5 Lab: LMS Algorithm (15 pts)

(a) (3pt) Directly compute the least square (Wiener) solution with the provided dataset. What is the optimal weight W^*? What is the MSE loss of the whole dataset when the weight is set to W^*?

Solution: The optimal weight W_star is

[[0.99769073]

[-2.00001451]

[2.99870453]]

Given W_star, the MSE loss of the whole dataset is (6.145138742034196e-05).

(b) (4pt) Now consider that you can only train with 1 pair of data point and target each time. In such case, the LMS algorithm should be used to find the optimal weight. Please initialize the weight vector as $W^0 = [0, 0, 0]^T$, and update the weight with the LMS algorithm. After each *epoch* (every time you go through all the training data and loop back to the beginning), compute and record the MSE loss of the current weight on the whole dataset. Run LMS for 20 epochs with learning rate $r = 0.01$, report the weight you get in the end and plot the MSE loss *in log scale* vs. Epochs.

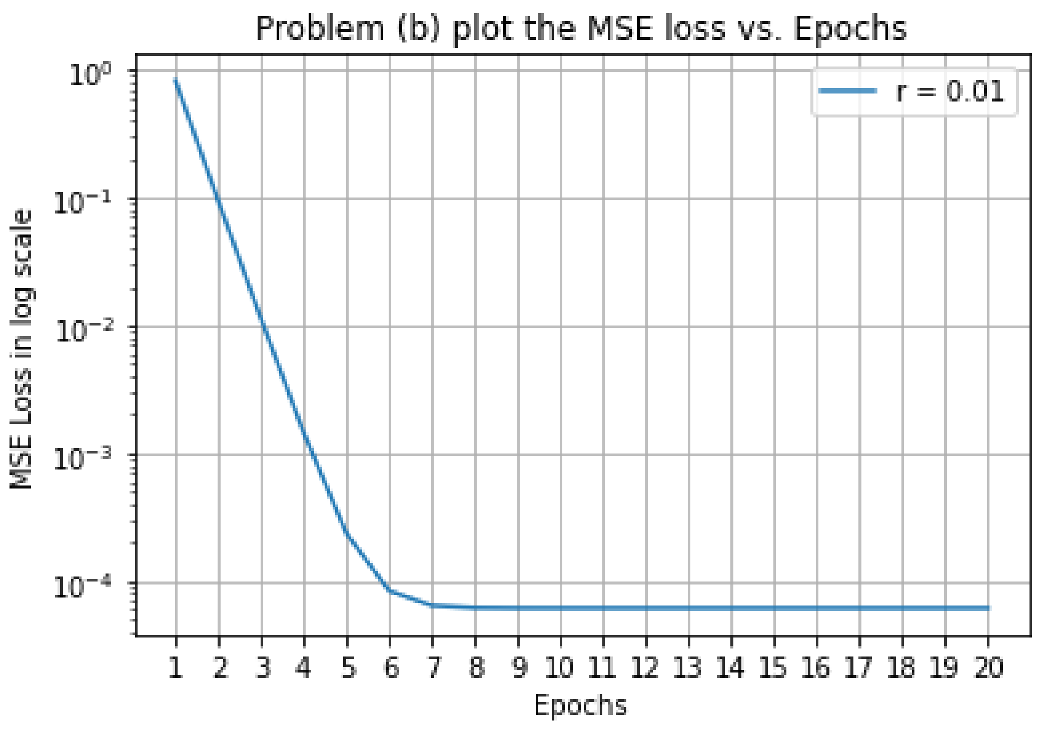
Solution: The weight I get after 20 epochs (W_20_Epochs in code) is

[[0.99761689]

[-1.99979502]

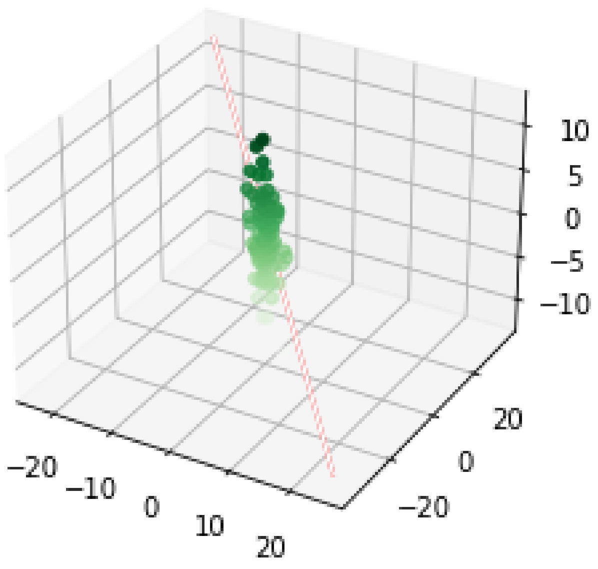
[2.99898621]]

Plot the MSE loss in log scale vs. Epochs



(c) (3pt) Scatter plot the points (x_{1k}, x_{2k}, d_k) for all 100 data-target pairs in a 3D figure[b], and plot the lines corresponding to the linear models you got in (a) and (b) respectively in the same figure. Observe if the linear models fit the data well.

