ECSE 4850/6850

Introduction to Deep Learning

**Program Assignment 2**

Due date: March 5, 11:59pm

In this programming assignment, you will implement techniques to learn a deep (two hidden layers) neural network (NN) for image digit classification on the MNIST dataset (https://en.wikipedia.org/wiki/MNIST\_database). The NN will take an image of a hand-written numerical digit as input and classify it into one of 10 classes corresponding to digits 0 to 9 respectively. Given the training data, you will follow the equations in the lecture notes to train the NN using the back propagation method and then evaluate its performance on the given testing data. The NN has one input layer (784 nodes), two hidden layers with 100 nodes for each layer, and one output layer with 10 nodes, corresponding to the 10 digit classes respectively. For training, you will learn the parameters ******W1**, **W10, W2**, **W20, W3**, **W30**] by minimizing the cross-entropy of the output nodes, where **W1** and **W2** are the weight matrices for the hidden layers, **W3**is the weight matrix for the output layer, and **W10 ,W20, W30** are the corresponding bias vectors.

Specifically, given the training data **D**={**x**[m], ***y***[m]}, m=1,2, …, M, where **x**[m] is a grayscale image of 28x28 (a 784x1 vector) and ***y***[m] is ground truth output vector that follows the 1 of K (one hot) encoding, with kth element of being 1 and the rest being 0. Let **[**m] be the NN output corresponding to **x**[m]. Use ReLU activation function for the hidden nodes and the softmax function for the output **[** [m]. You will implement the following methods to learn the parameters ****

1. Implement within **PyTorch** the back propagation method to solve for **** iteratively using all data in **D**. Each time, choose a mini-batch of **50** data from **D**. For each point in the mini-batch, perform forward propagation and back propagation to compute the gradients of weight matrix and weight bias vector for each layer. Then update the weights and bias using the average gradient of the gradients computed from all the **50** samples. Initialize weight matrix****to small different values. Iteratively update each **W**with an appropriate learning rate until convergence. Note students taking this class at 6000 level are also required to implemented the L1 norm on weight matrices and bias vectors. Save **** using pickle.dump functions (see below for details).

2. Using the given testing dataset **T**, evaluate the performance of your trained NN by computing the classification error for each digit as well as the average classification errors for all digits. The classification error for each digit is computed as the ratio of incorrect classification to the total number images for that digit. Plot the average training classification error, average testing classification error, and value of the loss function after each parameters update.

3. It is ok to use NumPy within **PyTorch** but do not use **PyTorch** back propagation functions. You however may use it to verify your implementation.

4. Submit the following

1. a report that summarizes the theories for multi-layer neural network. Discuss the model architecture (number of hidden layers and nodes for each layer), the loss function, the regularization, experimental settings, and hyper-parameters. Plot the training and testing losses over epochs, training and testing accuracy over epochs, and the classification error for each digit over epochs.
2. The implementation (code).
3. The saved weights **W** in the required format (see below).

**Data format**

The training data **D** contains 50000 training images (00001.jpg – 50000.jpg) in train\_data folder, and their labels are stored in train\_label.txt with each raw being the label of the corresponding image.

The testing data **T** contains 4000 testing images (00001.jpg – 04000.jpg) in test\_data folder, and their labels are stored in test\_label.txt.

Another 1000 testing images have been held as validation data. The accuracy of the learned parameters will be tested on those validation data.

**Preprocessing Images**

Student should follow the same preprocessing steps as in Programming Assignment 1. The prepocess.py could still be used, but you may need to create the train\_porcessed and test\_processed folder first.

**Output**

Use the code below to save the learned parameters ****in the following format

****= [**W1**, **W1,0, W2**, **W2,0, W3**, **W3,0**]

**** is a list objects. **W1** and **W2** are the matrix of the weights for the hidden layers, **W3** is the matrix of the weights for the output layer, and **W1,0, W2,0, W3,0** are the corresponding bias. Please note:

1. ****is a Python list.
2. **W1** , **W2 , W3** are 2-d Numpy array.
3. **W1,0, W2,0, W3,0** are 1-d Numpy array.

