ECE590-03: Homework #3
Deep Compression

Hai Li

ECE Department, Duke University — September 22, 2020

Objectives

Homework #3 mainly focuses on the contents of Lectures 11 and introduces the basic concepts on how to use pruning, quantization, and Huffman coding to compress a neural network model for efficient inference and deployment. In this assignment, you will gain hands-on experience in applying Deep Compression [1] techniques on CNN models for CIFAR-10, a real computer-vision dataset. Deep Compression is a very powerful technique, and it can reduce the storage of AlexNet by 35 $\times$, from 240 MB to 6.9 MB, without the loss of accuracy on the ImageNet dataset.

This assignment will start with ResNet-20, the CNN model that you learned and exercised previously in Homework #2. The objective of this assignment is to compress the size of a ResNet-20 model by $\geq 20\times$ via pruning, quantization and Huffman coding, without significant accuracy drop (i.e., $< 1\%$). Upon completing Homework #3, you will be able to obtain a good understanding of:

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

We encourage you to complete the Homework #3 on the JupyterLab server or Google CoLab since the model training will require the computing power of GPUs. Please refer to the **NumPy/PyTorch tutorial** slides on Sakai for the environment setup and instructions for NumPy/PyTorch utilities.

Warning: You are asked to complete the assignment independently.

This lab has **100** points plus **5** bonus points, yet your final score cannot exceed 100 points. The submission deadline will be **11:59pm, Sunday, October 4**. Here, we provide an Jupyter notebook template to start with. You are asked to develop your own code based on this template. At the end, You need to submit three independent files including: (1) *a self-contained report in PDF format* that provides answers to all the conceptual questions and clearly demonstrates all your lab results and observations, (2) `code.zip`, a zipped code file which contains all of the codes to reproduce the results and observations for your result; and (3) the requested checkpoints in lab 4 zipped in a single file `lab4_submission.zip`. **Note that 20 percent of the grade will be deducted if the submissions doesn't follow the above guidance.**

1 True/False Questions (10 pts)

For each question, please provide a short explanation to support your judgment.

Problem 1.1 (2 pts) Generally speaking, the weight pruning does not intervene the weight quantization, as they are orthogonal techniques in compress the size of DNN models.

Problem 1.2 (2 pts) One practical way to achieve an efficient inference of a DNN model is to save potential multiplication results in a look-up table (LUT), as pruning greatly reduces the number of weight variables. As a result, pruning directly benefits the following Huffman coding process.

Problem 1.3 (2 pts) Pruning removes (zeros) part of the weight parameters from the neural network models. When deploying on general hardware platforms (e.g., GPUs), these zeroed parameters don't need to be computed during inference. Thus, we anticipate that training a pruned model on GPUs is much faster than training its original model.

Problem 1.4 (2 pts) For a uniform distribution with cardinality N (random variables can be chosen from N values with equal probability), Huffman coding has an expected average code length of $O(\log(N))$.

Problem 1.5 (2 pts) Intuitively, k-bit quantization with K-means centroid requires Nk-bit storage for N weight variables.

2 Lab 1: Pruning ResNet-20 without Retraining (26 pts)

ResNet-20 is a popular convolutional neural network (CNN) architecture for image classification. Compared to early CNN designs such as VGG-16, ResNet-20 is much more compact. Thus, conducing the model compression on ResNet-20 is more challenging.

Instead of using the provided data loader as in HW2, the official CIFAR-10 data loader from torchvision will be taken in HW 3. This ensures a fair comparison of the results of your compressed ResNet-20 model with others', including the official results.

Lab 1 (26 points)

(a) (4 pts) Setting up the baseline is very important for deep learning. For the given ResNet-20 model, leverage your previous training experience to tune the hyperparameters. At this step, you should be able to obtain >91% accuracy for the given implementation.

(b) (2 pts) Recall that in Lecture 11, we introduced two popular pruning methods: **pruning by percentage** and **pruning by standard deviation**. Conceptually, which method do you think is likely to yield better results? Explain why.

(c) (6 pts) Please refer to `pruned_layers.py` to complete the implementation of *pruning by standard deviation*. This method determines the pruning threshold in each DNN layer using the standard deviation of the weight parameters with a given sensitivity s.

(d) (4 pts) Use the '`prune`' function to find out how well *pruning by standard deviation* can do. Set and test a few values of the sensitivity parameter s = 0.0, 0.25, 0.50, and 1.0. Report the test accuracy and the sparsity of your pruned network for each s. What do you observe from the sparsity pattern for each layer? E.g., is the sparsity structured or unstructured? Which layers are more likely to be sparse? etc. Give explanations to support your findings.

(e) (6 pts) Refer `pruned_layers.py` to complete the implementation of *pruning by percentage*. This method determines the pruning threshold in each DNN layer by the **'q-th percentile'** value in the weight distribution.

(f) (4 pts) Again, use the '`prune`' function to find out how well you did with both *pruning by percentage*. Try the following percentage values q = 0.0, 0.25, 0,5, 0.75. Report the test accuracy and the sparsity of your pruned network for each q. What can you observe from the sparsity pattern for each layer? Give explanations to support your findings.

