Homework 6 Written Assignment

10-405/10-605: Machine Learning with Large Datasets

Due Thursday, April 21st at 3:00 PM Eastern Time

Instructions: Submit your solutions via Gradescope, *with your solution to each subproblem on a separate page*, i.e., following the template below. Note that Homework 6 consists of only programming portions, although you are required to answer some questions in the text below. Please check out the Homework 6 release post on Piazza for details on how to submit each file.

Submitting via Gradescope: When submitting on Gradescope, you must assign pages to each question correctly (it prompts you to do this after submitting your work). This significantly streamlines the grading process for the course staff. Failure to do this may result in a score of 0 for any questions that you didn't correctly assign pages to. It is also your responsibility to make sure that your scan/submission is legible so that we can grade it.

1 Model Compression Competition (60 pts)

In this portion of the homework, you will explore methods for *neural network pruning* as part of a class-wide competition—attempting to compress a simple neural network which has 592,933 parameters. This network can achieve about 63% test accuracy in a 5-class classification task after training for 50 epochs with the default parameters in the notebook provided.

1.1 Logistics

We have provided a notebook for you which provides data loading helper functions, defines the model architecture, and provides some simple functions to save and load model weights. To download the notebook for this part, use this link. Make sure to use your andrew id. This homework is done on Colab.

1.1.1 Getting the data

Download the compressed training and validation data from this link using your andrew email ID. Upload it to Colab by going to the `Files` tab in the sidebar and clicking on the upload button. The first few cells in the notebook untar and load the training and validation sets. We will use hidden tests on Gradescope to test the final performance of your model. To avoid uploading the dataset every time the runtime disconnects, you may want to upload the dataset to drive and mount it from colab.

1.1.2 Training and pruning the network

You need to first train the network in the notebook. Starting from this trained network, you can then explore different strategies to prune the network (i.e., setting some weights to zero) while attempting to retain accuracy. Your output should be a network with the '.h5' extension with the same architecture as the neural network we define in the notebook. *Changing the architecture or designing a new model architecture yourself is not allowed and will cause the autograder to fail.* It might be helpful to save the weights of your model before proceeding with the pruning methods in order to save progress. You can do this buy using `model.get_weights()` and `model.set_weights(saved_weights)`.

Note that you can't use any pre-existing network pruning libraries to perform pruning; we expect you to implement your pruning methods from scratch.

1.1.3 Submission

You must submit your final pruned model with the name **"my_model_weights.h5"**. In the notebook, we specify how to download the model weights to your local workspace. You will then upload the model weights to Gradescope submission slot: **Homework 6: Model Compression** for autograding. You can submit an unlimited number of times; the final ranking will be determined by your last submission. Also you must submit your code for the model compression to Gradescopes **Homework 6: MC Code** submission slot.

1.1.4 Evaluation

Each submission will receive a score, which is a function of the accuracy on test data and the sparsity of the network. We assume there is an efficient way to encode the indices of sparse matrices, and therefore do not take into account the storage overhead for sparse matrix formats. In particular, the autograder uses the following functions to calculate the score of a model:

```
if accuracy $> 0.6$ and sparsity $> 0$:
    score = (accuracy + num_zero_weights / total_parameters) / 2
else:
    score = 0
```

This evaluation metric encourages to a trade-off between accuracy and sparsity (the ratio between the number of zero weights and the number of total parameters) while ensuring a reasonable accuracy (accuracy > 0.6). You must obtain an accuracy higher than 0.6 and a total **score higher than 0.55** to receive full credit (though we encourage you to search for even higher scores as part of the competition!).

1.2 The competition (50 pts)

We will maintain a leaderboard on Gradescope and plan to give bonus points (5 points on the assignment) to the top 3 submissions in this competition. **You must only use the provided train/validation data to train your model.** At the end of the competition, we will ask the top 3 submissions to submit their code and we will manually check to make sure that they used only the provided train/validation data. **As a reminder, you also cannot use any existing network pruning libraries to perform pruning. We will manually verify the submissions to make sure that's not the case.**

1.3 Reflection (10 pts)

You must implement at least 2 different methods for neural network pruning from scratch. See below for examples (note: you are *not* limited to the methods below—these are just examples).

