Machine Learning
Homework #5

Daniel Ho Kwan Leung
104360098
CSIE, Year 3

Question 1:

The cross entropy cost function,

$$C = -d\log_2 y - (1-d)\log_2(1-y)$$

Assume all the activation function is sigmoid. That is,

$$y = f(x) = \frac{1}{1+e^{(-w^T x)}}$$

The derivative term,

$$\begin{aligned}\frac{\partial C}{\partial w_j} &= \frac{\partial - d\log_2 y - (1-d)\log_2(1-y)}{\partial w_j} \\ &= -\frac{\partial d\log_2 y}{\partial w_j} - \frac{\partial(1-d)\log_2(1-y)}{\partial w_j} \\ &= -d\frac{\partial \log_2 y}{\partial y}\frac{\partial y}{\partial z}\frac{\partial z}{\partial w_j} - (1-d)\frac{\partial \log_2(1-y)}{\partial(1-y)}\frac{\partial(1-y)}{\partial y}\frac{\partial y}{\partial z}\frac{\partial z}{\partial w_j} \\ &= -d\frac{1}{\ln 2}\frac{1}{y}y(1-y)x_j + (1-d)\frac{1}{\ln 2}\frac{1}{1-y}y(1-y)x_j \\ &= \left[-d(1-y)x_j + (1-d)yx_j\right]\frac{1}{\ln 2}\end{aligned}$$

Since the coefficient $\frac{1}{\ln 2}$ is not important at all, we can just consider the term $\left[-d(1-y)x_j + (1-d)yx_j\right]$.

If $d = 1$, only the former term $-d(1-y)x_j$ contributes. If $d = 0$, only the latter term $(1-d)yx_j$ contributes. That is,

$$\begin{aligned}\frac{\partial C}{\partial w_j} &= \begin{cases}(y-1)x_j, & \text{if } d = 1 \\ yx_j, & \text{if } d = 0\end{cases} \\ &= \begin{cases}(y-d)x_j, & \text{if } d = 1 \\ (y-d)x_j, & \text{if } d = 0\end{cases} \\ &= (y-d)x_j\end{aligned}$$

Question 2:

Assume we are using one single tuple of data, we can omit the summation along the data. The loss function is cross entropy,

$$L_i(d_i, y_i | w_1, \dots, w_8) = -d_i\log_2 y_i - (1-d_i)\log_2(1-y_i)$$

$$C(d_1, y_1, d_2, y_2 | x_1, x_2, w_1, \dots, w_8) = L_1 + L_2$$

In order to use back propagation, we have to find the gradient of the cost function,

$$\nabla C_{w_{1\sim 8}} = \begin{bmatrix} \frac{\partial C}{\partial w_1} \\ \frac{\partial C}{\partial w_2} \\ \frac{\partial C}{\partial w_3} \\ \frac{\partial C}{\partial w_4} \\ \frac{\partial C}{\partial w_5} \\ \frac{\partial C}{\partial w_6} \\ \frac{\partial C}{\partial w_7} \\ \frac{\partial C}{\partial w_8} \end{bmatrix}$$

Same as the previous homework, chain rule is used to figure out the derivative terms. Let us consider all the weights at time t. In addition, the $\ln 2$ is ignored for simplicity.

$$\because \textit{ReLU is used in hidden layer}$$
$$\because \textit{Sigmoid is used in the output layer}$$

$$\frac{\partial C}{\partial w_1} = -\frac{\partial d_1 \log_2 y_1}{\partial w_1} - \frac{\partial (1-d_1)\log_2(1-y_1)}{\partial w_1} - \frac{\partial d_2 \log_2 y_2}{\partial w_1} - \frac{\partial (1-d_2)\log_2(1-y_2)}{\partial w_1}$$

$$= -d_1 \frac{\partial \ln y_1}{\partial y_1}\frac{\partial y_1}{\partial z_1}\frac{\partial z_1}{\partial h_1}\frac{\partial h_1}{\partial q_1}\frac{\partial q_1}{\partial w_1} - (1-d_1)\frac{\partial \ln(1-y_1)}{\partial (1-y_1)}\frac{\partial (1-y_1)}{\partial y_1}\frac{\partial y_1}{\partial z_1}\frac{\partial z_1}{\partial h_1}\frac{\partial h_1}{\partial q_1}\frac{\partial q_1}{\partial w_1}$$
$$- d_2 \frac{\partial \ln y_2}{\partial y_2}\frac{\partial y_2}{\partial z_2}\frac{\partial z_2}{\partial h_1}\frac{\partial h_1}{\partial q_1}\frac{\partial q_1}{\partial w_1} - (1-d_2)\frac{\partial \ln(1-y_2)}{\partial (1-y_2)}\frac{\partial (1-y_2)}{\partial y_2}\frac{\partial y_2}{\partial z_2}\frac{\partial z_2}{\partial h_1}\frac{\partial h_1}{\partial q_1}\frac{\partial q_1}{\partial w_1}$$