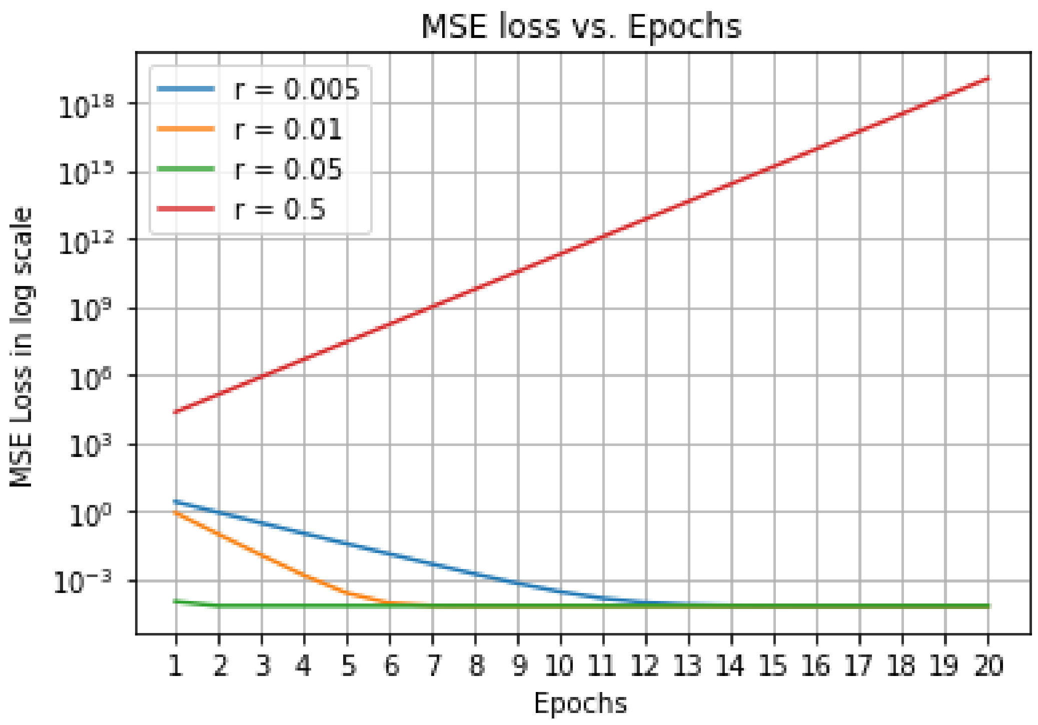
Solution: The 3D scatter plot along with two linear models is shown below.



In this 3D figure, a group of green points belong to all 100 data-target pairs, the linear model I got in (a) (W_star) is the red line, and the linear model I got in (b) (W_20_Epochs) is the white line. It could be seen that the two linear models are very close to each other, and both models fit the 100 data-target pairs relatively well.

(d) (5pt) Learning rate r is an important hyperparameter for the LMS algorithm, as well as for CNN optimization. Here, try repeat the process in (b) with r set to 0.005, 0.05 and 0.5 respectively. Together with the result you got in (b), plot the MSE losses of the 4 sets of experiments in log scale vs. Epochs in one figure. Then try further enlarge the learning rate to $r = 1$ and observe how the MSE changes. Base on these observations, comment on how learning rate affects the speed and quality of the learning process. (Note: The learning rate tuning for the CNN optimization will be introduced in Lecture 7.)

Solution: Plot the MSE losses of the 5 sets of experiments in log scale vs. Epochs in one figure.



Enlarge the learning rate to r = 1 and observe how the MSE changes. Shown below are MSE values after each Epoch.

MSE after Epoch 1: 3.6798545341445733e+31

MSE after Epoch 2: 5.402768778759977e+60

MSE after Epoch 3: 7.932354446216017e+89

MSE after Epoch 4: 1.164629648926877e+119

MSE after Epoch 5: 1.7099112607185237e+148

MSE after Epoch 6: 2.5104946643133094e+177

MSE after Epoch 7: 3.685912599287265e+206

MSE after Epoch 8: 5.411663240200822e+235

MSE after Epoch 9: 7.945413309855397e+264

```
MSE after Epoch 10: 1.1665469535403863e+294
MSE after Epoch 11: inf
MSE after Epoch 12: inf
MSE after Epoch 13: inf
MSE after Epoch 14: inf
MSE after Epoch 15: inf
MSE after Epoch 16: inf
MSE after Epoch 17: inf
MSE after Epoch 18: inf
MSE after Epoch 19: inf
MSE after Epoch 20: inf
```

Given learning rate r = 1 and its MSE losses shown above, the MSE loss gradually becomes so large that finally leads to numerical difficulty, making the LMS model training fail.

Comment: Based on all 6 observations, if the learning rate is too small, it would take a very long time for model training to achieve convergence (finding the optimal solution), although the model accuracy could be relatively good.

Although increasing the learning rate could speed up model training, if the learning rate is too large, then the model training might fail (training loss becomes so large that the model fails to converge), which means the model accuracy would be really bad.

Therefore, it is very important to maintain the trade-off between a small learning rate and a large learning rate, through either empirical knowledge or many experiments.

6 Lab: Simple NN (40 pts)

Lab 2 (50 points)

In the notebook, first run through the first two code blocks, then follow the instructions in the following questions to complete each code block and acquire the answers.

(a) (10pt) Complete code block 3 for defining the adapted SimpleNN model. Note that customized `CONV` and `FC` classes are provided in code block 2 to replace the nn.Conv2d and nn.Linear classes in PyTorch respectively. The usage of the customized classes are exactly the same as their PyTorch counterparts, the only difference is that in the customized class the input and output feature maps of the layer will be stored in `self.input` and `self.output` respectively after the forward pass, which will be helpful in question (b). After the code is completed, run through the block and make sure the model forward pass in the end throw no errors. Please copy your code of the completed `LeNet5` class into the report PDF.

Solution: My code of the completed LeNet5 class is shown below.

```
# Create the neural network module: LeNet-5
class SimpleNN(nn.Module):
    def __init__(self):
        super(SimpleNN, self).__init__()
        # Layer definition
        self.conv1 = CONV(in_channels = 3,
                          out_channels = 32,
                          kernel_size = 5,
                          stride = 1,
                          padding = 2)        #Your code here
        self.conv2 = CONV(in_channels = 32,
                          out_channels = 32,
                          kernel_size = 5,
                          stride = 1,
                          padding = 2)        #Your code here
        self.conv3 = CONV(in_channels = 32,
                          out_channels = 64,
                          kernel_size = 5,
                          stride = 1,
                          padding = 2)        #Your code here
        self.fc1    = FC(in_features = (64*3*3), out_features = 64)    #Your code here
        self.fc2    = FC(in_features = 64, out_features = 10)          #Your code here

    def forward(self, x):
        # Forward pass computation
        # Conv 1
        #Your code here
        #print("dim of x = " + str(x.shape))
        out = self.conv1(x)
        out = F.relu(out)
        #print("dim after CONV1 = " + str(out.shape))
        # MaxPool
        #Your code here
```

```
out = F.max_pool2d(out, kernel_size = 3, stride = 2)
#print("dim after MaxPool1 = " + str(out.shape))
# Conv 2
#Your code here
out = self.conv2(out)
out = F.relu(out)
#print("dim after CONV2 = " + str(out.shape))
# MaxPool
#Your code here
out = F.max_pool2d(out, kernel_size = 3, stride = 2)
#print("dim after MaxPool2 = " + str(out.shape))
# Conv 3
#Your code here
out = self.conv3(out)
out = F.relu(out)
#print("dim after CONV3 = " + str(out.shape))
# MaxPool
#Your code here
out = F.max_pool2d(out, kernel_size = 3, stride = 2)
#print("dim after MaxPool3 = " + str(out.shape))
# Flatten
#Your code here
out = out.view(out.size(0), -1)
#print("dim after Flatten = " + str(out.shape))
# FC 1
#Your code here
out = self.fc1(out)
out = F.relu(out)
#print("dim after FC1 = " + str(out.shape))
# FC 2
#Your code here
out = self.fc2(out)
out = F.relu(out)
#print("dim after FC2 = " + str(out.shape))
# return out
return out
```

