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)

(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.)

Solution for (a):

$$L = \sum_i (X_i W - y_i)^2,\ i \in \{1,2,3\},\ X_i \in \mathbb{R}^{1\times5},\ W \in \mathbb{R}^{5\times1}$$

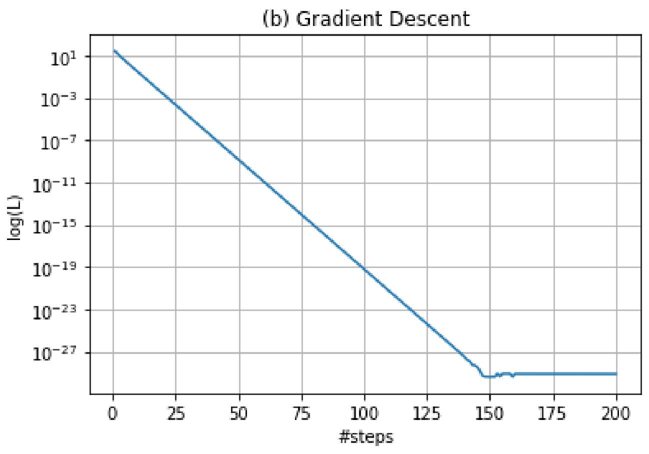
$$\frac{\partial L}{\partial W} = 2\cdot \sum_i \underbrace{(X_i W - y_i)}_{1\times1}\cdot \underbrace{X_i^T}_{5\times1} = 2\cdot \sum_{i=1}^{3}\left[(X_i W - y_i)\cdot X_i^T\right],\ i \in \{1,2,3\}.$$

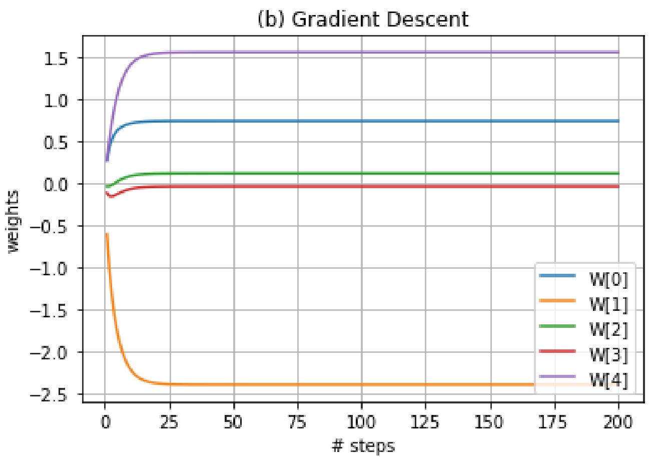
$$W^{k+1} = W^k - \mu \cdot \frac{\partial L}{\partial W}(W^k)$$

$$W^{k+1} = W^k - \mu\cdot 2\cdot \sum_{i=1}^{3}\left[(X_i W - y_i)\cdot X_i^T\right],\ i \in \{1,2,3\}.$$

(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?

Solution for (b):

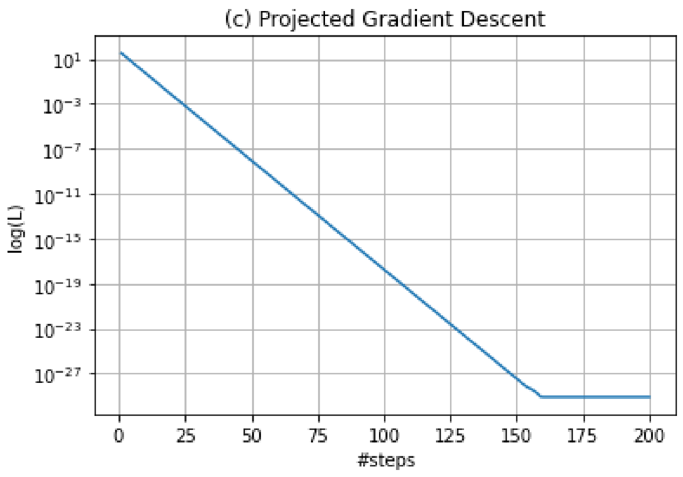


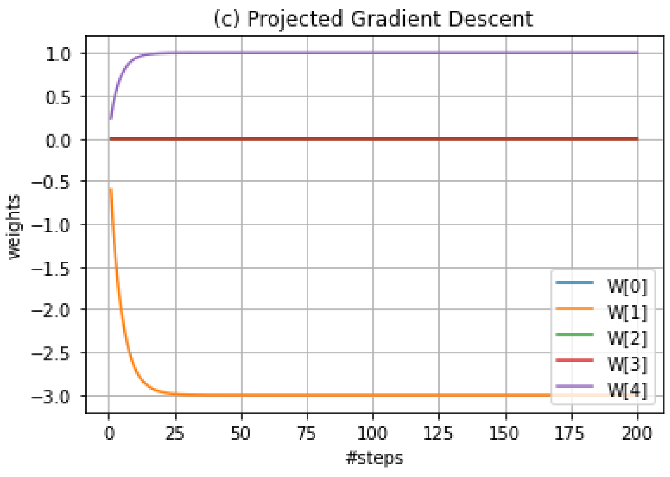


From my result, W is converging to an optimal solution, but W is not converging to a sparse solution.

(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 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?

Solution for (c):

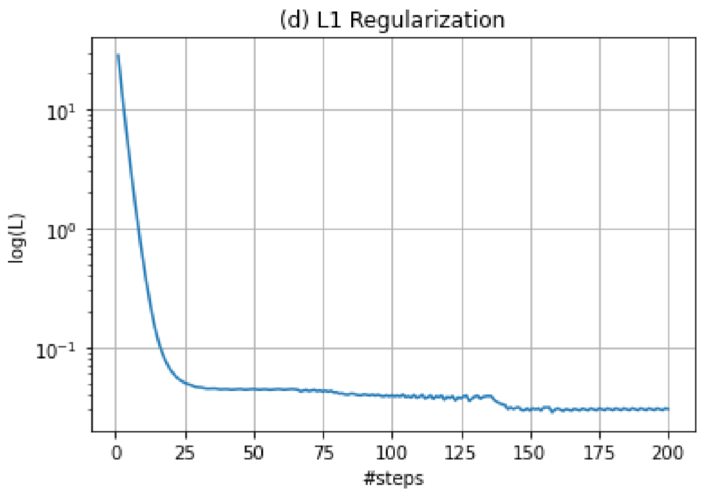


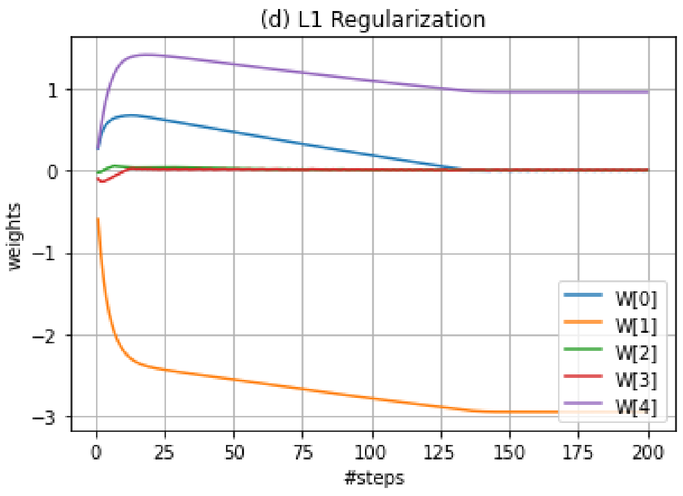


From my result, W is converging to an optimal solution, and W is converging to a sparse solution.

