1 Outline

In this assignment, you are asked to design a convolutional layer and maxpool layer.

2 Specification

In this assignment, you are asked to write a Python code, writing methods for classes implementing convolutional and maxpool layer for neural networks.

We will make use of PyTorch tool for the implementation. The main reason is to simplify the back-propagation operation. Backpropagataion is used to verify whether your neural network layer is correctly implemented. Note PyTorch library can be used by

```
import torch
```

The implementation details are provided in the following sections.

3 Implementation notes: convolutional and maxpool layer

There are classes defined for each layer:

1. `nn_convolutional_layer`
2. `nn_maxpool_layer`

3.1 class `nn_convolutional_layer`

`class nn_convolutional_layer(f_height, f_width, input_size, in_ch_size, out_ch_size)`

Parameters:

- `f_height`: height of convolutional filter
- `f_width`: width of convolutional filter
- `input_size`: input size (height/weight)
- `in_ch_size`: input channel size (depth)
- `out_ch_size`: output channel size (# of filters)

The class method to be implemented: `forward`.

3.1.1 `forward` method

Forward method has the following format:

`nn_convolutional_layer.forward(x)`: Performs a forward pass of convolutional layer. Input parameters are as follows:

Parameters: x:*tensor*.

4-dimensional `torch.tensor` input map to the convolutional layer. Each dimension of x has meaning

```
x.shape=(batch_size, input_channel_size, in_height, in_width)
```

For example, if `x.shape` returns $(16, 3, 32, 32)$, it means that `x` is an image/activation map of size 32×32, with three input channels (like RGB), and there are 16 of them treated as one batch (as in mini-batch SGD).

Returns: `out`:*tensor*.

4-dimensional `torch.tensor` output feature map as a result of convolution. Each dimension of `out` has meaning

```
out.shape=(batch_size, num_filter, out_height, out_width)
```

For example, if `out.shape` will return $(16, 8, 28, 28)$, for the input `x` with shape $(16, 3, 32, 32)$: if the convolutional layer has each filter size $5 \times 5 \times 3$, and there are a total of 8 filters. Note that the input width and height changes from 32 to 28. Batch size remains the same.

The forward pass of convolution will be done without zero-padding and stride of 1. So, if the input map has width N, and the filter size is given by F, the width of output map will be $N - F + 1$.

3.1.2 Tips for convolution operations

Implementing convolutional layer can be tricky. It may be implemented in many ways, but if it is not done properly, the operation can be slow. This is because our data is usually high-dimensional. We provide a useful function to implement multi-dimensional convolution called `view_as_windows` which is implemented in `hw4.py`. It is similar to the function `https://scikit-image.org/docs/dev/api/skimage.util.html#skimage.util.view_as_windows`: the only difference is that, our version uses `tensor` objects from PyTorch. Check out the description of the function in the provided link for usage guidelines.

`view_as_windows` function provides a *rearranged* view of the input map, which makes it easier to perform convolution with filters. Our approach for convolution is

- Rearrange the input map using `view_as_windows`.
- perform **inner product** operation (that is, matrix multiplication) between input map and the filter.

Example: For simplicity, let's consider 2-D case. Consider an input map x given by

$$\begin{bmatrix} 1 & 2 & 3 \\ 4 & 5 & 6 \\ 7 & 8 & 9 \end{bmatrix}$$

Suppose we would like to do convolution of x and 2-by-2 filter

$$\begin{bmatrix} 1 & 1 \\ 1 & 1 \end{bmatrix}$$

The convolution output will be 2-by-2. If you manually perform convolution by sliding the filter over x, you can check that the result is

$$\begin{bmatrix} 12 & 16 \\ 24 & 28 \end{bmatrix}$$

This is done by, first use `view_as_windows` to create a view into x through 2-by-2 window in a sliding manner. This is done by calling

```
y=view_as_windows(x,(2,2))
```

This will create y, which is a 2-by-2 (output map size) where each output element is a 2-by-2 (filter size) sliding view of input x. Thus the shape of y is (2,2,2,2). Then y is reshaped to (2,2,4) by "flattening" the windowed view – this is then inner-product with filter which is reshaped to (4,1).

