ECE 661: Homework #4

Pruning and Fixed-point Quantization

NetID: wh162

1. True/False Questions
    a. Problem 1.1: True, weight pruning is a technique that discard unnecessary data in a model without affecting the accuracy, whereas weight optimization is how the weight is optimize for reducing the bit representative. So these two method doesn't intervene with each other.

    b. Problem 1.2: False, even though the parameters are pruned to zero, the GPU still will compute the data when feed into it.

    c. Problem 1.3: False, though pruning will reduce the number of weight, the bit representative for a weight is still in 32/64 bit, which would take Huffman encoding more time to compute.

    d. Problem 1.4: False, pruning is to remove the weights that would less affect the accuracy of the model, it is not guarantee that the value of the weight would be exact zero, so pruning is still necessary.

    e. Problem 1.5: False, as from the slides, thought soft thresholding reveals the bias problem of L1, the issue was solved by SCAD and MCP.

    f. Problem 1.6: True, refer to lec13 page 21.

    g. Problem 1.7: True, refer to lec 14 page 16.

h. Problem 1.8: True, refer to lec 14 page 27-28

i. Problem 1.9: True, refer to lec15 page 10.

j. Problem 1.10: True, refer to Lec 15 page 28.

2. Lab1: Sparse optimization of linear model
   a. Problem 2.1:

$$L = (XW - Y)^2$$

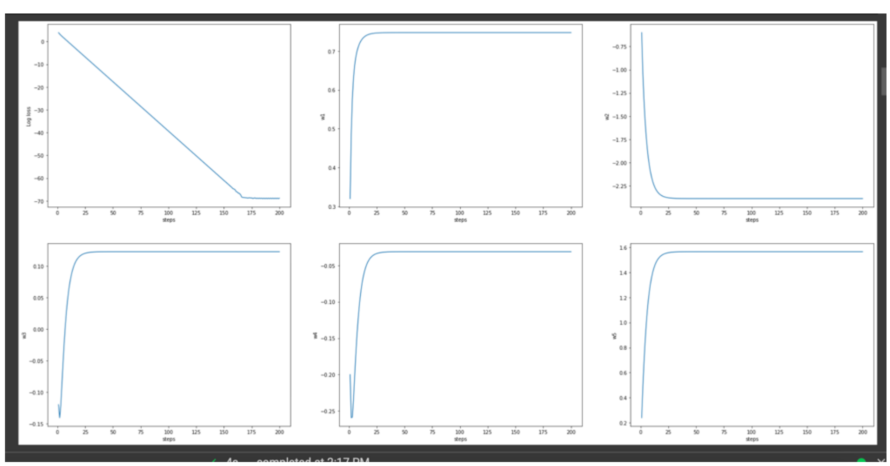
$$dL/dW = d(X^2W^2 - 2XWY + Y)^2 / dW$$

$$= 2X^2W - 2XY$$

$$= 2X \cdot (XW - Y)$$

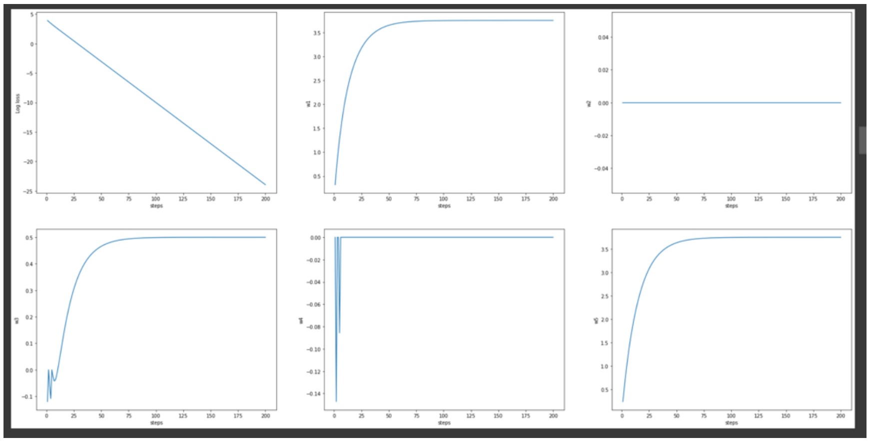
   b. Problem 2.2:

   From the plot below, we can see that W is converging to a value after several steps are executed. Yes, W is converging to a optimal solution but not a sparse solution.



