CS 4644/7643: Deep Learning
Assignment 2

Instructor: Zsolt Kira
TAs: Krishanu Agarwal, Manav Agrawal, Aditya Akula, Will Held, Vikranth Keerthipati, Pranay Mathur, Avinash Prabhu, Katie Stevo, Wei Zhou, Bowen Zuo
Discussions: https://piazza.com/gatech/spring2024/cs4644acs7643a

Deadline: 11:59 pm February 19, 2024

- 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: https://osi.gatech.edu/students/honor-code. **We will apply anti-cheating software to check for plagiarism**. We cross-check source code within the class, with previous years' solutions, and with online solutions. Any case that deemed substantial by the teaching team will be reported to OSI and receive a 0.
- 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 a 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 frameworks. You will also run different experiments on imbalanced datasets to evaluate your model and techniques to deal with imbalanced data.

Assignment Setup

This assignment will be hosted **entirely on Google Colab** to avoid any environment specific issues. Upload the files to Google Drive and then follow the Google Cloud setup directions in the *main.ipynb* notebook.

Code Test

There are two ways (steps) to 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 running the third block in the colab file and replacing test_network with your test of choice. 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 are not present in your local unit tests.

2. Gradescope Autograder: You may also submit your code to Gradescope. The auto-grader will return the results of public tests, but not the private tests; both of which are used to calculate grades for the coding sections. Note that we do not recommend using Gradescope Autograder as your primary testing method during development because the utility may **NOT** be available at all times.

1 Implementing CNN from Scratch [7 points]

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

1.1 Module Implementation

You will now learn how to build a CNN from scratch. Typically, a convolutional neural network is composed of several different modules and these modules work together to make the network effective. 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. After completing each of the 4 modules, run these unit tests:

```
$ python -m unittest tests.test_conv
$ python -m unittest tests.test_maxpool
$ python -m unittest tests.test_relu
$ python -m unittest tests.test_linear
```

1.2 Network Implementation

After finishing each module, it's time 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:

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

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 running these commands in a cell block

```
$ cd data
$ sh get_data.sh
$ 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 slightly under 0.9 with the given network in the script.

2 PyTorch [5 points + 2 Extra Credits]

You will work in *./part2-pytorch* for this part of the assignment. The main function in *main.py* contains the major logic of the code and can be run 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 the two layers. Please implement the model as instructed in *./models/twolayer.py*

    ```
    $ python -m unittest tests.test_twolayer
    ```

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**. The fully connected layer should have **10 output features**. Please implement the model as instructed in *./models/cnn.py*

    ```
    $ python -m unittest tests.test_vanilla_cnn
    ```

3. Your Own Network. You are now free to build your own model. Notice that it's okay for you to borrow some insights from existing well-known networks, however, directly using those networks as-is is **NOT** allowed. In other words, you have to build your model from scratch, which also means using any sort of pre-trained weights is also **NOT** allowed. 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. We will evaluate your implementation by loading your model weights and evaluating 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 credit**. Please refer to the GradeScope auto-test results for the requirement of full credits. All in all, please make sure the checkpoints of each model are saved into *./checkpoints*.

3 Deliverables

3.1 Coding

To submit your code to Gradescope, you will need to submit a zip file containing all your codes in structure. For your convenience, we have provided a cell in the Colab notebook that will zip the required files. Simply follow the instructions until the last cell to generate the zip file.

3.2 Writeup [6 points]

You will also need to submit a report summarizing your experimental results and findings as specified in the previous sections. We have provided a starting template for you in the Colab notebook under the assignment writeup part. Just follow the instructions to complete the writeup.

Before exporting the notebook, make sure to remove all outputs of cells except for plots so that the resulting pdf is of a reasonable length. You will need to export your notebook in **pdf** format and submit to Gradescope. The best way to do this is:

- Download the colab notebook as a .ipynb file locally
- Open it in jupyter notebook and click download as pdf

Remember to submit your coding report deliverables to Gradescope's **HW2 - Report** assignment. **When submitting to Gradescope, make sure you select ALL corresponding pages for each question. Failing to do so will result in -1 point for each incorrectly tagged question, with future assignments possibly having a more severe penalty.**