ECE 661: Homework #1
Linear Model, Back Propagation and Building a CNN

Yuanjing Zhu (yz792)
September 10, 2023

1 True/False Questions (10 pts)

For each question, please provide a short explanation to support your judgment.

Problem 1.1 (2 pts) The overfitting models can perfectly fit the training data. We should increase the noise in the training data and the number of parameters the models to improve NN's generalization ability.

FALSE. The overfitting problem impairs model's generalization ability. While an overfitting model may fit the training data exceptionally well, it usually fails to predict effectively on the unseen dataset, indicating poor generalization capability. Increasing noise in the training data can lead model to focus on unnecessary details, rather than the general pattern. Similarly, increasing the number of parameters will worsen the overfitting problem, especially when the dataset is not large enough. Some effective ways to reduce overfitting include L-norm regularization, weight-decay, dropout, early stopping, etc.

Problem 1.2 (2 pts) Given a learning task that can be perfectly learned by a Madaline model, the same set of weight values will be achieved after training, no matter how the Madaline is initialized.

FALSE. The result of a Madaline model is sensitive to random selection pattern and initialization value. Different initialized weights can lead to different local minimum when loss function is non-convex. Even worse, a Madaline model may not converge if initialized improperly. Therefore, even if a task can be perfectly learned by a Madaline model, we cannot guarantee the exact same weight values can be achieved with different initiation processes.

Problem 1.3 (2 pts) The error surface can be complicated. The direction of steepest descent is not always the direction towards the minimum. Full batch size can keep the direction of steepest descent perpendicular to the contour lines. Thus, we should increase the batch size when the error surface is complicated.

FALSE. Full batch size cannot guarantee convergence to the global minimum. While full batch follows the steepest descent on the error surface, which is perpendicular to the contour lines, the direction of the steepest descent may not necessarily direct to the global minimum. For instance, if error surface is a very elongated ellipse, the direction of the steepest descent can deviate from the direction towards the minimum. However, when error surface is complicated (e.g. with local minimums), full batch may lead to suboptimal model trapped in local minimum while a mini-batch with zig-zag routes can help jump out of the local minimum. Additionally, increasing batch size will lead to more computational resources, which is not always the best strategy.

Problem 1.4 (2 pts) In the following code, "If-else" splits the modified Adalines model into two parts. Each part is differentiable. The backpropagation algorithm can be applied to the training of the entire model.

Algorithm 1 A modified Adalines with branches

```
Require: $w_1, w_2, x_1, x_2, n$
Ensure: $n \neq 0$
Ensure: $(x_1w_1 + x_2w_2) \neq 0$
  if $n > 0$ then
    $y \leftarrow Sign(x_1w_1 + x_2w_2)$
  else
    $y \leftarrow Sign(x_1w_1 + x_2w_2) + 5$
  end if
```

FALSE. The signum function is differentiable with derivative 0 everywhere except at 0. This means that during backward propagation, the gradient will always be 0 and the update of weights is meaningless. Additionally, the sign function itself is not differentiable, and the if-else structure leads to a discontinuous function, which is also inherently not differentiable.

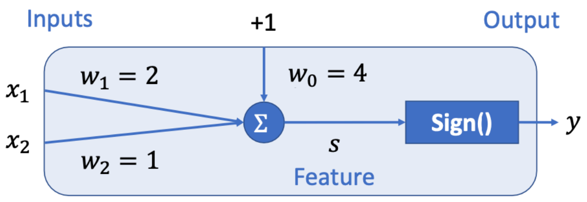
Problem 1.5 (2 pts) According to the "convolution shape rule," for a convolution operation with a fixed input feature map, increasing the height and width of kernel size will always lead to a larger output feature map size.

FALSE. According to the formula: $W_{output} = \frac{W_{input} - K + 2P}{S} + 1,\ H_{output} = \frac{H_{input} - K + 2P}{S} + 1$, increasing the height and width of kernel size while keeping padding and stride fixed will lead to a smaller output feature map. Additionally, the size of the output feature map can be larger, equal to, or smaller than the input size with certain padding and stride values.

2 Adalines (15 pts)

In the following problems, you will be asked to derive the output of a given Adaline, or propose proper weight values for the Adaline to mimic the functionality of some simple logic functions. For all problems, please consider +1 as **True** and −1 as **False** in the inputs and outputs.

Problem 2.1 (3 pts) Observe the Adaline shown in Figure 1, fill in the feature s and output y for each pair of inputs given in the truth table. What logic function is this Adaline performing?

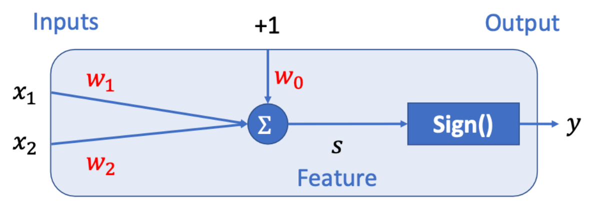


x_1	x_2	s	y
-1	-1	1	1
-1	+1	3	1
+1	-1	5	1
+1	+1	7	1

Figure 1: Problem 2.1.

ANSWER:
According to the equation: $s = w_0 + w_1x_1 + w_2x_2 = 4 + 2x_1 + x_2, y = Sign(s)$, the values of feature s and output y for each pair are shown in the table. This Adaline is performing **constant True** regardless of the inputs.

Problem 2.2 (4 pts) Propose proper values for weight w_0, w_1 and w_2 in the Adaline shown in Figure 2 to perform the functionality of a logic **NAND** function. Fill in the feature s for each pair of inputs given in the truth table to prove the functionality is correct. [**Hint:** The truth table of NAND function can be found here. https://en.wikipedia.org/wiki/NAND_logic]

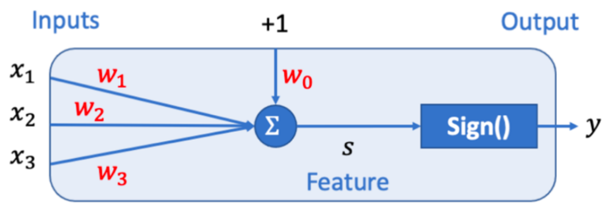


x_1	x_2	s	y
-1	-1	13	+1
-1	+1	3	+1
+1	-1	3	+1
+1	+1	-7	-1

Figure 2: Problem 2.2.

ANSWER: w_0, w_1, w_2 must follow: $\begin{cases} w_0 - w_1 - w_2 > 0 \\ w_0 - w_1 + w_2 > 0 \\ w_0 + w_1 - w_2 > 0 \\ w_0 + w_1 + w_2 < 0 \end{cases}$, one possible solution can be: $w_0 = 3, w_1 = -5, w_2 = -5$

Problem 2.3 (4 pts) Propose proper values for weight w$_0$, w$_1$, w$_2$ and w$_3$ in the Adaline shown in Figure 3 to perform the functionality of a **Majority Vote** function. Fill in the feature s for each triplet of inputs given in the truth table to prove the functionality is correct. [**Hint:** The truth table of Majority Vote function can be found here. https://en.wikichip.org/wiki/boolean_algebra/majority_function]

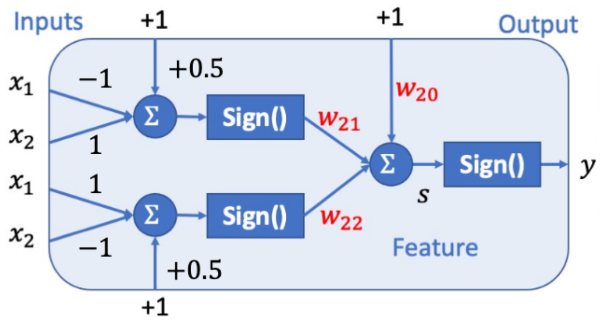


x_1	x_2	x_3	s	y
-1	-1	-1	-2.5	-1
-1	-1	+1	-0.5	-1
-1	+1	-1	-0.5	-1
-1	+1	+1	1.5	+1
+1	-1	-1	-0.5	-1
+1	-1	+1	1.5	+1
+1	+1	-1	1.5	+1
+1	+1	+1	3.5	+1

Figure 3: Problem 2.3.

ANSWER: w_0, w_1, w_2, w_3 must follow: $\begin{cases} w_0 - w_1 - w_2 - w_3 < 0 \\ w_0 - w_1 - w_2 + w_3 < 0 \\ w_0 - w_1 + w_2 - w_3 < 0 \\ w_0 - w_1 + w_2 + w_3 > 0 \\ w_0 + w_1 - w_2 - w_3 < 0 \\ w_0 + w_1 - w_2 + w_3 > 0 \\ w_0 + w_1 + w_2 - w_3 > 0 \\ w_0 + w_1 + w_2 + w_3 > 0 \end{cases}$, one possible solution can be: $w_0 = 0.5, w_1 = 1, w_2 = 1, w_3 = 1$

Problem 2.4 (4 pts) As discussed in Lecture 2, the XOR function cannot be represented with a single Adaline, but can be represented with a 2-layer Madaline. Propose proper values for second-layer weight w$_{20}$,w$_{21}$ and w$_{22}$ in the Madaline shown in Figure 4 to perform the functionality of a **XOR** function. Fill in the feature s for each pair of inputs given in the truth table to prove the functionality is correct.



x_1	x_2	s	y
-1	-1	-10	-1
-1	+1	2	+1
+1	-1	2	+1
+1	+1	-10	-1

Figure 4: Problem 2.4.