3 Lab 2: Pruning ResNet-20 with Retraining (24 pts)

Pruning a network without any retraining could result in accuracy loss, especially at a high compression rate. This lab introduces the retraining, which finetunes a pruned model to recovery the performance loss while maintaining its sparsity.

Lab 2 24 points

(a) (6 pts) Explain or illustrate the workflow/pipeline of retraining pruned models. Compared to the regular training pipelines in HW2 (our provided implementation), what additional components does this new pipeline contain?

(b) (6 pts) Follow the workflow/pipeline in Part (a) to setup the retraining for pruned models in `train_util.py`. Note that you only need to fill in the `finetune_after_prune` function to complete this step.

(c) (6 pts) Use your implementation of retraining to recover the loss induced by *pruning by standard deviation*. Take the exactly same sensitivity parameters $s = 0.0$, 0.25, 0.50, and 1.0 as in Lab 1, and observe how the retraining recovers the model accuracy. Please report the test accuracy before/after retraining, the model sparsity, and your retraining settings. Provide intuitive explanations. (**Hint:** Retraining for 20 epochs should be sufficient for recovering the accuracy induced by *pruning by standard deviation*).

(d) (6 pts) Similarly, use your implementation of retraining to recover the loss induced by *pruning by percentage*. Take the exactly same percentage values $q = 0.0$, 0.25, 0,5, 0.75 as in Lab 1, and see how the retraining recovers your model accuracy. Report the test accuracy before/after retraining, model sparsity, and your retraining settings. Explain your observations intuitively. (**Hint:** Retraining for 20 epochs should be sufficient for recovering the accuracy induced by *pruning by percentage*).

4 Lab 3: Quantization & Huffman coding (18 pts)

Quantization can further compress models by reducing the bit precision of weight parameters. A type of quantization is weight clustering, which searches for a number of clusters to approximate weight parameters. To further compress weights, we use Huffman coding after applying quantization.

Lab 3 (18 points)

(a) (6 pts) Implement the quantization (weight sharing) method by using *K*-means as the clustering method in `quantize.py`. Here, you can define a quantization bit, and the quantization will retrieve a set of weight clusters to approximate the weight parameters for each layer.

(b) (4 pts) Plot and compare the weight distributions before and after applying the quantization.

(c) (4 pts) Try more quantization bits: 6, 5, 4, 3 to see how quantization affects the test accuracy. According to your results, what is the optimal quantization bit with the best trade-off between memory consumption and model performance in this example? Briefly describe your findings and explain why.

(d) (**Bonus**, 5 pts) By default, the linear initialization is adopted to initialize K-means clusters. Try other initialization approaches in Lecture 11 and report the results on the test dataset. According to your investigation, what is the best way of initializing the K-means centroid? Briefly describe your findings.

(e) (4 pts) Implement Huffman coding in `huffman_coding.py` to transform weight parameters in each layer into variable-length encoding. When taking 5 bits for quantization, what is the average bit length for the weight parameters with Huffman encoding?

5 Lab 4: Putting All Together (22 pts)

The model compression rate in the deep compression scheme so far 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 $quant\_bits$).

Lab 4 (22 points)

(22 pts) By applying the pruning, quantization and Huffman coding pipeline, we aim to compress the model by a compression rate of $\geq 20\times$, with $< 1\%$ accuracy loss. Additional techniques (e.g., iterative pruning) might be needed to reach this target. The baseline accuracy for this model is about 91%. So your submission should achieve at least 90% test accuracy on the CIFAR-10 test dataset.

You need 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 weight parameters.
- **net_after_pruning.pt**: The checkpoint of the pruned model. It is the set of weight parameters after applying pruning on weight parameters.
- **net_after_quantization.pt**: The checkpoint for the model after quantization. It is the set of weight parameters after applying quantization on weight parameters.
- **codebook_resnet20.npy**: The quantization codebook (i.e., clusters) of the weight parameters in each layer of the ResNet-20 architecture.
- **huffman_encoding.npy**: The encoding map of each item within the quantization codebook in each layer of the ResNet-20 architecture.
- **huffman_freq.npy**: The frequency map of each item within the quantization codebook in each layer of the ResNet-20 architecture.

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 in Lab 1∼3 run correctly. Zip all of the requested checkpoints into a single submission file `lab4_submission.zip`. We also attach a sample submission, which can be used as a reference for creating the checkpoints. Note that in the sample submission, all of the weight parameters in the checkpoint are randomly distributed.

Grading Rubrics: 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 ($>90\%$) on CIFAR-10 test dataset;
- Your submission does not meet the required compression rate ($>20\times$).

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.
- **DO NOT overtrain your ResNet-20 model before pruning!** You are required to train your ResNet-20 model for a maximum of 100 epochs to get the baseline accuracy before applying any model compression techniques. **Training more than 100 epochs is prohibited, and we will check that from your submitted checkpoints.**

Info: As this assignment requires much computing power Please use your previous training experience to configure the hyperparameters for the given ResNet-20 model. We also suggest:

- Plan your work in advance and start early. We will **NOT** extend the deadline because of the unavailability of computing resources.
- Be considerate and kill Jupyter Notebook instances when you do not need them.
- **DO NOT** run your program forever. Please follow the recommended/maximum training budget in each lab.

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.