Assignment 3

Arturo Abril Martinez (2813498)

December 2024

1 Answers

question 1 The learnable weights of the conv layer are represented as a tensor of shape:

`(output_channels, input_channels, kernel_height, kernel_width)`

This is the set of *filters* of the convolutional layer. More explicitly (and excluding the bias terms) we will have `W0, W1, W2, ...` and so on up until the number given by `output_channels`. The total number of parameters will then be:

`kernel_width * kernel_height * input_channels * output_channels.`

The pseudocode to implement the convolution is:

```
Y = \
torch.zeros((batch_size, output_channels, output_height, \
    output_width))

for b in range(batch_size):
    for ch in range(output_channels)
        for row in range(output_height):
            for col in range(output_width):
                Y[b,ch,row,col] = \
                ( \
                    X[b,:,row*S:hk+row*S,col*S:wk+col*S] \
                    * W[ch,:,:,:]
                ).sum()
return Y
```

Where `S` represents the stride and `hk` and `wk` the kernel (or filter) height and width respectively. Note that the multiplication inside the `sum()` is element-wise, and in the pseudocode `W` is the set of all weight tensors, so that `W[0,:,:,:] = W0`. *Note: I assume the input images are already padded, if they need to be.*

question 2 Given that the size of the input is (C_{in}, H_{in}, W_{in}), we can calculate the size of the output as:

$$W_{out} = \frac{W_{in} - K + 2P}{S} + 1$$

H_{out} can be calculated using the same formula by symmetry, changing $W_{in} \to H_{in}$. The size of the output is then $(C_{out}, H_{out}, W_{out})$ where C_{out} can be chosen to be any number. K, S and P are the kernel size, stride and padding respectively[1].

question 3 Below is the pseudocode for the unfold operation. `p`, `k` and `S` represent the total number of patches, number of values per patch and stride respectively. `w` (`wk`) and `h` (`hk`) are the input (kernel) width and height respectively. *Note: I assume the input images are already padded, if needed to be.*

```
Y = torch.zeros((batch_size, p, k))
for b in range(batch_size):
    # iterate over rows in the output
    # each row will be a flattened image patch
    for row in range(p):
        # select patch
        i = row//w_out  # transition row every `w_out` patches
        j = row%(w-wk+1) # when `row` exceeds maximum, start over
        start_row, start_col = i*S, j*S
        patch = X[
            b, :, start_row : start_row+hk, \
            start_col : start_col+wk
        ]

        # flatten patch
        counter = 0
        for ch_patch in range(input_channels):
            for row_patch in range(hk):
                for col_patch in range(wk):
                    Y[b, row, counter] = \
                    patch[ch_patch, row_patch, col_patch]
                    counter += 1
# This way the output will have size (batch_size, p, k)
return Y
```

question 4 To calculate the backward of the con2d operation, we first present some notation. We first present all the tensors involved and what each of their indices represent. Later, we will use dummy indices for the caclculations, always

[1] I have ommited the batch size in the calculation. Of course, for every tensor in the input batch we would have an output tensor, which dimensions can be calculated with the formula provided.

keeping in mind what each of them represent, which will be determined by their postion. First, the constants:

- b, c, h, w: Batch size, input channels, input width and input height.
- h_k, w_k: Kernel height and kernel width.
- $c_{out}, h_{out}, w_{out}$: Output channels, output height and output width.
- p: Total number of patches. Resulting from doing $h_{out} * w_{out}$.
- k: Total number of values per patch. Resulting from doing $c * h_k * w_k$

Now we define the tensors involved. The indices are now named with what they represent. We will later have to remember this:

- Input batch: $\mathbf{X} = X_{bchw}$
- Unfolded input: $\mathbf{U} = U_{bkp}$
- Unfolded input transposed: $\tilde{\mathbf{U}} = \tilde{U}_{bpk} = \mathbf{U}.\text{transpose}(1, 2)$
- Kernel matrix: $\mathbf{W} = W_{c_{out}ch_kw_k}$
- Kernel matrix reshaped and transposed: $\tilde{\mathbf{W}} = \tilde{W}_{kc_{out}} =$
  $= \mathbf{W}.\text{reshape}(c_{out}, c * h_k * w_k).\text{transpose}()$
- Output without reshaping: $\mathbf{Y}' = Y'_{bpc_{out}} = \sum_k \tilde{U}_{bpk}\tilde{W}_{kc_{out}}$
- Output: $\mathbf{Y} = Y_{bc_{out}h_{out}w_{out}} =$
  $= \mathbf{Y}'.\text{transpose}(1, 2).\text{reshape}(b, c_{out}, h_{out}, w_{out})$

Before calculating $\mathbf{W}^\nabla$ and $\mathbf{X}^\nabla$ we present some useful derivatives. Note that there will be matrices which elements have a one to one correspondence, but their shapes are different. This is the case for example of $\mathbf{Y}$ and $\mathbf{Y}'$. When doing derivatives of one w.r.t. the other, we will have simple deltas for the dimensions that match, and "special" deltas for those dimensions that not match, but we still have a one to one correspondence of elements. This will become clear in the following derivatives.

$$\frac{\partial Y_{abcd}}{\partial Y'_{efg}} = \delta_{ae}\delta_{bg}\delta_{cd,f} \;\; ; \;\; \frac{\partial Y'_{efg}}{\partial \tilde{W}_{hi}} = \tilde{U}_{efh}\delta_{gi} \;\; ; \;\; \frac{\partial \tilde{W}_{hi}}{\partial W_{abcd}} = \delta_{ia}\delta_{h,bcd}$$

The deltas of the type $\delta_{ab,c}$ appear when we are dealing with different shapes but there is still a one to one correspondence of elements. In this case we want to pick the element ab that corresponds to c[2].

[2]If this is not still clear, imagine that we have a tensor $\mathbf{X}$ with elements X_{11}, X_{12}, X_{21} and X_{22}. We can "flatten" this into a vector of four values $\mathbf{x}$. X_{11} corresponds to x_1, X_{12} to x_2 and so on. The derivative $\partial X_{ab}/\partial x_c = \delta_{ab,c}$.

A Appendix