ECE 661: Homework #4
Pruning and Fixed-point Quantization

Hai Li

ECE Department, Duke University — October 18, 2023

Objectives

Homework #4 covers the contents of Lectures 12 $\sim$ 15. This assignment starts with conceptual questions on model pruning and quantization techniques, followed by lab questions evaluating the effectiveness of different sparse optimization methods on linear models, iterative pruning of a pretrained CNN model, and training fixed-point quantized CNNs.

Warning: You are asked to complete the assignment independently.

This lab has a total of **100** points. You must submit your report in PDF format and your original codes for the lab questions through **Sakai** before **11:59:00pm, Wednesday, November 8**. You need to submit **three individual 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) *a single code/notebook file* used to produce the results for Lab1: Sparse optimization of linear models, and (3) *the completed notebook file* for Lab 2 and Lab 3.
Note that 20 percent of the grade will be deducted for the submissions uploaded in a zip file.

1 True/False Questions (20 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 compressing the size of DNN models.

Problem 1.2 (2 pts) In weight pruning techniques, the distribution of the remaining weights does not affect the inference latency.

Problem 1.3 (2 pts) In deep compression pipeline, even if we skip the quantization step, the pruned model can still be effectively encoded by the following Huffman coding process as pruning greatly reduces the number of weight variables.

Problem 1.4 (2 pts) Directly using SGD to optimize a sparsity-inducing regularizer (i.e. L-1, DeepHoyer etc.) with the training loss will lead to exact zero values in the weight elements, there's no need to apply additional pruning step after the optimization process.

Problem 1.5 (2 pts) Using soft thresholding operator will lead to better results comparing to using L-1 regularization directly as it solves the "bias" problem of L-1.

Problem 1.6 (2 pts) Group Lasso can lead to structured sparsity on DNNs, which is more hardware-friendly. The idea of Group Lasso comes from applying L-2 regularization to the L-1 norm of all of the groups.

Problem 1.7 (2 pts) Proximal gradient descent introduces an additional proximity term to minimize regularization loss in the proximity of weight parameters. The proximity term allows smoother convergence of the overall objective.

Problem 1.8 (2 pts) Models equipped with early exits allows some inputs to be computed with only part of the model, thus overcoming the issue of overfitting and overthinking.

Problem 1.9 (2 pts) When implementing quantization-aware training with STE, gradients are quantized during backpropagation, ensuring that updates are consistent with the quantized weights.

Problem 1.10 (2 pts) Comparing to quantizing all the layers in a DNN to the same precision, mixed-precision quantization scheme can reach higher accuracy with a similar-sized model.

2 Lab 1: Sparse optimization of linear models (30 pts)

By now you have seen multiple ways to induce a sparse solution in the optimization process. This problem will provide you some examples under linear regression setting so that you can compare the effectiveness of different methods. For this problem, consider the case where we are trying to find a sparse weight W that can minimize $L = \sum_i (X_i W - y_i)^2$. Specifically, we have $X_i \in \mathbb{R}^{1\times 5}$, $W \in \mathbb{R}^{5\times 1}$ and $||W||_0 \leq 2$.

For Problem (a) - (f), consider the case where we have 3 data points: $(X_1 = [1, -2, -1, -1, 1], y_1 = 7)$; $(X_2 = [2, -1, 2, 0, -2], y_2 = 1)$; $(X_3 = [-1, 0, 2, 2, 1], y_3 = 1)$. For stability the objective L should be minimized through full-batch gradient descent, with initial weight W^0 set to $[0; 0; 0; 0; 0]$ and use learning rate $\mu = 0.02$ throughout the process. Please run gradient descent for 200 steps for all the following problems.

You will need to use NumPy to finish this set of questions, please put all your code for this set of questions into one python/notebook file and submit it on Sakai. Please include all your results, figures and observations into your PDF report.

Lab 1 (30 points)

(a) (4 pts) Theoretical analysis: with learning rate μ, suppose the weight you have after step k is W^k, derive the symbolic formulation of weight W^{k+1} after step k+1 of full-batch gradient descent with $X_i, y_i, i \in \{1, 2, 3\}$. (Hint: note the loss L we have is defined differently from standard MSE loss.)

(b) (3 pts) In Python, directly minimize the objective L without any sparsity-inducing regularization/constraint. Plot the value of $log(L)$ vs. #steps throughout the training, and use another figure to plot how the value of each element in W is changing throughout the training. From your result, is W converging to an optimal solution? Is W converging to a sparse solution?

(c) (6 pts) Since we have the knowledge that the ground-truth weight should have $||W||_0 \leq 2$, we can apply **projected gradient descent** to enforce this sparse constraint. Redo the optimization process in (b), this time prune the elements in W after every gradient descent step to ensure $||W^l||_0 \leq 2$. Plot the value of $log(L)$ throughout the training, and use another figure to plot the value of each element in W in each step. From your result, is W converging to an optimal solution? Is W converging to a sparse solution?

(d) (5 pts) In this problem we apply ℓ_1 regularization to induce the sparse solution. The minimization objective therefore changes to $L + \lambda||W||_1$. Please use full-batch gradient descent to minimize this objective, with $\lambda = \{0.2, 0.5, 1.0, 2.0\}$ respectively. For each case, plot the value of $log(L)$ throughout the training, and use another figure to plot the value of each element in W in each step. From your result, comment on the convergence performance under different λ.

