CS 7643/4644: Deep Learning
Homework 1 Coding

Instructor: Zsolt Kira
TAs: Mili Das, Yipu Chen, Kausar Patherya
Discussions: https://piazza.com/class/mafsg3dobtu42c

Deadline: 11:59pm June 5th, 2025

- 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/content/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 0 score if your code prints out anything that is not asked in each question.

Overview

Deep Neural Networks are becoming more and more popular and widely applied to many ML-related domains. In this assignment, you will complete a simple pipeline of training neural networks to recognize MNIST Handwritten Digits: http://yann.lecun.com/exdb/mnist/. You'll implement two neural network architectures along with the code to load data, train and optimize these networks. You will also run different experiments on your model to complete a short report. Be sure to use the template of report we give to you and fill in your information on the first page.

The `main.ipynb` contains the major logic of this assignment. You will use this notebook to download the data, train your models and plot your curves.

There are three pre-defined config files under ./`configs`. Two of them are default hyper-parameters for models that you will implement in the assignment (Softmax Regression and 2-layer MLP). The correctness of your implementation is partially judged by the model performance on these default hyper-parameters; therefore, do **NOT** modify values in these config files. The third config file, `config_exp.yaml`, is used for your hyper-parameter tuning experiments (details in Section 5) and you are free to modify values of the hyper-parameters in this file.

The script trains a model with the number of epochs specified in the config file. At the end of each epoch, the script evaluates the model on the validation set. After the training completes, the script finally evaluates the best model on the test data.

Assignment Setup

This assignment will be hosted entirely on Google Colab to avoid any environment specific issues. Follow the below steps before editing any code files.

1. Open main.ipynb in Google Colab
2. Run cells 1-2 to allow Colab to access your Google Drive and to download the necessary data

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 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 as specified in Section 6 for testing. 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 Data Loading [2 points]

1.1 Data Preparation

To avoid overfitting the training data, it is a common practice to split the training dataset into the actual training data and validation data and perform hyper-parameter tuning based on results on validation data. Additionally, in deep learning, training data is often forwarded to models in **batches** for faster training time and noise reduction.

In our pipeline, we first load the entire MNIST data into the system, followed by a training/validation split on the training set. We simply use **the first 80% of the training set as our training data and use the rest training set as our validation data**. We also want to organize our data (training, validation, and test) in batches and use different combination of batches in different epochs for training data. Therefore, your tasks are as follows:

(a) follow the instruction in code to complete `load_mnist_trainval` in ./`utils.py` for training/validation split

(b) follow the instruction in code to complete `generate_batched_data` in ./`utils.py` to organize data in batches

You can test your data loading code by running the following command in cell 3:

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

2 Model Implementation [8 points]

You will now implement two networks from scratch: a simple softmax regression and a two-layer multi-layer perceptron (MLP). Definitions of these classes can be found in `./models`.

Weights of each model will be randomly initialized upon construction and stored in a weight dictionary. Meanwhile, a corresponding gradient dictionary is also created and initialized to zeros. Each model only has one public method called `forward`, which takes input of batched data and corresponding labels and returns the loss and accuracy of the batch. Meanwhile, it computes gradients of all weights of the model (even though the method is called forward!) based on the training batch.

2.1 Utility Function [4 points]

There are a few useful methods defined in `./models/_base_network.py` that can be shared by both models. Your first task is to implement them based on instructions in `_base_network.py`:

(a) **Activation Functions**. There are two activation functions needed for this assignment: `ReLU` and `Sigmoid`. Implement both functions as well as their derivatives in `./models/_base_network.py` (i.e, `sigmoid`, `sigmoid_dev`, `ReLU`, and `ReLU_dev`). Test your methods by running the following command in cell 3:

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

(b) **Loss Functions**. The loss function used in this assignment is Cross Entropy Loss. You will need to implement both Softmax function and the computation of Cross Entropy Loss in `./models/_base_network.py`. **HINT**: You may want to checkout the numerically stable version of softmax: https://blester125.com/blog/softmax.html. Run this command in cell 3 (this test also tests the accuracy method):

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

(c) **Accuracy**. We are also interested in knowing how our model is doing on a given batch of data. Therefore, you may want to implement the `compute_accuracy` method in `./models/_base_network.py` to compute the accuracy of given batch.

2.2 Model Implementation [4 points]

You will implement the training processes of a simple Softmax Regression and a two-layer MLP in this section. The Softmax Regression is composed by a fully-connected layer followed by a ReLU activation. The two-layer MLP is composed by two fully-connected layers with a Sigmoid Activation in between. Note that the Sofmax Regression model has no bias terms, while the two-layer MLP model does use biases. Also, don't forget the softmax function before computing your loss!