- Magnitude-based pruning: `https://arxiv.org/abs/1506.02626`
- L1-norm based filter pruning: `https://arxiv.org/abs/1608.08710`
- ThiNet: `https://arxiv.org/abs/1707.06342`
- Regression based Feature Reconstruction: `https://arxiv.org/abs/1707.06168`
- Network Slimming: `https://arxiv.org/abs/1708.06519`
- Sparse Structure Selection: `https://arxiv.org/abs/1707.01213`

You will receive 50 points for submitting a model with a score higher than **0.55** to the leaderboard on Gradescope. In addition, please answer the following two reflection questions.

1.3.1 *[2 points]* **What 2+ methods for pruning did you use? Please describe each briefly.**

1.3.2 *[8 points]* **Compare the performance of the pruning methods: For example, which method led to the best model score? Which was easiest to implement? Provide a plot exploring the methods in terms of the Pareto frontier of sparsity vs. accuracy (report results across at least 3 different sparsity levels).**

2 Federated Averaging (40 pts)

In this portion of the homework, you will explore a popular method for federated learning ('FederatedAveraging' or `FedAvg`). In particular, you will implement `FedAvg` from scratch, and then perform two experiments.

First, you will compare `FedAvg` with traditional mini-batch SGD in order to see the effects of `FedAvg` on convergence; here we expect you to see that performing more local work ('local updating') can improve the speed of convergence in terms of iterations. Second, you will explore the performance of `FedAvg` on two variants of a synthetic dataset that differ in terms of their levels of statistical heterogeneity. Here you should see that despite its possible convergence benefits of `FedAvg`, its performance can degrade in highly heterogeneous settings.

2.1 Logistics

We have provided a notebook for you which provides data loading helper functions, defines the model architecture, and provides some simple functions to compute gradients and report training and test accuracies. You can find the data and the starter notebook as part of the handout. This homework should be done on Google Colab.

2.2 Implementation

To complete the homework, you will need to implement: (1) the the forward and gradient calculation of the linear layer in the MLP; (2) mini-batch SGD; and (3) FedAvg. We've marked all the parts you need to complete with `TODO`s in the code. We expect you to implement these from scratch and not to use any pre-existing libraries.

2.3 Grading + Submission

For this portion you must turn in your notebook on Gradescope to the **Homework 6: FA Code submission slot** in case we need to verify your code. You will receive *[10 points]* for submitting your code. In addition, you will be graded on the following reflection questions. **For the questions below and following the plotting code in the notebook, please plot 'convergence' in terms of the number of epochs vs. loss and/or accuracy on both training and validation data.**

2.3.1 *[10 points]* **Using the IID synthetic data, compare the performance of mini-batch SGD vs. FedAvg. Provide two sets of plots: one tracking the convergence of mini-batch SGD, and one tracking the convergence of FedAvg.**

2.3.2 *[5 points]* **Which method (mini-batch SGD or FedAvg) converges faster in terms of server iterations? How does the number of local steps used in FedAvg affect convergence? Provide details about the number of local steps you used in your experiments.**

2.3.3 *[10 points]* **Now compare the performance of FedAvg on the IID synthetic dataset vs. the heterogeneous synthetic dataset. Provide a set of plots tracking the convergence of FedAvg on each dataset.**

2.3.4 *[5 points]* **What happens to the convergence behavior of FedAvg on the heterogeneous dataset? How does the number of local steps affect this behavior? Provide details about the number of local steps you used in your experiments.**

3 Collaboration Questions

1. (a) Did you receive any help whatsoever from anyone in solving this assignment?

   (b) If you answered 'yes', give full details (e.g. "Jane Doe explained to me what is asked in Question 3.4")

2. (a) Did you give any help whatsoever to anyone in solving this assignment?

   (b) If you answered 'yes', give full details (e.g. "I pointed Joe Smith to section 2.3 since he didn't know how to proceed with Question 2")

3. (a) Did you find or come across code that implements any part of this assignment?

   (b) If you answered 'yes', give full details (book & page, URL & location within the page, etc.).