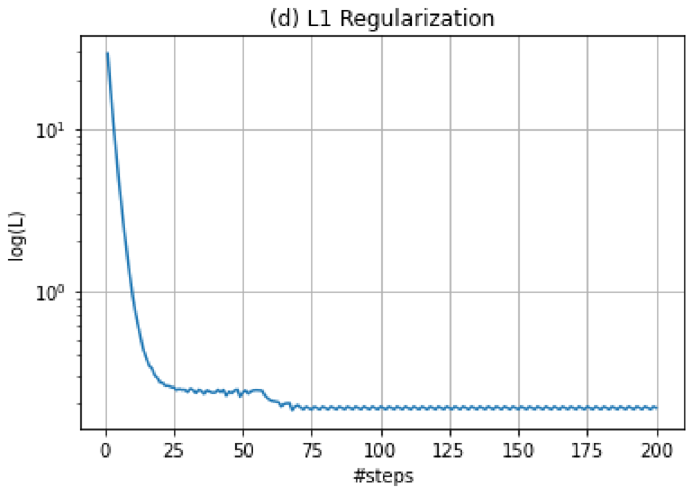
(d) (5 pts) In this problem we apply ℓ_1 regularization to induce the sparse solution. The minimization objective therefore changes to $L + \lambda||W||_1$. Please use full-batch gradient descent to minimize this objective, with $\lambda = \{0.2, 0.5, 1.0, 2.0\}$ respectively. For each case, plot the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

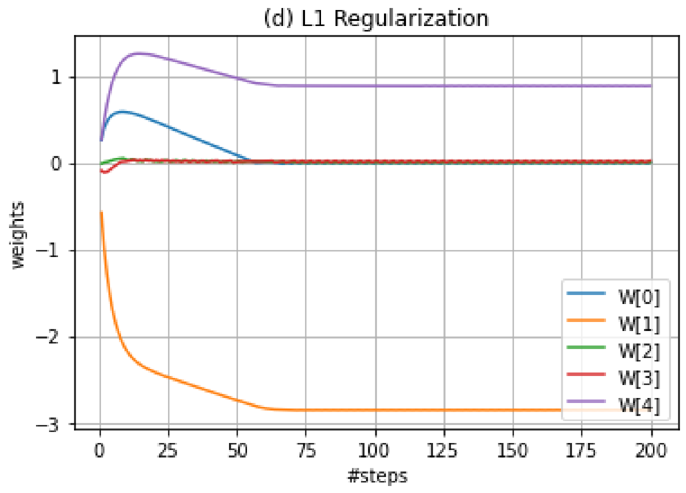
Solution for (d): First, when lambda = 0.2.



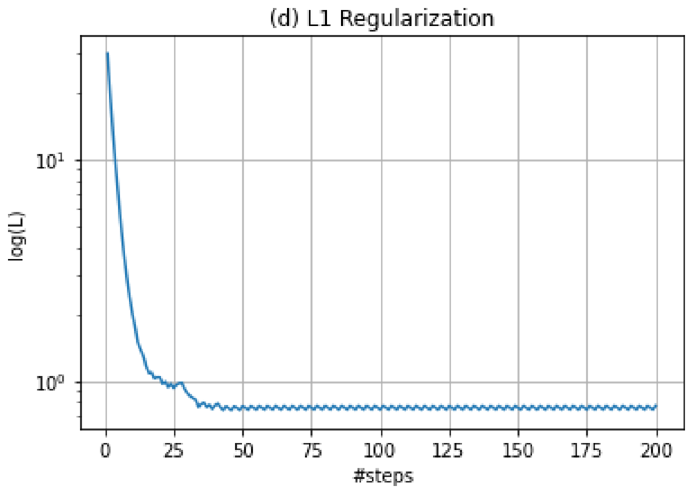


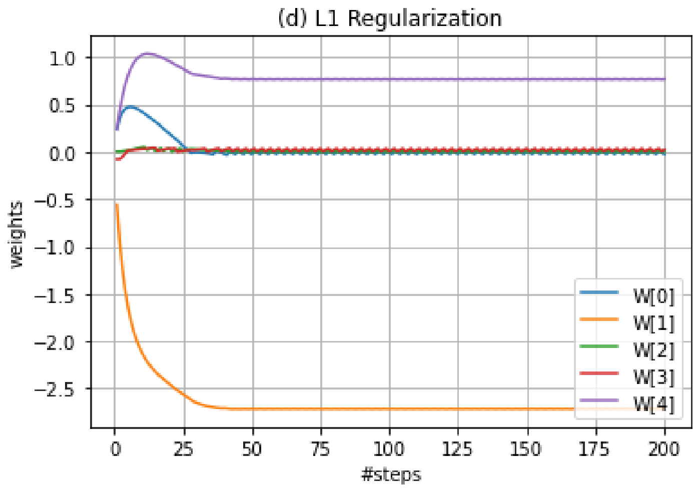
Second, when lambda = 0.5.



