ECE 590-03: Homework #4
Advanced Sparse Optimization and Fixed-point Quantization

Hai Li

ECE Department, Duke University — October 4, 2020

Objectives

Homework #4 covers the contents of Lectures 12 $\sim$ 14. 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, and training fixed-point quantized CNNs.

Warning: You are asked to complete the assignment independently.

This lab has a total of **100** points plus 10 bonus points, yet your final score cannot exceed 100 points. You must submit your report in PDF format and your original codes for the lab questions through **Sakai** before **11:55:00pm, Thursday, October 15**. You need to submit **two 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 file* used to produce all the results for Lab1: Sparse optimization of linear models. **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) 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 1.2 (2 pts) The sparsity of a model will typically not change if hard thresholding (iterative pruning) with a fixed percentile threshold is applied, while the sparsity will gradually increase when soft thresholding operator is applied in the training.

Problem 1.3 (2 pts) Using soft thresholding operator will lead to better results comparing to using L-1 regularization directly as it makes the optimization smoother and solves the "bias" problem of L-1.

Problem 1.4 (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 1.5 (2 pts) The Hoyer regularizer is similar to L-0 regularizer in that it is scale-invariant. The Hoyer-square regularizer shows an automatic 'trimming' effect to preserve larger weights while penalizing smaller ones. Therefore, it has a great approximation of the L-0 norm and thus it is beneficial for crafting sparse DNN models.

Problem 1.6 (2 pts) Proximal gradient descent introduces an additional proximity term to minimize regularization loss in the proximity of weight parameters. While the proximity term does not change the optimal solution, it allows smoother convergence of the overall objective.

Problem 1.7 (2 pts) The objective of trimmed L1 is equivalent to setting up an L-0 constraint on sparse minimization problem. This gives a much stronger guarantee on the sparsity than Lasso (i.e. L-1 regularizer).

Problem 1.8 (2 pts) When applied to pruning deep neural network, ADMM-based method can lead to higher sparsity comparing to L-1 based methods under same accuracy without additional training cost.

Problem 1.9 (2 pts) When training a quantized model with STE, both the forward-pass and the gradient accumulation will be performed with the full-precision weight kept in the training process.

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 (40+10 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 (40+10 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) (7 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) (7 pts) Since we have the knowledge that the ground-truth weight should have $||W||_0 \leq 2$, we can apply iterative pruning 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 the 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) (7 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 the plot the value of each element in W in each step. From your result, comment on the convergence performance under different λ.

(e) (7 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 the 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) (8 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 the 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.

(g) (Bonus 10 pts) Redo Problem (c) - (f), this time with 2 data points (X_1, y_1) and (X_2, y_2) only. For each method you may only try one λ or threshold that you think works the best. How is the convergence performance different from using 3 data points?

3 Lab 2: Fixed-point quantization and finetuning (40 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. Everything you need for this lab can be found in `HW4.zip`.

The coding of this lab is fairly simple, so except for the part where you are asked to include a screenshot in the report, no code file needs to be submitted for this lab question. Please include all your results and observation into your PDF report.

Lab 2 (40 points)

(a) (10 pts) As is mentioned in lecture 14, 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 14. 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.

(b) (4 pts) In `hw4.ipynb`, run through the first three code block, 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) (6 pts) With `Nbits` set to 4, 3, and 2 respectively, run code block 4 and 5 to finetune the quantized model. Please run 10 epochs of finetuning for 4-bit and 3-bit model, and run 20 epochs for the 2-bit model. You do not need to change other parameter in the `finetune` function except the number epochs. 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) (6 pts) Now we move on to code block 6 and 7, where we explore the effectiveness of mixed-precision quantization. Here we use `Nbits1, Nbits2` and `Nbits3` to control the precision of the three stages in the ResNet-20 models respectively. You may take a look at `resnet20.py` if you are not sure what these three stages correspond to.

In Code block 6, set {4,4,2}, {4,2,4} and {2,4,4} precision to the three stages respective. In each case, run through block 6 and 7 and report the testing accuracy before and after finetuning. All the finetuning should be performed for 10 epochs.

(e) (9 pts) For each mixed-precision quantization scheme in (d), compute the average precision of all the weight elements in the model (consider float weight in first conv and final FC as 32-bit, do not count BN layers and bias). Please show how you compute this precision in detail in the report. (Hint: for average precision computation, recall what you did for the Huffman coding length computation in HW3. You can do this either by code or by hand.)

(f) (5 pts) Combine the results in (d) and (e), which scheme do you think leads to the best accuracy-average precision tradeoff? Based on this observation, infer which of the three stages in the ResNet is the least important for reaching high accuracy?