ECE590-10/11: Lab #3
Pruning, Quantization and Huffman Coding to Compress Deep Neural Networks

Hai Li and Yiran Chen

ECE Department, Duke University — October 14, 2019

Objectives

Lab #3 covers most of the core ideas in Prof. Song Han's **Deep Compression paper** [1]. DNNs can be compressed in a 3-stage pipeline: **Pruning**, **Quantization**, and **Huffman Coding**. This 3-stage pipeline can significantly reduce the memory usage of DNNs without inducing much performance degradation. Upon completing Lab #3, you will be able to obtain a good understanding of:

- How to apply weight pruning to remove redundant weights from a large DNN so as to reduce its memory consumption;
- How to apply quantization (weight sharing) to encode the weights of a large DNN with fewer bits for further memory consumption reduction; and
- How to apply Huffman coding to optimize the storage of a large DNN model.

We anticipate that the Lab #3 will consume a lot of computing power of GPUs and therefore encourage you to complete the lab on the JupyterLab server.

Warning: You are asked to complete the assignment independently.

This lab has total **105** points which includes 5 extra points in Assignment 5. The submission deadline will be **11:59pm, Thursday, November 7**.

We provide a Python package for Lab #3 and a Jupyter notebook template named **DeepCompression.ipynb**. To complete Assignments 1$\sim$5, you are asked to follow the flows in the template, and fill all the missing parts in the given package according to your understanding of the Deep Compression pipeline. Assignment 6 asks you to design your own compression pipeline to achieve 90% accuracy on the CIFAR-10 within the given compression rate requirement. You may need to apply compression mechanisms beyond those embedded in the template (e.g. iterative pruning) to achieve the target in Assignment 6.

Considering that different DNNs have different redundancy thus can achieve completely different results within the given compression rate requirement, we fix the model of choice in Lab #3 to **VGG-16_half**. **VGG-16_half** is derived from VGG-16 architecture by applying 0.5 width multiplier (i.e. multiply the number of channels/units by 0.5 in every convolutional/FC layer). This architecture is imported as **VGG16_half** in the given template.

You are asked to submit all the following **3** components by the deadline: (1) a PDF report that describe your results, analysis and lessons learned in the lab; (2) your code within the given Lab #3 package for Assignments 1$\sim$6; (3) A total of **6 checkpoint files** required for grading the Assignment 6 in Lab #3. Please refer the instructions in **DeepCompression.ipynb** and the instructions for **Assignment 6** for details. To reduce the load of the Sakai server, please compress all of the required checkpoints into a **'.zip'** file named with your NetID, i.e., **'NetID.zip'**.

1 Warmup

We need to retrieve a full-precision model by employing regular training mechanisms as a baseline before applying any kind of model compression mechanisms. In this lab, **VGG-16_half** (i.e., VGG-16 with a width-multiplier of 0.5) will be used as our backbone model. Please refer to **vgg16.py** for more details of the implementation. In this lab, we will also use the CIFAR-10 dataset for model training and evaluating.

Assignment 1 (5 points)

As the warmup, tune the hyperparamters and train the given **VGG-16_half** model so that it can achieve >90% accuracy on the CIFAR-10 test dataset. For this part, you may tune only the hyper-parameters in the *DeepCompression.ipynb*. It is not necessary to modify the part of the code beyond *DeepCompression.ipynb*.

2 Weight Pruning

Weight pruning removes the **unimportant** weight parameters in DNNs. More specifically, weight pruning removes the weight parameters which have a magnitude lower than a given pruning threshold. There are two ways to determine the pruning threshold:

- The q-th percentile value in the weight distribution: For a DNN layer, weight parameters with magnitude (i.e., absolute value) lower than the q-th percentile value will be pruned.
- The distribution of the weight parameters: For a DNN layer, the pruning threshold is defined by the standard deviation of the weight parameters multiplied by a sensitivity parameter **s**.

Assignment 2 (35 points)

(a) (10 pts) Complete the method **prune_by_percentage** in **pruned_layers.py**. It determines the pruning threshold in each DNN layer by the **'*q*-th percentile'** value in the weight distribution.

(b) (10 pts) Complete the method **prune_by_std** in **pruned_layers.py**. This method determines the pruning threshold in each DNN layer using the standard deviation of the weight parameters with given sensitivity **s**.

(c) (5 pts) Prune the DNN by calling the 'prune' function. Set the **method** variable to 'std' and set the sensitivity parameter **s** to 0.75. Observe the test accuracy on the CIFAR-10 dataset after you have applied weight pruning. Does the test accuracy on the CIFAR-10 dataset remain the same? Give explanations to support your findings. Also, report the sparsity of your pruned network.

(d) (5 pts) By following the configuration in (c), change the value of sensitivity parameter **s**. Note that both accuracy and sparsity of the DNN model will change accordingly. Plot the relationship between the sparisty of the model and accuracy drop induced by the change of sensitivity parameter **s**. What do you observe from your plot?

(e) (5 pts) Change the **method** variable from **'std'** to **'percentage'**. As such, the pruning threshold will be determined by the ***q*-th percentile**. Choose a ***q*** of your own to retrieve the same sparsity as problem (c). Observe the test accuracy on the CIFAR-10 dataset after you have applied weight pruning. Does the test accuracy on the CIFAR-10 dataset remain the same? Compared to the pruning threshold selection method in problem (c), which method is better? Give explanation to support your judgement.