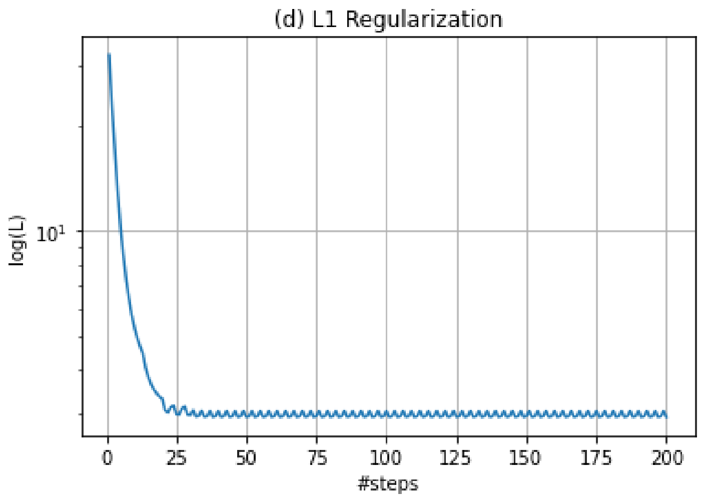


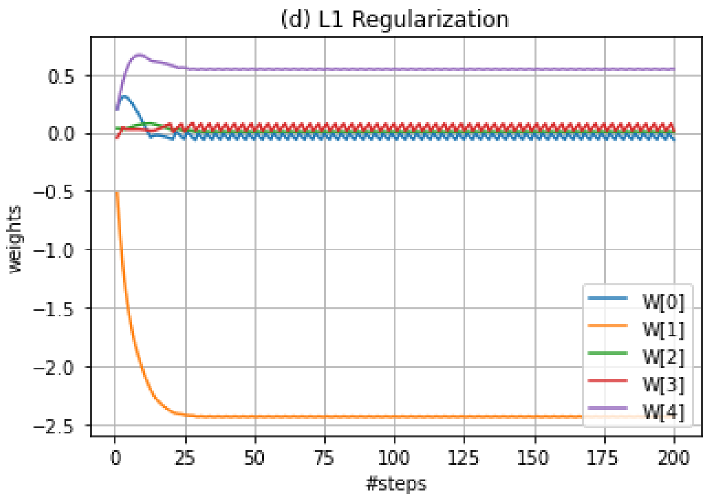
Third, when lambda = 1.0.





Fourth, when lambda = 2.0.

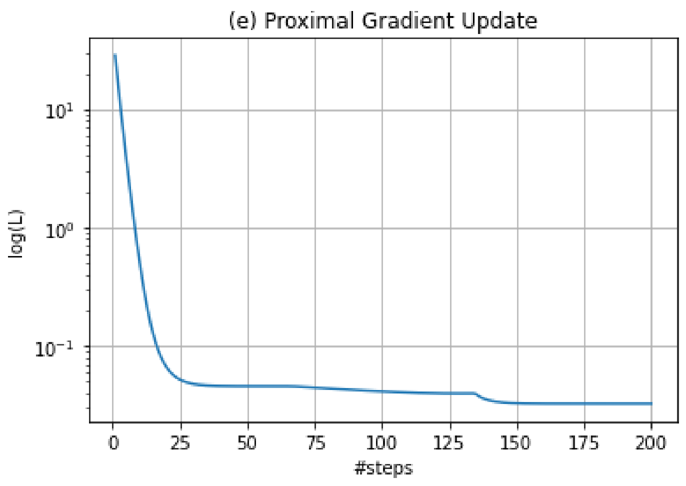


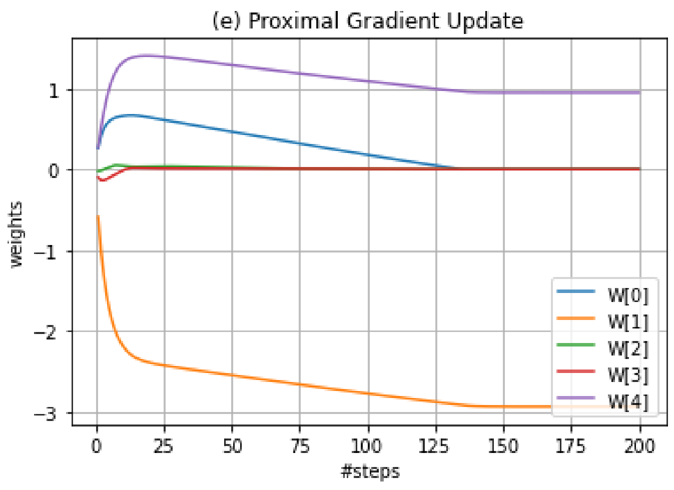


Comment: From my result, when I increase the lambda value, the model converges faster, but the log loss increases a bit, from below 0.1 to above 1 (still below 10), and the convergence performance becomes noisy. The log loss and weights fluctuate more drastically as increasing the lambda value.

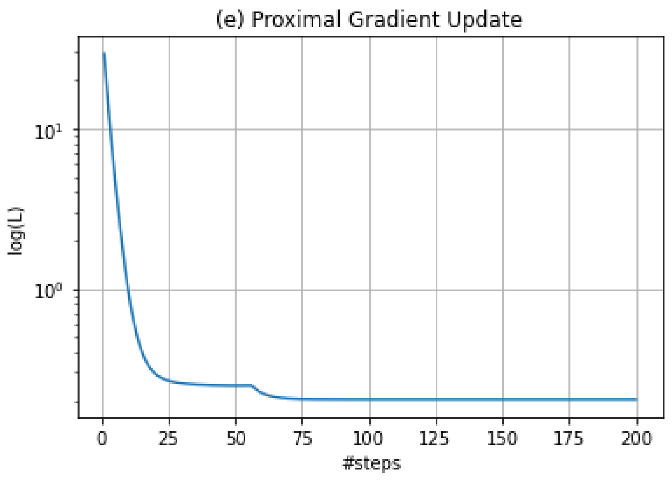
(e) (6 pts) Here we optimize the same objective as in (d), this time using **proximal gradient update.** Recall that the proximal operator of the ℓ_1 regularizer is the soft thresholding function. Set the threshold in the soft thresholding function to $\{0.004, 0.01, 0.02, 0.04\}$ respectively. Plot the value of $log(L)$ throughout the training, and use another figure 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 $\mu\lambda$)

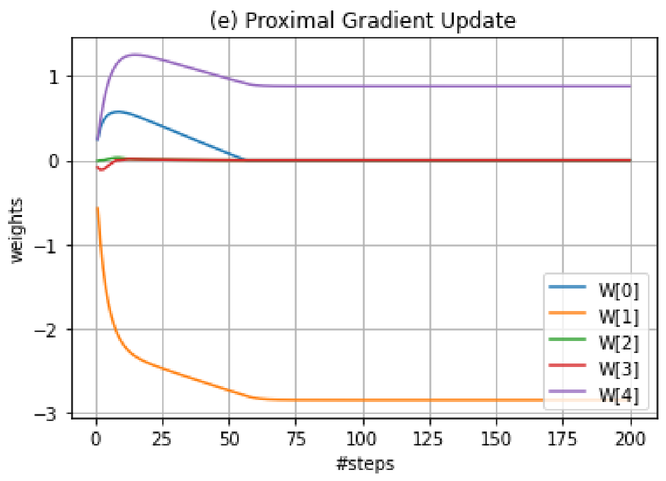
Solution for (e): First, when threshold = 0.004.