(b) (30pt) Complete the for-loop in code block 4 to print the shape of the input feature map, output feature map and the weight tensor of the 5 convolutional and fully-connected layers when processing a single input. Then compute the number of parameters and the number of MACs in each layer with the shapes you get. In your report, use your results to fill in the blanks in Table 2.

Solution: Table 2 is completed below.

Layer	Input shape	Output shape	Weight shape	# Param	# MAC
Conv 1	(1, 3, 32, 32)	(1, 32, 32, 32)	(32, 3, 5, 5)	2400	2457600
Conv 2	(1, 32, 15, 15)	(1, 32, 15, 15)	(32, 32, 5, 5)	25600	5760000
Conv 3	(1, 32, 7, 7)	(1, 64, 7, 7)	(64, 32, 5, 5)	51200	2508800
FC1	(1, 576)	(1, 64)	(64, 576)	36864	36864
FC2	(1, 64)	(1, 10)	(10, 64)	640	640

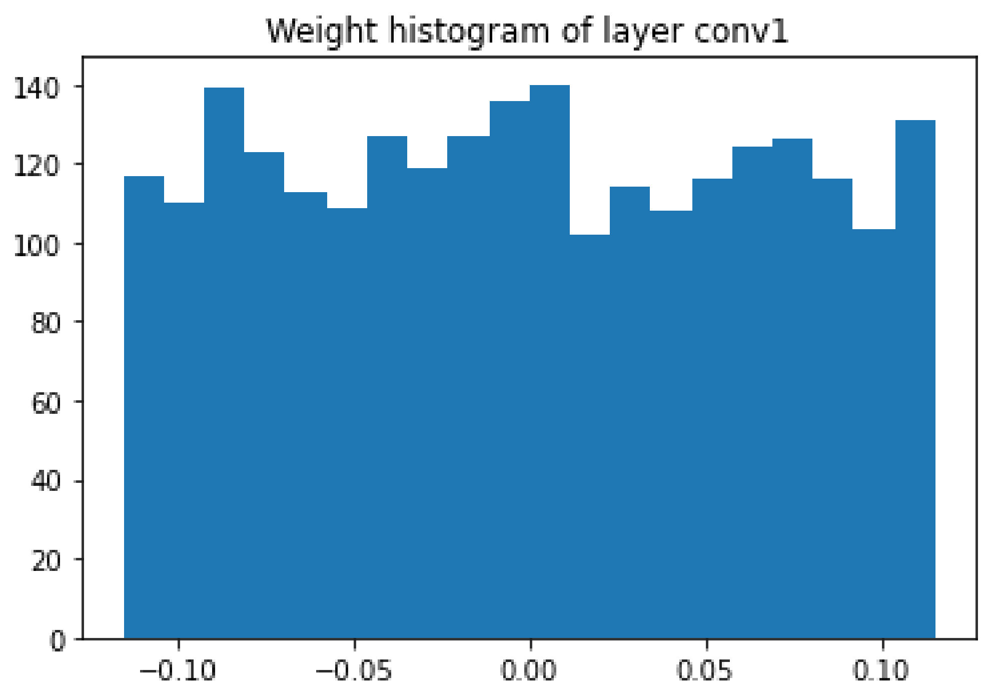
Table 2: Results of Lab 2(b).

