Advanced Deep Learning - Exercise 1

Fredrik Reinholdsen -frerei-6

April 2020

1 Theoretical Task

In this theoretical exercise we wish to follow the path of an imagined input through a convolutional neural network, by doing all of the matrix operations by hand.

1.1 Convolution

The first par is the actual convolution step. We have some imagined input I, and a convolutional kernel k defined as the following:

$$\mathrm{I} = \begin{bmatrix} 1 & 1 & 1 & 1 \\ 1 & 1 & 2 & 1 \\ 1 & -3 & -4 & 1 \\ 1 & 1 & 1 & 1 \end{bmatrix} \text{ and } \mathrm{k} = \begin{bmatrix} 0 & 1 & 0 \\ 1 & 2 & 1 \\ 0 & 1 & 1 \end{bmatrix}$$

More specifically we wish to do a *same* convolution meaning that the size of the input matrix I is preserved after the convolution. Meaning the size of the output matrix will also be $4x4$, the same as I. We do the convolution by sliding the flipped kernel matrix k over I padded with a margin of zeros around it, moving it one step at them time and doing element wise multiplication at each step.

$$C = I *_{same} k \tag{1}$$

So the elements of C are the following:

$$C_{i,j} = \sum_{n=i}^{i+2} \sum_{l=j}^{j+2} I'_{n,l} \cdot k'_{n,l} \quad for \;\; i = 1,2,3,4 \;\; j = 1,2,3,4 \tag{2}$$

Where I' denotes the zero-padded I matrix that has a boundary of zeroes around it, and k' denotes the flipped version of the kernel k, which in this case is just equal to k, because k is flip symmetric. If we apply the last equation all values of i and j, we find that the convolution matrix is the following.

$$C = I *_{same} k = \begin{bmatrix} 4 & 5 & 6 & 4 \\ 5 & 3 & 3 & 6 \\ 1 & -7 & -7 & 0 \\ 4 & 1 & 0 & 4 \end{bmatrix} \tag{3}$$

1.2 Non Linearity

Next we have some form of activation function that applies some form of non-linearity. In our case we wish to calculate the output of what happens if we apply the ReLU function to the output from the convolutional layer C.

$$ReLU(x) = \begin{cases} x, & \text{if } x > 0 \\ 0, & \text{otherwise} \end{cases} \tag{4}$$

We define C' as $ReLU(C)$ and so we get the following.

$$C' = ReLU(C) = \begin{bmatrix} 4 & 5 & 6 & 4 \\ 5 & 3 & 3 & 6 \\ 1 & 0 & 0 & 0 \\ 4 & 1 & 0 & 4 \end{bmatrix} \tag{5}$$

1.3 Max Pooling

Next we have a so called Max pooling layer. Which pools neurons of the feature maps reducing the dimensionality. Max pooling is simply selecting the mapping a collection of values from the feature maps to the max value of that collection of values. More specifically we wish to do a (2,2) max pooling with a stride of (2,2) which is going to result in a pooled output that is 2x2. So in our case if $Y = MaxPool(X)$ this can be described by

$$Y_{i,j} = max(X_{2(i-1),2(j-1)}, X_{2(i-1)+1,2(j-1)}, X_{2(i-1),2(j-1)+1}, X_{2(i-1)+1,2(j-1)+1}) \tag{6}$$

For $i = 1, 2$ and $j = 1, 2$. If we do the max pooling operation on the matrix C' we find the max value of the top left quadrant is 5, for the top right quadrant is 6, for both bottom quadrants it is 4.

$$MaxPool(C') = C'' = \begin{bmatrix} 5 & 6 \\ 4 & 4 \end{bmatrix} \tag{7}$$

1.4 Flattening

Next we wish to flatten the Max Pooled output before we finally send this into the last fully connected classification layer. Flattening is basically just turning this matrix into a vector. So we get the following:

$$C''_{flat} = \begin{bmatrix} 5 & 6 & 4 & 4 \end{bmatrix} \tag{8}$$

After extracting the features in the previous steps, the flattened output from the convolution and Max Pooling operations is sent into a fully connected layer that performs the classification. The fully connected layer has a the weight matrix W. The output from sending the input x to the layer is defined by the following simple matrix multiplication:

$$Y = W \cdot x \tag{9}$$

In our case we have a weight matrix that is the following:

$$W = \begin{bmatrix} 1 & 2 & 3 & 4 \\ 5 & 6 & 7 & 8 \end{bmatrix} \tag{10}$$

By multiplying the output from the flattening operation with the weight matrix, we find that the output from the fully connected layer Y is:

$$Y = W \cdot C''_{flat} = \begin{bmatrix} 45 \\ 121 \end{bmatrix} \tag{11}$$

1.5 Softmax

Finally, for classification the output from the fully connected layer is put through a Softmax function which computes essentially the probabilities of the output belonging to each class. The Softmax function is described by:

$$Softmax(y)_i = \frac{e^{y_i}}{\sum_{j=1}^{M} e^(y_j)} \tag{12}$$

In our case $M = 2$ since we have two classes. Applying the Softmax function to our output from the fully connected layer finally yields:

$$Softmax(Y) = \begin{bmatrix} 0 \\ 1 \end{bmatrix} \tag{13}$$

This means that our network has predicted the input to belong to class number 2.

Github Link for the practical part: https://github.com/Frezo666/Advanced-Deep-Learning—D7047E-Course.git