ANSWER: w_0, w_1, w_2 must follow: $\begin{cases} w_{20} + w_{21} + w_{22} < 0 \\ w_{20} + w_{21} - w_{22} > 0 \\ w_{20} - w_{21} + w_{22} > 0 \end{cases}$, one possible solution can be: $w_{20} = 2, w_{21} = -6, w_{22} = -6$

3 Back Propagation (10 pts)

Problem 3.1 (5 pts) Consider a 2-layer fully-connected NN, where we have input $x_1 \in R^{n\times1}$, hidden feature $x_2 \in R^{m\times1}$, output $x_3 \in R^{k\times1}$ and weights and bias $W_1 \in R^{m\times n}$, $W_2 \in R^{k\times m}$, $b_1 \in R^{m\times1}$, $b_2 \in R^{k\times1}$ of the two layers. The hidden features and outputs are computed as follows

$$x_2 = \text{Sigmoid}(W_1x_1 + b_1) \tag{1}$$

$$x_3 = W_2x_2 + b_2 \tag{2}$$

A MSE loss function $L = \frac{1}{2}(t - x_3)^T(t - x_3)$ is applied in the end, where $t \in R^{k\times1}$ is the target value. Following the chain rule, derive the gradient $\frac{\delta L}{\delta W_1}, \frac{\delta L}{\delta W_2}, \frac{\delta L}{\delta b_1}, \frac{\delta L}{\delta b_2}$ in a **vectorized format**.

ANSWER:
According to the chain rule:

$$\frac{\partial L}{\partial W_1} = \frac{\partial L}{\partial x_3}\frac{\partial x_3}{\partial x_2}\frac{\partial x_2}{\partial W_1}$$

$$\begin{aligned}\frac{\partial L}{\partial x_3} &= \frac{d}{dx_3}\left(\frac{1}{2}(t^Tt - t^Tx_3 - x_3^Tt + x_3^Tx_3)\right) \\ &= \frac{d}{dx_3}\left(\frac{1}{2}(-2t^Tx_3 + x_3^Tx_3)\right) \\ &= x_3 - t \quad \in R^{k\times1}\end{aligned}$$

$$\frac{\partial x_3}{\partial x_2} = \frac{\partial(W_2x_2 + b_2)}{\partial x_2} = W_2 \quad \in R^{k\times m}$$

$$\frac{\partial x_2}{\partial W_1} = \frac{\partial Sigmoid(W_1x_1 + b_1)}{\partial W_1} = \text{Sigmoid}(W_1x_1 + b_1)\big(1 - Sigmoid(W_1x_1 + b_1)\big)x_1^T$$

Therefore, $\frac{\boldsymbol{\partial L}}{\boldsymbol{\partial W_1}} = [W_2^T(x_3 - t) \odot Sigmoid(W_1x_1 + b_1)\big(1 - Sigmoid(W_1x_1 + b_1)\big)]x_1^T \quad \in R^{m\times n}$

$$\frac{\boldsymbol{\partial L}}{\boldsymbol{\partial W_2}} = \frac{\partial L}{\partial x_3}\frac{\partial x_3}{\partial W_2} = (x_3 - t){x_2}^T \quad \in R^{k\times m}$$

$$\frac{\partial L}{\partial b_1} = \frac{\partial L}{\partial x_3}\frac{\partial x_3}{\partial x_2}\frac{\partial x_2}{\partial b_1}$$

$$\frac{\partial L}{\partial x_3} = x_3 - t, \qquad \frac{\partial x_3}{\partial x_2} = W_2, \qquad \frac{\partial x_2}{\partial b_1} = Sigmoid(W_1x_1 + b_1)\big(1 - Sigmoid(W_1x_1 + b_1)\big)$$

Therefore, $\frac{\boldsymbol{\partial L}}{\boldsymbol{\partial b_1}} = W_2^T(x_3 - t) \odot Sigmoid(W_1x_1 + b_1)\big(1 - Sigmoid(W_1x_1 + b_1)\big) \quad \in R^{m\times 1}$

$$\begin{aligned}\frac{\boldsymbol{\partial L}}{\boldsymbol{\partial b_2}} &= \frac{\partial L}{\partial x_3}\frac{\partial x_3}{\partial b_2}\\ &= (x_3 - t)\ \frac{\partial(W_2x_2 + b_2)}{\partial b_2}\\ &= x_3 - t \quad \in R^{k\times 1}\end{aligned}$$

Problem 3.2 (5 pts) Replace the Sigmoid function with ReLU function. Given a data $x_1 = [0, 1, 2]^T$,

target value $t = [1, 2]^T$, weights and bias at this iteration are

$$W_1 = \begin{bmatrix} 3 & -1 & 1 \\ -5 & 2 & -1 \end{bmatrix}, b_1 = \begin{bmatrix} 1 \\ 2 \end{bmatrix}$$

$$W_2 = \begin{bmatrix} 1 & -2 \\ -3 & 1 \end{bmatrix}, b_1 = \begin{bmatrix} 1 \\ 1 \end{bmatrix}$$

Following the results in Problem 3.1, calculate the values of $L, \frac{\delta L}{\delta W_1}, \frac{\delta L}{\delta W_2}, \frac{\delta L}{\delta b_1}, \frac{\delta L}{\delta b_2}$

ANSWER:

$$x_2 = ReLU(W_1x_1 + b_1) \ = \ ReLU(\begin{bmatrix} 3 & -1 & 1 \\ -5 & 2 & -1 \end{bmatrix}\begin{bmatrix} 0 \\ 1 \\ 2 \end{bmatrix} + \begin{bmatrix} 1 \\ 2 \end{bmatrix}) \ = \ ReLU(\begin{bmatrix} 2 \\ 2 \end{bmatrix}) \ = \ \begin{bmatrix} 2 \\ 2 \end{bmatrix}$$

$$x_3 = W_2x_2 + b_2 = \begin{bmatrix} 1 & -2 \\ -3 & 1 \end{bmatrix}\begin{bmatrix} 2 \\ 2 \end{bmatrix} + \begin{bmatrix} 1 \\ 1 \end{bmatrix} = \begin{bmatrix} -1 \\ -3 \end{bmatrix}$$

$$\boldsymbol{L} = \frac{1}{2}(t-x_3)^T(t-x_3) = \frac{1}{2}\left(\begin{bmatrix}1\\2\end{bmatrix} - \begin{bmatrix}-1\\-3\end{bmatrix}\right)^T\left(\begin{bmatrix}1\\2\end{bmatrix} - \begin{bmatrix}-1\\-3\end{bmatrix}\right) = \frac{1}{2}\left(\begin{bmatrix}2 & 5\end{bmatrix}\begin{bmatrix}2\\5\end{bmatrix}\right) = \mathbf{14.5}$$

The derivative of ReLU function is $ReLU'(x) = \begin{cases}0 & if\ x < 0\\ 1 & if\ x > 0\end{cases}$

$$\begin{aligned}
\frac{\boldsymbol{\partial L}}{\boldsymbol{\partial W_1}} &= [W_2^T(x_3 - t) \odot ReLU'(W_1x_1 + b_1)]x_1^T\\
&= \left[\begin{bmatrix}1 & -3\\-2 & 1\end{bmatrix}\begin{bmatrix}-2\\-5\end{bmatrix} \odot ReLU'\left(\begin{bmatrix}2\\2\end{bmatrix}\right)\right]\begin{bmatrix}0 & 1 & 2\end{bmatrix}\\
&= \begin{bmatrix}13\\-1\end{bmatrix}\begin{bmatrix}0 & 1 & 2\end{bmatrix}\\
&= \begin{bmatrix}0 & 13 & 26\\0 & -1 & -2\end{bmatrix}
\end{aligned}$$

$$\begin{aligned}
\frac{\boldsymbol{\partial L}}{\boldsymbol{\partial W_2}} &= (x_3 - t){x_2}^T\\
&= \left(\begin{bmatrix}-1\\-3\end{bmatrix} - \begin{bmatrix}1\\2\end{bmatrix}\right)\begin{bmatrix}2 & 2\end{bmatrix}\\
&= \begin{bmatrix}-4 & -4\\-10 & -10\end{bmatrix}
\end{aligned}$$

$$\begin{aligned}
\frac{\boldsymbol{\partial L}}{\boldsymbol{\partial b_1}} &= W_2^T(x_3 - t) \odot ReLU'(W_1x_1 + b_1)\\
&= \left[\begin{bmatrix}1 & -3\\-2 & 1\end{bmatrix}\begin{bmatrix}-2\\-5\end{bmatrix} \odot ReLU'\left(\begin{bmatrix}2\\2\end{bmatrix}\right)\right]\\
&= \begin{bmatrix}13\\-1\end{bmatrix}
\end{aligned}$$

$$\begin{aligned}
\frac{\boldsymbol{\partial L}}{\boldsymbol{\partial b_2}} &= x_3 - t\\
&= \begin{bmatrix}-2\\-5\end{bmatrix}
\end{aligned}$$

4 2D Convolution (10 pts)

Problem 4.1 (5 pts) Derive the 2D convolution results of the following 5 × 9 input matrix and the 3 × 3 kernel. Consider 0s are padded around the input and the stride is 1, so that the output should also have shape 5 × 9.

$$\begin{bmatrix}0 & 0 & -1 & 0 & 0 & 0 & 1 & 0 & 0\\0 & -1 & -1 & -1 & 0 & 1 & 1 & 1 & 0\\-1 & -1 & -1 & -1 & 0 & 1 & 1 & 1 & 1\\0 & -1 & -1 & -1 & 0 & 1 & 1 & 1 & 0\\0 & 0 & -1 & 0 & 0 & 0 & 1 & 0 & 0\end{bmatrix} * \begin{bmatrix}0 & -1/2 & 0\\-1/2 & 1 & -1/2\\0 & -1/2 & 0\end{bmatrix}$$

ANSWER:
According to the formula: $W_{output} = \frac{W_{input} - K + 2P}{S} + 1$, $H_{output} = \frac{H_{input} - K + 2P}{S} + 1$, padding size = 1 to make sure the output shape is still 5 × 9.

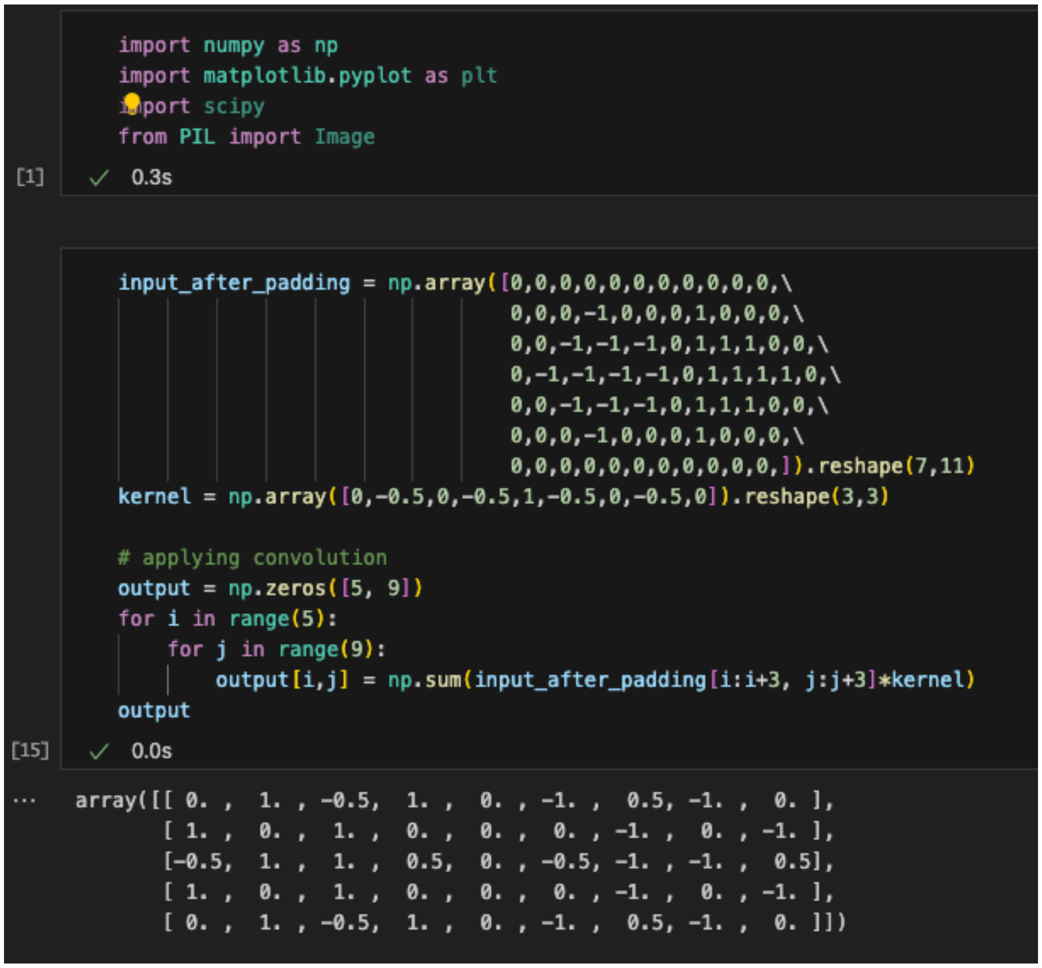
The input matrix after padding is : $\begin{bmatrix} 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & -1 & 0 & 0 & 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & -1 & -1 & -1 & 0 & 1 & 1 & 1 & 0 & 0 \\ 0 & -1 & -1 & -1 & -1 & 0 & 1 & 1 & 1 & 1 & 0 \\ 0 & 0 & -1 & -1 & -1 & 0 & 1 & 1 & 1 & 0 & 0 \\ 0 & 0 & 0 & -1 & 0 & 0 & 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \end{bmatrix}$

Applying 3×3 kernel on the input:

$$a_{11} = 0 \times 0 + 0 \times (-1/2) + 0 \times 0 + 0 \times (-1/2) + 0 \times 1 + 0 \times (-1/2) + 0 \times 0 + 0 \times (-1/2) + (-1) \times 0 = 0$$
$$a_{12} = 0 \times 0 + 0 \times (-1/2) + 0 \times 0 + 0 \times (-1/2) + 0 \times 1 + (-1) \times (-1/2) + 0 \times 0 + (-1) \times (-1/2) + (-1) \times 0 = 1$$
……

So the output matrix is: $\begin{bmatrix} 0 & 1 & -1/2 & 1 & 0 & -1 & 1/2 & -1 & 0 \\ 1 & 0 & 1 & 0 & 0 & 0 & -1 & 0 & -1 \\ -1/2 & 1 & 1 & 1/2 & 0 & -1/2 & -1 & -1 & 1/2 \\ 1 & 0 & 1 & 0 & 0 & 0 & -1 & 0 & -1 \\ 0 & 1 & -1/2 & 1 & 0 & -1 & 1/2 & -1 & 0 \end{bmatrix}$

I also validate the manual calculation by using *numpy* library:



```
import numpy as np
import matplotlib.pyplot as plt
import scipy
from PIL import Image
```
[1] ✓ 0.3s

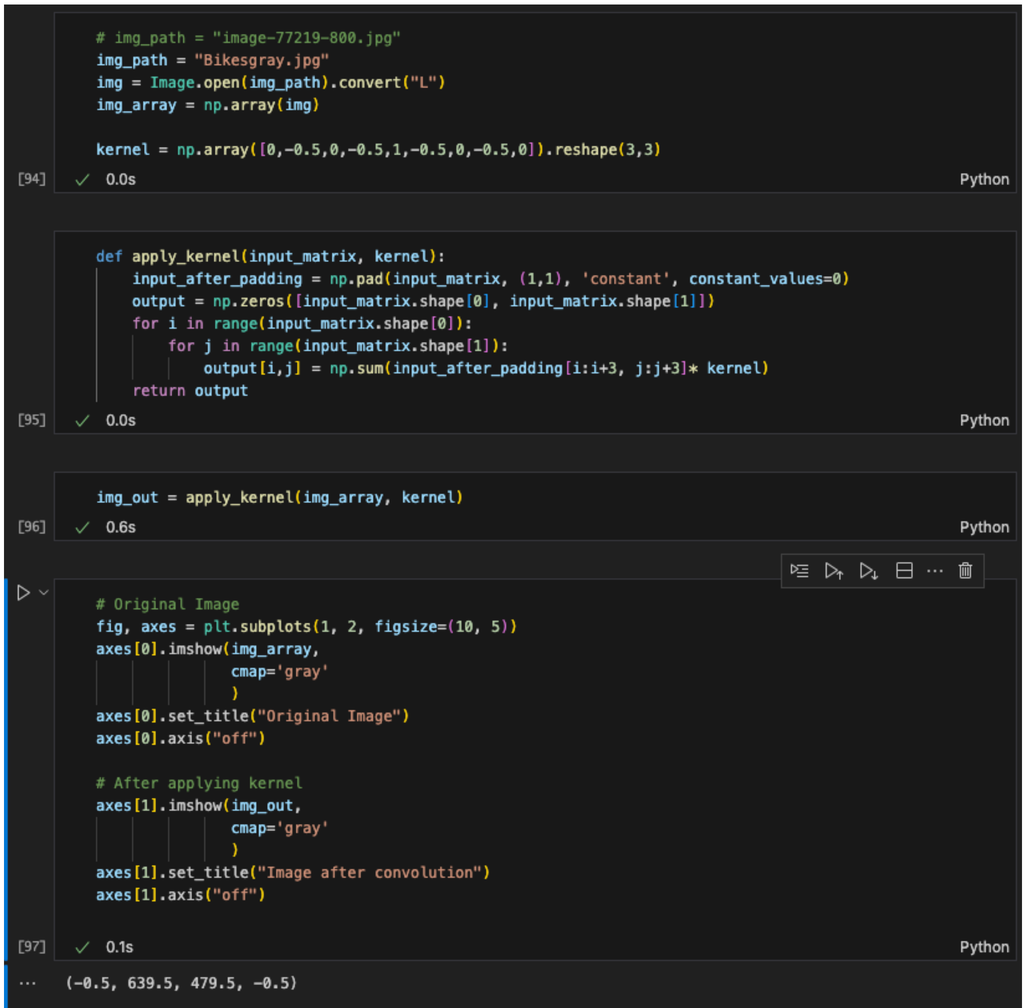
```
input_after_padding = np.array([0,0,0,0,0,0,0,0,0,0,0,\
                                0,0,0,-1,0,0,0,1,0,0,0,\
                                0,0,-1,-1,-1,0,1,1,1,0,0,\
                                0,-1,-1,-1,-1,0,1,1,1,1,0,\
                                0,0,-1,-1,-1,0,1,1,1,0,0,\
                                0,0,0,-1,0,0,0,1,0,0,0,\
                                0,0,0,0,0,0,0,0,0,0,0,]).reshape(7,11)
kernel = np.array([0,-0.5,0,-0.5,1,-0.5,0,-0.5,0]).reshape(3,3)

# applying convolution
output = np.zeros([5, 9])
for i in range(5):
    for j in range(9):
        output[i,j] = np.sum(input_after_padding[i:i+3, j:j+3]*kernel)
output
```
[15] ✓ 0.0s

```
array([[ 0. ,  1. , -0.5,  1. ,  0. , -1. ,  0.5, -1. ,  0. ],
       [ 1. ,  0. ,  1. ,  0. ,  0. ,  0. , -1. ,  0. , -1. ],
       [-0.5,  1. ,  1. ,  0.5,  0. , -0.5, -1. , -1. ,  0.5],
       [ 1. ,  0. ,  1. ,  0. ,  0. ,  0. , -1. ,  0. , -1. ],
       [ 0. ,  1. , -0.5,  1. ,  0. , -1. ,  0.5, -1. ,  0. ]])
```

Problem 4.2 (5 pts) Compare the output matrix and the input matrix in Problem 4.1, briefly analyze the effect of this 3 × 3 kernel on the input. (Hint: apply this kernel to an image to see the outputs)

ANSWER:
After applying the kernel to an image and comparing it with the transformed image, I found the effect of this 3 × 3 kernel is to **flip black and white** in the original image. The negative / inversion effect of the kernel can help highlight intensity transitions in the image. Moreover, the surrounding negative 0.5 values around the central 1 brings a **blurring** effect to the image. The image after transformation has a smoother look, especially around the edges and regions of high contrast.



```
# img_path = "image-77219-800.jpg"
img_path = "Bikesgray.jpg"
img = Image.open(img_path).convert("L")
img_array = np.array(img)

kernel = np.array([0,-0.5,0,-0.5,1,-0.5,0,-0.5,0]).reshape(3,3)
```
[94] ✓ 0.0s Python

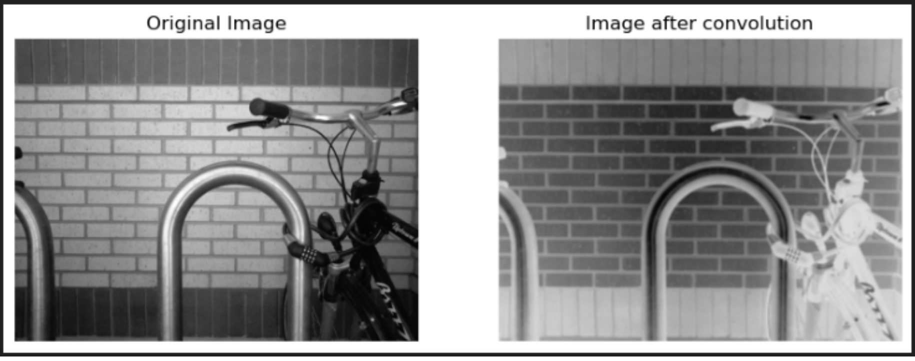
```
def apply_kernel(input_matrix, kernel):
    input_after_padding = np.pad(input_matrix, (1,1), 'constant', constant_values=0)
    output = np.zeros([input_matrix.shape[0], input_matrix.shape[1]])
    for i in range(input_matrix.shape[0]):
        for j in range(input_matrix.shape[1]):
            output[i,j] = np.sum(input_after_padding[i:i+3, j:j+3]* kernel)
    return output
```
[95] ✓ 0.0s Python

```
img_out = apply_kernel(img_array, kernel)
```
[96] ✓ 0.6s Python

```
# Original Image
fig, axes = plt.subplots(1, 2, figsize=(10, 5))
axes[0].imshow(img_array,
               cmap='gray'
               )
axes[0].set_title("Original Image")
axes[0].axis("off")

# After applying kernel
axes[1].imshow(img_out,
               cmap='gray'
               )
axes[1].set_title("Image after convolution")
axes[1].axis("off")
```
[97] ✓ 0.1s Python

```
(-0.5, 639.5, 479.5, -0.5)
```



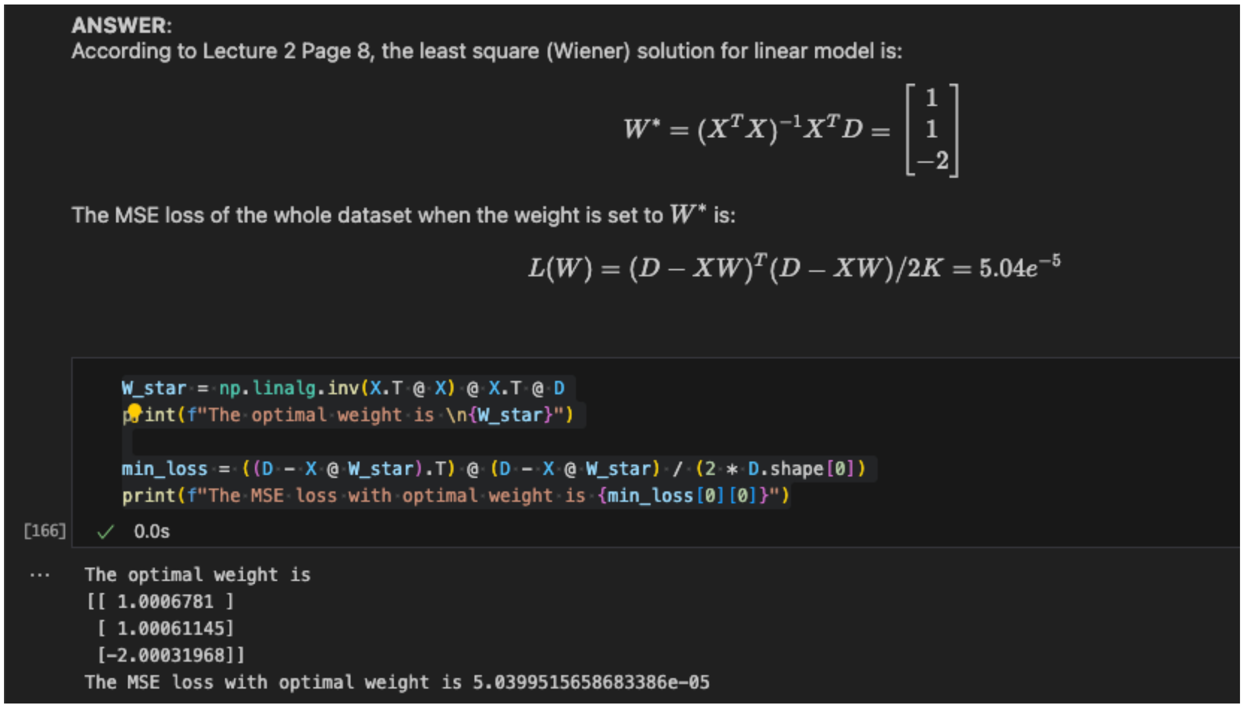
5 Lab: LMS Algorithm (15 pts)

In this lab question, you will implement the LMS algorithm with NumPy to learn a linear regression model for the provided dataset. You will also be directed to analyze how the choice of learning rate in the LMS algorithm affect the final result. All the codes generating the results of this lab should be gathered in one file and submit to Sakai.

To start with, please download the dataset.mat file from Sakai and load it into NumPy arrays[a]. There are two variables in the file: data $X \in R^{100\times3}$ and target $D \in R^{100\times1}$. Each individual pair of data and target is composed into X and D following the same way as discussed on Lecture 2 Page 8. Specifically, each row in X correspond to the transpose of a data point, with the first element as constant 1 and the other two as the two input features x_{1k} and x_{2k}. The goal of the learning task is finding the weight vector $W \in R^{3\times1}$ for the linear model that can minimize the MSE loss, which is also formulated on Lecture 2 Page 7.

