Problem Set 2
Deep Learning E1394

Group g: Danial Riaz, Luke Smith

Out on Oct 4, 2022
Due on Oct 18, 2022, at 23:59

Submit your written answers as a pdf typed in LaTeXtogether with your code. Submit one answer per group (as assigned on Moodle) and include names of all group members in the document. Round answers to two decimal places as needed. Include references to any external sources you have consulted (points are deducted if those were used but not cited). See "Submission" at the bottom of the problem set for more details on how to submit using Github classroom.

1 Convolutional neural networks

1.1 Kernels (7 pts)

(a) Design a 3×3 kernel that leaves the input image unchanged. Describe also how you may need to modify the input image before applying the kernel so that the output stays the same.

Here we can use the identity kerned like the one below:

$$k = \begin{bmatrix} 0 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 0 \end{bmatrix}$$

We know that our kernel is 3×3 hence $f = 3$ Say we have an input image of size 4,4 . So $n = 1$. Our output image will only be 4×4 (preserving size) if:

$$\begin{aligned} n + 2p - f + 1 &= 4 \\ 4 + 2p - 3 + 1 &= 4 \\ p &= 1 \end{aligned}$$

where n = input dimension f = kernel dimension p = padding Hence only with a padding of size 1 on the input image, can we ensure that

the output dimensions stay the same as the input dimensions. e.g. Input image: $\begin{bmatrix} 10 & 10 & 0 & 0 \\ 10 & 10 & 0 & 0 \\ 10 & 10 & 0 & 0 \\ 10 & 10 & 0 & 0 \end{bmatrix}$

$$\begin{bmatrix} 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 10 & 10 & 0 & 0 & 0 \\ 0 & 10 & 10 & 0 & 0 & 0 \\ 0 & 10 & 10 & 0 & 0 & 0 \\ 0 & 10 & 10 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 \end{bmatrix} * \begin{bmatrix} 0 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 0 \end{bmatrix}$$

$$\Rightarrow \begin{bmatrix} 10 & 10 & 0 & 0 \\ 10 & 10 & 0 & 0 \\ 10 & 10 & 0 & 0 \\ 10 & 10 & 0 & 0 \end{bmatrix}$$

(b) How does the following kernel modify an input image: $K = \frac{1}{16}\begin{bmatrix} 1 & 1 & 1 & 1 \\ 1 & 1 & 1 & 1 \\ 1 & 1 & 1 & 1 \\ 1 & 1 & 1 & 1 \end{bmatrix}$?

Since K is a 4×4 kernel of 1s and is being multiplied by $\frac{1}{16}$, it is essentially taking an average of each offset.

Thus, this functions as an 'Average Pooling' An input image convolved with this kernel would perhaps be blurred with fine details and noise being reduced.

(c) Design a custom kernel of any size and describe how it transforms an input image or what it highlights in an image.

$$\begin{bmatrix} 1 & 1 & 1 \\ 0 & 0 & 0 \\ -1 & -1 & -1 \end{bmatrix}$$

The above kernel is an example of a horizontal edge detecting filter. If there are any significant intensity changes in the horizontal direction, this kernel will detect them.

1.2 Convolution and pooling (15 pts)

1.2.1 Convolutional layer (10/15 pts)

Given an input image

$$X = \begin{bmatrix} 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 1 \\ 1 & 0.5 & 1 & 0.5 \\ 0 & 1 & 0 & 1 \end{bmatrix}, \tag{1}$$

and a kernel $K = \begin{bmatrix} 0 & 2 & 0 \\ 2 & 1 & 1 \\ 0 & 2 & 0 \end{bmatrix}$, compute the feature map (output after the convolution and applying a sigmoid activation function). Modify the input such that the feature map has the same dimension as the original input image. Show the intermediate result after the convolution and before applying the activation function as well.