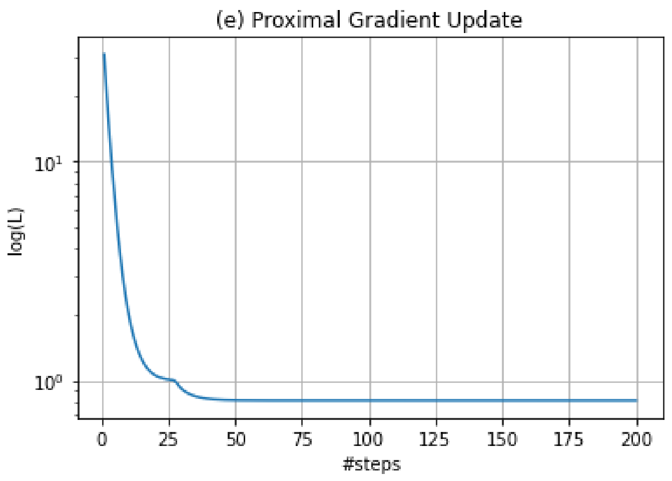


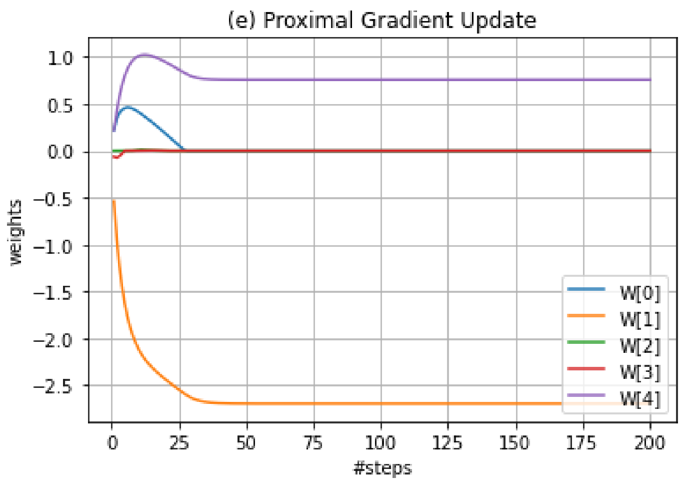
Second, when threshold = 0.01.



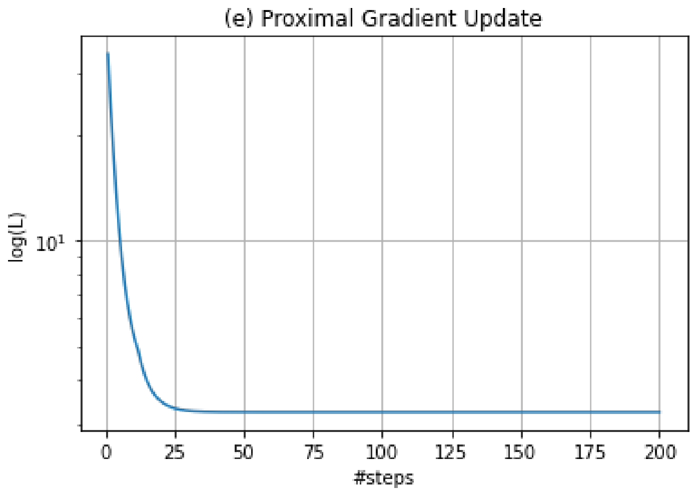


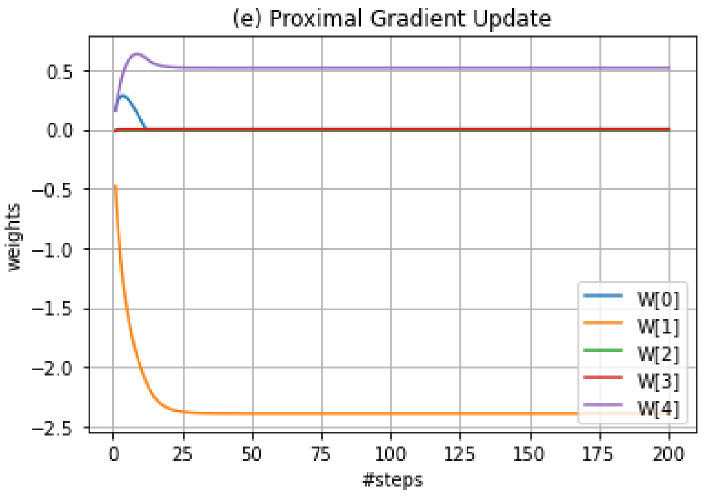
Third, when threshold = 0.02.





Fourth, when threshold = 0.04.



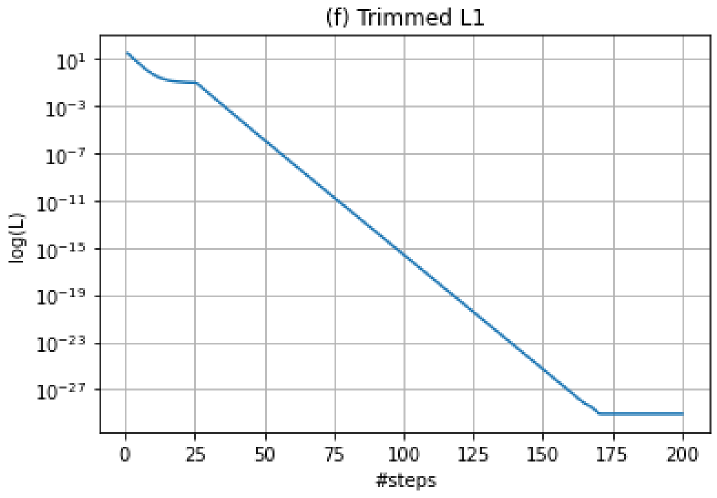


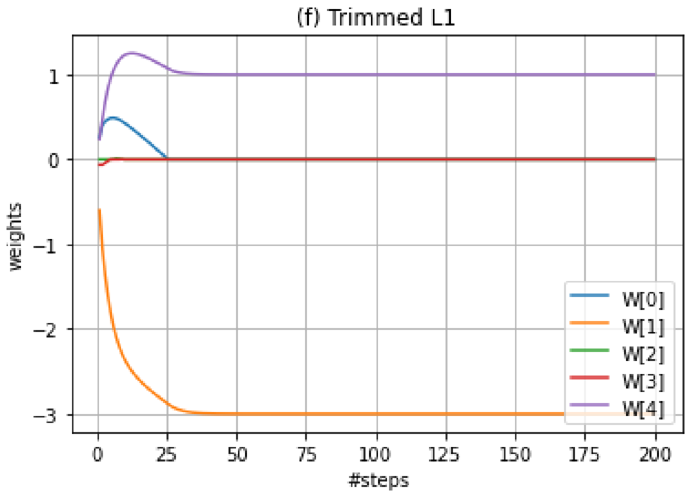
Compare the convergence performance with the results in (d), for one specific threshold corresponding to one particular lambda, the proximal gradient update with soft thresholding function in (e) solved the "noisy performance problem" of L1 regularization in (d). Here in (e) the log loss and weights 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.

(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 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.