(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^*?

ANSWER:



ANSWER:
According to Lecture 2 Page 8, the least square (Wiener) solution for linear model is:

$$W^* = (X^TX)^{-1}X^TD = \begin{bmatrix}1\\1\\-2\end{bmatrix}$$

The MSE loss of the whole dataset when the weight is set to W^* is:

$$L(W) = (D - XW)^T(D - XW)/2K = 5.04e^{-5}$$

```
W_star = np.linalg.inv(X.T @ X) @ X.T @ D
print(f"The optimal weight is \n{W_star}")

min_loss = ((D - X @ W_star).T) @ (D - X @ W_star) / (2 * D.shape[0])
print(f"The MSE loss with optimal weight is {min_loss[0][0]}")
```

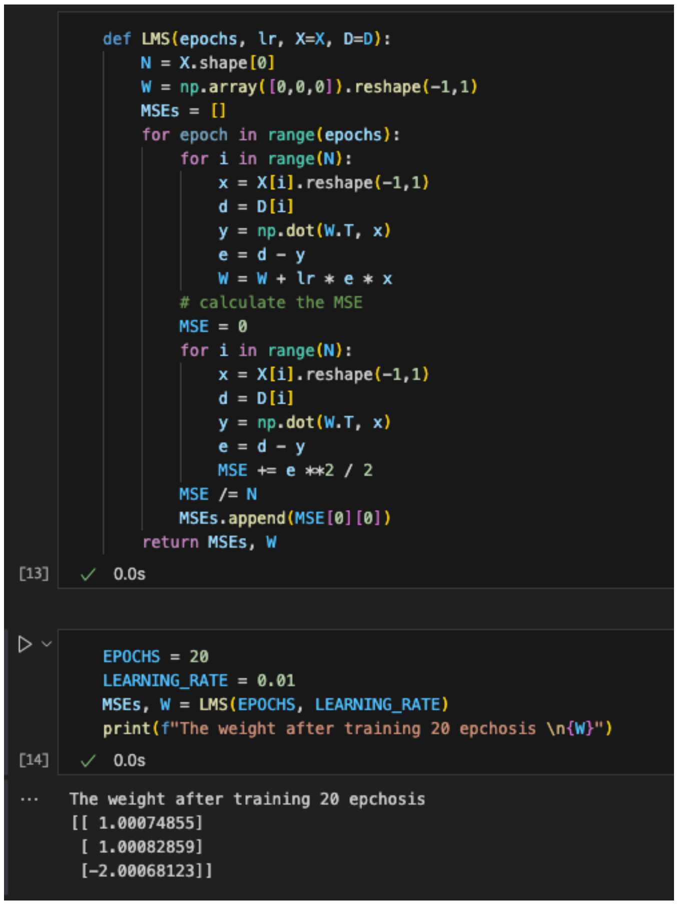
```
[166]  ✓ 0.0s
```

```
The optimal weight is
[[ 1.0006781 ]
 [ 1.00061145]
 [-2.00031968]]
The MSE loss with optimal weight is 5.0399515658683386e-05
```

Computing Wiener solution, the optimal weight W* is [[1], [1], [-2]], the MSE loss of the whole dataset when the weight is set to W* is 5.04e-5.

(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.

ANSWER: The weight I get in the end is also [[1], [1], [-2]].



```
def LMS(epochs, lr, X=X, D=D):
    N = X.shape[0]
    W = np.array([0,0,0]).reshape(-1,1)
    MSEs = []
    for epoch in range(epochs):
        for i in range(N):
            x = X[i].reshape(-1,1)
            d = D[i]
            y = np.dot(W.T, x)
            e = d - y
            W = W + lr * e * x
        # calculate the MSE
        MSE = 0
        for i in range(N):
            x = X[i].reshape(-1,1)
            d = D[i]
            y = np.dot(W.T, x)
            e = d - y
            MSE += e **2 / 2
        MSE /= N
        MSEs.append(MSE[0][0])
    return MSEs, W
```

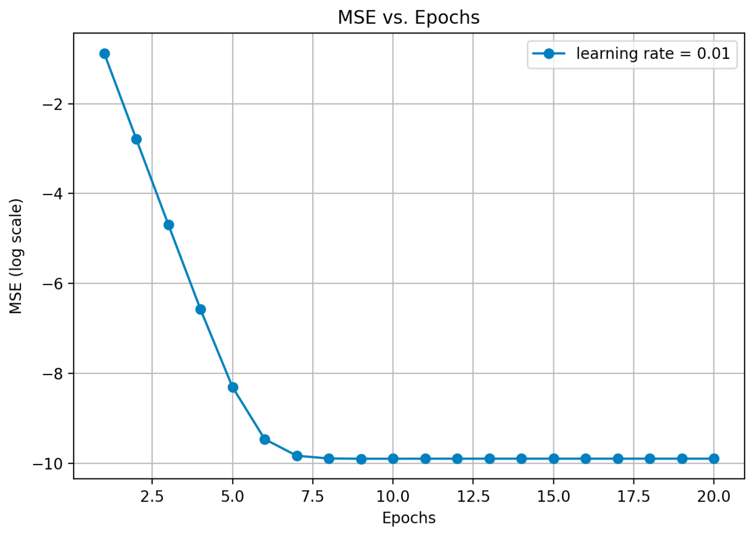
```
[13]  ✓ 0.0s
```

```
EPOCHS = 20
LEARNING_RATE = 0.01
MSEs, W = LMS(EPOCHS, LEARNING_RATE)
print(f"The weight after training 20 epchosis \n{W}")
```

```
[14]  ✓ 0.0s

...  The weight after training 20 epchosis
     [[ 1.00074855]
      [ 1.00082859]
      [-2.00068123]]
```

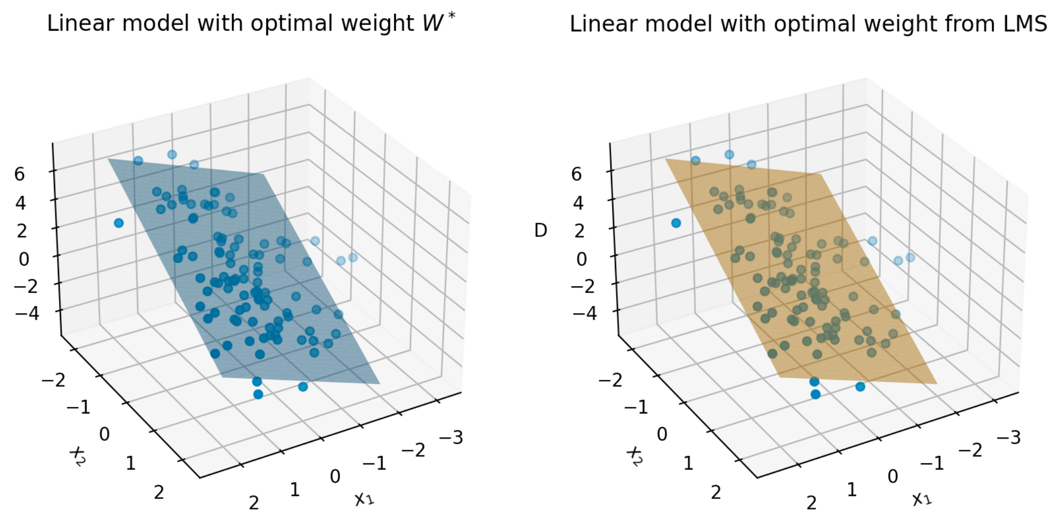
Here is the plot the MSE loss *in log scale* vs. Epochs:



(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.

ANSWER:

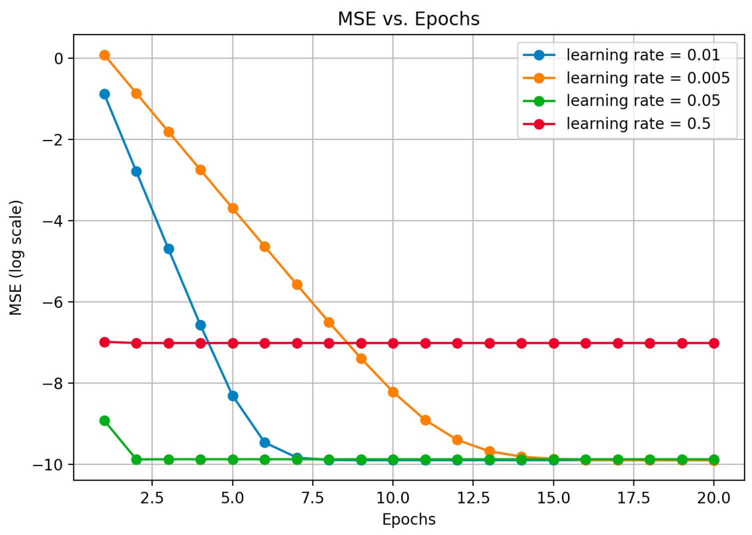
As shown in the 3D plot, the linear model **fits the data well**. Most of the data points lie on the contour of the linear model with optimal weight from LMS.



