**1 True/False Questions (20 pts)**

Problem 1.1 – True

Explanation: weight pruning and weight quantization generally do not intervene because they are orthogonal techniques. In some cases, weight pruning and weight quantization can also coexist as long as we define a reasonable weight mask to tell which weight values have been pruned.

Problem 1.2 – False

Explanation: The inference with the pruned network may not bring much speedup on traditional platforms like GPU, even with specifically designed sparse matrix multiplication algorithm. Specialized hardware accelerator should be designed to store and compute non-structured sparse weights.

Problem 1.3 – False

Explanation: Pruning and quantization are both important steps in deep compression pipeline, because pruning reduces the number of weights while quantization reduces the bits per weight. In addition, if there is no quantization process, Huffman encoding cannot be implemented.

Problem 1.4 – False

Explanation: During the training, the Lasso will automatically guide the parameters in the DNN model towards zero. Only one final pruning step with a small constant threshold is needed to reach a sparse model, can reach similar sparse level as the iterative pruning.

Problem 1.5 – False

Explanation: Soft thresholding reveals the “bias” problem of L1, partially solved by SCAD & MCP etc. The Trimmed L1 combines hard and soft thresholding and solved the “bias” problem of L1.

Problem 1.6 – True

Explanation: Group Lasso applies L1 regularization to the L2 norms of all the groups to induce all-zero groups. For L2 norm to be 0, all elements within the group have to be 0 simultaneously, leading to structured sparsity. The overall sparsity is less, but the speedup on GPU is higher.

Problem 1.7 – True

Explanation: Proximal gradient update has an added proximity term. This proximity term will allow smoother convergence of the overall objective.

Problem 1.8 – True

Explanation: The effectiveness of early exiting is overcoming overfitting and overthinking and identifying easy data at early stage.

Problem 1.9 – False

Explanation: DNN training with STE: train with full precision weight, loss computation with quantized weight.

Problem 1.10 – True

Explanation: Perform mixed-precision quantization (assign different precisions to different layers) can provide better size/latency-accuracy tradeoff than fixed quantization.

Lab 1 results and analysis are shown in the next page.

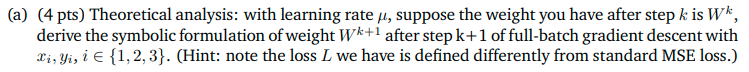
**Important Note for Lab 1 after Seeking Advice from Lead TA during OH:**

In Lab 1, we are asked to use full-batch gradient descent to minimize loss. However, I mistakenly updated my weight matrix for each data point within each training epoch, which means I am using mini-batch gradient descent to minimize the loss, and the batch size is 1.

We have 3 data points in total, and the total number of gradient descent steps is 200. For full batch case, the weight matrix is updated 200 times. For my case (mini-batch and batch size is 1), however, the weight matrix is updated 3 \* 200 = 600 times. Consequently, for all of my figures in Lab 1, there could be such an effect that “converges faster”. However, when we try to compare the convergence performance among different models, such an effect will not influence the comparing result, because all weights of all models in Lab 1 have been updated 600 times.

Due to limited time, I cannot modify my Lab 1 to fix this issue. After seeking advice from the Lead TA during Office Hour, I wrote this note to make it clear. I apology for my mistake, and I feel much appreciated for your understanding. Thank you.

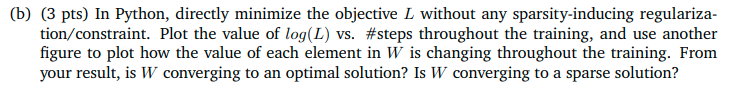
**2 Lab 1: Sparse optimization of linear models (30 pts)**



Solution for (a):

Diagram, letter

Description automatically generated



Solution for (b):

A screenshot of a computer

Description automatically generated with medium confidenceChart, line chart

Description automatically generated

From my result, W is converging to an optimal solution, but W is not converging to a sparse solution.

Text, letter

Description automatically generated

Solution for (c):

Chart, line chart

Description automatically generatedA picture containing text, shoji

Description automatically generated

From my result, W is converging to an optimal solution, and W is converging to a sparse solution.

Text

Description automatically generated

Solution for (d): First, when lambda = 0.2.

Chart, line chart

Description automatically generatedChart, line chart

Description automatically generated

Second, when lambda = 0.5.

Chart, line chart

Description automatically generatedChart, histogram

Description automatically generated

Third, when lambda = 1.0.