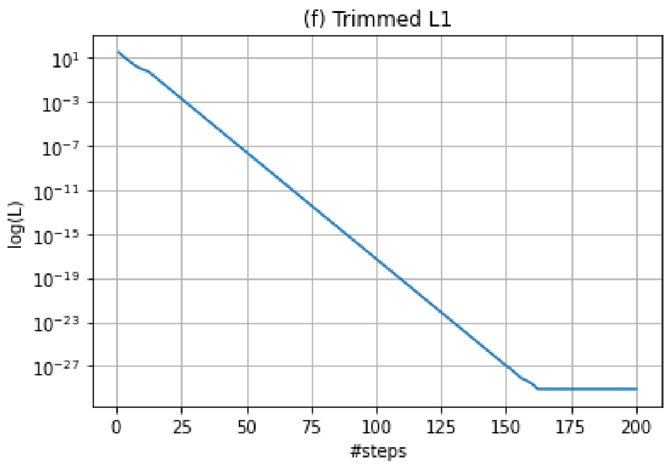
Solution for (f):

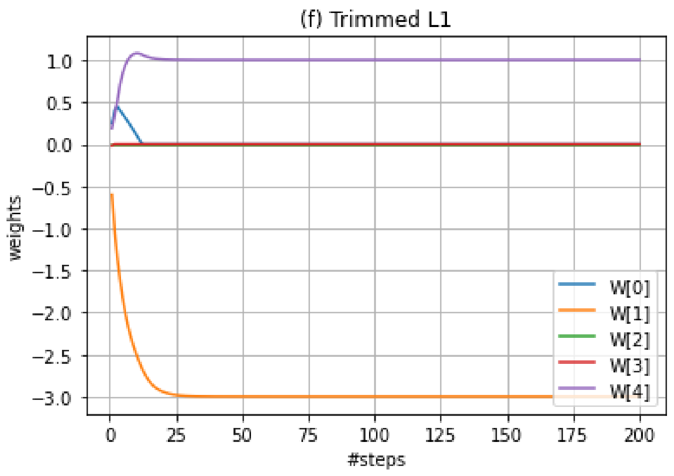
First, when lambda = 1.0, threshold = 0.02.



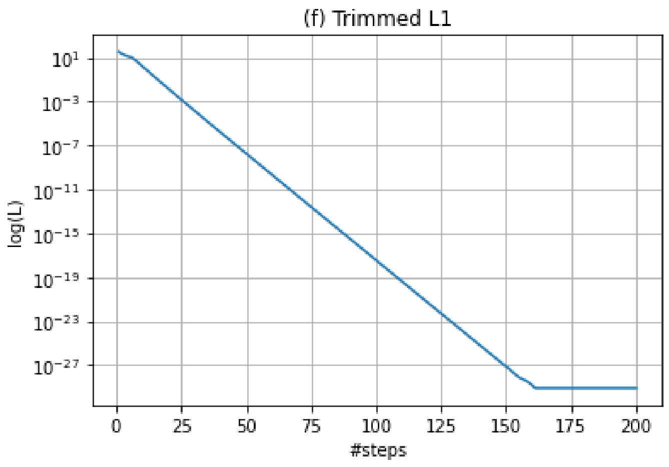


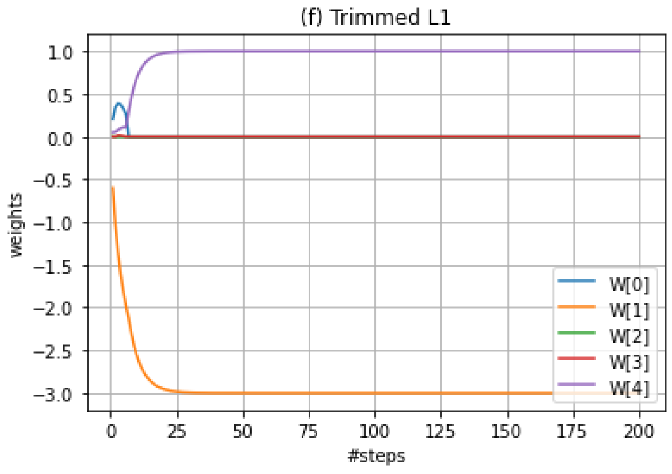
Second, when lambda = 2.0, threshold = 0.04.



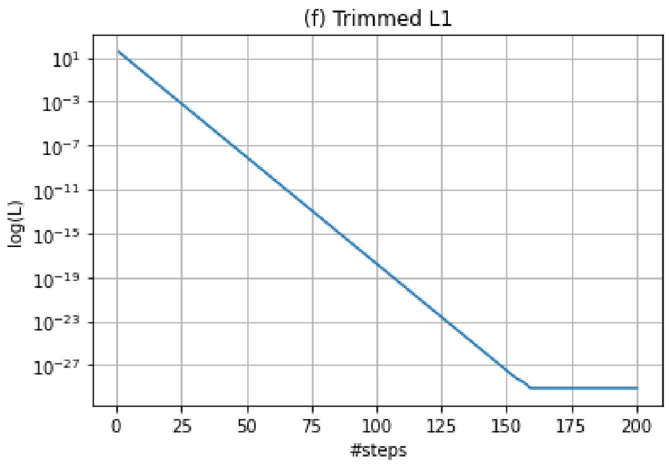


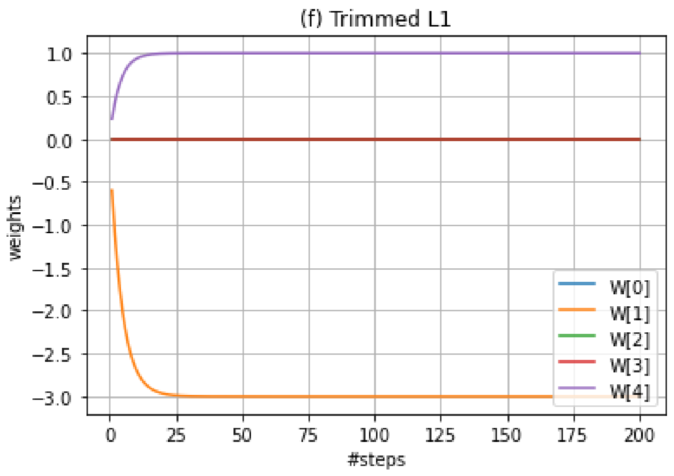
Third, when lambda = 5.0, threshold = 0.1.





Fourth, when lambda = 10.0, threshold = 0.2.





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)

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

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

Test Accuracy = 0.9151.

(b) (8 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 $q = 0.4, 0.6, 0.8$. Report the test accuracy q. (**Hint:** You need to reload the full model checkpoint before applying the prune function with a different q).

Solution for (b):

When q = 0.4, Test Accuracy = 0.8874.

When q = 0.6, Test Accuracy = 0.7226.