$$= -\mathrm{d}_1 \frac{1}{y_1} y_1(1-y_1) w_5 (h_1)' x_1 + (1-d_1)\frac{1}{1-y_1} y_1(1-y_1) w_5 (h_1)' x_1$$
$$- \mathrm{d}_2 \frac{1}{y_2} y_2(1-y_2) w_7 (h_1)' x_1 + (1-d_2)\frac{1}{1-y_2} y_2(1-y_2) w_7 (h_1)' x_1$$

$$= (y_1 - d_1) w_5 (h_1)' x_1 + (y_2 - d_2) w_7 (h_1)' x_1$$

$$\frac{\partial C}{\partial w_2} = (y_1 - d_1) w_5 (h_1)' x_2 + (y_2 - d_2) w_7 (h_1)' x_2$$

$$\frac{\partial C}{\partial w_3} = (y_1 - d_1) w_6 (h_2)' x_1 + (y_2 - d_2) w_8 (h_2)' x_1$$

$$\frac{\partial C}{\partial w_4} = (y_1 - d_1) w_6 (h_2)' x_2 + (y_2 - d_2) w_8 (h_2)' x_2$$

$$\frac{\partial C}{\partial w_5} = (y_1 - d_1) h_1$$

$$\frac{\partial C}{\partial w_6} = (y_1 - d_1)h_2$$

$$\frac{\partial C}{\partial w_7} = (y_2 - d_2)h_1$$

$$\frac{\partial C}{\partial w_8} = (y_2 - d_2)h_2$$

where

$$(h_1)' = \begin{cases} 1, & q_1 > 0 \\ 0, & q_1 \leq 0 \end{cases}$$

$$(h_2)' = \begin{cases} 1, & q_2 > 0 \\ 0, & q_2 \leq 0 \end{cases}$$

After getting all the derivative terms, we can simply update the weights by,

$$w_{t+1,i} = w_{t,i} - \alpha \frac{\partial C}{\partial w_i}$$

where α is the learning rate.

Question 3:

The code was uploaded to GitHub:

https://github.com/DHKLeung/NTUT_Machine_Learning/blob/master/HW5_Q3.ipynb

Note that there is $epsilon = 1 \times 10^{-6}$ for tackling the numerical problem.

```
import numpy as np
import matplotlib.pyplot as plt
import pandas as pd
```

```
"""
Load UCI ML Iris data
Return: data(shape = (150, 4)) and labels(shape = (150, 1)) in numpy array(rank 2)
"""
def load_data(classes):
    data = pd.read_csv('Iris.csv', index_col=0).as_matrix()
    features = data[:, :-1]
    labels = data[:, -1].reshape(-1, 1)
    for class_id in classes:
        labels[labels == class_id[0]] = class_id[1]
    return features.astype(np.float32), labels.astype(np.float32)
```

```
"""
Perform Data Shuffling
Return: shuffled features and labels
"""
def data_shuffling(features, labels):
    shuffle_array = np.random.permutation(features.shape[0])
    features = features[shuffle_array]
    labels = labels[shuffle_array]
    return features, labels
```

```
"""
Splitting the data into train and test set
```

```
Variable: ratio = percentage of data for train set
Return: train set and test set
"""
def data_split(features, labels, ratio=0.7):
    train_end_index = np.ceil(features.shape[0] * 0.7).astype(np.int32)
    train_features = features[:train_end_index]
    train_labels = labels[:train_end_index]
    test_features = features[train_end_index:]
    test_labels = labels[train_end_index:]
    return train_features, train_labels, test_features, test_labels
```

```
"""
Perform standardization for features
Return: standardized features, standard deviation, mean
"""
def standardization(features, exist_params=False, std=None, mean=None, colvar=True):
    features = features.T if not colvar else features
    if not exist_params:
        std = np.std(features, axis=0, keepdims=True)
        mean = np.mean(features, axis=0, keepdims=True)
    standard_features = (features - mean) / std
    return standard_features, std, mean
```

```
"""
Preparation of data, Performing shuffling, splitting, one-hotting
"""
def data_preparation(data, labels, classes):
    labels = np.eye(len(classes), dtype=np.float32)[np.squeeze(labels).astype(int)]
    data, labels = data_shuffling(data, labels)
    train_data = np.empty((0, data.shape[1]), dtype=np.float32)
    train_labels = np.empty((0, labels.shape[1]), dtype=np.float32)
    validation_data = np.empty((0, data.shape[1]), dtype=np.float32)
    validation_labels = np.empty((0, labels.shape[1]), dtype=np.float32)
    for each_class in classes:
        data_class = data[np.argmax(labels, axis=1).astype(np.float32) ==
float(each_class[1])]
        labels_class = labels[np.argmax(labels, axis=1).astype(np.float32) ==
float(each_class[1])]
        a, b, c, d = data_split(data_class, labels_class)
        train_data = np.append(train_data, a, axis=0)
        train_labels = np.append(train_labels, b, axis=0)
        validation_data = np.append(validation_data, c, axis=0)
        validation_labels = np.append(validation_labels, d, axis=0)
    train_data, train_labels = data_shuffling(train_data, train_labels)
    validation_data, validation_labels = data_shuffling(validation_data, validation_labels)
    train_data, std, mean = standardization(train_data)
    validation_data, _, _ = standardization(validation_data, exist_params=True, std=std,
mean=mean)
    return train_data, train_labels, validation_data, validation_labels
```

```
"""
Sigmoid function
"""
def sigmoid(z):
    return 1 / (1 + np.exp(-z).astype(np.float32))
```

```
"""
Derivative of Sigmoid
"""
def sigmoid_diff(a):
    return a * (1 - a)
```

```
"""
ReLU function
"""
def relu(z):
    temp_matrix = (z > 0).astype(np.float32)
    return np.multiply(z, temp_matrix)
```

```
"""
Derivative of ReLU
"""
def relu_diff(z):
    temp_matrix = (z > 0).astype(np.float32)
    return temp_matrix
```

```
"""
Define the architecture of neural networks
"""
def neural_networks(x, w1, w2, b1, b2):
    Z1 = np.dot(x, w1) + b1
    A1 = relu(Z1)
    Z2 = np.dot(A1, w2) + b2
    A2 = sigmoid(Z2)
    return Z1, A1, Z2, A2
```

```
"""
Define the Cost Function, Cross Entropy
"""
def cross_entropy(outputs, ground_truth, epsilon=10e-6):
    cost_of_all_data = -np.multiply(ground_truth, np.log2(outputs + epsilon)) -
np.multiply(1. - ground_truth, np.log2(1. - outputs + epsilon))
    cost = np.mean(np.mean(cost_of_all_data, axis=1), axis=0)
    return cost
```

```
"""
Compute the accuracy
"""
def get_accuracy(predictions, ground_truth):
    return np.mean(np.squeeze(predictions == ground_truth).astype(np.float32))
```

```
"""
Training of neural networks
"""
def training(train_data, train_labels, validation_data, validation_labels,
learning_rate=0.01, epochs=35000, epsilon=10e-6):
    #Initialize the weights and bias#
    w1 = (np.random.randn(2, 2) / np.sqrt(2)).astype(np.float32)
    w2 = (np.random.randn(2, 2) / np.sqrt(2)).astype(np.float32)
    b1 = np.zeros((1, 2))
    b2 = np.zeros((1, 2))

    #Accuracy and cost history#
    accs = list()
    costs = list()
    accs_val = list()
    costs_val = list()

    #Save the initial training cost and accuracy#
    Z1, A1, Z2, A2 = neural_networks(train_data, w1, w2, b1, b2)
    costs.append(cross_entropy(A2, train_labels))
    accs.append(get_accuracy(np.argmax(A2, axis=1), np.argmax(train_labels, axis=1)))

    #Save the initial validation cost and accuracy#
    Z1, A1, Z2, A2 = neural_networks(validation_data, w1, w2, b1, b2)
    costs_val.append(cross_entropy(A2, validation_labels))
    accs_val.append(get_accuracy(np.argmax(A2, axis=1), np.argmax(validation_labels,
axis=1)))

    #Training#
    for i in range(epochs):
        #Feed Forward#
        Z1, A1, Z2, A2 = neural_networks(train_data, w1, w2, b1, b2)

        #Back Propagation and Update weights, bias#
        dZ2 = A2 - train_labels
```