(e) (6 pts) Here we optimize the same objective as in (d), this time using **proximal gradient update**. Recall that the proximal operator of the ℓ_1 regularizer is the soft thresholding function. Set the threshold in the soft thresholding function to $\{0.004, 0.01, 0.02, 0.04\}$ respectively. Plot the value of $log(L)$ throughout the training, and use another figure to plot the value of each element in W in each step. Compare the convergence performance with the results in (d). (Hint: Optimizing $L + \lambda||W||_1$ using gradient descent with learning rate μ should correspond to proximal gradient update with threshold $\mu\lambda$)

(f) (6 pts) Trimmed ℓ_1 ($T\ell_1$) regularizer is proposed to solve the "bias" problem of ℓ_1. For simplicity you may implement the $T\ell_1$ regularizer as applying a ℓ_1 regularization with strength λ on the 3 elements of W **with the smallest absolute value**, with no penalty on other elements. Minimize $L + \lambda T\ell_1(W)$ **using proximal gradient update** with $\lambda = \{1.0, 2.0, 5.0, 10.0\}$ (correspond the soft thresholding threshold $\{0.02, 0.04, 0.1, 0.2\}$). Plot the value of $log(L)$ throughout the training, and use another figure to plot the value of each element in W in each step. Comment on the convergence comparison of the Trimmed ℓ_1 and the ℓ_1. Also compare the behavior of the early steps (e.g. first 20) between the Trimmed ℓ_1 and the iterative pruning.

3 Lab 2: Pruning ResNet-20 model (25 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.

This lab explores the element-wise pruning of ResNet-20 model on CIFAR-10 dataset. We will observe the difference between single step pruning and iterative pruning, plus exploring different ways of setting pruning threshold. Everything you need for this lab can be found in `HW4.zip`.

Lab 2 (25 points)

(a) (2 pts) In `hw4.ipynb`, run through the first three code block, report the accuracy of the floating-point pretrained model.

(b) (6 pts) Complete the implementation of *pruning by percentage* function in the notebook. Here we determines the pruning threshold in each DNN layer by the **'q-th percentile'** value in the absolute value of layer's weight element. Use the next block to call your implemented *pruning by percentage*. Try pruning percentage q = 0.3, 0.5, 0.7. Report the test accuracy q. (**Hint:** You need to reload the full model checkpoint before applying the prune function with a different q).

(c) (6 pts) Fill in the `finetune_after_prune` function for pruned model finetuning. Make sure the pruned away elements in previous step are kept as 0 throughout the finetuning process. Finetune the pruned model with q=0.7 for 20 epochs with the provided training pipeline. Report the best accuracy achieved during finetuning. Finish the code for sparsity evaluation to check if the finetuned model preserves the sparsity.

(d) (5 pts) Implement iterative pruning. Instead of applying single step pruning before finetuning, try iteratively increase the sparsity of the model before each epoch of finetuning. Linearly increase the pruning percentage for 10 epochs until reaching 70% in the final epoch (prune $(7 \times e)\%$ before epoch e) then continue finetune for 10 epochs. Pruned weight can be recovered during the iterative pruning process before the final pruning step. Compare performance with (c)

(e) (6 pts) Perform magnitude-based global iterative pruning. Previously we set the pruning threshold of each layer following the weight distribution of the layer and prune all layers to the same sparsity. This will constrain the flexibility in the final sparsity pattern across layers. In this question, Fill in the `global_prune_by_percentage` function to perform a global ranking of the weight magnitude from all the layers, and determine a single pruning threshold by percentage for all the layers. Repeat iterative pruning to 70% sparsity, and report final accuracy and the percentage of zeros in each layer.

4 Lab 3: Fixed-point quantization and finetuning (25 pts)

Besides pruning, fixed-point quantization is another important technique applied for deep neural network compression. In this Lab, you will convert the ResNet-20 model we used in previous lab into a quantized model, evaluate is performance and apply finetuning on the model.

Lab 3 (25 points)

(a) (10 pts) As is mentioned in lecture 15, to train a quantized model we need to use floating-point weight as trainable variable while use a straight-through estimator (STE) in forward and backward pass to convert the weight into quantized value. Intuitively, the forward pass of STE converts a float weight into fixed-point, while the backward pass passes the gradient straightly through the quantizer to the float weight.

To start with, implement the STE forward function in `FP_layers.py`, so that it serves as a linear quantizer with dynamic scaling, as introduced on page 9 of lecture 15. Please follow the comments in the code to figure out the expected functionality of each line. **Take a screen shot** of the finished STE class and paste it into the report. Submission of the `FP_layers.py` file is not required. (**Hint:** Please consider zeros in the weight as being pruned away, and build a mask to ensure that STE is only applied on non-zero weight elements for quantization.)

(b) (2 pts) In `hw4.ipynb`, load pretrained ResNet-20 model, report the accuracy of the floating-point pretrained model. Then set `Nbits` in the first line of block 4 to 6, 5, 4, 3, and 2 respectively, run it and report the test accuracy you got. (Hint: In this block the line defining the ResNet model (second line) will set the residual blocks in all three stages to `Nbits` fixed-point, while keeping the first conv and final FC layer still as floating point.)

(c) (5 pts) With `Nbits` set to 4, 3, and 2 respectively, run code block 4 and 5 to finetune the quantized model for 20 epochs. You do not need to change other parameter in the `finetune` function. For each precision, report the highest testing accuracy you get during finetuning. Comment on the relationship between precision and accuracy, and on the effectiveness of finetuning.

(d) (4 pts) In practice, we want to apply both pruning and quantization on the DNN model. Here we explore how pruning will affect quantization performance. Please load the checkpoint of the 70% sparsity model with the best accuracy from Lab 2, repeat the process in (c), report the accuracy before and after finetuning, and discuss your observations comparing to (c)'s results.

(e) (4 pts) Symmetric quantization is a commonly used and hardware-friendly quantization approach. In symmetric quantization, the quantization levels are symmetric to zero. Implement symmetric quantization in the STE class and repeat the process in (b). Compare and analyze the performance of symmetric quantization and asymmetric quantization.