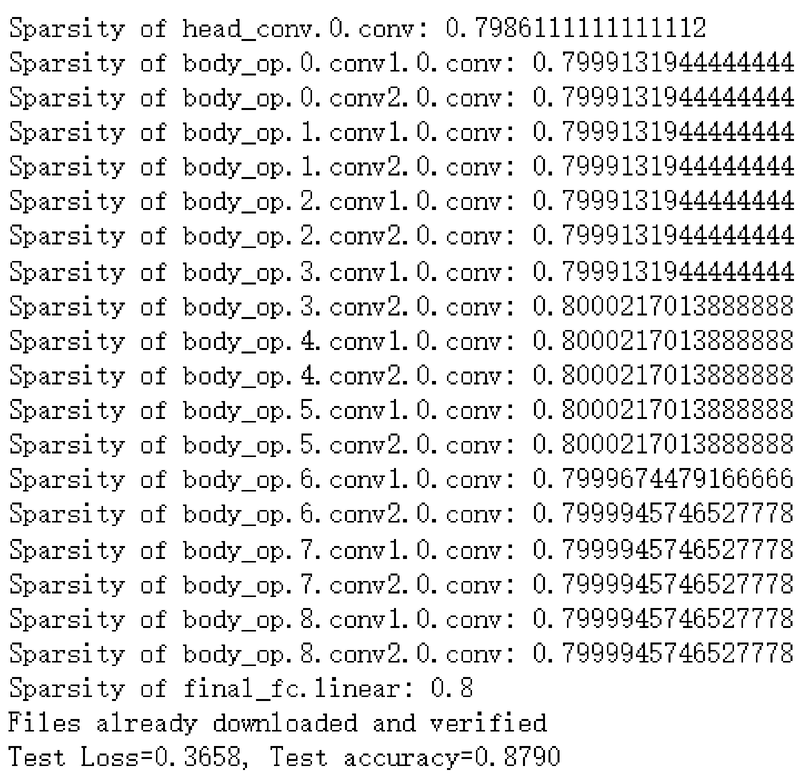
When q = 0.8, Test Accuracy = 0.1003.

(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=0.8 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.

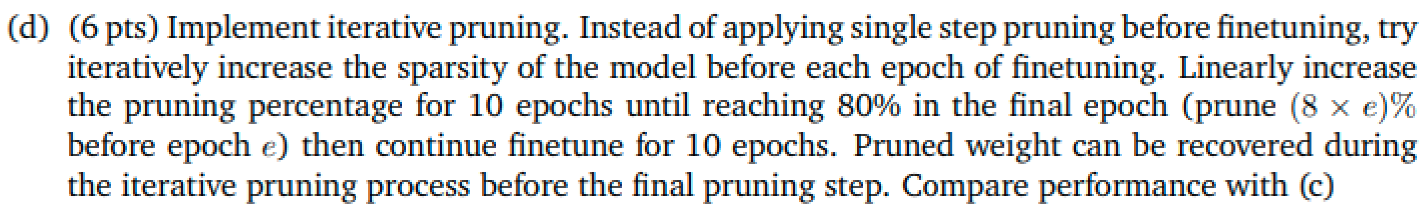
Solution for (c):

The best Test Accuracy achieved during finetuning is 0.8790.

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



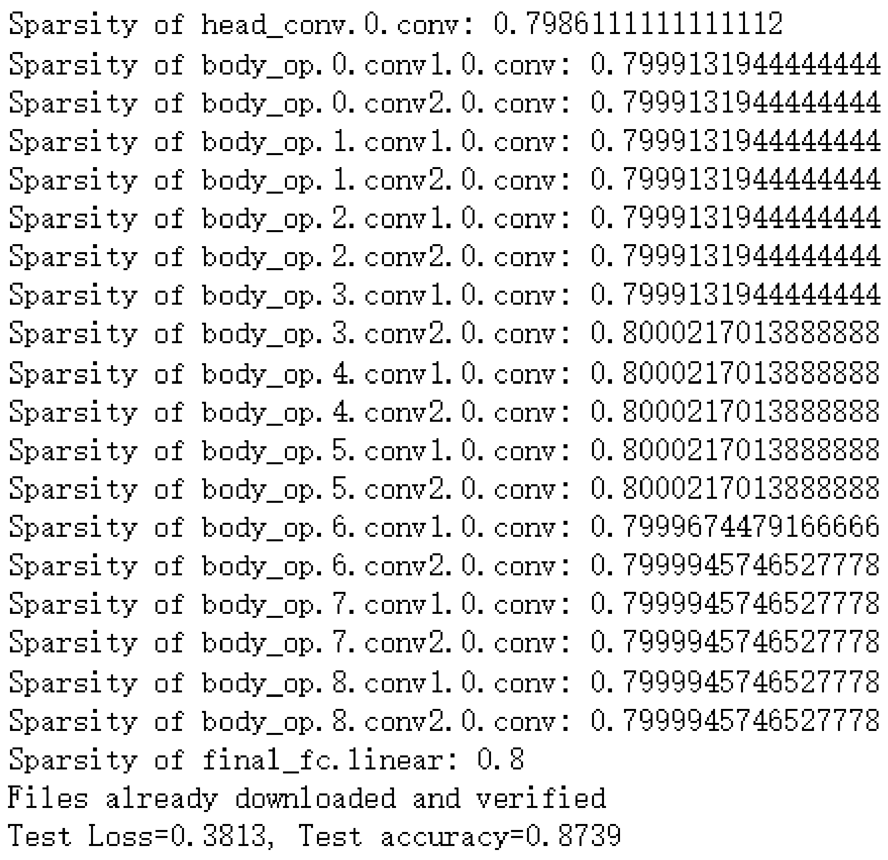
```
Sparsity of head_conv.0.conv: 0.7986111111111112
Sparsity of body_op.0.conv1.0.conv: 0.7999131944444444
Sparsity of body_op.0.conv2.0.conv: 0.7999131944444444
Sparsity of body_op.1.conv1.0.conv: 0.7999131944444444
Sparsity of body_op.1.conv2.0.conv: 0.7999131944444444
Sparsity of body_op.2.conv1.0.conv: 0.7999131944444444
Sparsity of body_op.2.conv2.0.conv: 0.7999131944444444
Sparsity of body_op.3.conv1.0.conv: 0.7999131944444444
Sparsity of body_op.3.conv2.0.conv: 0.8000217013888888
Sparsity of body_op.4.conv1.0.conv: 0.8000217013888888
Sparsity of body_op.4.conv2.0.conv: 0.8000217013888888
Sparsity of body_op.5.conv1.0.conv: 0.8000217013888888
Sparsity of body_op.5.conv2.0.conv: 0.8000217013888888
Sparsity of body_op.6.conv1.0.conv: 0.7999674479166666
Sparsity of body_op.6.conv2.0.conv: 0.7999945746527778
Sparsity of body_op.7.conv1.0.conv: 0.7999945746527778
Sparsity of body_op.7.conv2.0.conv: 0.7999945746527778
Sparsity of body_op.8.conv1.0.conv: 0.7999945746527778
Sparsity of body_op.8.conv2.0.conv: 0.7999945746527778
Sparsity of final_fc.linear: 0.8
Files already downloaded and verified
Test Loss=0.3658, Test accuracy=0.8790
```


(d) (6 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 80% in the final epoch (prune $(8 \times e)\%$ 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)