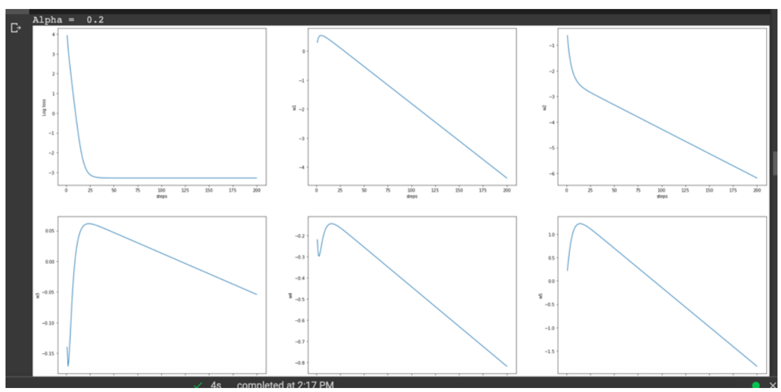
   c. Problem 2.3:

From the plot below, we can see that W is converging to a value after several steps are executed. Yes, W is converging to an optimal solution and a sparse solution.

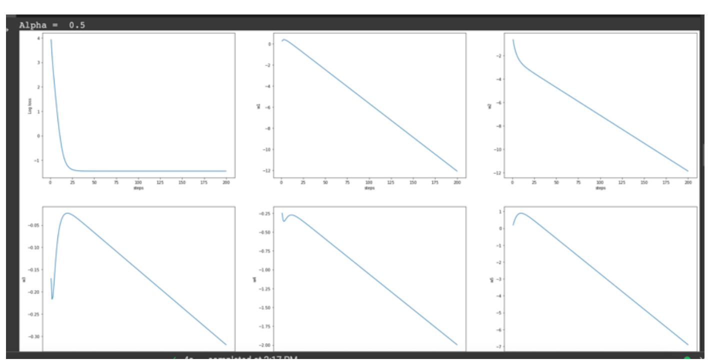


d. Problem 2.4:

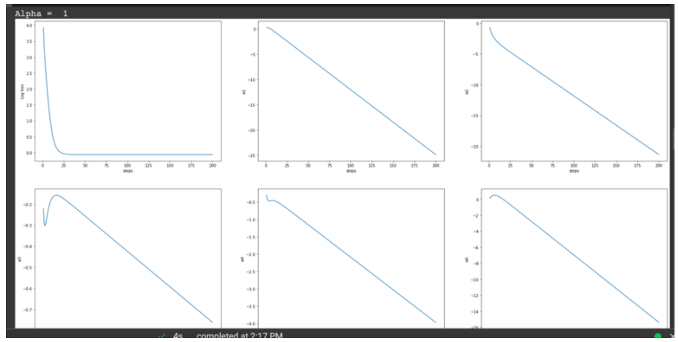
   i. Alpha = 0.2, from the plot we can see that loss converged to -3 after some steps and W hasn't reach a steady point after 200 steps