Below is the example Python code.

```
x=torch.arange(9, dtype=torch.double).reshape(3,3)+1
print('x.shape',x.shape)
print('x')
print(x)

# filter
filt=torch.ones((2,2))
filt=filt.reshape((-1,1))
print('reshaped filt', filt)

y=view_as_windows(x,(2,2))
print('y.shape',y.shape)
print('y')
print(y)

y=y.reshape((2,2,-1))
print('reshaped y')
print(y)

# perform convolution by matrix multiplication
print('convolution result:')
print('y, filt size', y.shape, filt.shape)
result=torch.matmul(y, filt)
result=torch.squeeze(result,axis=2)

print('result is: ', result)
```

This will produce result output:

```
x.shape torch.Size([3, 3])
x
tensor([[1., 2., 3.],
        [4., 5., 6.],
        [7., 8., 9.]])
reshaped filt tensor([[1.],
        [1.],
        [1.],
        [1.]])
y.shape torch.Size([2, 2, 2, 2])
y
tensor([[[[1., 2.],
          [4., 5.]],

         [[2., 3.],
          [5., 6.]]],


        [[[4., 5.],
          [7., 8.]],

         [[5., 6.],
          [8., 9.]]]])
reshaped y
tensor([[[1., 2., 4., 5.],
         [2., 3., 5., 6.]],

        [[4., 5., 7., 8.],
         [5., 6., 8., 9.]]])
convolution result:
y, filt size torch.Size([2, 2, 4]) torch.Size([4, 1])
result is:  tensor([[12., 16.],
        [24., 28.]])
```

Also see Figure 1 for this example.

If possible you can use other functions to do convolution – as long as they are not pre-built public libraries from Pytorch/Tensorflow/Keras, etc. Also, you can define other functions as many as you want, if needed.

3.2 class `nn_maxpool_layer`

Parameters:

- `pool_size`: window size of pooling
- `stride`: stride of pooling

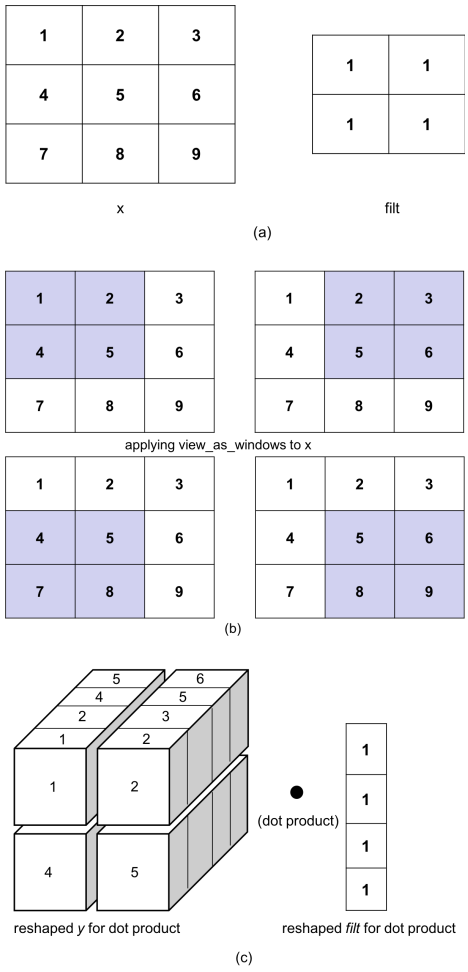

Figure 1: (a) Input map x and filter to perform convolution. (b)apply `view_as_window` to x to create a sliding-window view y into x. (c) reshape y and filter, and perform a dot product for the convolution.

The objects of class `nn_maxpool_layer` are created as follows:

```
maxpool_obj=nn_maxpool_layer(pool_size=2, stride=2)
```

Here we pass two parameters, `stride` is the stride of `pool_size` is the size of window over which we take the max pooling. In this example, we slide 2-by-2 window, with stride 2, and take the maximum value from the 2-by-2 windowed view of input.