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.


```
Sparsity of head_conv.0.conv: 0.7986111111111112
Sparsity of body_op.0.conv1.0.conv: 0.7999131944444444
Sparsity of body_op.0.conv2.0.conv: 0.7999131944444444
Sparsity of body_op.1.conv1.0.conv: 0.7999131944444444
Sparsity of body_op.1.conv2.0.conv: 0.7999131944444444
Sparsity of body_op.2.conv1.0.conv: 0.7999131944444444
Sparsity of body_op.2.conv2.0.conv: 0.7999131944444444
Sparsity of body_op.3.conv1.0.conv: 0.7999131944444444
Sparsity of body_op.3.conv2.0.conv: 0.8000217013888888
Sparsity of body_op.4.conv1.0.conv: 0.8000217013888888
Sparsity of body_op.4.conv2.0.conv: 0.8000217013888888
Sparsity of body_op.5.conv1.0.conv: 0.8000217013888888
Sparsity of body_op.5.conv2.0.conv: 0.8000217013888888
Sparsity of body_op.6.conv1.0.conv: 0.7999674479166666
Sparsity of body_op.6.conv2.0.conv: 0.7999945746527778
Sparsity of body_op.7.conv1.0.conv: 0.7999945746527778
Sparsity of body_op.7.conv2.0.conv: 0.7999945746527778
Sparsity of body_op.8.conv1.0.conv: 0.7999945746527778
Sparsity of body_op.8.conv2.0.conv: 0.7999945746527778
Sparsity of final_fc.linear: 0.8
Files already downloaded and verified
Test Loss=0.3813, Test accuracy=0.8739
```

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.

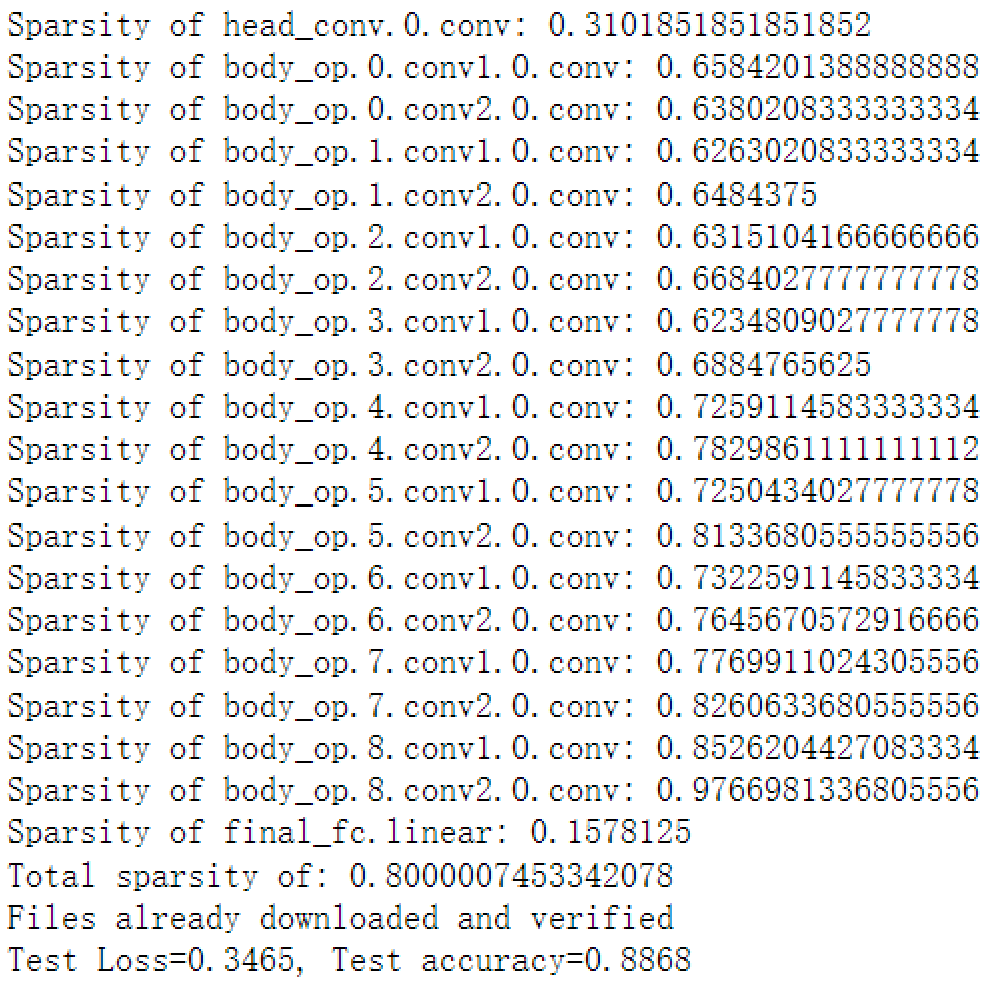
Please continue to the next page.

(e) (8 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 constraint 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 80% sparsity, report final accuracy and the percentage of zeros in each layer.

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.



```
Sparsity of head_conv.0.conv: 0.3101851851851852
Sparsity of body_op.0.conv1.0.conv: 0.6584201388888888
Sparsity of body_op.0.conv2.0.conv: 0.6380208333333334
Sparsity of body_op.1.conv1.0.conv: 0.6263020833333334
Sparsity of body_op.1.conv2.0.conv: 0.6484375
Sparsity of body_op.2.conv1.0.conv: 0.6315104166666666
Sparsity of body_op.2.conv2.0.conv: 0.6684027777777778
Sparsity of body_op.3.conv1.0.conv: 0.6234809027777778
Sparsity of body_op.3.conv2.0.conv: 0.6884765625
Sparsity of body_op.4.conv1.0.conv: 0.7259114583333334
Sparsity of body_op.4.conv2.0.conv: 0.7829861111111112
Sparsity of body_op.5.conv1.0.conv: 0.7250434027777778
Sparsity of body_op.5.conv2.0.conv: 0.8133680555555556
Sparsity of body_op.6.conv1.0.conv: 0.7322591145833334
Sparsity of body_op.6.conv2.0.conv: 0.7645670572916666
Sparsity of body_op.7.conv1.0.conv: 0.7769911024305556
Sparsity of body_op.7.conv2.0.conv: 0.8260633680555556
Sparsity of body_op.8.conv1.0.conv: 0.8526204427083334
Sparsity of body_op.8.conv2.0.conv: 0.9766981336805556
Sparsity of final_fc.linear: 0.1578125
Total sparsity of: 0.8000007453342078
Files already downloaded and verified
Test Loss=0.3465, Test accuracy=0.8868
```

