CS7643: Deep Learning
Assignment 2

Instructor: Zsolt Kira

Deadline: February 28, 2021, 11:59 PM EST

- This assignment is due on the date/time posted on canvas. We will have a 48-hour grace period for this assignment. However, no questions regarding the assignment are answered during the grace period in any form.
- Discussion is encouraged, but each student must write his/her own answers and explicitly mention any collaborators.
- Each student is expected to respect and follow the GT Honor Code. **We will apply anti-cheating software to check for plagiarism**. Anyone who is flagged by the software will automatically receive 0 for the homework and be reported to OSI.
- Please **do not change the filenames and function definitions** in the skeleton code provided, as this will cause the test scripts to fail and you will receive no points in those failed tests. You may also **NOT** change the import modules in each file or import additional modules.
- It is your responsibility to make sure that all code and other deliverables are in the correct format and that your submission compiles and runs. We will not manually check your code (this is not feasible given the class size). Thus, **non-runnable code in our test environment will directly lead to a score of 0**. Also, your entire programming parts will **NOT** be graded and given 0 score if your code prints out anything that is not asked in each question.

Overview

Convolutional Neural Networks (CNNs) are one of the major advancements in computer vision over the past decade. In this assignment, you will complete a simple CNN architecture from scratch and learn how to implement CNNs with PyTorch, one of the most commonly used deep learning framework. You will also run different experiments on imbalanced datasets to evaluate your model and techniques to deal with imbalanced data.

Python and dependencies

In this assignment, we will work with **Python 3**. If you do not have a python distribution installed yet, we recommend installing Anaconda (or miniconda) with Python 3. We provide `environment.yaml` (present under *./part1-convnet*) which contains a list of libraries needed to set the environment for this assignment. You can use it to create a copy of conda environment. Refer to the users' manual for more details.

```
$ conda env create -f environment.yaml
```

Please note that our environment does **NOT have PyTorch Installation** for you because you may use cpu/gpu or a different version of CUDA. To install PyTorch, please refer to the official documentation and select the options based on your local OS.

If you already have your own Python development environment, please refer to this file to find necessary libraries, which is used to set the same coding/grading environment.

Code Test

There are two ways (steps) that you can test your implementation:

1. Python Unit Tests: Some public unit tests are provided in the `tests/` in the assignment repository. You can test each part of your implementation with these test cases by:

   ```
   $ python -m unittest tests.<name_of_tests>
   ```

   However, passing all local tests neither means your code is free of bugs nor guarantees that you will receive full credits for the coding section. Your code will be graded by GradeScope Autograder(see below for more details). There will be additional tests on GradeScope which does not present in your local unit tests.

2. Gradescope Autograder: You may also submit your code as specified in Section 6 for testing. Gradescope would only reveal the results of public tests. Your final grades of the coding section are based on both public and hidden private tests. However, we do not recommend using Gradescope as your primary testing method during development because the private test cases will **NOT** be available to you at any time.

1 Implementing CNN from Scratch

You will work in *./part1-convnet* for this part of assignment.

1.1 Module Implementation

You will now learn how to build CNN from scratch. Typically, a convolutional neural network is composed by several different modules and these modules work together to make the network great. For each module, you will implement a forward pass (computing forwarding results) and a backward pass (computing gradients). Therefore, your tasks are as follows:

(a) Follow the instructions in the code to complete each module in *./modules*. Specifically, modules to be implemented are: 2D convolution, 2D Max Pooling, ReLU, and Linear. These will be the building blocks of the full network. The file *./modules/conv_classifier.py* ties each of the aforementioned modules together, and is the subject of the next section.

1.2 Network Implementation

After finishing each module, it's now to put things together to form a real convolutional neural network. Your task is:

(a) Follow the instructions in the code to complete a CNN network in *./modules/conv_classifier.py*. The network is constructed by a list of module definitions **in order** and should handle both forward and backward communication between modules.

1.3 Optimizer

You have implemented a simple SGD optimizer in assignment-1. In practice, it is common to use a momentum term in SGD for better convergence.

Specifically, we introduce a new velocity term v_t and the update rule is as follows:

$$\begin{aligned} v_t &= \beta v_{t-1} - \eta \frac{\partial L}{\partial w} \\ w &= w + v_t \end{aligned}$$

where β denotes the momentum coefficient and η denotes the learning rate

(a) Follow the instructions in the code to complete SGD with momentum in *./optimizer/sgd.py*.

You might have noticed that the training process of your implementation can be extremely slow. Therefore, we only want to deliberately overfit the model with a small portion of data to verify whether the model is learning something or not. First, you should download the dataset by

```
$ cd data
$ sh get_data.sh
$ cd ../
```

Microsoft Windows 10 Only

```
C:\assignmentfolder> cd data
C:\assignmentfolder\data> get_data.bat
C:\assignmentfolder\data> cd ..
```

You can then simply run:

```
$ python train.py
```

which trains a small CNN with only 50 samples in CIFAR-10 dataset. The script will make a plot on the training data only and **be sure to include the plot in your report**. Your final accuracy should be above 0.9 with the given network in the script.

2 PyTorch

You will work in *./part2-pytorch* for this part of assignment. The main function in *main.py* contains the major logic of the code and you can run it by

```
$ python main.py --config configs/<name_of_config_file>.yaml
```

2.1 Training

The first thing of working with PyTorch is to get yourself familiarized with the basic training step of PyTorch.

1. Complete *train* and *validate* functions in *main.py*.

2.2 PyTorch Model

You will now implement some actual networks with PyTorch. We provide some starter files for you in *./models*. The models for you to implement are as follows:

1. Two-Layer Network. This is the same network you have implemented from scratch in assignment 1. You will build the model with two fully-connected layers and a sigmoid activation function in between of the two layers. Please implement the model as instructed in *./models/twolayer.py*

2. Vanilla Convolutional Neural Network. You will build the model with a convolution layer, a ReLU activation, a max-pooling layer, followed by a fully-connected layer for classification. Your convolution layer should use **32 output channels**, a **kernel size of 7** with **stride 1** and **zero padding**. You max-pooling should use a **kernel size of 2** and **stride of 2**. Fully connected layer should have **10 output features**. Please implement the model as instructed in *./models/cnn.py*

3. Your Own Network. You are now free to build any model of your choice. Please implement your model in *./models/my_model.py*

We provide you configuration files for these three models respectively. For Two-Layer Network and Vanilla CNN, you need to train the model without modifying the configuration file. The script automatically saves the weights of the best model at the end of training and we will evaluate your implementation by loading your model weights and evaluate the model on CIFAR-10 test data. You should expect the accuracy of Two-Layer Network and Vanilla CNN to be around 0.3 and 0.4 respectively.

For your own network, you are free to tune any hyper-parameters to obtain better accuracy. Your final accuracy must be above 0.5 to receive at least partial credits.

All in all, please make sure the checkpoints of each model are saved into *./checkpoints*.

3 Data Wrangling

So far we have worked with well-balanced datasets (samples of each class are evenly distributed). However, in practice, datasets are often not balanced. In this section, you will explore the limitation of standard training strategy on this type of dataset.

You will work with an inbalanced version of CIFAR-10 in this section.

3.1 Class-Balanced Focal Loss

You will implement one possible solution to the imbalance problem: Class-Balanced Focal Loss. It is described in Section 4 of this CVPR-19 paper: Class-Balanced Loss Based on Effective Number of Samples. You may also refer to the original paper of Focal Loss for more details if you are interested. Please implement CB Focal Loss in *./losses/focal_loss.py*

4 Deliverables

4.1 Code Submission

4.1.1 Part-1 ConvNet

Simply run `bash collect_submission.sh` or `collect_submission.bat` if running **Microsoft Windows 10** under *part1-convnet* and upload the zip file to Gradescope (part1-code).

4.1.2 Part-2 PyTorch

Simply run `bash collect_submission.sh` or `collect_submission.bat` if running **Microsoft Windows 10** under *part2-pytorch* and upload the zip file to Gradescope (part2-code).

4.2 Write-up

Please follow the report template in the starter folder to complete your write-up. You will need to explain your design of network of Section 2.2, submit your experimental results of data wrangling. Output your report to pdf format and submit it to Gradescope.