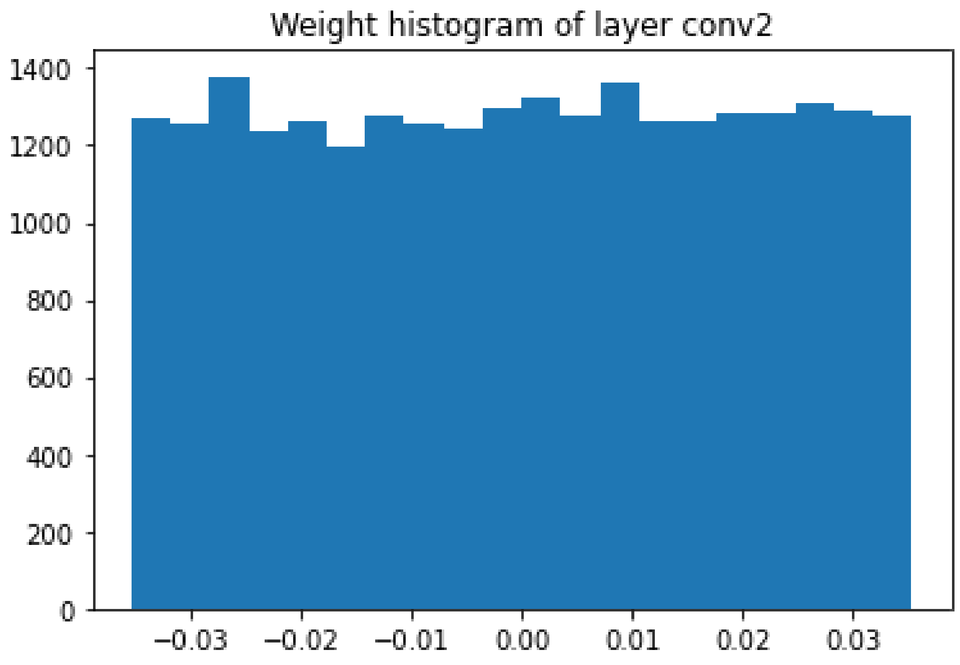
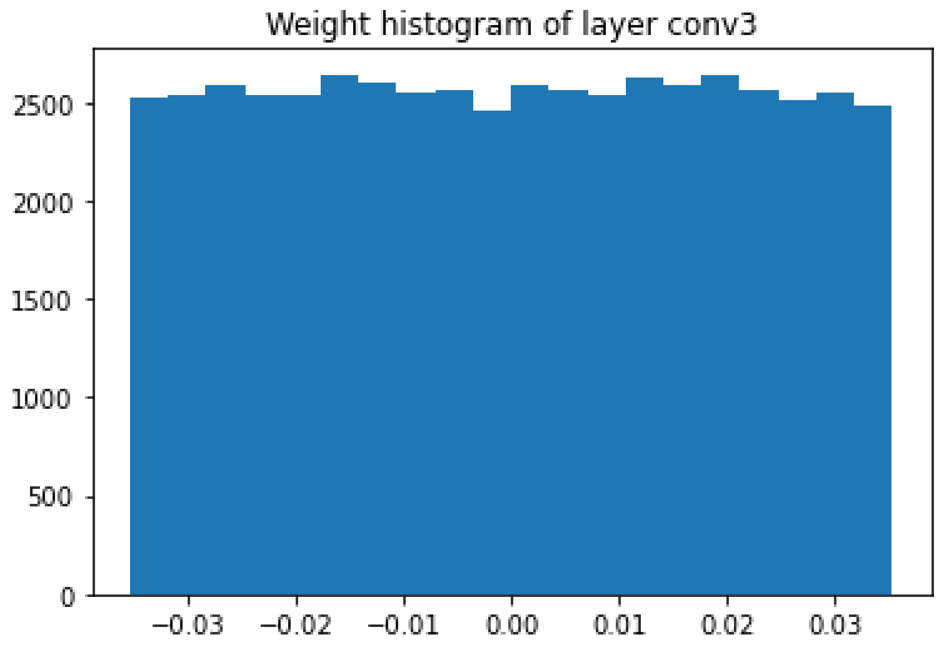
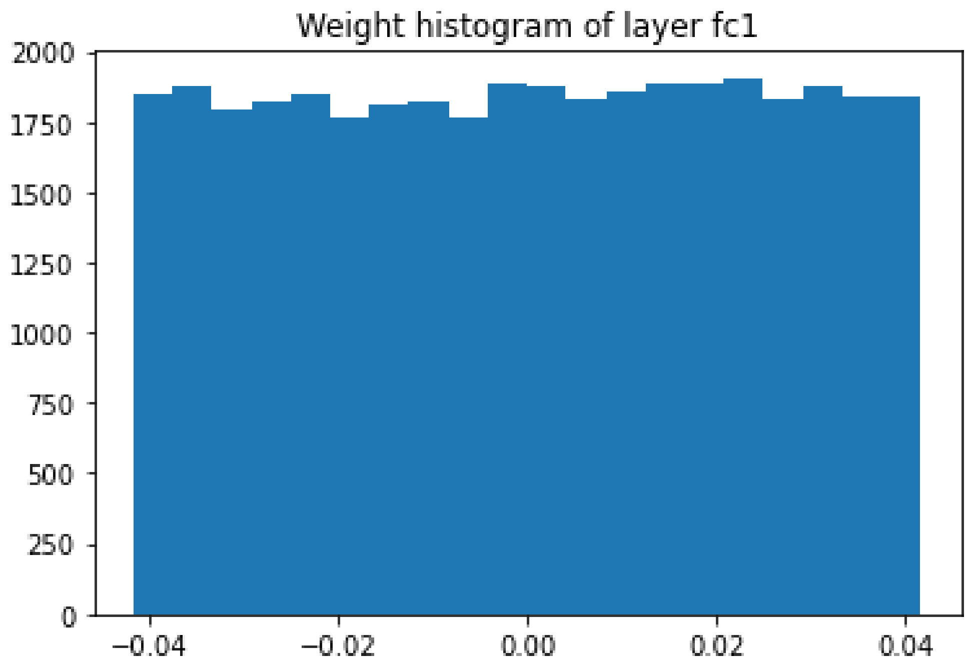
Lab 3 (Bonus 10 points)

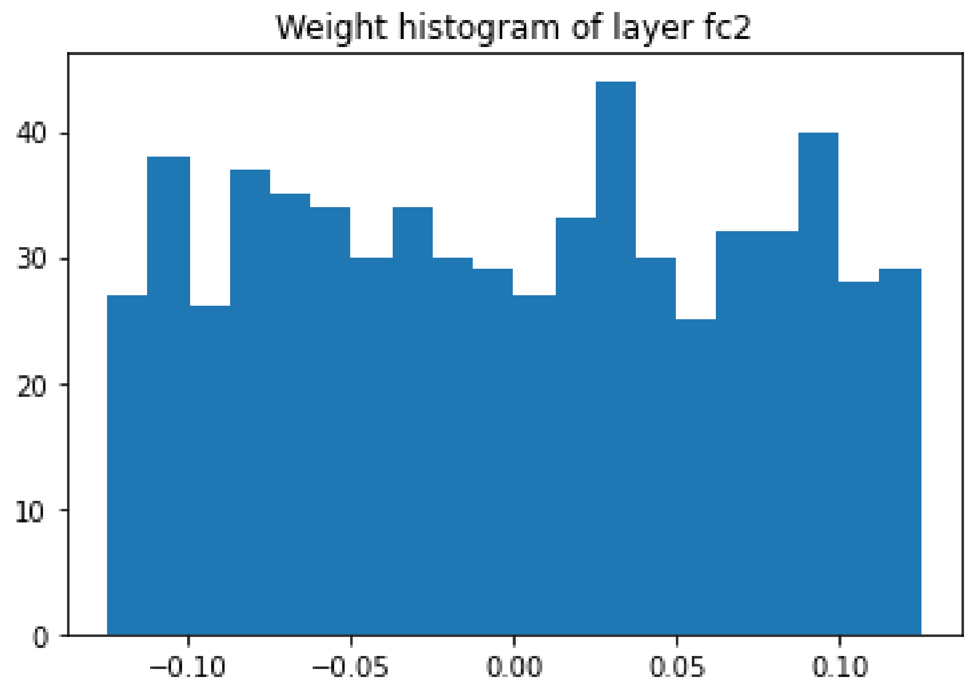
Please first finish all the required codes in Lab 2, then proceed to code block 5 of the notebook file.