(a) Implement the forward method in `softmax_regression.py` as well as `two_layer_nn.py`. If the mode argument is train, compute gradients of weights and store the gradients in the gradient dictionary. Otherwise, simply return the loss and accuracy. Run the following command in cell 3:

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

3 Optimizer [4 points]

We will use an optimizer to update weights of models. An optimizer is initialized with a specific learning rate and a regularization coefficients. Before updating model weights, the optimizer applies L2 regularization on the model:

$$J = L_{CE} + \frac{1}{2}\lambda\sum_{i=1}^{N} w_i^2 \tag{1}$$

where J is the overall loss and L_{CE} is the Cross-Entropy loss computed between predictions and labels.

You will also implement a vanilla SGD optimizer.The update rule is as follows:

$$\theta^{t+1} = \theta^t - \eta\nabla_\theta J(\theta) \tag{2}$$

where θ is the model parameter, η stands for learning rate and the ∇ term corresponds to the gradient of the parameter.

In summary, your tasks are as follows:

(a) Follow instructions in the code and implement `apply_regularization` in `_base_optimizer.py` Remember, you may **NOT** want to apply regularization on **bias** terms!

(b) Implement the `update` method in `sgd.py` based on the discussion of update rule above.

Test your optimizer by running the following command in cell 3:

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

4 Visualization [2 points]

It is always a good practice to monitor the training process by monitoring the learning curves. Our training code in `main.ipynb` stores averaged loss and accuracy of the model on both training and validation data at the end of each epoch. Your task is to plot the learning curves by leveraging these values. A sample plot of learning curves can be found in Figure 1. After you implement the method run cells 4-9 to train and plot the curves for the softmax and two layer neural networks.

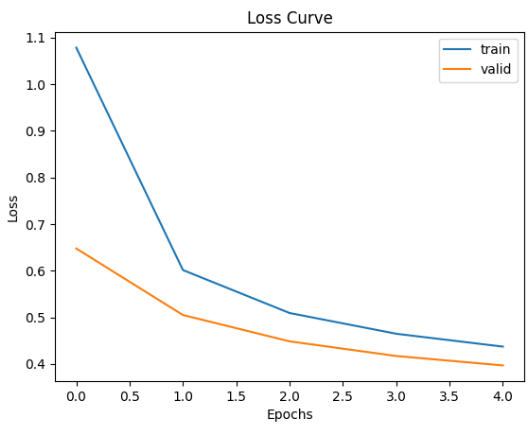

Figure 1: Example plot of learning curves

(a) Implement `plot_curves` in `./utils.py`. You'll get full marks on this question as long as your plot makes sense.

5 Experiments [6 points]

Now, you have completed the entire training process. It's time to play with your model a little. You will use your implementation of the two-layer MLP for this section. There are different combinations of your hyper-parameters specified in the report template and your tasks are to tune those parameters and report your observations by answering questions in the report template. We provide a default config file `config_exp.yaml` in `./configs`. When tuning a specific hyper-parameter (e.g, the learning rate), please leave all other hyper-parameters as-is in the default config file.

(a) You will try out different values of learning rates and report your observations in the report file. Use cells 10-17 to experiment with learning rates. [**2 points**]

(b) You will try out different values of regularization coefficients and report your observations in the report file. Use cells 18-27 to experiment with regularization coefficients. [**2 points**]

(c) You will try your best to tune the hyper-parameters for best accuracy. Use cells 28-29 for hyperparameter tuning. [**2 points**]

6 Deliverables

6.1 Coding [14 points]

Submit this part to the **HW1 Coding** assignment under Gradescope.

To submit your code to Gradescope, you will need to submit a zip file containing all your codes in structure. For your convenience, we provide a notebook collect_submission.ipynb that will zip up all your files.

6.2 Writeup [8 points]

Submit this part to the **HW1 Writeup** assignment under Gradescope.

You will also need to submit your learning curves plots as specified in Section 4 as well as a report summarizing your experimental results and findings as specified in Section 5. For the experimental results report, We provide a starting template for you in the Google Colab notebook under the assignment writeup part. Just follow the instructions to generate your writeup.

Note: Explanations should go into *why* things work the way they do with proper deep learning theory. For example, with hyperparameter tuning you should explain the reasoning behind your choices and what behavior you expected. If you need more than one cell for a question, you are free to create new cells **right after the given one**.

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. For section 5, don't worry about listing the accuracies of the different experiments as long as the plots are shown, we will grade based on those. You will need to export your notebook in **pdf** format and submit to Gradescope. The best way to do this is to use the collect_submission.ipynb notebook that we provide. You should submit your notebook to the "HW1 Code PDF" assignment in Gradescope. **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 having a more severe penalty.**