer making use of the above 2 functions. You can store information that you compute that will be needed in the backward pass in the variable self.data
e. gradient_relu_of_X: Assume that the output and input are represented by X and Y, respectively such that Y = ReLU(X). This function should take as input dLoss/dY and return dLoss/dX.
f. gradient_softmax_of_X: Like gradient_relu_of_X, this function takes as input dLoss/dY and should return dLoss/dX. You should try to work the gradient out on paper first and then try to implement it in the most efficient way possible. A "for" loop over the batch is an acceptable implementation for this subtask. [Hint: An output element y_j is not dependent on only x_j, so you may need to use the Jacobian Matrix here.]
g. backwardpass: Implement the backward pass here, using the above 2 functions. This function should only compute the gradients and store them in the appropriate member variables (which will be checked by the autograder), and not update the parameters. The function should also return the gradient with respect to its input (dLoss/dX), taking the gradient with respect to its output (dLoss/dY) as an input parameter.
h. updateWeights: This function uses the learning rate and the stored gradients to make actual updates.

Task 2

The NeuralNetwork class already has a defined __init__ function as well as a function to add layers to the network. You need to understand these functions and implement the following functions in the class:

a. crossEntropyLoss: Computes the cross entropy loss using the one-hot encoding of the groundtruth label and the output of the model.
b. crossEntropyDelta: Computes the gradient of the loss with respect to the model predictions P i.e. d[crossEntropy(P, Y)] / dP, where Y refers to the ground-truth labels.
c. train: This function should use the batch size, learning rate and number of epochs to implement the entire training loop. Make sure that the activation used in the last layer is the softmax function, so that the output of the model is a probability distribution over the classes. You can use the validation set to compute validation accuracy at different epochs (using the member functions of the NeuralNetwork class). Feel free to print different accuracies, losses and other statistics for debugging and hyperparameter tuning. It would, however, be preferable if you commented or deleted all the print commands in your final submission.

The train function will not be graded using the autograder. You will receive the marks for the datasets (Part B) only if you have satisfactorily implemented the train function.

There are also functions to make predictions and test accuracy that should not be modified.

Task 3

Finally, in main.py you need to define appropriate neural networks which give the best accuracy on both datasets. You also need to specify all the other hyperparameters like the batch size, learning rate and the number of epochs to train for. The relevant functions are:

a. taskXor
b. taskMnist
c. preprocessMnist: Perform any preprocessing you wish to do on the data here. [Hint: Some minimal basic preprocessing is needed to train with stability]

Do not modify the code in the rest of the function since it will be used to generate your final predictions.

You are encouraged to make plots of the validation / training loss versus the number of epochs for different hyperparameters and note down any interesting or unusual observations. You can submit the same in a folder called “extra” in the main directory.

Tip: In case you are getting NaN as the loss value, ensure that if you are dividing by a variable that might be 0, add a small constant to it, i.e., 1/x -> 1/(x + 1e-8)

Marks Distribution (Part A: 30 Marks)

Students will be awarded marks based on the test cases that they pass from the autograder which we will use (Hidden Test Cases) to evaluate your code. However, you are also given one small test case (Visible Test Case) which you can evaluate on your own to get an understanding of how it will work, rest all test cases will not be shared with you. You can use the following command to evaluate your code on the testcase_01 file.

```
python3 autograder.py
```

Note: your autograder should be placed along with main.py. It will check your code for just one test case, we will be using different and more complex test cases to evaluate your submission later.

Autograder will evaluate following function of your program

1. forwardpass	- 4 Marks
2. forwardpass + backwardpass	- 6 Marks
3. updateWeights	- 2 Marks
4. relu_of_X	- 1 Marks
5. gradient_relu_of_X	- 1 Marks
6. softmax_of_X	- 3 Marks
7. gradient_softmax_of_X	- 7 Marks
8. crossEntropyLoss	- 3 Marks

9. crossEntropyDelta	- 3 Marks
Total Marks from autograder	- 30 Marks

Rest 20 Marks will be based on your accuracy on different datasets which is mentioned in the next section (Datasets).

Datasets (Part B: 20 Marks)

For this assignment, you will be using two different datasets :

● XOR dataset (Toy dataset)	- 7 Marks
● MNIST dataset	- 13 Marks

The marks for each dataset will be given to you based on the accuracy you achieve on test data, which will be different for both datasets.