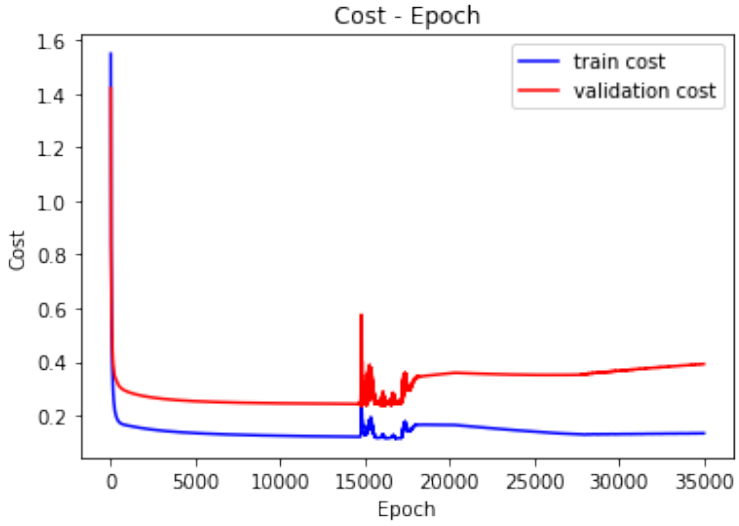
```
        dZ1 = np.multiply(np.dot(dZ2, w2.T), relu_diff(Z2))
        dw2 = np.dot(A1.T, dZ2) / train_data.shape[0]
        dw1 = np.dot(train_data.T, dZ1) / train_data.shape[0]
        db2 = np.sum(dZ2, axis=0)  / train_data.shape[0]
        db1 = np.sum(dZ1, axis=0) / train_data.shape[0]
        w1 = w1 - learning_rate * dw1
        w2 = w2 - learning_rate * dw2
        b1 = b1 - learning_rate * db1
        b2 = b2 - learning_rate * db2

        #Save the initial training cost and accuracy#
        Z1, A1, Z2, A2 = neural_networks(train_data, w1, w2, b1, b2)
        costs.append(cross_entropy(A2, train_labels))
        accs.append(get_accuracy(np.argmax(A2, axis=1), np.argmax(train_labels, axis=1)))

        #Save the initial validation cost and accuracy#
        Z1, A1, Z2, A2 = neural_networks(validation_data, w1, w2, b1, b2)
        costs_val.append(cross_entropy(A2, validation_labels))
        accs_val.append(get_accuracy(np.argmax(A2, axis=1), np.argmax(validation_labels,
axis=1)))
    return w1, w2, b1, b2, costs, accs, costs_val, accs_val
```

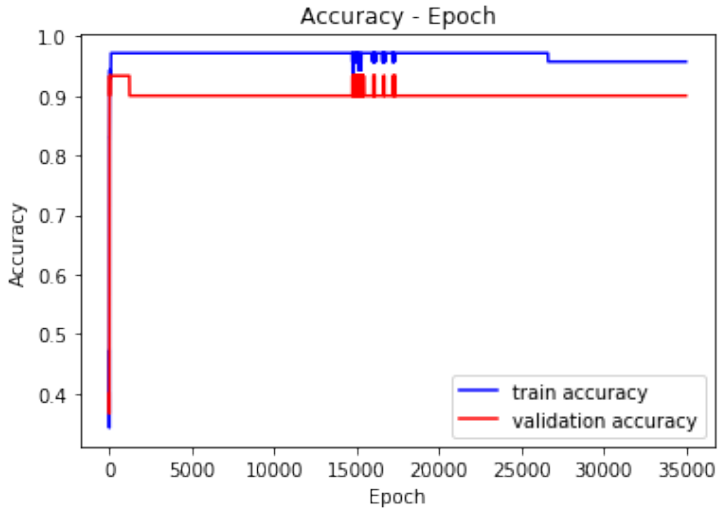
```
"""
Plot the scatter graphs
"""
def plot_scatter(features, labels):
    plt.plot(features[np.squeeze(labels == float(1))][:, 0], features[np.squeeze(labels ==
float(1))][:, 1], 'gx', label='Iris-versicolor')
    plt.plot(features[np.squeeze(labels == float(2))][:, 0], features[np.squeeze(labels ==
float(2))][:, 1], 'bx', label='Iris-virginica')
    plt.title('Petal Length - Petal Width')
    plt.legend(loc='best')
    plt.show()
```

```
"""
Settings
"""
classes = [('Iris-setosa', 0), ('Iris-versicolor', 1), ('Iris-virginica', 2)]
cross_val_times = 10
```

```
data, labels = load_data(classes)
data = data[np.logical_or(np.squeeze(labels == 1), np.squeeze(labels == 2))][:, 2:]
labels = labels[np.logical_or(np.squeeze(labels == 1), np.squeeze(labels == 2))] - 1.
classes = [('Iris-versicolor', 0), ('Iris-virginica', 1)]
cross_val_acc = 0.
for i in range(cross_val_times):
    train_data, train_labels, validation_data, validation_labels = data_preparation(data,
labels, classes)
    w1, w2, b1, b2, costs, accs, costs_val, accs_val = training(train_data, train_labels,
validation_data, validation_labels)
    cross_val_acc += accs_val[-1] / cross_val_times
plt.plot(costs, 'b-', label='train cost')
plt.plot(costs_val, 'r-', label='validation cost')
plt.title('Cost - Epoch')
plt.ylabel('Cost')
plt.xlabel('Epoch')
plt.legend(loc='best')
plt.show()
plt.plot(accs, 'b-', label='train accuracy')
plt.plot(accs_val, 'r-', label='validation accuracy')
plt.title('Accuracy - Epoch')
plt.ylabel('Accuracy')
plt.xlabel('Epoch')
plt.legend(loc='best')
plt.show()
print('Average Accuracy ({} times): {}'.format(cross_val_times, cross_val_acc))
```

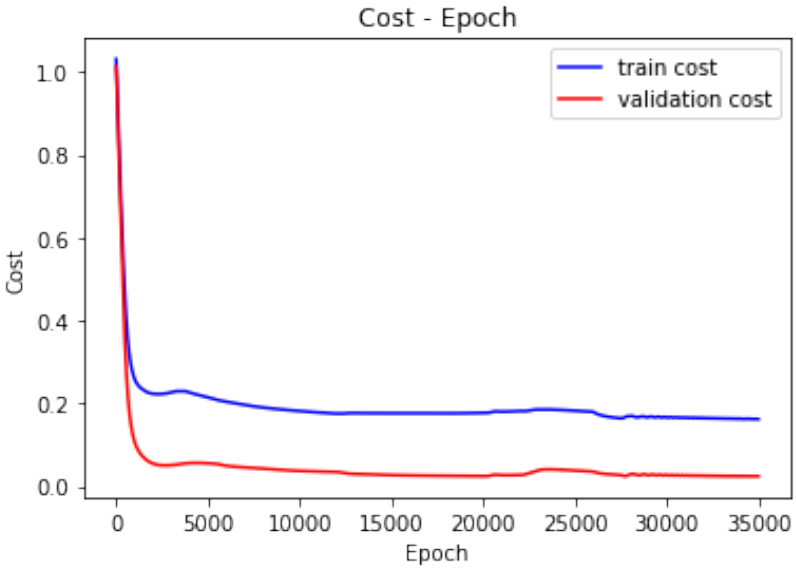
If the learning rate used is the same as the one in Homework #4, the result will be

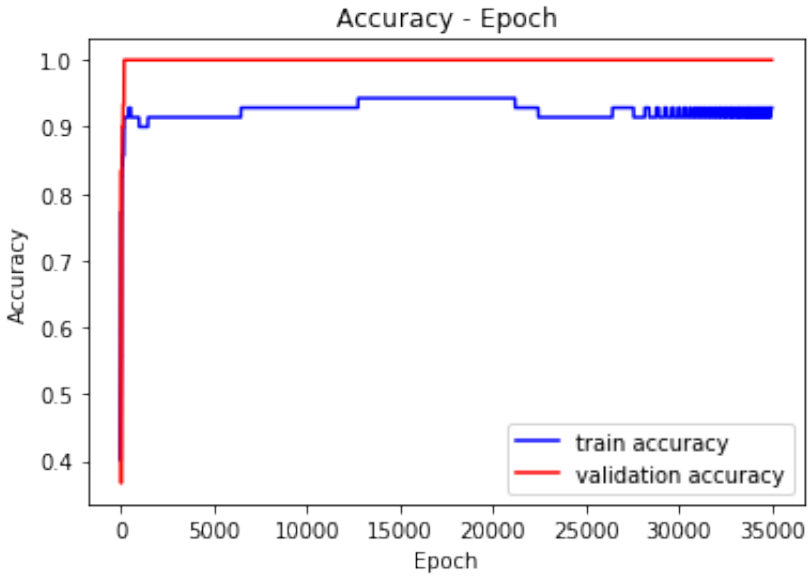




There is overshooting problem and instability.

If learning rate are set as 0.01,





The average accuracy of 10 times is 0.9366666555404662.

By comparing to the Homework #4 used MSE as cost function, we can see that using ReLU and Cross Entropy as cost function, in this case, can boost the convergence in a short time. It works well in classification task.

Question 4:

Assume the input image contains only one channel and one single duple of image. That is, the input is an image of $32 \times 32 \times 1$. Since there are 6 feature maps of 28×28 in the layer C1, this implies that there are 6 kernels used in C1 layer.

$$\because kernel\ size = 5 \times 5$$

$$\because stride = 1$$

$$\because no\ zero\ padding$$

Since each pixel in a feature map is computed by 25 connections. Therefore, there are a total of $28 \times 28 \times 25 = 19600$ connections for one feature map. For 6 feature maps, there are $19600 \times 6 = 117600$ connections in total.

The number of weights is $5 \times 5 \times 6 = 150$.

Question 5:

The code was uploaded to GitHub:

https://github.com/DHKLeung/NTUT_Machine_Learning/blob/master/HW5_Q5.ipynb

```
import numpy as np
import matplotlib.pyplot as plt
def conv(input_plane, kernel):
```

```
    kernel_size = kernel.shape[0]
    input_plane_size = input_plane.shape[0]
    total_move = input_plane_size - kernel_size + 1
    feature_map_before_activation = np.zeros((total_move, total_move))
    for i in range(total_move):
        for j in range(total_move):
            sub_input_plane = input_plane[i:i + total_move, j:j + total_move]
            feature_map_before_activation[i, j] = np.sum(np.multiply(sub_input_plane,
kernel))
    return feature_map_before_activation
```

```
def relu(feature_map_before_activation):
    temp_matrix = (feature_map_before_activation > 0).astype(np.float64)
    feature_map = np.abs(np.multiply(feature_map_before_activation, temp_matrix))
    return feature_map
```

```
def create_data():
    input_plane_1 = np.array([
        [6., 0., -4., 0., 1.],
        [4., 4., 0., 2., 1.],
        [3., -7., 1., 4., 2.],
        [-2., 2., 1., -4., 2.],
        [5., 1., 2., 4., -1.]
    ])
    input_plane_2 = np.array([
        [3., 1., -4., 0., -1.],
        [3., 0., 3., 4., -1.],
        [3., 7., -2, -2, 2.],
        [-5., -2., 2., 0., -2],
        [2., 1., -1., 1., 2.]
    ])
    kernel_1 = np.array([
        [3., -1, 2.],
        [-2., 1., -3.],
        [-2., 0., 3.]
    ])
    kernel_2 = np.array([
        [0., -1., 0.],
        [-1., 6., -1.],
        [0., -1., 0.]
    ])
    return input_plane_1, input_plane_2, kernel_1, kernel_2
```

```
input_plane_1, input_plane_2, kernel_1, kernel_2 = create_data()
sub_feature_map_1_before_activation = conv(input_plane_1, kernel_1)
print('Sub Feature Map 1 before activation:
\n{}\n'.format(sub_feature_map_1_before_activation))
sub_feature_map_2_before_activation = conv(input_plane_2, kernel_2)
print('Sub Feature Map 2 before activation:
\n{}\n'.format(sub_feature_map_2_before_activation))
feature_map_before_activation = sub_feature_map_1_before_activation +
sub_feature_map_2_before_activation
print('Feature Map before activation: \n{}\n'.format(feature_map_before_activation))
feature_map = relu(feature_map_before_activation)
print('Feature Map: \n{}\n'.format(feature_map))
```

Sub Feature Map 1 before activation:

$$\begin{bmatrix} 3 & 16 & -7 \\ -1 & 3 & 0 \\ 17 & 5 & -16 \end{bmatrix}$$

Sub Feature Map 2 before activation:

$$\begin{bmatrix} -14 & 20 & 24 \\ 43 & -22 & -16 \\ -17 & 17 & 1 \end{bmatrix}$$

Feature Map before activation:

$$\begin{bmatrix} -11 & 36 & 17 \\ 42 & -19 & -16 \\ 0 & 22 & -15 \end{bmatrix}$$

Feature Map:

$$\begin{bmatrix} 0 & 36 & 17 \\ 42 & 0 & 0 \\ 0 & 22 & 0 \end{bmatrix}$$