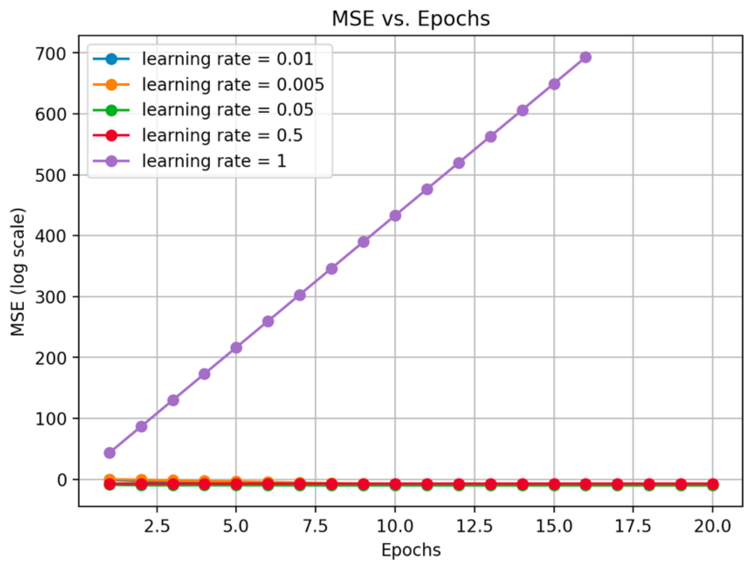
(d) (5pt) Learning rate r is an important hyperparameter for the LMS algorithm, as well as for CNN optimization. Here, try repeat the process in (b) with r set to 0.005, 0.05 and 0.5 respectively. Together with the result you got in (b), plot the MSE losses of the 4 sets of experiments in log scale vs. 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.)

ANSWER:

Plot of MSE loss vs Epochs with learning rate set to 0.01, 0.005, 0.05, and 0.5 respectively:



Plot of MSE loss vs Epochs with learning rate increasing to 1:



At $lr = 0.005$, the training process slowly converges to its gloabl minimum. It is very stable and the quality of training is essured, but it takes more epoches to reach a minimum MSE compared to a higher learning rate. Increasing lr to 0.05, it shows a good balance between speed and the quality. It converges faster but still stably reaches minimum MSE in the training process. With a higher learning rate ($lr = 0.05$), the drop of MSE is much faster in the first epoch. Further increasing learning rate ($lr = 0.5$) led to overshooting the minimum, resulting in suboptimal model performance. Finally at a much higher learning rate ($lr = 1$), the MSE loss continully increases, leading to divergence in the end. Therefore, properly tuning learning rate is crucial to strike a balance between speed and model accuracy.

6 Lab: Simple NN (40 pts)

For getting started with deep Neural Network model easily, we consider a simple Neural Network model here and details of the model architecture is given in Table 1. This lab question focuses on building the model in PyTorch and observing the shape of each layer's input, weight and output. Please refer to the **NumPy/PyTorch Tutorial slides** on Sakai and the official documentations if you are unfamiliar with PyTorch syntax.

Please finish this lab by completing the SimpleNN.ipynb notebook file provided on Sakai. The com- pleted notebook file should be submitted to Sakai.

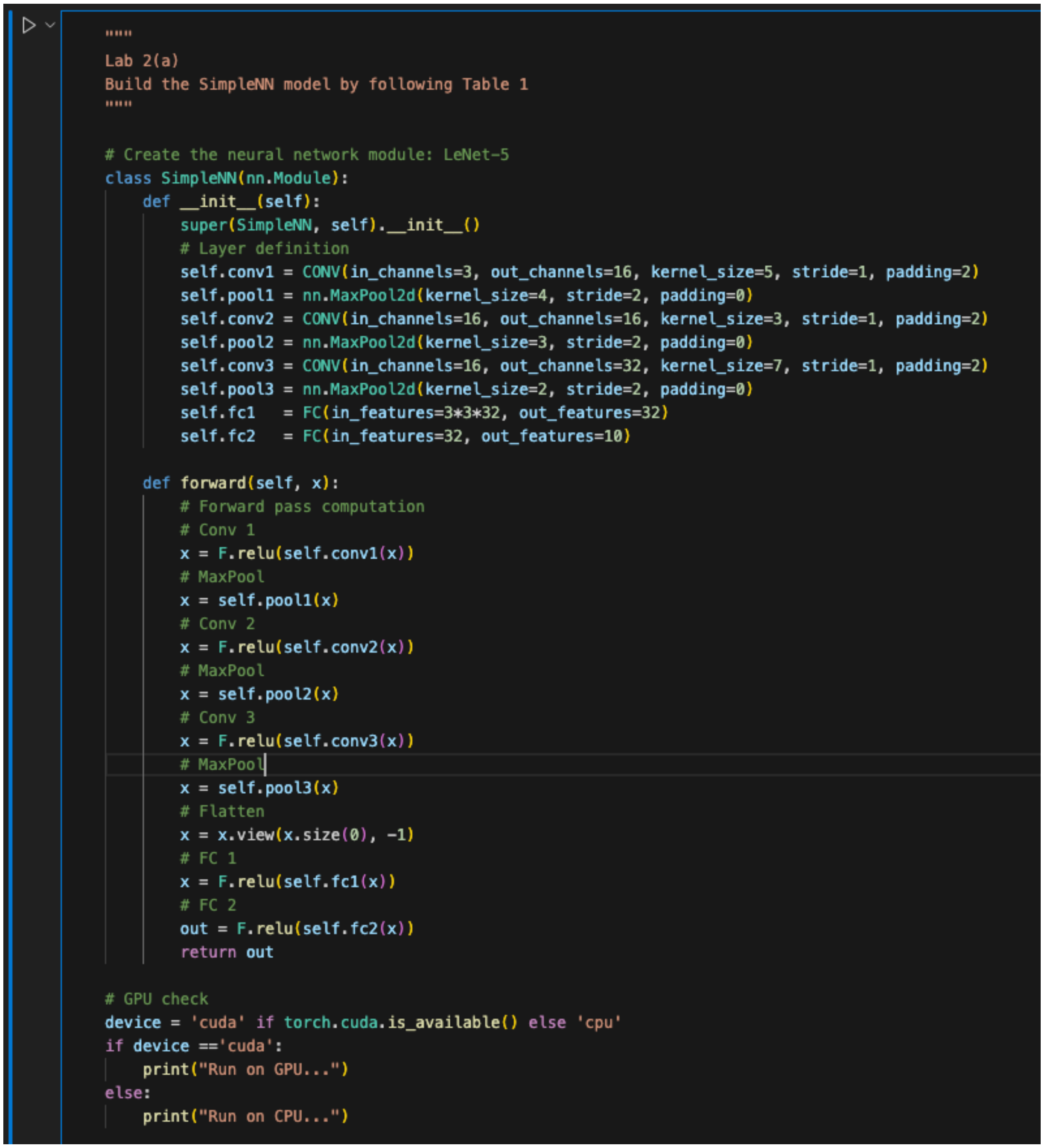
Name	Type	Kernel size	depth/units	Activation	Strides
Conv 1	Convolution	5	16	ReLU	1
MaxPool	MaxPool	4	N/A	N/A	2
Conv 2	Convolution	3	16	ReLU	1
MaxPool	MaxPool	3	N/A	N/A	2
Conv 3	Convolution	7	32	ReLU	1
MaxPool	MaxPool	2	N/A	N/A	2
FC1	Fully-connected	N/A	32	ReLU	N/A
FC2	Fully-connected	N/A	10	ReLU	N/A

Table 1: The padding for all three convolution layers is 2. The padding for all three MaxPool layers is 0. A flatten layer is required before FC1 to reshape the feature.

Lab 2 (40 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 SimpleNN class into the report PDF.