As shown in Answer 1.1(a), for an input image of 4×4 and a kernel of size 3×3 we need a padding of 1 . We therefore pad I as follows:

$$I' = \begin{bmatrix} 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 1 & 0 \\ 0 & 1 & 0.5 & 1 & 0.5 & 0 \\ 0 & 0 & 1 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 \end{bmatrix}$$

To make our convolution caries, we flip K as:

$$K_f = \begin{bmatrix} 0 & 2 & 0 \\ 1 & 1 & 2 \\ 0 & 2 & 0 \end{bmatrix}$$

Performing an element wise multiplication and summation at each stride, we get the following output matrix :

$$a_1 = I' \cdot K_f = \begin{bmatrix} 0 & 2 & 0 & 2 \\ 4 & 2 & 5 & 2 \\ 2 & 7.5 & 2.5 & 5.5 \\ 4 & 2 & 5 & 2 \end{bmatrix}$$

Calculation of a_1 :

$$
\begin{aligned}
e_{1,2} &= 2 + 2 \\
e_{2,2} &= 1 + \frac{1}{2} \cdot 2 = 2 \\
e_{2,3} &= 1 + 2 + 2 = 5 \\
e_{2,4} &= 1 + \frac{1}{2} \cdot 2 = 2 \\
e_{3,1} &= 1 + \frac{1}{2} \cdot 2 + 0 = 2 \\
e_{3,2} &= 2 + 1 + 0.5 + 2 + 2 = 7.5 \\
e_{3,3} &= 0.5 + 1 + 1 = 2 \cdot 5 \\
e_{3,4} &= 2 + 1 + 0 \cdot 5 + 2 = 5.5
\end{aligned}
$$

Finally applying the sigmoid to a_1 giver us:

$$
\sigma(a_1) = \begin{bmatrix} 0.50 & 0.88 & 0.50 & 0.88 \\ 0.98 & 0.88 & 0.99 & 0.98 \\ 0.88 & 1.00 & 0.92 & 1.00 \\ 0.98 & 0.88 & 0.99 & 0.88 \end{bmatrix}
$$

where $\sigma(x) = \frac{1}{1+e^{-x}}$ for each element x in a_1,

1.2.2 Pooling layer (2/15 pts)

Apply 2×2 max pooling to the feature map (no overlap/stride 2).

Applying a 2×2 maxpool to $\sigma(a_1)$ gives us:

$$
\begin{bmatrix} 0.98 & 0.99 \\ 1.00 & 1.00 \end{bmatrix} \approx \begin{bmatrix} 1 & 1 \\ 1 & 1 \end{bmatrix}
$$

1.2.3 Discussion (3/15 pts)

Describe any problem(s) that you may see in training the model if a feature map like this one was typical for your CNN.

If a feature map like this was typical, then whatever pooling mechanism we were to use; max or average, we would see the neurons to be activated, since the elements are close to 1 . The change in the Loss function with respect to our network parameters will be very small as they are back-propagated through the layers of the network. This is also known as the vanishing gradient problem.

1.3 Pooling transformed into convolution (8 pts)

How do you represent 2×2 average pooling as a convolution? You may show this by the example of 4×4 input data. Provide the kernel size, kernel values,

stride, etc. as appropriate.

Say we have an input image of 4×4 with the following elements:

$$A = \begin{bmatrix} a & b & c & d \\ e & f & g & h \\ i & j & k & l \\ m & n & o & p \end{bmatrix}$$

A 2×2 average pooling would produce the following output.

$$O = \begin{bmatrix} \frac{a+b+e+f}{4} & \frac{c+d+g+h}{4} \\ \frac{i+j+m+n}{4} & \frac{k+L+o+p}{4} \end{bmatrix}$$

In order to produce the same output through a convolution, we can use a kernel of size 2×2 with a stride of 2 like below:

$$K = \frac{1}{4} \begin{bmatrix} 1 & 1 \\ 1 & 1 \end{bmatrix}$$

We now have:

$$A * K = \begin{bmatrix} \frac{a+b+e+f}{4} & \frac{c+d+g+h}{4} \\ \frac{i+j+m+n}{4} & \frac{k+L+o+p}{4} \end{bmatrix} = O$$

as we wanted.