--------------------

import pickle

filehandler = open("nn\_parameters.txt","wb")

pickle.dump(Theta, filehandler, protocol=2)

filehandler.close()

Convert a tensor variable to a numpy variable: A = A.numpy()

**Batch computing**

To speed up the training, you can simultaneously compute the gradients of 50 samples at the same time using 3-D tensor. For example, given two tensors AN\*p\*q and B N\*q\*m, torch.matmul(A,B) will compute matrix multiplication N time and return a tensor with shape N\*p\*m. More details and example can be found at:

<https://pytorch.org/docs/stable/generated/torch.matmul.html>

**Other requirements:**

* 1. To grade assignments fairly with our grading policy, we only accept codes written in python3 (Please take this rule as the standard!).
  2. Since **PyTorch** has been announced at the beginning of this class, our assignments will be based on **PyTorch**. Please do not refer to previous assignments on github or other websites! A large penalty will be given if you try to copy the codes from those projects.
  3. You are not allowed to use any built-in functions such as **Dataloader**, **optimizer**, or **loss.backward()** in **PyTorch**. You may use them for verification of your own implementation.
  4. Academical plagiarism will lead to a strict penalty! We will carefully check your codes to identify plagiarism. Writing your own code is more important than having a good model. We set a relatively small percentage of total score for model performance according to our grading policy.
  5. The students are required to submit a report and the code. No handwritten report will be accepted. All reports must be written with Microsoft Word, Latex or Markdown language. Submitted report must be in PDF format.
  6. The template code has a line which tells us which ‘device’ (GPU or CPU) to use for the calculation. The ‘device’ will be ‘cuda’ (if GPU is enabled on the machine) or it will be ‘cpu’ (CPU).
  7. We encourage you to leave possible comments in your codes for the grading TA to understand your codes clearly. Unclear coding may lead to misunderstanding.
  8. This assignment will be time consuming if it is your first time to train a model, start early! Late submission will not be accepted unless the student acquires permission from the professor in advance (before the deadline). The report and implementation should be submitted to LMS. TA does not accept submission via email.

**Grading Policy:**

The programming assignment #2: Total **100** points.

1. **Model performance** (**30 points):** Grading TA will use the saved parameter to check model performance on validation dataset. A correct model, which achieves a testing accuracy above 97%, gets 30 points. A good model, which achieves a testing accuracy above 93%, gets 20 points.
2. **Report (50 points)**:
   1. **Problem statement (5 points)** - A brief introduction to the problem you are going to solve. Please summarize the task in your own words and do not copy the sentences in the assignment.
   2. **Model description (15 points)** - Clearly describe your model including the structure, equations for performing forward-propagation and back-propagation, equations of the gradients of loss with respect to parameters. You should include the derivations of the equations. Symbols should be clearly defined.
   3. **Experimental setup (5 points) –** Describe how you set up your experiments. You can talk about how the training and testing split is performed. Also, you can report your batch size, how you initialize the weight matrix, how you change the learning rate or any other hyper-parameter you used.
   4. **Experimental results (10 points)** - Plot the training losses versus epochs. Also, plot the training and testing accuracy of each digit versus epochs. Report the final average classification error on testing dataset and visualize the confusion matrix based on the testing performance.
   5. **Analysis (10 point) –** Based onyour setup and results, analyze your findings. High quality analysis will get full marks. Be creative here!
   6. **Conclusion (5 point)** – Describe briefly what you did in this report and what can be a good future direction based on the learning from this programming assignment.
3. **Code (20 points):** Grading TA should be able to run the implementation without any bugs or software related errors. In addition, the code should be properly documented with proper comments and should follow proper convention of coding.
4. Well documented source code that follows the requirements **(15 points)**
5. Clearly identify external or other functions used **(5 points)**
6. Copy of a significant portion of others code without acknowledgment will be counted as plagiarism and will lead to 0 for the project for the first occurrence and an F grade for this course and report to RPI if it occurs again.

**Tips on programming:**

TA’s office hour is limited, and TA is not responsible for debugging in principle. Hence, a few tips are given. Please check those tips before you turn to TA for help.

1. **Good python IDEs may enhance your efficiency.** Powerful python IDEs such as PyCharm, Spyder, Xcode, and Visual Studio contain debug tools and can help you to automatically correct syntax errors.
2. **The solutions to beginners’ issues can be found online.** PyTorch is a commonly- used deep learning platform, the solutions to the issues that the students may encounter can be found on StackOverflow, PyTorch community, and Github.
3. **Choose the proper learning rate:** It is very important to choose a proper learning rate during deep learning model training. A large learning rate may block the convergence of the model. It may take a long time for the model to converge with small learning.
4. **Prevent overfitting.** The students should carefully monitor the loss and testing accuracy to prevent the model from overfitting.