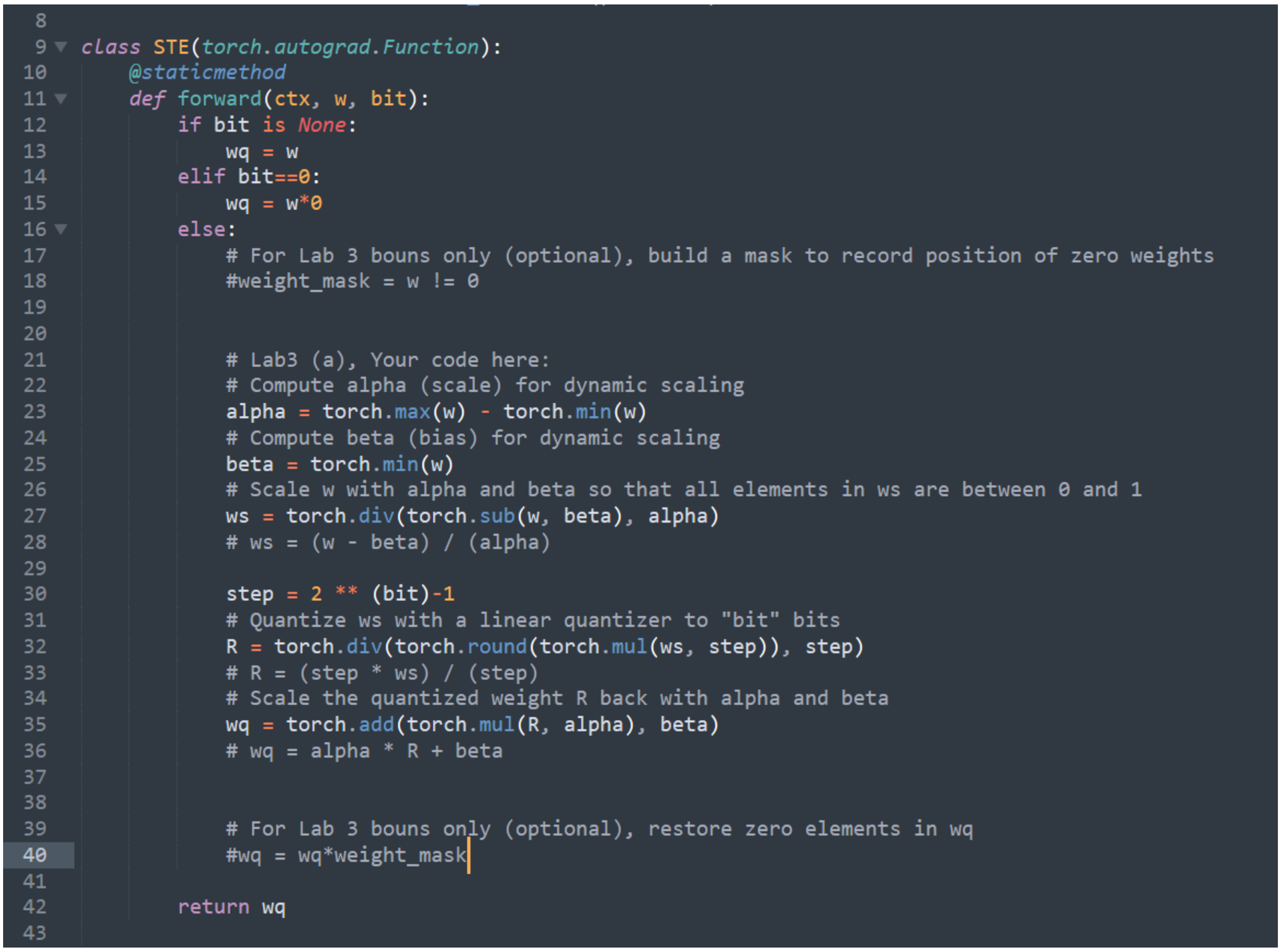
Please continue to the next page.

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

(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.

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).



```python

class STE(torch.autograd.Function):
    @staticmethod
    def forward(ctx, w, bit):
        if bit is None:
            wq = w
        elif bit==0:
            wq = w*0
        else:
            # For Lab 3 bouns only (optional), build a mask to record position of zero weights
            #weight_mask = w != 0


            # Lab3 (a), Your code here:
            # Compute alpha (scale) for dynamic scaling
            alpha = torch.max(w) - torch.min(w)
            # Compute beta (bias) for dynamic scaling
            beta = torch.min(w)
            # Scale w with alpha and beta so that all elements in ws are between 0 and 1
            ws = torch.div(torch.sub(w, beta), alpha)
            # ws = (w - beta) / (alpha)

            step = 2 ** (bit)-1
            # Quantize ws with a linear quantizer to "bit" bits
            R = torch.div(torch.round(torch.mul(ws, step)), step)
            # R = (step * ws) / (step)
            # Scale the quantized weight R back with alpha and beta
            wq = torch.add(torch.mul(R, alpha), beta)
            # wq = alpha * R + beta


            # For Lab 3 bouns only (optional), restore zero elements in wq
            #wq = wq*weight_mask

        return wq

```

Please continue to the next page.

(b) (4 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.)

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.

(c) (6 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.

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.

(d) (Bonus 5 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 80% 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. (**Hint:** Please consider zeros in the weight as being pruned away, and only apply STE on non-zero weight elements for quantization. Modification on STE implementation may be needed. For analysis you may focus on the accuracy drop after quantization.)

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.

(e) (Bonus 5 pts) Another way to have both pruning and quantization is to directly apply quantization aware training during the iterative pruning process. Starting from the pretrained ResNet-20 model, repeat the iterative pruning process of Lab 2(e) with 4-bit quantization. Report the final accuracy and the percentage of zeros in each layer. Compare your result with Lab2(e) results and the result of the previous question and discuss your observations.

Solution for (e):

Final Testing Accuracy = 0.7479.

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

```
Sparsity of head_conv.0.conv: 0.30787037037037035
Sparsity of body_op.0.conv1.0.conv: 0.6575520833333334
Sparsity of body_op.0.conv2.0.conv: 0.6397569444444444
Sparsity of body_op.1.conv1.0.conv: 0.6280381944444444
Sparsity of body_op.1.conv2.0.conv: 0.6488715277777778
Sparsity of body_op.2.conv1.0.conv: 0.6315104166666666
Sparsity of body_op.2.conv2.0.conv: 0.6684027777777778
Sparsity of body_op.3.conv1.0.conv: 0.6252170138888888
Sparsity of body_op.3.conv2.0.conv: 0.6882595486111112
Sparsity of body_op.4.conv1.0.conv: 0.7261284722222222
Sparsity of body_op.4.conv2.0.conv: 0.7820095486111112
Sparsity of body_op.5.conv1.0.conv: 0.7249348958333334
Sparsity of body_op.5.conv2.0.conv: 0.8132595486111112
Sparsity of body_op.6.conv1.0.conv: 0.7331814236111112
Sparsity of body_op.6.conv2.0.conv: 0.7640245225694444
Sparsity of body_op.7.conv1.0.conv: 0.7770724826388888
Sparsity of body_op.7.conv2.0.conv: 0.8261176215277778
Sparsity of body_op.8.conv1.0.conv: 0.852701822916666
Sparsity of body_op.8.conv2.0.conv: 0.9764539930555556
Sparsity of final_fc.linear: 0.159375
Total sparsity of: 0.8000007453342078
Files already downloaded and verified
Test Loss=0.7748, Test accuracy=0.7479
```

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.