A picture containing text, shoji

Description automatically generatedChart, histogram

Description automatically generated

Fourth, when lambda = 2.0.

Chart, line chart

Description automatically generatedChart, histogram

Description automatically generated

Comment: From my result, when I increase the lambda value, the model converges faster, but the log loss increases a bit, from below 0.1 to above 1 (still below 10), and the convergence performance becomes noisy. The log loss and weights fluctuate more drastically as increasing the lambda value.

Text

Description automatically generated

Solution for (e): First, when threshold = 0.004.

Chart, line chart

Description automatically generatedA picture containing text, shoji, tiled

Description automatically generated

Second, when threshold = 0.01.

Chart, line chart

Description automatically generatedChart, line chart

Description automatically generated

Third, when threshold = 0.02.

A picture containing text, shoji

Description automatically generatedChart, histogram

Description automatically generated

Fourth, when threshold = 0.04.

A picture containing text, shoji

Description automatically generatedChart, histogram

Description automatically generated

Compare the convergence performance with the results in (d), for one specific threshold corresponding to one particular lambda, the proximal gradient update with soft thresholding function in (e) solved the “noisy performance problem” of L1 regularization in (d). Here in (e) the log loss and weights do not fluctuate after convergence. In addition, as the hint mentions, for each threshold in (e) corresponding to its particular lambda value in (d), if we do not consider the “noisy problem”, both models would have the same convergence performance, in terms of log loss value, weight values and convergence speed.

Text

Description automatically generated

Solution for (f):

First, when lambda = 1.0, threshold = 0.02.

Chart, line chart

Description automatically generatedChart, line chart

Description automatically generated

Second, when lambda = 2.0, threshold = 0.04.

Chart, line chart

Description automatically generatedA picture containing text, shoji

Description automatically generated

Third, when lambda = 5.0, threshold = 0.1.

A picture containing text, shoji

Description automatically generatedChart, line chart

Description automatically generated

Fourth, when lambda = 10.0, threshold = 0.2.

A picture containing text, shoji

Description automatically generatedChart, line chart

Description automatically generated

Compare the Trimmed L1 and L1: First, it is very clear that the convergence performance is not noisy, its log loss and weights do not fluctuate after convergence, while the log loss and weights fluctuate drastically if we have a large lambda value such as 1.0 and 2.0. Besides, Trimmed L1’s weights converge faster and achieves lower log loss than L1. Trimmed L1 generally has a log loss below 10 to the power of minus 27, while L1 achieves a log loss ranging from above 0.01 and below 10.

Compare the Trimmed L1 and the iterative pruning: First, for the log loss figure, Trimmed L1 and iterative pruning are a bit similar, the log loss continues to decrease and stabilize at a value below 10 to the power of minus 27 after around 160 epochs. However, as for weights, especially for the first 20 epochs, we find that for Trimmed L1, if we have a relatively small threshold value, then some weights would fluctuate a bit, and then finally converge to a stable value. Maybe this could to some extent indicate the “soft thresholding” for the Trimmed L1 model and the “hard thresholding” for the iterative pruning model.

If we have a relatively large threshold value, then the Trimmed L1’s weights are just like the iterative pruning ‘s weights in (c): what weights should be 0 would remain 0, what weights should be nonzero would gradually converge to a nonzero value without noise and fluctuation.

Lab 2 results and analysis are shown in the next page.

**3 Lab 2: Pruning ResNet-20 model (30 pts)**



Solution for (a): The accuracy of the floating-point pretrained model is

Test Accuracy = 0.9151.

Text

Description automatically generated

Solution for (b):

When q = 0.4, Test Accuracy = 0.8874.

When q = 0.6, Test Accuracy = 0.7226.

When q = 0.8, Test Accuracy = 0.1003.

Text

Description automatically generated

Solution for (c):

The best Test Accuracy achieved during finetuning is 0.8790.

The finetuned model preserves the sparsity, and the model sparsity is shown below.

Text

Description automatically generated

Text

Description automatically generated

Solution for (d):

The best Test Accuracy achieved during iterative pruning is 0.8739.

The iteratively pruned model preserves the sparsity, and the model sparsity is shown below.

Text

Description automatically generated

Compare performance with (c): The finetuning model achieves 0.8790 test accuracy, while the iterative pruning model achieves 0.8739 test accuracy, and it seems that iterative pruning performs slightly worse than the finetuning model. In theory, iterative pruning will work better if we train the model for more epochs. Considering that in Lab 2 and 3 we only train one model for 20 epochs, we might observe that iterative pruning performs slightly worse than finetuning.