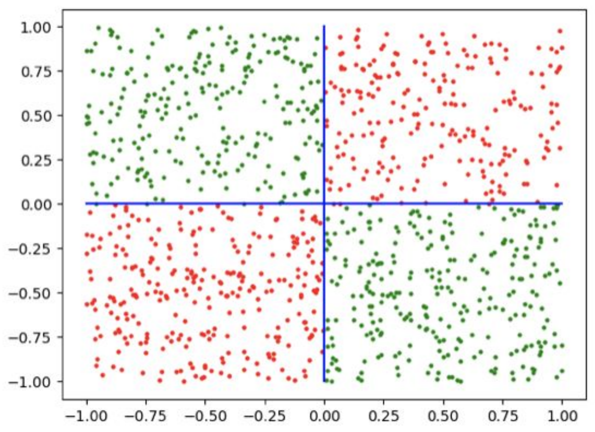
The formula we will use to award marks to students will be as follows:

$$Obtained\ Marks = \frac{(your\ Acc\ on\ Dataset * Max\ Marks\ for\ that\ dataset\ )}{100}$$

An accuracy over 97.5% and 95% on XOR and MNIST respectively is satisfactory, and easily achievable in the given time constraints. If the time constraints are violated, no marks will be awarded for the question, so make sure you set hyperparameters such that training completes well within the specified time limit. We will be using the SL2 machines to evaluate submissions.

XOR Dataset

The input X is a list of 2-dimensional vectors. Every example X_i is represented by a 2-dimensional vector [x,y]. The output y_i corresponding to the ith example is either a 0 or 1. The labels follow XOR-like distribution. That is, the first and third quadrant have the same label (y_i = 1) and the second and fourth quadrants have the same label (y_i = 0) as shown in the figure below:



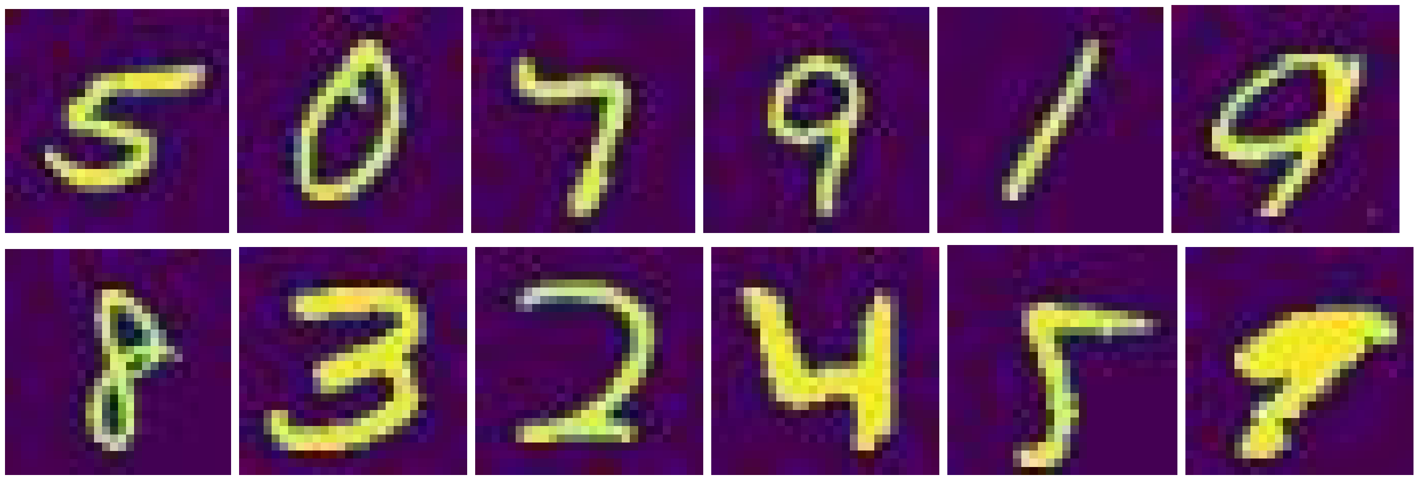
There are a total of 10000 points, and the training, validation and test splits contains 7000, 2000 and 1000 points respectively. As discussed in class, the decision boundaries can be learnt exactly for this dataset using a Neural Network. Hence, if your implementation is correct, the accuracy on the train, validation and test sets should be close to 100%

MNIST Dataset

We use the MNIST data set which contains a collection of handwritten numerical digits (0-9) as 28x28-sized binary images. Therefore, input X is represented as a vector of size 784 and the number of output classes is 10 (1 for each digit). In this case, the features are the grayscale image values at each of the pixels in the image. These images have been size-normalised and centred in a fixed-size image. MNIST provides a total 70,000 examples, divided into a test set of 10,000 images and a training set of 60,000 images. In this assignment, we will carve out a validation set of 10,000 images from the MNIST training set, and use the remaining 50,000 examples for training.

Simple feedforward neural networks (consisting of fully connected layers separated by non-linear activation functions) can be used to achieve a fairly high accuracy (even over 97%), but achieving this accuracy might require some careful tuning of the hyperparameters like the number of layers, number of hidden nodes and the learning rate.

Here are a few examples from the dataset:



Have Fun !