Exercise 1 - Transposed Convolution Output Sizes

What is the size of the output for a input tensor and a transposed convolutional layer if the parameters are given as follows (assume the number of channels is given in the first dimension).

(a) Input tensor size: $3 \times 2 \times 2$

Transposed convolution: 3×3 kernel, stride 1, output channels: 2

(b) Input tensor size: $3 \times 5 \times 5$

Transposed convolutional: 2×2 kernel, stride 2, output channels: 4

(c) Input tensor size: $c_{in} \times h \times w$

Transposed convolutional: 3×3 kernel, stride 2, output channels: 2

(d) Input tensor size: $c_{in} \times h \times w$

Transposed convolutional: $h_k \times w_k$ kernel, stride s, output channels: c_{out}

Solution

(a) Output tensor size: $2 \times 4 \times 4$

(b) Output tensor size: $4 \times 10 \times 10$

(c) The first dimension is always the number of input channels, i.e. c_{out}. The output for the first value of the input tensor is of size 3×3. For all remaining values, the stride determines the additional values of the ouput. Stride 2 means, that we need two values for each additional value. Thus, the output height h_{out} and width are w_{out}:

$$h_{out} = 3 + (h-1) \cdot 2 = 2h + 1$$
$$w_{out} = 3 + (w-1) \cdot 2 = 2w + 1$$

Thus, the resulting size is $2 \times (2h+1) \times (2w+1)$.

(d) This generalization requires a similar consideration and we can basically reuse the equations from above:

$$h_{out} = h_k + (h-1) \cdot s$$
$$w_{out} = w_k + (w-1) \cdot s$$

Exercise 2 - Transposed Convolution Parameter Sizes

What is the number of learnable parameters for each of the following transposed convolution layers defined in PyTorch. Try to calculate those by hand first and use pytorch later to verify your results.

(a) `nn.ConvTranspose2d(in_channels=3, out_channels=2, kernel_size=3, stride=1)`

(b) `nn.ConvTranspose2d(in_channels=3, out_channels=10, kernel_size=3, stride=2)`

(c) `nn.ConvTranspose2d(in_channels=3, out_channels=2, kernel_size=4, stride=5)`

(d) `nn.ConvTranspose2d(in_channels=3, out_channels=4, kernel_size=3, stride=23)`

Solution

The number of parameters is the product of input channels, output channels and kernel size. For each output channel, there is one bias parameter (if `bias=True` which is default in PyTorch). The stride does not matter for the number of paramters.

(a) $3 \cdot 2 \cdot 9 + 2 = 56$
(b) $3 \cdot 10 \cdot 9 + 10 = 280$
(c) $3 \cdot 2 \cdot 16 + 2 = 98$
(d) $3 \cdot 4 \cdot 9 + 4 = 112$

Exercise 3 - Transposed Convolution by Hand

You are given an input matrix X (consisting of a single channel) and a kernel K as follows:

$$X = \begin{pmatrix} 1 & 0 & 2 \\ 2 & 3 & 0 \\ -1 & 0 & 3 \end{pmatrix}, \quad K = \begin{pmatrix} 1 & 0 \\ 1 & 2 \end{pmatrix}$$

(a) Compute the transposed convolution by hand assuming stride 2.

(b) Compute the transposed convolution by hand assuming stride 1.

(c) Use PyTorch to verify your answers.

(d) Implement the transposed convolution in PyTorch without using its own implementaiton. You can assume no bias term a square kernel and no separate batch dimension. I.e. your task is to implement the following function

```
def conv_transpose2d(inp, weight, stride=1):
    # inp - input of shape (C_in, H, W)
    # weight - kernel of shape (C_in, C_out, K, K)
    # stride - stride of the transposed convolution
    # RETURNS
    # output - output of shape (C_out, H_out, W_out)
    #
    # YOUR CODE HERE
    return output
```

Solution

(a) The resulting output tensor is of shape 6×6 and is given by:

$$Y = \begin{pmatrix} 1 & 0 & 0 & 0 & 2 & 0 \\ 1 & 2 & 0 & 0 & 2 & 4 \\ 2 & 0 & 3 & 0 & 0 & 0 \\ 2 & 4 & 3 & 6 & 0 & 0 \\ -1 & 0 & 0 & 0 & 3 & 0 \\ -1 & -2 & 0 & 0 & 3 & 6 \end{pmatrix}$$

(b) The resulting output tensor is of shape 4×4 and is given by:

$$Y = \begin{pmatrix} 1 & 0 & 2 & 0 \\ 3 & 5 & 2 & 4 \\ 1 & 7 & 9 & 0 \\ -1 & -2 & 3 & 6 \end{pmatrix}$$

(c) We can verify our answers with PyTorch with the following commands:

```
nn.functional.conv_transpose2d(X.unsqueeze(0), K, stride=2)

nn.functional.conv_transpose2d(X.unsqueeze(0), K)
```

The call to `.unsqueeze(0)` can be left out if `X` has a batch dimension and is already a 4D tensor.

(d) A potential implementaiton (that isn't optimized in any way) of transposed convolutions looks like this:

```
def conv_transpose2d(input, weight, stride=1):
  # input - input of shape (C_in, H, W)
  # weight - kernel of shape (C_in, C_out, K, K)
  # stride - stride of the transposed convolutio
  # RETURNS
  # output - output of shape (C_out, H_out, W_out)
  (c_in, h_in, w_in) = X.size()
  (c2_in, c_out, k, k2) = K.size()

  assert c_in == c2_in, "Number of input channels must match"
  assert k == k2, "Kernel must be square"

  h_out = (h_in - 1) * stride + k
  w_out = (w_in - 1) * stride + k
  output = torch.zeros((c_out, h_out, w_out))

  for c_cur_in in range(c_in):
    for c_cur_out in range(c_out):
      for h in range(0, h_in):
        for w in range(0, w_in):
          output[c_cur_out, h*stride:h*stride+k, w*stride:w*stride+k] \
              += weight[c_cur_in, c_cur_out,:,:] * input[c_cur_in,h,w]

  return output
```