3 Fine-tuning Pruned DNNs

Accuracy drop is inevitable after pruning a considerable number of weight parameters. We usually implement a retraining process to recover the accuracy loss.

Assignment 3 (5 points)

(a) (5 pts) Complete **train_util.py** to set up the fine-tuning process. Report the test accuracy on the CIFAR-10 dataset after fine-tuning with proper hyperparameter tuning. Is the fine-tuning process able to recover the accuracy loss induced by weight pruning? (*Hint:* The gradient for the pruned weight connection should be set to 0 before weight update. This guarantees that the pruned weight variables remains 0 during the fine-tuning process.)

4 Quantization and Weight Sharing

Next, we will apply quantization on DNN models. Quantization reduces the memory footprint of DNN models by reducing the number of bits required to represent weight parameters. A common form of quantization is weight sharing, which searches for clusters to approximate weights in DNNs. In this way, the weight parameters in DNN can be represented with fewer bits. The quantization scheme is applied after the weight pruning scheme.

Assignment 4 (15 points)

(a) (10 pts) Implement the quantization (i.e., weight sharing) method by using K-means as the clustering method in **quantize.py**. After this quantization, we can retrieve a set of weight clusters to approximate the weight parameters for each layer. Set the quantization bits to 5 and report the test accuracy after quantization. How does the quantization affect the performance of the DNN model? Give explanations to support your findings.

(b) (5 pts) Change the quantization bits and see how it influences the test accuracy on CIFAR-10 dataset. What is the optimal quantization bit with the best trade-off between memory consumption and model performance in this example? Explain.

5 Huffman Coding

Finally, we employ the Huffman coding to further reduce the memory footprints of DNNs. Huffman coding is a common method which uses variable-length encoding for data compression. The Huffman coding scheme is applied after the quantization scheme.

Assignment 5 (20 points)

(a) (5 pts) **(Extra Credit)** Describe how Huffman coding can reduce the memory footprints of DNNs.

(b) (10 pts) Implement the Huffman coding in **huffman_coding.py** to transform weight parameters in each DNN layer into variable-length encoding.

(c) (5 pts) Generate the encoding map and frequency map of weight parameters in each DNN layer and calculate the average encoding length of the clustered weight variables. Give a quantitative analysis on the additional memory reduction with the usage of Huffman coding.

6 Putting All Together

The model compression rate in this deep compression scheme can be formulated as follows:

$$ratio = \frac{nonzero\_params}{total\_params} \times \frac{quant\_bits}{32} \times \frac{huffman\_length}{original\_length}. \quad (1)$$

Where $nonzero\_params$ denotes the number of non-zero weight parameters, $total\_params$ indicates the total number of weight parameters, $quant\_bits$ is the quantization bits, $huffman\_length$ represents the average Huffman encoding length of the clustered weight variables, and $orginal\_length$ indicates the encoding length of weight variables before Huffman coding (i.e. equal to $quant\_bits$).

Assignment 6 (25 points)

(a) (25 pts) By applying the pruning, quantization and Huffman coding pipeline, we aim to compress the model by a compression rate of 40$\times$ without significant drop in accuracy. Additional techniques (e.g., iterative pruning) might be needed to reach this target. The baseline accuracy for this model is about 90%. So your submission should achieve at least 89.5% test accuracy on the CIFAR-10 test dataset. For this assignment, you are asked to submit the following checkpoints:

- **net_before_pruning.pt**: The checkpoint of the full precision model. It is the set of weight parameters before applying pruning on DNN weight parameters.
- **net_after_pruning.pt**: The checkpoint of the pruned model. It is the set of weight parameters after applying pruning on DNN weight parameters.
- **net_after_quantization.pt**: The checkpoint for the model after quantization. It is the set of weight parameters after applying quantization on DNN weight parameters.
- **codebook_vgg16.npy**: The quantization codebook (i.e., clusters) of the weight parameters in each DNN layer.
- **huffman_coding.npy**: The encoding map of each item within the quantization codebook in each DNN layer.
- **huffman_freq.npy**: The frequency map of each item within the quantization codebook in each DNN layer.

Note that you do not need to write I/O protocols to generate these files. These files will be automatically generated if all the codes required in Assignments 1$\sim$5 run correctly. We also attach a sample submission. Please **use it as your reference** when creating the checkpoints. Note that in the sample submission, all of the weight parameters in the checkpoint are randomly distributed.

Grading Rubics 50% of the grading will depend on your final test accuracy as well as compression rate on the test dataset of CIFAR10. Considerable points will be taken if:

- Your submission does not meet the accuracy requirement ($>89.5\%$) on the CIFAR-10 test dataset;
- Your submission does not meet the required compression rate ($>40\times$).

Additional requirements:

- **DO NOT** train on the testset or use pretrained models for unfair advantage.
- **DO NOT** copy code directly online or from other classmates. We will check it! You are at your own risk of failing the code check.

References

[1] S. Han, H. Mao, and W. J. Dally, "Deep compression: Compressing deep neural networks with pruning, trained quantization and huffman coding," *arXiv preprint arXiv:1510.00149*, 2015.