The following methods `forward` need to be implemented.

3.2.1 `forward` method

Forward method has the following format:

`nn_maxpool_layer.forward(x)`: Performs a forward pass of convolutional layer. Input parameters are as follows:

Parameters: x:*tensor*.

4-dimensional `torch.tensor` input map to the convolutional layer. Each dimension of x has meaning

```
x.shape=(batch_size, input_channel_size, in_height, in_width)
```

For example, if `x.shape` returns $(16, 3, 32, 32)$, it means that `x` is an image/activation map of size 32×32, with three input channels (like RGB), and there are 16 of them treated as one batch (as in mini-batch SGD).

Returns: `out`:*tensor*.

4-dimensional `torch.tensor` output feature map as a result of convolution. Each dimension of `out` has meaning

```
out.shape=(batch_size, input_channel_size, out_height, out_width )
```

For example, if `out.shape` will return $(16, 3, 16, 16)$, for the input `x` with shape $(16, 3, 32, 32)$, assuming that the `stride`=2 and `pool_size`=2.

3.2.2 Tips for maxpool operations

Similar to convolutional layer, it is convenient to use `view_as_windows` function. The function takes the parameter `step`, which is equivalent to the stride. So you can create, for example 2-by-2 windowed view into the input, with the stride 2.

Below is an example. For simplicity, let's consider 2-D case. Consider an input map x given by

$$\begin{bmatrix} 1 & 2 & 3 & 4 \\ 5 & 6 & 7 & 8 \\ 9 & 10 & 11 & 12 \\ 13 & 14 & 15 & 16 \end{bmatrix}$$

Suppose we would like to perform max pooling over 2-by-2 sliding window with stride of 2. Below is the example Python code.

```
import torch

x=torch.arange(16).reshape(4,4)+1
print('x.shape',x.shape)
print('x', x)

y=view_as_windows(x,(2,2),step=2)
print('y.shape',y.shape)
print('y', y)
```

This will produce result output:

```
x.shape torch.Size([4, 4])
x tensor([[ 1,  2,  3,  4],
        [ 5,  6,  7,  8],
        [ 9, 10, 11, 12],
        [13, 14, 15, 16]])
y.shape torch.Size([2, 2, 2, 2])
y tensor([[[[ 1,  2],
          [ 5,  6]],

         [[ 3,  4],
          [ 7,  8]]],


        [[[ 9, 10],
          [13, 14]],

         [[11, 12],
          [15, 16]]]])
```

4 What to submit

You will be given file `hw4.py` . **You are asked to complete Q1–Q2 parts in** `hw4.py`**.**

- Submit the modified Python file `hw4.py`.
- Upload your files at Blackboard before deadline. (Please submit the file in time, no late submission will be accepted).

5 How your module will be graded

The accuracy of your implementation of convolutional/pooling layer will be measured by comparing their outputs with `Conv2d` and `MaxPool2d` modules from PyTorch. You can read their documents in the link

`https://pytorch.org/docs/stable/generated/torch.nn.Conv2d.html`

`https://pytorch.org/docs/stable/generated/torch.nn.MaxPool2d.html`

We will create a random PyTorch tensor, and feed it to your module and test module from PyTorch. Their outputs are compared, and if your module is correctly designed, you will see the following output stating that the accuracy is close to 100%.

```
conv test
pooling test
```

```
accuracy results
forward accuracy tensor(100.) %
pooling accuracy tensor(100.) %
```

As a general advice, first test your module with very simple inputs and filters – so that you can easily calculate the forward by hand. Start with setting batch size and input channel size, filter numbers to 1, and see if you can get correct results.

6 Grading

- 20 points for Q1–Q2 done correctly. Specifically, if each of 2 accuracy tests gets more than 90%, you get 10 points for each. So the total is 20 points.
- 4 points for each of 2 accuracy test results between 50–90%.
- 2 points for each of 2 accuracy test results below 50%.

In the blackboard, you can upload your files as many times as you like, before the deadline. The last uploaded file will be used for grading. After deadline, the submission menu will be closed and you will not be able to make submission.