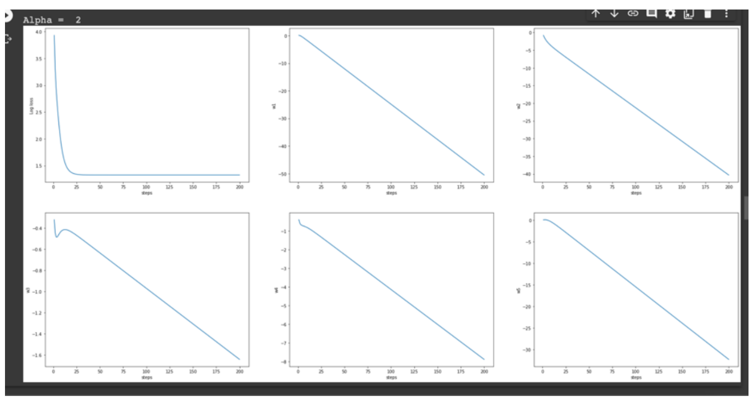
   ii. Alpha = 0.5, we can see that the loss is much lower the alpha=0.2, however, the weight is still dropping and not yet reach a steady state.



iii. Alpha = 1, loss converge at 0 and this might the best case, weight has been dropping drastically.

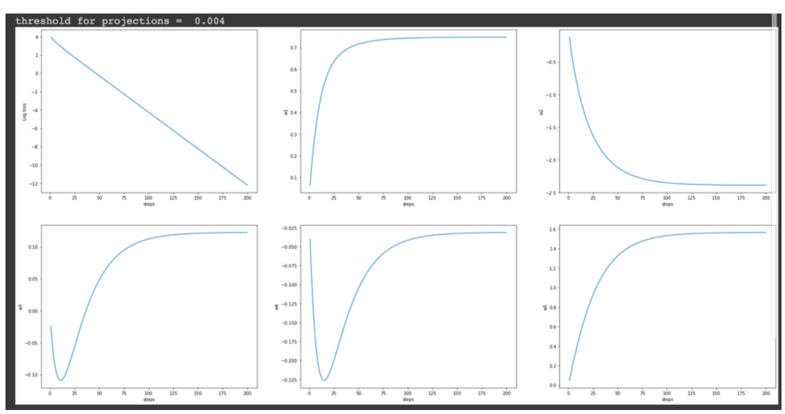


iv. Alpha = 2, loss was greater than the previous alpha values. However, the weight turns into a steady drop, unlike the previous ones, which increases and decreases, when alpha = 2, the weight dropped consistency.

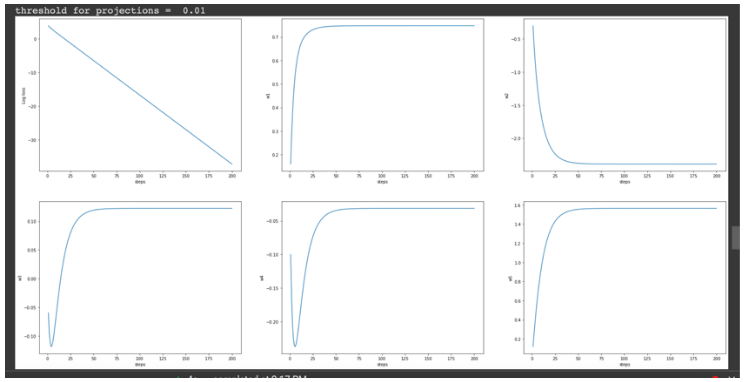


e. Problem 2.5:

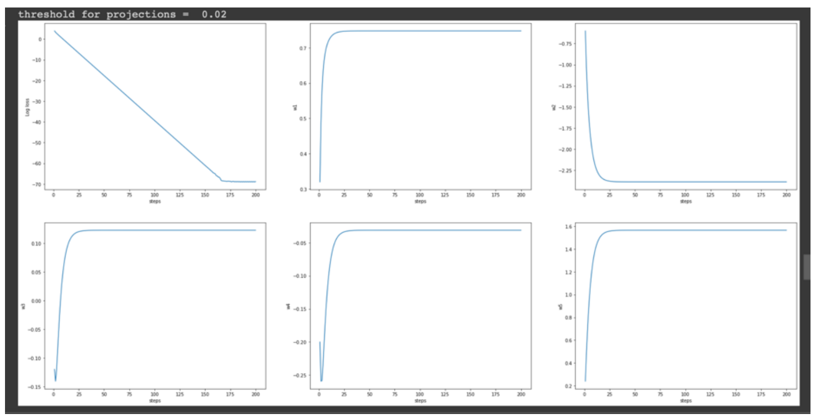
i. Threshold = 0.004, we can see that comparing with 2.4, the loss is dropping consistency, whereas the weight is converging, though some of the data have a bump.



