EECS 182 Deep Neural Networks
Fall 2022 Anant Sahai

Discussion 4

1. Weight Sharing in CNN

In this question we will look at the mechanism of weight sharing in convolutions.

(a) Suppose that we have a 9 dimensional input vector and compute a 1d convolution with the kernel filter that has 3 weights.

$$\mathbf{k} = \begin{bmatrix} k_1 & k_2 & k_3 \end{bmatrix}^T, \mathbf{x} = \begin{bmatrix} x_1 & x_2 & x_3 & ... & x_9 \end{bmatrix}^T$$

Find the matrix corresponding to this linear transformation from input vector $\mathbf{x}$ to output $\mathbf{x'}$ which is the 1d convolution of $\mathbf{x}$ and $\mathbf{k}$. Make sure that you should include "same padding" such that you can get a matrix transformation $\mathbf{K}$ from from $\mathbb{R}^9$ to $\mathbb{R}^9$ Write the transformation in the form of $\mathbf{x'} = \mathbf{K}\mathbf{x}$

Solution: This is a Topelitz matrix corresponding to discrete convolution.

$$\begin{bmatrix} k_2 & k_3 & 0 & 0 & 0 & ... & 0 \\ k_1 & k_2 & k_3 & 0 & 0 & ... & 0 \\ 0 & k_1 & k_2 & k_3 & 0 & ... & 0 \\ & & & \vdots & & & \\ 0 & ... & 0 & k_1 & k_2 & k_3 & 0 \\ 0 & ... & 0 & 0 & k_1 & k_2 & k_3 \\ 0 & ... & 0 & 0 & 0 & k_1 & k_2 \end{bmatrix}$$

(b) **Write down the weight matrix for a linear layer (weight matrix multiplication) that takes in a 9 dimensional vector as input and outputs a 9 dimensional vector. How many weights are there?**

Solution: Two things distinguish this from a linear layer 1) No shared weights in a linear layer 2) Linear layer won't have any zeros

Linear layer has 81 weights.

(c) In the question (a), we observed how weight are spatially shared in convolution operation: the same kernel is applied at every input location. Suppose that we no longer share weights spatially over the input (i.e. we go through the same mechanics of a convolution in the sliding window style, but the kernel can have different weights at the different locations). **How does this change our matrix?**

Solution: At each row, the weights (parameters) are different: whereas each row's k_i were shared in convolution kernel.

$$\begin{bmatrix} k_2 & k_3 & 0 & 0 & ... & 0 \\ k_1' & k_2' & k_3' & 0 & ... & 0 \\ & & & \vdots & & \\ 0 & ... & 0 & k_1'' & k_2'' & k_3'' \\ 0 & ... & 0 & 0 & k_1''' & k_2''' \end{bmatrix}$$

(d) We are now applying convolution kernel over an image with a 2D kernel. Just like the question (c), the kernel weights aren't spatially shared (meaning at each location in our convolution operation, the kernel weights can be different). Let's say it is the *new convolution*. Compared with the original convolutional layer that shares weights spatially, the learned features and weights are different in the *new convolutional layer*. **How do learned features change? What do the weights at any of the fixed in place windows learn?**

Hint: Imagine one region of the image has one repeated patch of pixels for certain classes. Why might our normal convolution not immediately create a filter specifically for this feature? And how does the *new convolution* quickly converges?

Solution: Before we were learning kernels that could act essentially as edge detectors. Now our fixed window weights would correspond to an edge or feature detector at a specific spatial location. Before our kernel weights had to be relevant to every patch of the image. After removing spatial weight sharing, they only have to be specific to the receptive field around that location. Our weights that are responsible for computing features no longer have to generalize across space.

(e) Without any zero padding, **how does output size changes as the kernel size increases? What is the maximum kernel size? What is the output size when the kernel size is maximized? Compare the results of our new kernels and weight matrices of fully connected layers**

Solution: Output size : H' = H - K + 1, W' = W - K + 1
Maximum kernel size: H (or W)
As the size of the kernel increases, we can observe that the convolutional layer slowly approaches to a fully connected layer. This means that the spatial output feature size shrinks down to 1.

2. Coding Question: Principles of CNN

Look at the CNNPrinciples.ipynb. In this notebook, you'll study principles and properties of CNN. You will see the three problems below in the notebook as well. Write down your answers in the notebook.

(a) Report the result. Which model performs better? Explain why.

Solution: CNN performs better than MLP. This is because CNN's inductive biases: sparse interactions, parameter sharing, and translational equivariance, are more suitable for the vision modality than MLP.

(b) What do you observe? Why is it different from the case of MLP?

Solution: CNN is translationally equivariant, while MLP is not. This is because CNN has parameter sharing spatially.

(c) Note that even though CNN is not trained, the feature maps not only are still translationally equivariant but also extract a quite good features. Why is it so?

Solution: Such properties are from the architecture of CNN not from the trained weights.

(d) Explain the results. Why do you think the performance is better than the MLP?

Solution: This implies that the function family of CNN locates the initial weights pretty close to the optimal solution. (Any other sensible answer is also acceptable.)