(a) (2pt) Complete the for-loop in code block 5 to plot the histogram of weight elements in each one of the 5 convolutional and fully-connected layers.

Solution: Histograms of weight elements in each one of the 5 convolutional and fully-connected layers are shown below.

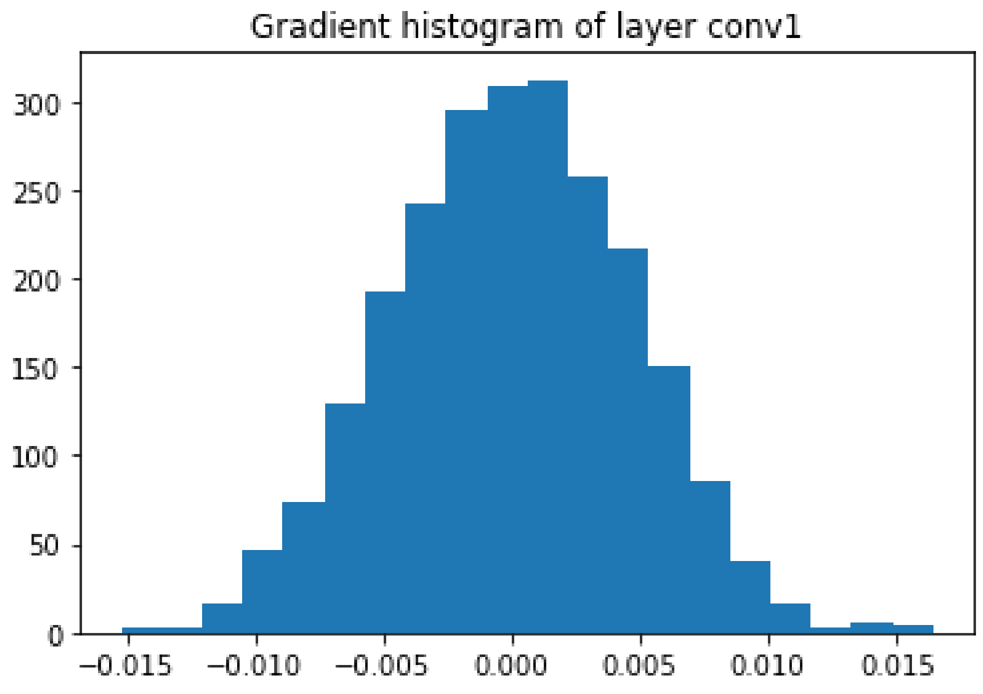


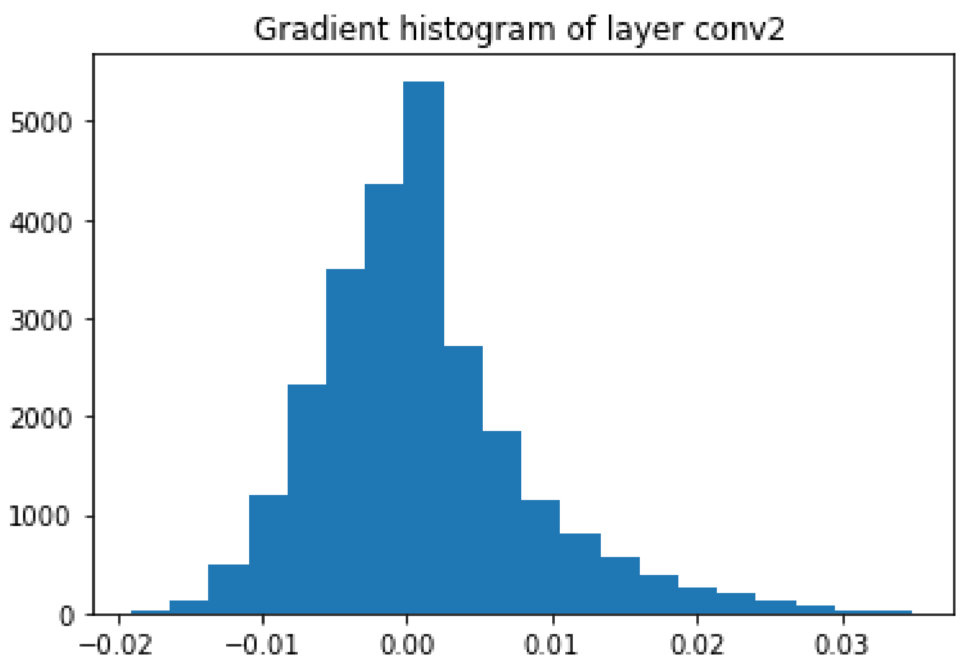

Weight histogram of layer conv2
1400
1200
1000
800
600
400
200
0
−0.03
−0.02
−0.01
0.00
0.01
0.02
0.03

Weight histogram of layer conv3
2500
2000
1500
1000
500
0
−0.03
−0.02
−0.01
0.00
0.01
0.02
0.03

Weight histogram of layer fc1
2000
1750
1500
1250
1000
750
500
250
0
−0.04
−0.02
0.00
0.02
0.04

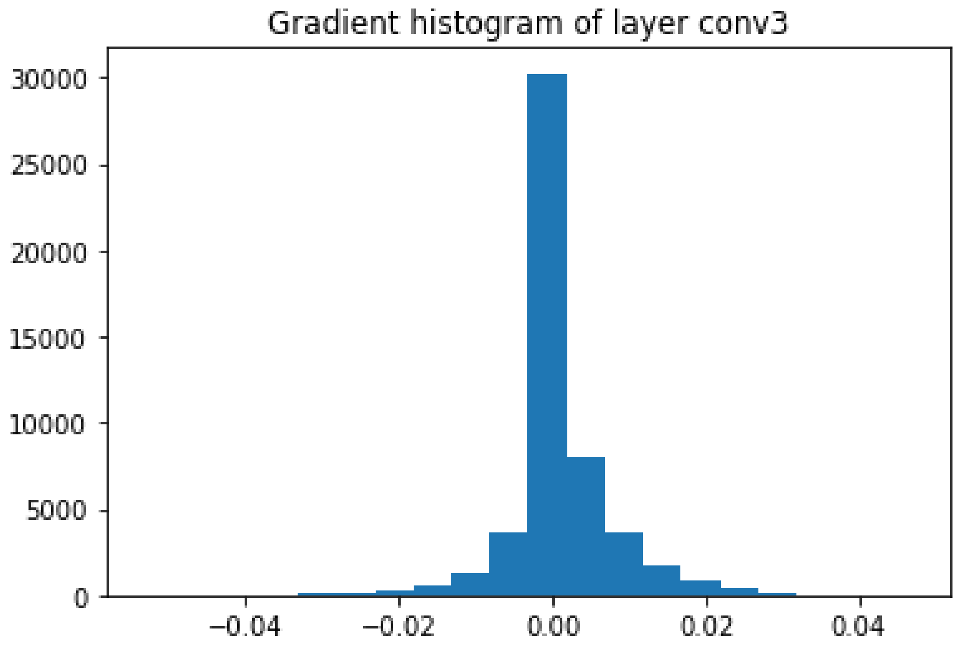


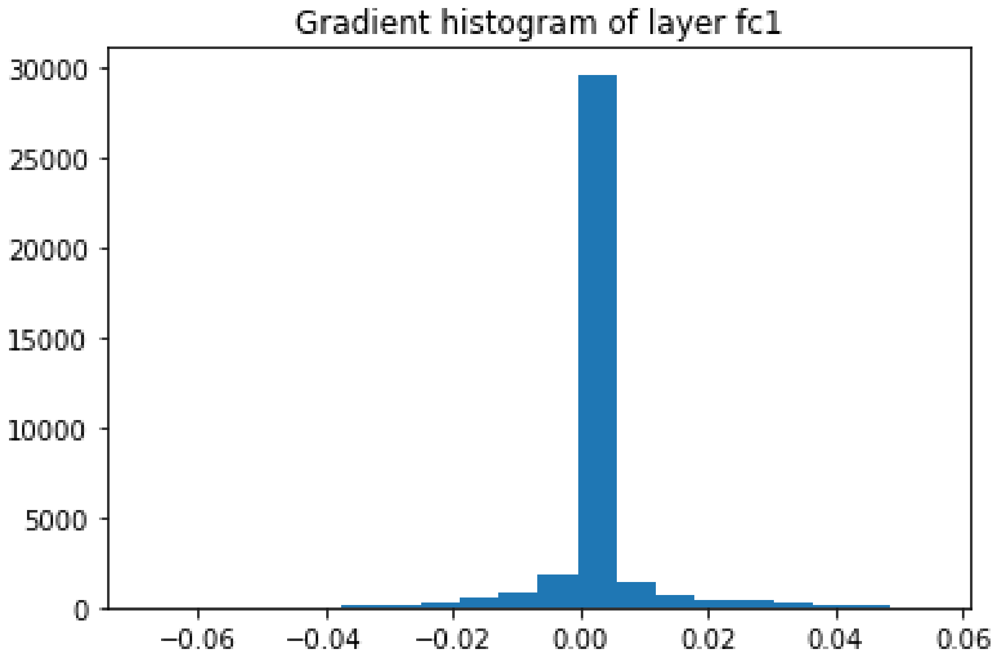
(b) (3pt) In code block 6, complete the code for backward pass, then complete the for-loop to plot the histogram of weight elements' gradients in each one of the 5 convolutional and fully-connected layers.

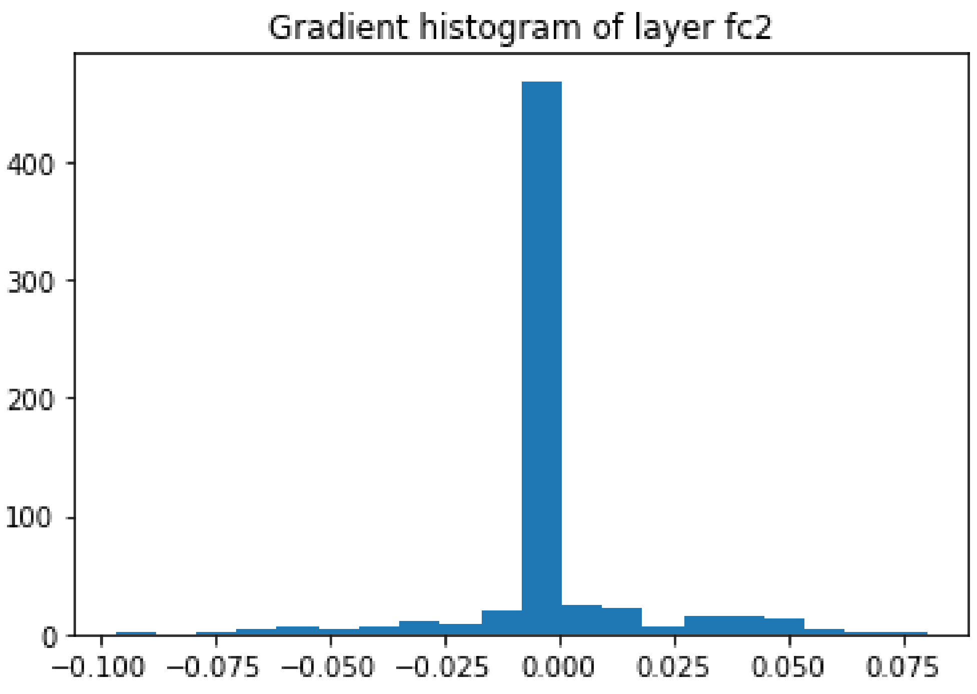
Solution: Histograms of weight elements' gradients in each one of the 5 convolutional and fully-connected layers are shown below.




Gradient histogram of layer conv2
5000
4000
3000
2000
1000
0
−0.02
−0.01
0.00
0.01
0.02
0.03

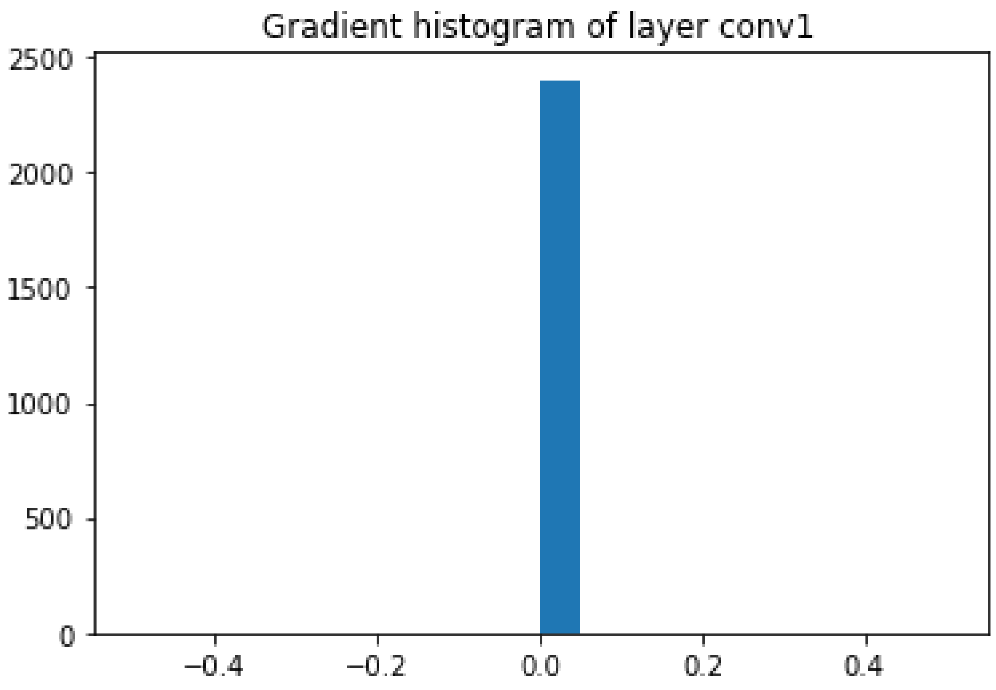

Gradient histogram of layer conv3
30000
25000
20000
15000
10000
5000
0
−0.04
−0.02
0.00
0.02
0.04

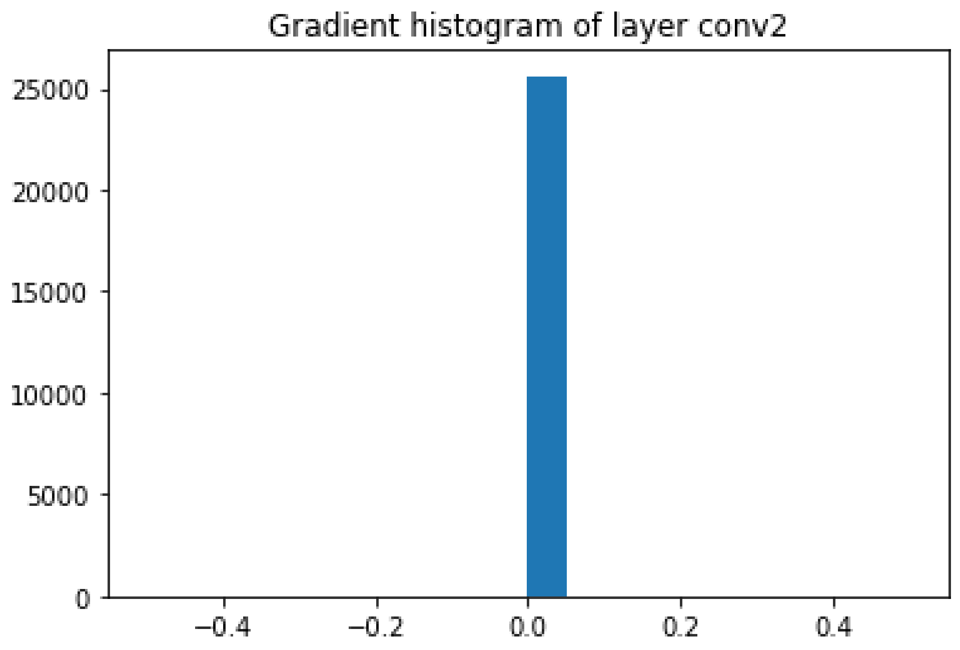
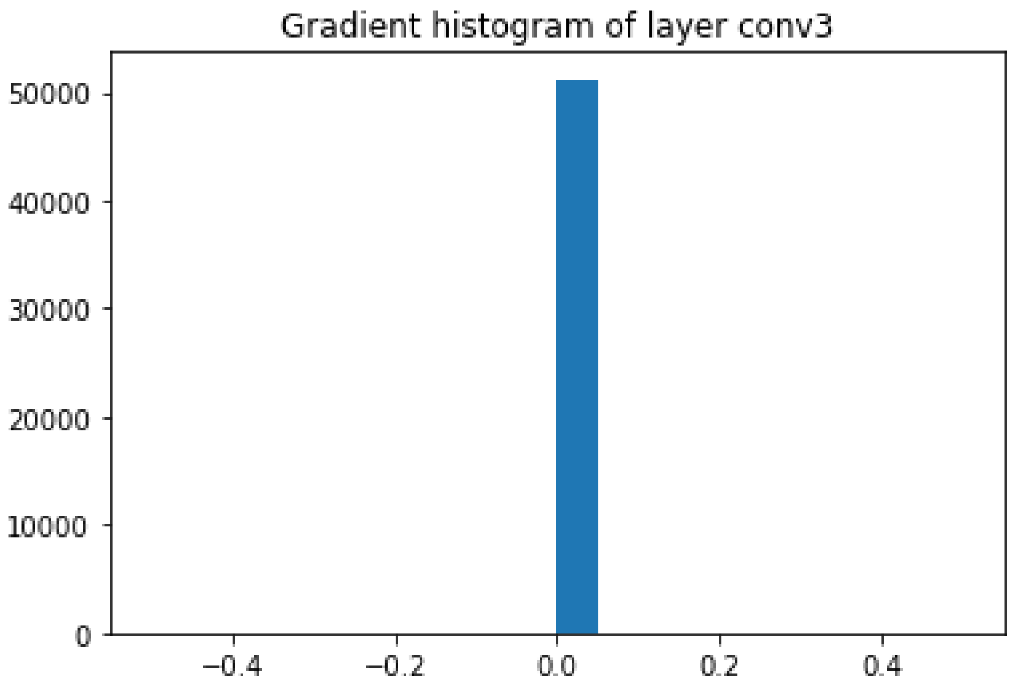
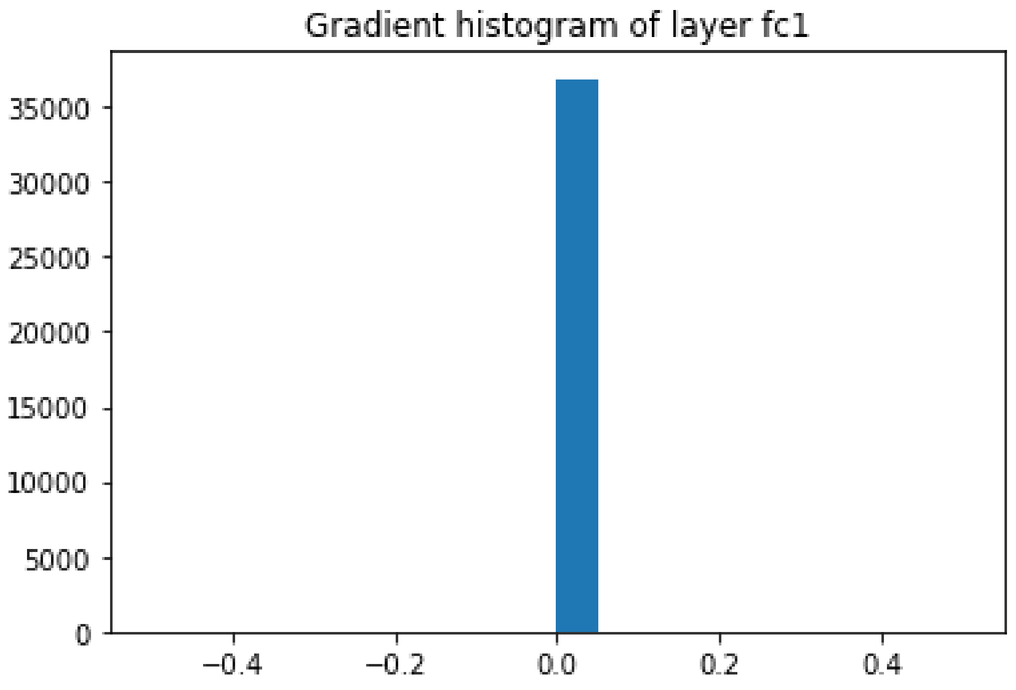

Gradient histogram of layer fc1
30000
25000
20000
15000
10000
5000
0
−0.06
−0.04
−0.02
0.00
0.02
0.04
0.06

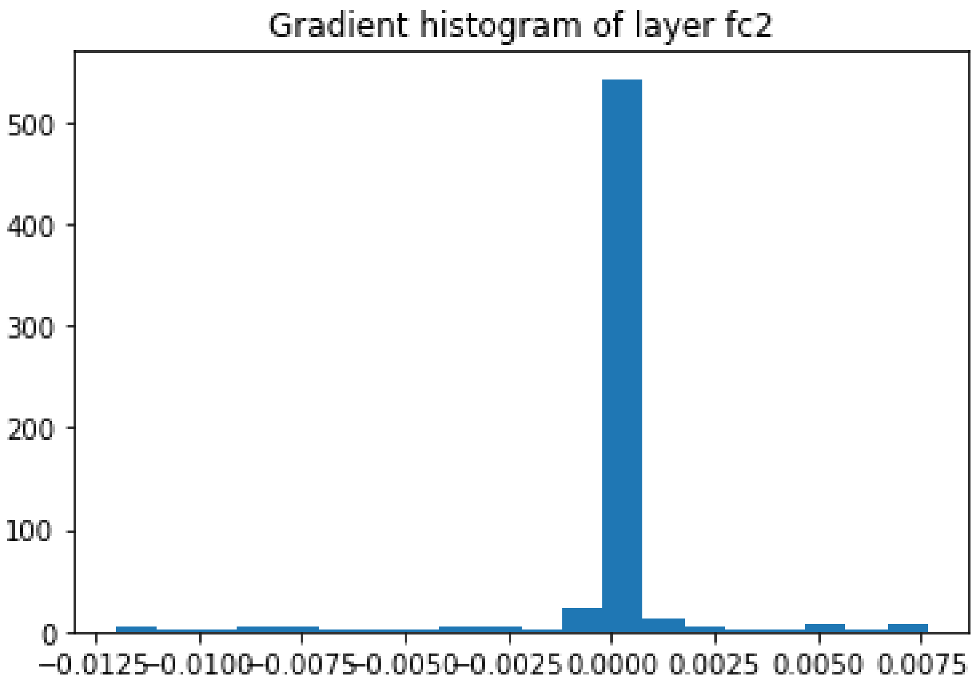


(c) (5pt) In code block 7, finish the code to set all the weights to 0. Perform forward and backward pass again to get the gradients, and plot the histogram of weight elements' gradients in each one of the 5 convolutional and fully-connected layers. Comparing with the histograms you got in (b), are there any differences? Briefly analyze the cause of the difference, and comment on how will initializing CNN model with zero weights will affect the training process. (Note: The CNN initialization methods will be introduced in Lecture 6.)

Solution: Histograms of weight elements' gradients in each one of the 5 convolutional and fully-connected layers are shown below.




Gradient histogram of layer conv2
25000
20000
15000
10000
5000
0
−0.4
−0.2
0.0
0.2
0.4

Gradient histogram of layer conv3
50000
40000
30000
20000
10000
0
−0.4
−0.2
0.0
0.2
0.4

Gradient histogram of layer fc1
35000
30000
25000
20000
15000
10000
5000
0
−0.4
−0.2
0.0
0.2
0.4



Analysis and Comment: Gradient Histograms in (b) generally tend to indicate the characteristic of Normal Distribution, whereas Gradient Histograms in (c) are monolithic, single and isolated. The cause of difference is exactly weight zero initialization. For a CNN model, if all weights are initialized to be 0, the gradient descent of backpropagation will yield the same derivative for each element in weight matrixes. Therefore, all convolutional layers will extract the same features from training samples from every epoch of model training. Finally, the CNN model will either fail during training or have very bad quality and accuracy.