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 13, 2019

Objectives

Lab #3 covers most of the core idea in 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 is able to reduce the memory storage of DNNs without suffering from significant performance degradation. Upon completing Lab #3, you will be able to obtain good understandings of:

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

We encourage you to complete the Lab #3 on the JupyterLab server since it consumes a lot of computing power of GPUs.

Warning: You are asked to complete the assignment independently.

This lab has **100** points in total. The submission deadline will be **11:59pm, Tuesday, November 5**.

We provide a Python package for Lab #3 and a Jupyter notebook template named **DeepCompression.ipynb**. To complete Assignments (1)∼(5), you are asked to follow the flows in the template, and fill all of the missing parts in the given package according to your understanding of the Deep Compression pipeline. For Assignment (6), you are asked 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 mechansims beyond the template (e.g. iterative pruning) to achieve the target in Assignment (6). As different DNNs have different redundancy thus can achieve completely difference within the given compression rate requirement, we fix the model of choice in this Lab #3 to be **VGG-16_half** for fair submission grading. **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 each convolutional/FC layer). This architecture is imported as **VGG16_half** in the given template.

you are asked to submit all the following **3** parts 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∼6; (3) A total of **6 checkpoint files** required for grading the Assignment 6 in Lab #3. Please see the instructions in **DeepCompression.ipynb** and the instructions for **Assignment (6)** for details. To reduce the load of the Sakai server, you should compress all of the required checkpoints into a **'.zip'** file named **'submission.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. We choose **VGG-16_half** (i.e. VGG-16 with width-

multiplier 0.5) as our backbone model for the model compression. You may refer to **vgg16.py** for more details of the implementation. In this lab, we will also use the CIFAR-10 dataset for training and evaluating our models.

Assignment 1 (5 points)

(a) (5 pts) As a warmup, tune the hyperparamters a bit and train the given **VGG-16_half** model to over 90% accuracy on the CIFAR-10 test dataset. For this part, you may only tune the hyperparameters within the *DeepCompression.ipynb*. You may not modify any part of the code beyond *DeepCompression.ipynb*.

2 Weight pruning

Weight pruning removes the **unimportant** weight parameters in each DNN layer. 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:

- For each DNN layer, determine the pruning threshold using the **n-th percentile** value in the weight distribution. Weight parameters with magnitude (i.e. absolute value) lower than the n-th percentile value will be pruned (i.e. set to 0).
- For each DNN layer, determine the pruning threshold using the distribution of the weight parameters. The pruning threshold is defined by the standard deviation of the weight parameters multiplied by a sensitivity parameter **s**. Weight parameters with magnitude (i.e. absolute value) lower than the the pruning threshold will be pruned (i.e. set to 0).

Assignment 2 (35 points)

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

(b) (10 pts) Complete the methods **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) After completing the above functions, try to 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 some explanations to support your findings. Also, report the sparsity of your pruned network.

(d) (5 pts) Following the configuration in (c), try to change the value of sensitivity parameter **s**. Note that the sparsity of the DNN model also changes accordingly. Plot the relationship between the sparisty of the model and accuracy drop induced by the change of sensitivity parameter **s**. What could you observe from your plot?

(e) (5 pts) Try to change the **method** variable from **'std'** to **'percentage'**. This will determine the pruning threshold 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 some explanations to support your judgement.

3 Fine-tuning Pruned DNNs

Usually, accuracy drop is inevitable if we prune a considerable number of weight parameters in the DNN without any retraining. Therefore, we need to implement a retraining process to recover the accuracy loss.

Assignment 3 (5 points)

(a) (5 pts) Try to complete **train_util.py** and 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 each weight update. This is an efficient method to guarantee that the pruned weight variables is always 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 each weight parameter. A common form of quantization is weight sharing, which aims to search for clusters to approximate weights in DNNs and represent them with fewer bits. In this way, the weight parameters in DNN can be represented with fewer bits. The quantization scheme is always applied after the weight pruning scheme.

Assignment 4 (15 points)

(a) (10 pts) Try to 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. What does quantization impact on the performance of the DNN model? Give some explanations to support your findings.

(b) (5 pts) Try to change the quantization bits and see how it influences the test accuracy on CIFAR-10 dataset. Can you find a quantization bit which have the best trade-off between memory consumption and model performance? Give some explanations to support your judgement.

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 encodings for data compression. The Huffman coding scheme is applied after the quantization scheme.

Assignment 5 (20 points)

(a) (5 pts) Give some description on how Huffman coding can reduce the memory footprints of DNNs.

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

(c) (5 pts) Try to 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 them together

The model compression rate in the deep compression scheme (i.e. pruning, quantization, Huffman coding) can be formulated as follows:

$$ratio = \frac{nonzero\_params}{total\_params} \times \frac{quant\_bits}{32} \times \frac{huffman\_length}{original\_length} \tag{1}$$

Where $nonzero\_params$ indicates the number of non-zero weight parameters in the DNN, $total\_params$ indicates the total number of weight parameters in the DNN, $quant\_bits$ indicates the quantization bits, $huffman\_length$ indicates 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) Use the pruning, quantization and Huffman coding pipeline, try to compress the model by a compression rate of 40× without significant drop in accuracy. You may need additional techniques (e.g. iterative pruning) to reach this target. As the baseline accuracy for this model is about 90%, 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 are automatically generated for you if your previous code runs correctly. To ensure that you understand the contents of your submitted checkpoints, we have attached a reference submission for your reference. Note that in the 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. You will lose points if:

- You do not achieve the expected accuracy on the CIFAR-10 test dataset.
- You do not achieve the expected compression rate (i.e. at least 40×).

Some additional requirements:

- **DO NOT** train on the testset or use pretrained models to get unfair advantage.
- **DO NOT** copy code directly online or from other classmates. We will check it! The result can be severe if your codes fail to pass our 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.