ANSWER:

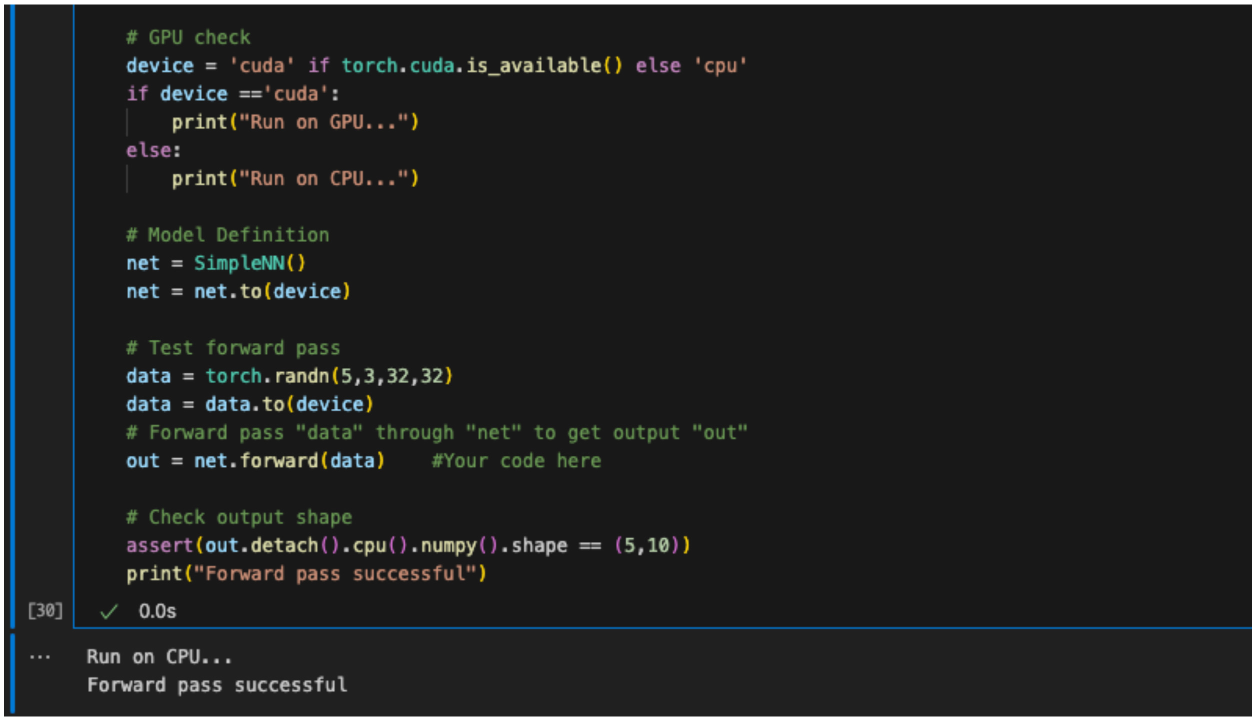


```
"""
Lab 2(a)
Build the SimpleNN model by following Table 1
"""

# 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=16, kernel_size=5, stride=1, padding=2)
        self.pool1 = nn.MaxPool2d(kernel_size=4, stride=2, padding=0)
        self.conv2 = CONV(in_channels=16, out_channels=16, kernel_size=3, stride=1, padding=2)
        self.pool2 = nn.MaxPool2d(kernel_size=3, stride=2, padding=0)
        self.conv3 = CONV(in_channels=16, out_channels=32, kernel_size=7, stride=1, padding=2)
        self.pool3 = nn.MaxPool2d(kernel_size=2, stride=2, padding=0)
        self.fc1   = FC(in_features=3*3*32, out_features=32)
        self.fc2   = FC(in_features=32, out_features=10)

    def forward(self, x):
        # Forward pass computation
        # Conv 1
        x = F.relu(self.conv1(x))
        # MaxPool
        x = self.pool1(x)
        # Conv 2
        x = F.relu(self.conv2(x))
        # MaxPool
        x = self.pool2(x)
        # Conv 3
        x = F.relu(self.conv3(x))
        # MaxPool
        x = self.pool3(x)
        # Flatten
        x = x.view(x.size(0), -1)
        # FC 1
        x = F.relu(self.fc1(x))
        # FC 2
        out = F.relu(self.fc2(x))
        return out

# GPU check
device = 'cuda' if torch.cuda.is_available() else 'cpu'
if device =='cuda':
    print("Run on GPU...")
else:
    print("Run on CPU...")
```



```
# GPU check
device = 'cuda' if torch.cuda.is_available() else 'cpu'
if device =='cuda':
    print("Run on GPU...")
else:
    print("Run on CPU...")

# Model Definition
net = SimpleNN()
net = net.to(device)

# Test forward pass
data = torch.randn(5,3,32,32)
data = data.to(device)
# Forward pass "data" through "net" to get output "out"
out = net.forward(data)     #Your code here

# Check output shape
assert(out.detach().cpu().numpy().shape == (5,10))
print("Forward pass successful")
[30]  ✓  0.0s

...  Run on CPU...
     Forward pass successful
```

(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 pro- cessing 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.

ANSWER:

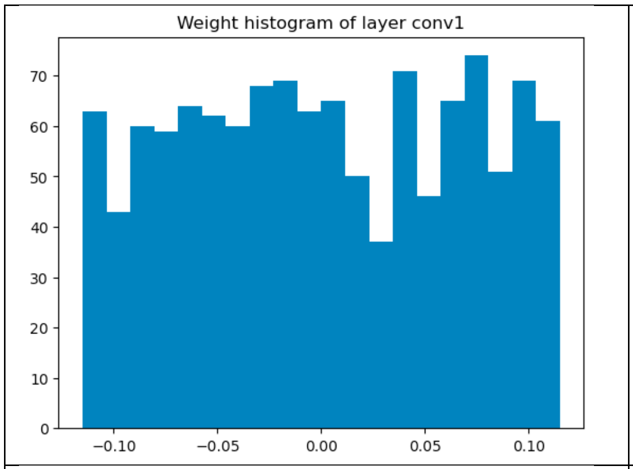
Formula for calculating weight elements and total MACs is shown in the code.

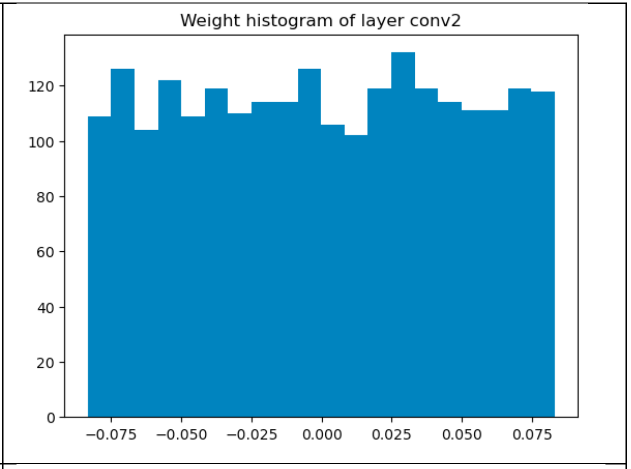
Layer	Input shape	Output shape	Weight shape	# Param	# MAC
Conv 1	[1, 3, 32, 32]	[1, 16, 32, 32]	[16, 3, 5, 5]	1200	1228800
Conv 2	[1, 16, 15, 15]	[1, 16, 17, 17]	[16, 16, 3, 3]	2304	665856
Conv 3	[1, 16, 8, 8]	[1, 32, 6, 6]	[32, 16, 7, 7]	25088	903168
FC1	[1, 288]	[1, 32]	[32, 288]	9216	9216
FC2	[1, 32]	[1, 10]	[10, 32]	320	320

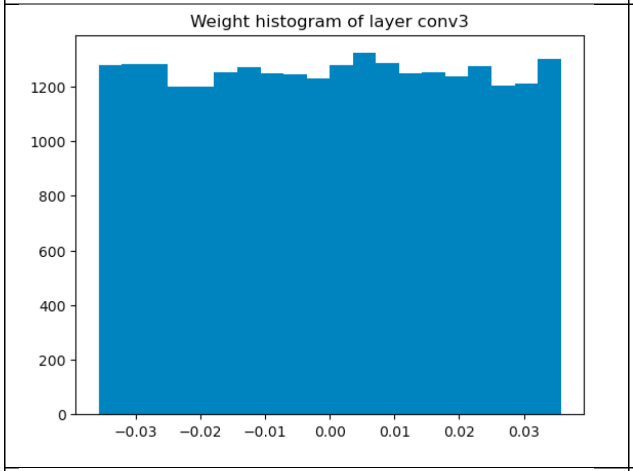
Table 2: Results of Lab 2(b).

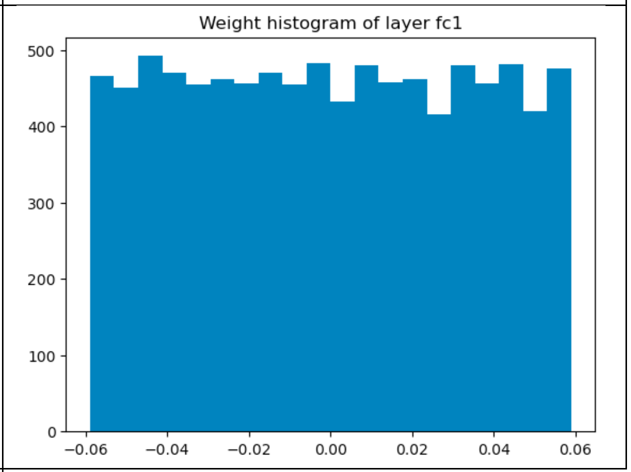
Lab 3 (Bonus 10 points)
Please first finish all the required codes in Lab 2, then proceed to code block 5 of the notebook file.

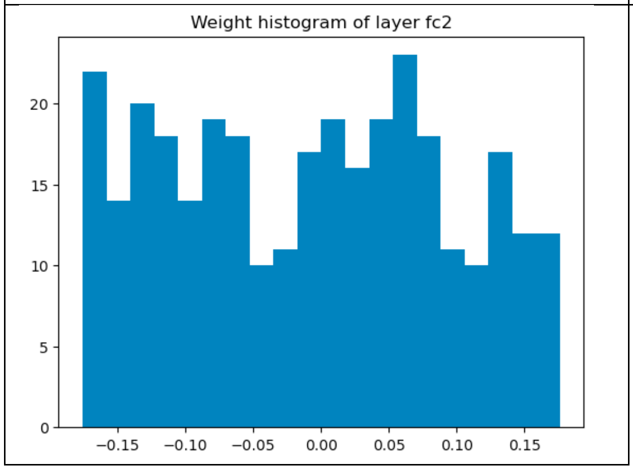
(a) (2pt) Complete the for-loop in code block 5 to plot the histogram of weight elements in each one of the 5 convolutional and fully-connected layers.