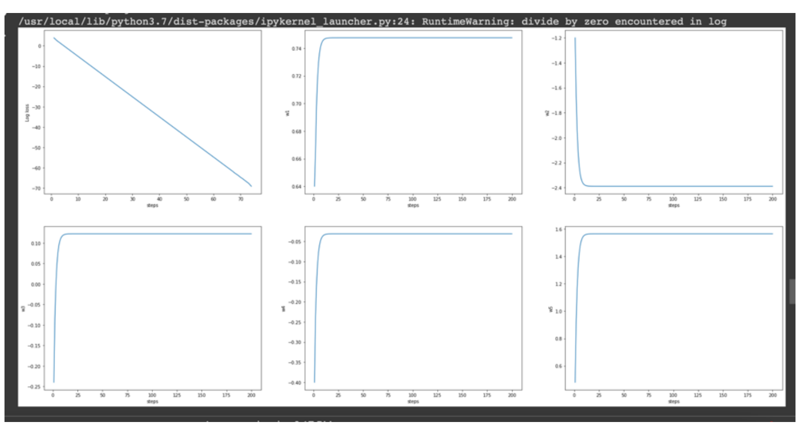
ii. Threshold = 0.01, we can see that all the weight converges when reached step 200, however the loss is not converging.



iii. Threshold = 0.02, all the weight had converged, and so does the loss, the performance was better than 2.4



iv. Threshold = 0.04, we can see that the convergence of the weight reaches their steady point much faster than the previous ones. Loss, on the other hand, doesn't converge yet after 200 steps.



f. Problem 2.6:

3. Lab2: Pruning ResNet-20 model

a. Problem 3.1:

The accuracy of the floating-point pretrained model is 0.9151

```
# Load the best weight paramters
net.load_state_dict(torch.load("pretrained_model.pt"))
test(net)

Downloading https://www.cs.toronto.edu/~kriz/cifar-10-python.tar.gz to ./data/cifar-10-python.tar.gz
100% 170498071/170498071 [00:05<00:00, 34187172.26it/s]
Extracting ./data/cifar-10-python.tar.gz to ./data
Test Loss=0.3231, Test accuracy=0.9151
```

b. Problem 3.2:

```
import numpy.ma as ma
def prune_by_percentage(layer, q=80.0):
    """
    Pruning the weight paramters by threshold.
    :param q: pruning percentile. 'q' percent of the least
    significant weight parameters will be pruned.
    """
    # Convert the weight of "layer" to numpy array
    new_layer = layer.weight.detach().cpu().numpy()

    # Compute the q-th percentile of the abs of the converted array
    temp_layer = np.absolute(new_layer)
    qth = np.percentile(temp_layer, q)#, axis = 0, keepdims=False)

    # Generate a binary mask same shape as weight to decide which element to prune
    mask = np.where(temp_layer < qth, 0, new_layer)

    # Convert mask to torch tensor and put on GPU
    new_mask = torch.tensor(mask)

    # Multiply the weight by mask to perform pruning
    layer.weight.data = new_mask.to(device)
    pass
```

c. Problem 3.3:

From the result shown in the image, the best accuracy is 88.05% after 20 epochs of training.

```
Epoch: 18
[Step=50]       Loss=0.2489     acc=0.9123      1635.6 examples/second
[Step=100]      Loss=0.2460     acc=0.9136      2534.9 examples/second
[Step=150]      Loss=0.2448     acc=0.9147      2553.7 examples/second
Test Loss=0.3675, Test acc=0.8793
Saving...

Epoch: 19
[Step=50]       Loss=0.2500     acc=0.9128      1600.0 examples/second
[Step=100]      Loss=0.2466     acc=0.9134      2570.4 examples/second
[Step=150]      Loss=0.2412     acc=0.9148      2539.9 examples/second
Test Loss=0.3642, Test acc=0.8805
Saving...
```

d. Problem 3.4:

From the testing result, the best accuracy occurs at the first two epochs with accuracy = 91.55%, which is different with the previous result, which increases gradually.

```
Epoch: 0
qth =  8
[Step=50]       Loss=0.0494     acc=0.9840      1581.2 examples/second
[Step=100]      Loss=0.0496     acc=0.9839      2545.3 examples/second
[Step=150]      Loss=0.0496     acc=0.9840      2560.3 examples/second
Test Loss=0.3231, Test acc=0.9155

Epoch: 1
qth =  16
[Step=50]       Loss=0.0495     acc=0.9832      1604.1 examples/second
[Step=100]      Loss=0.0486     acc=0.9840      2551.5 examples/second
[Step=150]      Loss=0.0489     acc=0.9839      2495.6 examples/second
Test Loss=0.3268, Test acc=0.9140

Epoch: 2
```

e. Problem 3.5:

The result shows the same which the best accuracy occurs during the first couple epochs. The total sparsity was 79%.

```
Epoch: 0
q =  8
[Step=50]       Loss=0.0473     acc=0.9853      1556.6 examples/second
[Step=100]      Loss=0.0464     acc=0.9855      2590.2 examples/second
[Step=150]      Loss=0.0471     acc=0.9854      2541.4 examples/second
Test Loss=0.3254, Test acc=0.9145

Epoch: 1
q =  16
[Step=50]       Loss=0.0471     acc=0.9858      1610.5 examples/second
[Step=100]      Loss=0.0479     acc=0.9850      2594.2 examples/second
[Step=150]      Loss=0.0479     acc=0.9845      2566.5 examples/second
Test Loss=0.3285, Test acc=0.9146
```

```
Sparsity of head_conv.0.conv: 0.3101851851851852
Sparsity of body_op.0.conv1.0.conv: 0.6575520833333334
Sparsity of body_op.0.conv2.0.conv: 0.6380208333333334
Sparsity of body_op.1.conv1.0.conv: 0.6267361111111112
Sparsity of body_op.1.conv2.0.conv: 0.6493055555555556
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: 0.6236979166666666
Sparsity of body_op.3.conv2.0.conv: 0.6883680555555556
Sparsity of body_op.4.conv1.0.conv: 0.7254774305555556
Sparsity of body_op.4.conv2.0.conv: 0.7829861111111112
Sparsity of body_op.5.conv1.0.conv: 0.7241753472222222
Sparsity of body_op.5.conv2.0.conv: 0.8132595486111112
Sparsity of body_op.6.conv1.0.conv: 0.7325303819444444
Sparsity of body_op.6.conv2.0.conv: 0.7645670572916666
Sparsity of body_op.7.conv1.0.conv: 0.7768825954861112
Sparsity of body_op.7.conv2.0.conv: 0.8259548611111112
Sparsity of body_op.8.conv1.0.conv: 0.8529730902777778
Sparsity of body_op.8.conv2.0.conv: 0.9767523871527778
Sparsity of final_fc.linear: 0.15625
Total sparsity of: 0.7999970186631685
Files already downloaded and verified
Test Loss=0.3473, Test accuracy=0.8834
```

4. Lab3: Fixed-point quantization and fine-tuning
    a. Problem 4.1:

```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 = ...

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

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

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

        return wq
```

b. Problem 4.2:

From the result, we can see that when Nbits = 6, the accuracy is the best, and it decreases as we lower the Nbit value.

```
---------------
current Nbits =  6
Files already downloaded and verified
Test Loss=0.3364, Test accuracy=0.9145
---------------
current Nbits =  5
Files already downloaded and verified
Test Loss=0.3390, Test accuracy=0.9112
---------------
current Nbits =  4
Files already downloaded and verified
Test Loss=0.3858, Test accuracy=0.8972
---------------
current Nbits =  3
Files already downloaded and verified
Test Loss=0.9874, Test accuracy=0.7662
---------------
current Nbits =  2
Files already downloaded and verified
Test Loss=9.5441, Test accuracy=0.0899
```

c. Problem 4.3:

As we can see from the following results, when the value of Nbit is greater, the accuracy is better, and so does the quantize results.

i. Nbits = 4

```
[Step=3550]     Loss=0.0547     acc=0.9815      1179.0 examples/second
[Step=3600]     Loss=0.0513     acc=0.9827      2337.2 examples/second
[Step=3650]     Loss=0.0530     acc=0.9826      2321.9 examples/second
[Step=3700]     Loss=0.0535     acc=0.9825      2494.7 examples/second
Test Loss=0.3357, Test acc=0.9127

Epoch: 19
[Step=3750]     Loss=0.0542     acc=0.9811      1153.1 examples/second
[Step=3800]     Loss=0.0538     acc=0.9815      2338.6 examples/second
[Step=3850]     Loss=0.0554     acc=0.9812      2362.4 examples/second
[Step=3900]     Loss=0.0547     acc=0.9813      2561.5 examples/second
Test Loss=0.3341, Test acc=0.9135
Saving...
Files already downloaded and verified
Test Loss=0.3341, Test accuracy=0.9135
---------------
current Nbits =  3
```

ii. Nbits = 3

```
Epoch: 15
[Step=2950]     Loss=0.0955     acc=0.9672      1183.0 examples/second
[Step=3000]     Loss=0.0943     acc=0.9661      2367.6 examples/second
[Step=3050]     Loss=0.0911     acc=0.9670      2342.5 examples/second
[Step=3100]     Loss=0.0900     acc=0.9678      2403.1 examples/second
Test Loss=0.3545, Test acc=0.9049

Epoch: 16
[Step=3150]     Loss=0.0789     acc=0.9729      1210.4 examples/second
[Step=3200]     Loss=0.0888     acc=0.9691      2321.8 examples/second
[Step=3250]     Loss=0.0889     acc=0.9689      2327.1 examples/second
[Step=3300]     Loss=0.0884     acc=0.9687      2335.9 examples/second
Test Loss=0.3513, Test acc=0.9076
Saving...

Epoch: 17
```

iii. Nbits = 2

```
Test Loss=0.4702, Test acc=0.8506

Epoch: 14
[Step=2750]     Loss=0.3100     acc=0.8900      1191.9 examples/second
[Step=2800]     Loss=0.3002     acc=0.8940      2365.7 examples/second
[Step=2850]     Loss=0.3004     acc=0.8950      2324.6 examples/second
[Step=2900]     Loss=0.3026     acc=0.8945      2337.3 examples/second
Test Loss=0.4701, Test acc=0.8499

Epoch: 15
[Step=2950]     Loss=0.3017     acc=0.8945      1191.8 examples/second
[Step=3000]     Loss=0.2951     acc=0.8962      2365.2 examples/second
[Step=3050]     Loss=0.2993     acc=0.8939      2384.7 examples/second
[Step=3100]     Loss=0.2983     acc=0.8951      2336.7 examples/second
Test Loss=0.4652, Test acc=0.8565
Saving...

Epoch: 16
[Step=3150]     Loss=0.2935     acc=0.8954      1204.8 examples/second
[Step=3200]     Loss=0.2924     acc=0.8990      2376.3 examples/second
```

d. Problem 4.4:

e. Problem 4.5: