Homework 5

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

Due: Monday, April 20th at 9:00 AM

1 Overview

The goal of this assignment is to gain familiarity with different machine learning optimization methods using TensorFlow, and to understand techniques for low-latency inference by implementing methods for neural network pruning.

Part A: Optimization Methods: In Part A, you will implement several of the optimization methods discussed in lecture on March 30th including Gradient Descent, SGD, AdaGrad, and Adam, and train a linear model with them using Tensorflow 2.0. [**Note: you can't use tf.compat.v1.train.Optimizer or tf.keras.optimizers in the final evaluation.**]

Part B: Model Compression Competition: In Part B, you will explore pruning methods for model compression. Starting with a fixed model architecture and a pre-trained model, the goal will be to obtain a final model that is *as sparse as possible* without a significant drop in accuracy. We will create a leaderboard to reflect the score for each student (described in detail below). [**Note: you can't use any network pruning libraries in Tensorflow for this portion of the assignment.**]

2 Part A: Optimization Methods

In this portion of the homework you will implement several common optimizers. You will first build a simple linear regression model and train it by calling existing optimizers provided by tf.keras. Then you will implement and evaluate your own optimizers. Finally you will have a better understanding of the mechanism of different optimizers and their effectiveness on simple models. Detailed instructions are included in the assignment notebook.

There are 4 optimizers you need to implement:

- Gradient Descent: update weights using all samples to reach the minimum of the loss function.
- Stochastic Gradient Descent: update weights using one sample at a time to reach the minimum of the loss function.
- AdaGrad: decrease the step size for coordinates with high gradient magnitudes with a cheap approximation of the hessian.

- Adam: combine momentum (move further in the correct direction and less in the wrong direction) and RMSprop (take a moving average of the coordinates of the gradient to put more emphasis on the current gradients).

The instructions in the notebook will guide you to implement each of the optimizers. To understand the theory behind them, we encourage you to read the update rule provided in the notebook, as well as these notes: https://drive.google.com/file/d/1PA0ARElkcUZYuu4fPk_AbrFKF1fwsfI-/view.

2.1 Logistics for Part A

We provide several public tests via `assert` in the notebook. You may want to pass all those tests before submitting your homework.

In order to enable auto-grading, please do not change any function signatures (e.g., function name, parameters, etc) or delete any cells. If you do delete any of the provided cells (even if you re-add them), the autograder will fail to grade your homework. If you do this, you will need to re-download the empty notebook file and fill in your answers again and resubmit.

To download the notebooks of this assignment, go to the following links and open the notebook in **playground** mode so that you can edit and complete the notebook on colab.

- Optimizer notebook

Make sure that you login to your CMU account when accessing to these notebooks.

Then go to `File -> Download .ipynb` to download the notebook to your local computer, and upload the 2 notebooks to the corresponding Gradescope grader.

Please make sure you name the notebook **assignment_notebook.ipynb** before you submit.

3 Part B: Model Compression Competition

In this portion of the homework you will explore methods for neural network pruning as part of a class-wide competition. In particular, you will attempt to compress a simple neural network [1] which has 1,250,858 parameters in total and a 15MB model size. It can achieve about 75 test accuracy after training for 50 epochs with the default parameters provided in the notebook.

To help you start, we provide a simple example of manipulating the weights, assigning the new weights to the model, and evaluating how that affects the accuracy in the notebook. You can explore various pruning techniques as you like. You might also want to read some references [e.g., 2, 3].

3.1 Logistics for Part B

We will provide a notebook for you for this part which provides data loading helper functions, defines the model architecture, and provides some simple functionalities to save and load model weights for you to start with.

To download the notebook for this part, go to this link and open it in the playground mode.

This part of the homework is also done on Colab. You need to first train a network in the notebook until convergence. Starting from this pre-trained full model, you need to explore different strategies to prune the network (i.e., setting some weights to zero) without degrading the accuracy. Your output should be a pre-trained model 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.

Submission. You do **not** need to submit the notebook, but instead, have to submit the final pruned model with a certain name (see the network) under the '.h5' extension. In the notebook, we specify how to download the model weights to your local work space. Similar as Part A, you will then upload the model weights to Gradescope for autograding. You can submit an unlimited number of times, and the final ranking will be determined by the last submission.

Evaluation. Each submission will receive a score, which is a function of the accuracy of the test data and the sparsity level of the model. We assume there is an efficient way to encode the indices of sparse matrices, 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 > 0.65:
    score = (accuracy + num_zero_weights/total_parameters)/2
else:
    score = 0
```

This evaluation metric encourages to achieve a good tradeoff between accuracy and sparsity (the ratio between the number of zero weights and the number of total parameters) while ensuring a reasonable accuracy (accuracy > 0.65).

References

[1] Keras documentation. https://keras.io/examples/cifar10_cnn/.

[2] Z. Liu, M. Sun, T. Zhou, G. Huang, and T. Darrell. Rethinking the value of network pruning. *arXiv preprint arXiv:1810.05270*, 2018.

[3] M. M. Pasandi, M. Hajabdollahi, N. Karimi, and S. Samavi. Modeling of pruning techniques for deep neural networks simplification. *arXiv preprint arXiv:2001.04062*, 2020.