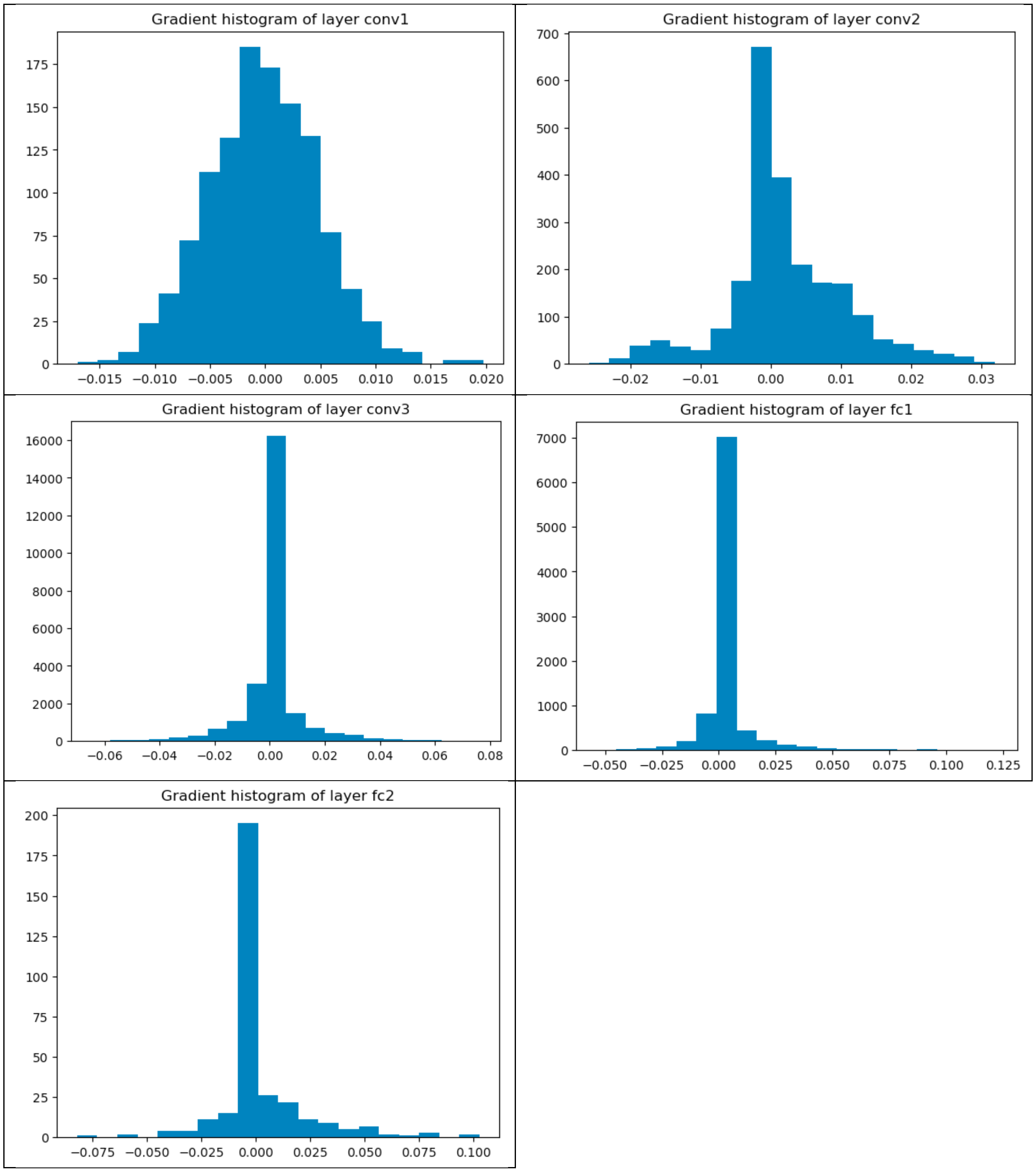




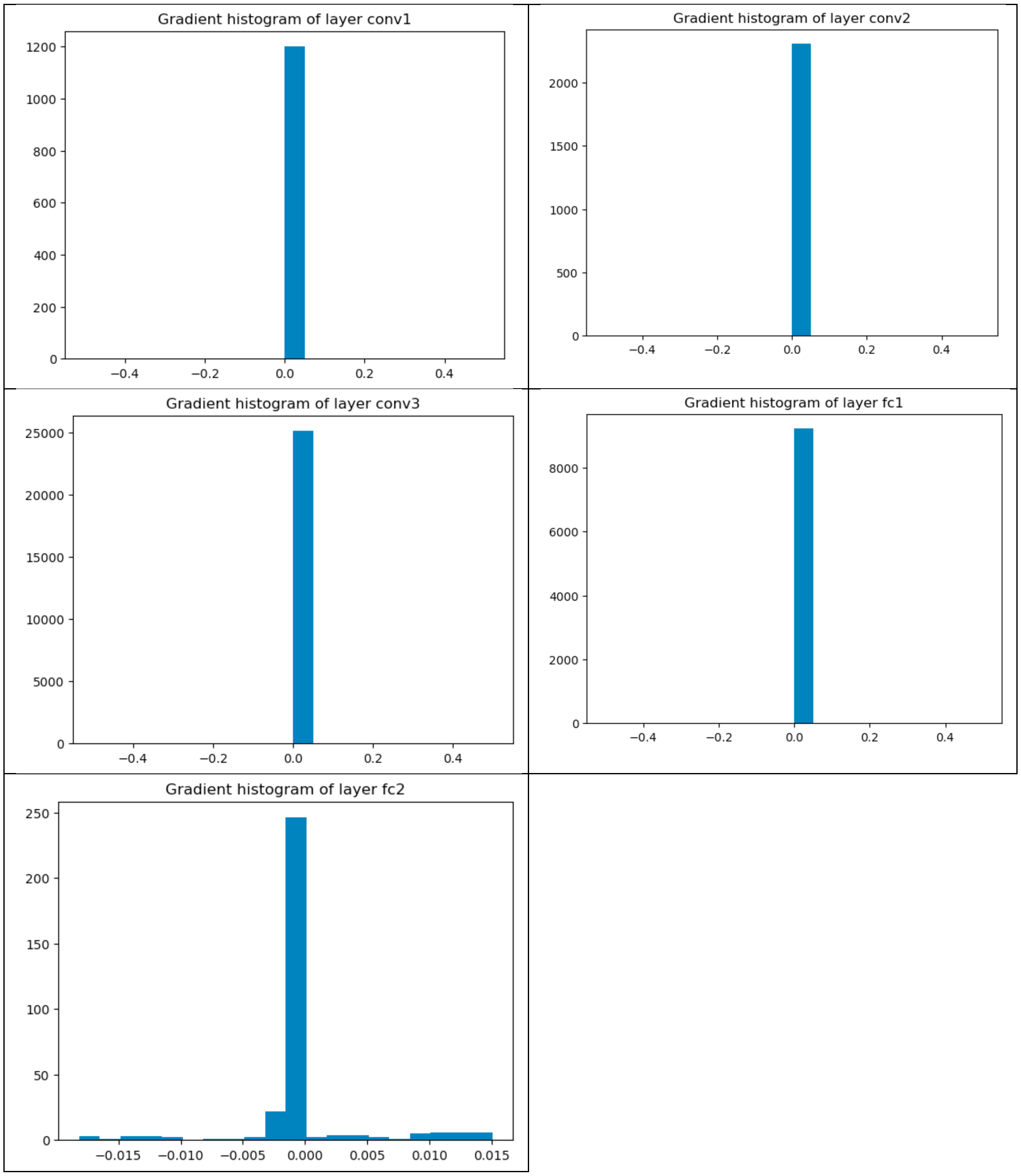




(b) (3pt) In code block 6, complete the code for backward pass, then complete the for-loop to plot the histogram of weight elements' gradients in each one of the 5 convolutional and fully-connected layers.



(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.)



ANSWER:

- After setting all the weights to 0, the gradients of nearly every weight element become zero for both convolutional and fully-connected layers.
- This is caused by the weight initialization of 0. The activation function is ReLU: $y = max(0, x). Its\ derivative\ is: \frac{dy}{dx} = \begin{cases} 0 & x \leq 0 \\ 1 & x > 0 \end{cases}$. If all the weights are set to be 0 from the beginning, the gradients of weights will also be 0 during backpropagation. Therefore, weights don't get updated despite of incoming training data, resulting in no learning.

- Initializing CNN model with zero weights will lead gradients to vanish. Weights will never be updated, and the training process is meaningless.