1.4 Dimensions of CNN layers (10 pts)

For the CNN shown in Figure 1, write down the dimensions of each each layer and how you computed them. The input image is first increased to $3 \times 227 \times 227$ with padding.

Tip: A pooling layer with 'stride 2' means that after each pooling operation the next pooling area is 2 pixels apart. A 2×2 pooling operation with stride 2 would result in our example from class with no overlap. A 3×3 pooling operation with stride 2 has overlap. 'Pad 2' refers to padding with 2 pixels on each side.

../img/PS2_network.png

Figure 1: Convolutional neural network in simplified form. Note that the input image is first increased to $3 \times 227 \times 227$ with padding.

Input image with padding:

$$227 \times 227 \times 3$$

First layer: 96 filters of size 11 x 11 and $s = 4$

$$\frac{n-f}{s} + 1 = \frac{227 - 11}{4} + 1 = 55$$

$55 \times 55 \times 96$

Second Layer: 3×3 MaxPool filter of $s = 2$

$$\frac{55-3}{2}+1=27$$
$$27 \times 27 \times 96$$

Third layer: 256 conv. filters of $5 \times 5; p = 2$

$$\frac{n+2p-f}{5}+1=\frac{27+2(2)-5}{1}+1=27$$
$$27 \times 27 \times 256$$

Fourth layer: 3×3 maxpool, $s = 2$

$$\frac{27-3}{2}+1=13$$
$$13 \times 13 \times 256$$

Fifth layer:

$$3 \times 3 \text{ Conv } (384), p = 1$$
$$\frac{13+2(1)-3}{1}+1=13$$
$$13 \times 13 \times 384$$

Sixth layer: 3×3 Conv (384), $p = 1$

$$13 \times 13 \times 384$$

Seventh layer:

$$3 \times 3 \text{ Conv } (256), \text{ pad } 1$$
$$13+2(1)-3+1=13$$
$$13 \times 13 \times 256$$

Eighth layer: 3×3 Maxpool; $s = 2$

$$\frac{13-3}{2}+1=6$$
$$6 \times 6 \times 256 = 9216 \text{ nodes}$$

Ninth layer: Fully connected layer (4096) A flattened vector of length 4096 Tenth layer: Flattened vector of length 4096 Eleventh layer: Vector with 1000 neurons.

2 Monitoring power plant operations with CNNs

You will create an image classifier using PyTorch to estimate power plant operation. Specifically, you will learn how to apply CNNs to binary cassify LAWS'

Sentinel 2 L2A satellite pictures (0: power plant is off, 1: power plant is on). All tasks are marked in their specific section and described in more detail in the Jupyter notebook. For some tasks, you may need to elaborate on your implementation. You can add the answers to these questions directly below the respective implementation task that you then upload to your GitHub repository. Remember to always show the code output in the final upload and use the "test your code" sections to test your implementation.

1. Implement your Convolutional Neural Network (20 pt)
2. Implement a hyperparameter tuner and fine-tune your network (20 pt)
3. Load and train a pre-trained model (10 pt)
4. Compare and discuss your results (10 pt)

Submission

The submission of the whole problem set is done via GitHub classroom. You have to register for the assignment with your GitHub account at `https://classroom.github.com/a/Ej3Bnsnb`. Please make sure that your GitHub account profile includes your real name. When starting the assignment, please create or join a team according to the teams assigned in Moodle (use the exact team name from Moodle). Please upload / push all solutions to the GitHub repository which was created for your team. Please upload your solutions for the theoretical part (1) as a PDF to GitHub. If you upload multiple PDF files, please indicate in the file name to which subtask they are corresponding (we much prefer one single pdf). For the practical part (2), just push your code changes to the existing files. Anybody in your team can push as often as they want. Once the deadline has passed, you are not able to push to your repository anymore and all changes on your **main** branch will be considered for grading. See the README.md in the repository for more details on how to push your changes to GitHub.