ELEC 4511/5511

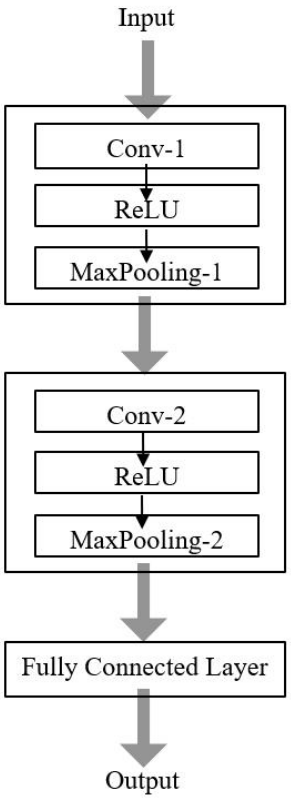
Lab 4

Build your own Convolutional Neural Network using NumPy

In this lab, you will use python and NumPy to design a simple two-layer CNN model , and test the handwritten digit recognition model by loading pre-trained weights. The template and most of the code have been provided to you below, and the contents of each part will be introduced separately. For details of the specific steps, please refer to the relevant content in the lecture.

1. Model

The model in this lab is composed of two sets of convolutional neural network modules and a fully connected layer as shown in the figure below. The CNN module contains a convolutional layer and a pooling layer. And the output of the convolutional layer needs to go through an activation function and then input to the pooling layer. Here we use Relu as the activation function and choose MaxPooling as the pooling scheme.




```
# define the forward process
def forward(X):

    conv1      =  conv_forward(X.astype(np.float64),conv1_w,conv1_b, padding=1)
    conv1_relu =  relu_forward(conv1)
    maxp1      =  max_pooling_forward(conv1_relu.astype(np.float64))

    conv2      =  conv_forward(maxp1,conv2_w,conv2_b, padding=1)
    conv2_relu =  relu_forward(conv2)
    maxp2      =  max_pooling_forward(conv2_relu.astype(np.float64))

    flatten    =  flatten_forward(maxp2)
    y          =  fc_forward(flatten,fc_w,fc_b)

    return y
```

2. Weights and Bias

The weight and bias of each layer are taken from a model designed based on the pytorch library and trained using GPU. The training set uses MNIST handwritten digit recognition data. You can download them directly from Canvas (conv1_w, conv1_b, conv2_w, conv2_b, fc_w, fc_b). You need to put them in your project path and load them with the following code. You can understand their shape through the comments after each line of code. This is important for you to write code for other functions.

```python
import numpy as np

conv1_w =np.load("conv1_w.npy") # (16, 1, 3, 3)   Kernal size: 3x3x1, 16 filters in total
conv1_b =np.load("conv1_b.npy") # (16,)
conv2_w =np.load("conv2_w.npy") # (32, 16, 3, 3)   Kernal size: 3x3x16, 32 filters in total
conv2_b =np.load("conv2_b.npy") # (16,)
fc_w =np.load("fc_w.npy") # (10, 1568)
fc_b =np.load("fc_b.npy") # (10,)
```

3. Convolution Layer (your work)

Since we do not need to train the model, the convolutional layer we designed here only needs to complete the forward propagation, so the design difficulty is greatly reduced. The template is shown below.

```python
def conv_forward(z, K, b, padding=0):
    """
     Multi-channel convolution forward process
    :param z: Convolutional layer matrix, shape (N,C,H,W), N is the batch_size, C is the number of channels
    :param K: Convolution kernel, shape (D,C,k1,k2), C is Number of input channels, D is Number of output channels
    :param b: bias, shape (D,)
    :param padding: paddingre
    :return: conv_z: Convolution result, shape [N,D,oH,oW]
    """

    return conv_z
```

For detailed information about the specific design steps of this function, please refer to the relevant content in the lecture.

4. Pooling Layer (your work)

Here we design the pooling layer as MaxPooling, and only need to consider forward propagation. Please note that *stride* must be an even number here. The template is shown below.

```python
def max_pooling_forward(z, stride=2):
    """
     Maximum pooling forward process
    :param z: Convolutional layer matrix, shape (N,C,H,W), N is the batch_size, C is the number of channels
    :param strides: length of step
    :return:
    """

    return pool_z
```

For detailed information about the specific design steps of this function, please refer to the relevant content in the lecture.

5. Other functions (have been designed for you)

```
def relu_forward(z):
    """
     relu() activation
    :param z: Layer to be activated
    :return: Result after activation
    """
    return np.maximum(0, z)

def flatten_forward(z):
    """
    Flatten the multidimensional array
    :param z: multidimensional array,shape (N,d1,d2,..)
    :return:
    """
    N = z.shape[0]
    return np.reshape(z, (N, -1))

def fc_forward(z, W, b):
    """
    Forward propagation of the fully connected layer
    :param z: The output of the current layer,shape (N,ln)
    :param W: The weight of the current layer
    :param b: The bias of the current layer
    :return: Output of the next layer
    """
    return np.dot(W, z.transpose())+np.expand_dims(b,1).repeat(z.shape[0],axis=1)
```

6. Code for test

The test code is also provided to you. "mnist.pkl.gz" is the test images package you need to use for this lab. Please put them in your project path. Every time, the test script can randomly select 5 images from the testset for testing. Please feel free to modify this code if you need.

```
import gzip, pickle, sys
import matplotlib.pyplot as plt

f = gzip.open('mnist.pkl.gz', 'rb')
if sys.version_info < (3,):
    (X_train, y_train), (X_test, y_test) = pickle.load(f)
else:
    (X_train, y_train), (X_test, y_test) = pickle.load(f, encoding="bytes")


idx=np.random.choice(X_test.shape[0],5)
x,y=X_test[idx],y_test[idx]
x = np.reshape(x,(-1,1,28,28))
y_predict = forward(x)

for i in range(5):
    plt.figure(figsize=(3,3))
    plt.imshow(np.reshape(x[i],(28,28)), cmap='gray')
    plt.show()
    print("y_true:{},y_predict:{}".format(y[i],np.argmax(y_predict[:,i])))
```

Finally, all in all, the task of this experiment is to complete the design of two functions: **conv_forward** and **max_pooling_forward**.

Canvas submission: Please submit your report in "**.pdf**" file format and compress your codes into a "**.zip**" file.

What needs to be included in your report:

1. Screenshots of the results you get after running the program.
2. Copy and paste your code. And write comments for each function/method.
3. Please write a short analysis of each problem, mainly explaining how you think about the design of the function, what troubles you encountered during the design process, and how you finally solved them, etc.

What needs to be included in your "**.zip**" file:

Your Python code (.py file) for each problem.