Please continue to the next page.

Text

Description automatically generated

Solution for (e):

The best/final Test Accuracy of magnitude-based global iterative pruning is 0.8868.

The global iterative pruning model preserves the Total Sparsity, and the model sparsity is shown below.

Graphical user interface, text, table

Description automatically generated

Please continue to the next page.

**4 Lab 3: Fixed-point quantization and finetuning (20 + 10 pts)**

A screenshot of a computer

Description automatically generated with medium confidence

Solution for (a): The screen shot is shown below. Please note that the two lines of codes for bonus part are now commented, these two lines of codes will not be commented when I work on the bonus part (d) and (e).

Text

Description automatically generated

Please continue to the next page.

Text

Description automatically generated

Solution for (b):

The Test Accuracy of the floating-point pretrained model is 0.9151.

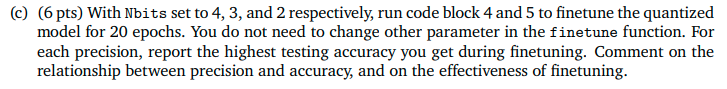
When Nbits = 6, Test Accuracy = 0.9145.

When Nbits = 5, Test Accuracy = 0.9112.

When Nbits = 4, Test Accuracy = 0.8972.

When Nbits = 3, Test Accuracy = 0.7662.

When Nbits = 2, Test Accuracy = 0.0899.



Solution for (c):

When Nbits = 4, Highest Testing Accuracy = 0.9149.

When Nbits = 3, Highest Testing Accuracy = 0.9069.

When Nbits = 2, Highest Testing Accuracy = 0.8591.

Comment:

Before finetuning, if we decrease Nbits (decrease precision) to the pretrained model weights, then the testing accuracy will decrease. If we decrease Nbits to a very small value (very low precision), such as 2, then the pretrained model will fail the testing, with extremely low accuracy.

With the help of finetuning, however, even though we might have very small Nbits value (very low precision), after finetuning for 20 epochs, we can still achieve relatively good testing accuracy (above 85%). Therefore, it could be concluded that finetuning can help significantly improve the performance of quantized model.

Please continue to the next page.

Text

Description automatically generated

Solution for (d):

When Nbits = 4,

Test Accuracy before finetuning = 0.1000.

Test Accuracy after finetuning = 0.8977.

When Nbits = 3,

Test Accuracy before finetuning = 0.1017.

Test Accuracy after finetuning = 0.8718.

When Nbits = 2,

Test Accuracy before finetuning = 0.1026.

Test Accuracy after finetuning = 0.3342.

First, before finetuning, all 3 models achieved about 10% testing accuracy, which means the models completely failed the testing.

After finetuning, when Nbits = 4, the model’s testing accuracy drops from 91.49% in (c), to 89.77% in (d).

When Nbits = 3, the model’s testing accuracy drops from 90.69% in (c), to 87.18% in (d).

When Nbits = 2, the model’s testing accuracy drops from 85.91% in (c), to 33.42% in (d).

For 3-bit and 4-bit quantization, we have relatively small accuracy drop, which is still acceptable.

For 2-bit quantization, however, the testing accuracy drops significantly, and a 33.42% testing accuracy means that this model failed the testing.

Please continue to the next page.

Text

Description automatically generated

Solution for (e):

Final Testing Accuracy = 0.7479.

The percentage of zeros in each layer (sparsity) is shown below.

Text

Description automatically generated

Compare with Lab 2 (e) results: The model in Lab 2 (e) achieves 88.68% testing accuracy, which is clearly better than what we have in Lab 3 (e), because Lab 3 (e) model achieves only 74.79% testing accuracy.

Compare with previous question’s results:

Consider only the pruning. If we set the sparsity as 80%, all pruning models can achieve above 85% testing accuracy after finetuning for 20 epochs, so we might not have a very good model here in Lab 3 (e).

Consider only the quantization. If we set the Nbits as 2, the quantized model can still achieve above 85% testing accuracy after finetuning for 20 epochs, so we might not have a very good model here in Lab 3 (e).

With 74.79 testing accuracy, it seems that model in Lab 3 (e) is only better than the 33.42% accuracy quantization pruning finetuning model in Lab 3 (d), which has Nbits = 2, 80% sparsity, and finetuning for 20 epochs.