EECS 182 Deep Neural Networks
Spring 2023 Anant Sahai

Discussion 4

1. Weight Sharing in CNN

In this question we will look at the mechanism of weight sharing in convolutions. Let's start with a 1-dimension example. Suppose that we have a 9 dimensional input vector and compute a 1D convolution with the kernel filter that has 3 weights (parameters).

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

(a) What's the **output dimension** if we apply filter $\mathbf{k}$ with no padding and stride of 1? What's the **first element** of the output? What's the **last element**?

Solution: The output dimension is 9 - 3 + 1 = 7. The first element is $k_1x_1 + k_2x_2 + k_3x_3$ and the last element is $k_1x_7 + k_2x_8 + k_3x_9$.

(b) What's the **output dimension** if we apply $\mathbf{k}$ with "same padding" and stride of 1? What's the **first element** of the output? What's the **last element**?

Solution: The output dimension is 9 by the definition of same padding. The first element is $k_2x_1 + k_3x_2$ and the last element is $k_1x_8 + k_2x_9$.

(c) Recall that in lecture, we discussed that CNN filters have the property of **weight sharing**, meaning that different portions of an image can share the same weight to extract the same set of features. Turns out convolution is a linear operation and we can express it in the form of linear layers, ie. $\mathbf{x'} = \mathbf{K}\mathbf{x}$ (assume the bias term is zero).

Find $\mathbf{K}$, the linear transformation matrix corresponding to the convolution applied in part (a). (*Hint: What is the dimension of* $\mathbf{K}$*?*)

Solution: This is a Topelitz matrix corresponding to discrete convolution. The output size shrinks by k-1, which is 2 in this case, so $\mathbf{K}$ has a dimension of $\mathbb{R}^{7\times 9}$.

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

(d) **Find K for part (b), where we apply "same padding".** (*Hint: You should only be adding rows to your answer in part c.*)

Solution: Since the only difference is the "same padding", we will be adding one row at the beginning and the end respectively. The output size is $\mathbb{R}^{9\times 9}$.

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

(e) Suppose that we no longer want to share weights spatially over the input, ie. we go though the same mechanics as convolution "sliding window", but for different locations within the input, we apply different kernel. **How does this change our matrix? How many weights do we have now?**

Solution:

We will still end up with a "sparse" weight matrix, but each row now represents a new kernel filter. In the case where no padding is applied, we end up with $(9-3+1)*3 = 21$ non-zero weights as shown below.

$$\begin{bmatrix} k_1 & k_2 & k_3 & 0 & ... & 0 & 0 & 0 \\ 0 & k_4 & k_5 & k_6 & ... & 0 & 0 & 0 \\ & & & \vdots & & & & \\ 0 & ... & 0 & 0 & k_{16} & k_{17} & k_{18} & 0 \\ 0 & ... & 0 & 0 & 0 & k_{19} & k_{20} & k_{21} \end{bmatrix}$$

(f) We want to know the general formula for computing the output dimension of a convolution operation. Suppose we have a square input image of dimension $W \times W$ and a $K \times K$ kernel filter. If we assume stride of 1 and no padding, **what's the output dimension W' ? What if we applied stride of s and padding of size p, how would the dimension change?**

Solution:

For the first question, the output dimension will be $W' = W - K + 1$.

For the second one, the output dimension will be $\left\lfloor \frac{W+2p-K}{s} \right\rfloor + 1$

Let's take what we've learnt into actual applications on image tasks. Suppose our input is a 256 by 256 RGB image. We are also given a set of 32 filters, each with kernel size of 5.

(g) Conventionally in frameworks such as PyTorch, images are 3D tensors arranged in the format of `[channels, height, width]`. In practice, it's more common to have an additional `batch_size` dimension at the front, but here we ignore that to simplify the math. **What's the shape our input tensor**? **What's the shape of each kernel filter**? (*Hint: Both your answers should have 3 dimensions.)*

Solution:

The input tensor is $3 \times 256 \times 256$. The Kernel size is $3 \times 5 \times 5$.

(h) Now apply convolution on our image tensor with no padding and stride of 2. **What is the output tensor's dimension?** Considering all kernel filters, **how many weights do we have**? Had we not use CNN but MLP instead (with flattened image), **how many weights does that linear layer contain?** Feel free to use a calculator for this question.

Solution: Using the formula we derived in part (f), the output shape can be computed as $\left\lfloor \frac{256-5}{2} \right\rfloor + 1 = 126$, so the output dimension is $32 \times 126 \times 126$. In total we have $32 * (3 * 5 * 5) = 2400$ weights.

Turning an input of $3 \times 256 \times 256$ to a $32 \times 126 \times 126$ tensor requires first flattening the image to a 196608 dimensional vector, and transforming it to a 508302 vector. The resulting transformation matrix will have $196608 * 508302 \approx 9.9936 * 10^{10}$.

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.)

Contributors:

- Jake Austin.
- Suhong Moon.
- Kevin Li.
- Anant Sahai.

