BackPropagation for Fully-Connected and Convolutional Neural Networks

Shusen Wang
Stevens Institute of Technology

August 30, 2019

Abstract

First, define a fully-connected (FC) layer. Second, derive the gradients using chain rule for one FC layer. Third, connect multiple FC layers to build a FC neural network. Fourth, perform backpropagation using the chain rule. Fifth, express convolution as a matrix multiplication. Last, derive gradients for convolutional layer.

1 Fully-Connected (FC) Layer

We consider one fully-connected (FC) layer and follow the convention of PyTorch. Let d_{in} be the input shape, d_{out} be the output shape, and b be the batch size. Let $\mathbf{X} \in \mathbb{R}^{b \times d_{\text{in}}}$ be a batch of input vectors, $\mathbf{W} \in \mathbb{R}^{d_{\text{out}} \times d_{\text{in}}}$ be the weight matrix, and $\mathbf{Z} = \mathbf{X}\mathbf{W}^T \in \mathbb{R}^{b \times d_{\text{out}}}$. The output of this FC layer is $\mathbf{X}' = \sigma(\mathbf{Z}) \in \mathbb{R}^{b \times d_{\text{out}}}$ where σ is an activation function that applies elementwisely. For example, if the activation function is ReLU, then the (i, j)-th entry of $\mathbf{X}'$ is

$$x'_{ij} = \begin{cases} z_{ij}, & \text{if } z_{ij} > 0; \\ 0, & \text{otherwise.} \end{cases}$$

The structure of a FC layer is illustrated in Figure 1(left).

2 Differentiation for FC Layer

Let Q be the loss function that depends on $\mathbf{X}'$. Suppose we know $\frac{\partial Q}{\partial \mathbf{X}'} \in \mathbb{R}^{b \times d_{\text{out}}}$ (the derivative of Q w.r.t. $\mathbf{X}'$). Since $\mathbf{X}$ and $\mathbf{W}$ influence Q via $\mathbf{X}'$:

$$\cdots \longrightarrow \cdots \longrightarrow \left.\begin{matrix} \mathbf{X} \\ \mathbf{W} \end{matrix}\right\} \xrightarrow{\text{multiply}} \mathbf{Z} \xrightarrow{\text{activation}} \mathbf{X}' \longrightarrow \cdots \longrightarrow Q,$$

we can let the gradient flow to $\mathbf{X}$ and $\mathbf{W}$ in the opposite direction:

$$\cdots \longleftarrow \cdots \longleftarrow \left.\begin{matrix} \frac{\partial Q}{\partial \mathbf{X}} \\ \frac{\partial Q}{\partial \mathbf{W}} \end{matrix}\right\} \longleftarrow \frac{\partial Q}{\partial \mathbf{Z}} \longleftarrow \frac{\partial Q}{\partial \mathbf{X}'}.$$

In the following, we compute $\frac{\partial Q}{\partial \mathbf{Z}}$ and then $\frac{\partial Q}{\partial \mathbf{X}}$ and $\frac{\partial Q}{\partial \mathbf{W}}$.

From $\mathbf{X}'$ to $\mathbf{Z}$. First, compute $\frac{\partial Q}{\partial \mathbf{Z}} \in \mathbb{R}^{b \times d_{\text{out}}}$. If $\mathbf{X}' = \mathsf{ReLU}(\mathbf{Z})$,[1] then the (i,j)-th entry of $\frac{\partial Q}{\partial \mathbf{Z}}$ is

$$\left[\frac{\partial Q}{\partial \mathbf{Z}}\right]_{ij} = \frac{\partial Q}{\partial z_{ij}} = \frac{\partial x'_{ij}}{\partial z_{ij}} \frac{\partial Q}{\partial x'_{ij}} = \begin{cases} \frac{\partial Q}{\partial x'_{ij}}, & \text{if } z_{ij} > 0; \\ 0, & \text{otherwise.} \end{cases}$$

Let $\mathbf{A} \in \mathbb{R}^{b \times d_{\text{out}}}$ be such as matrix that

$$a_{ij} = \begin{cases} 1, & \text{if } z_{ij} > 0; \\ 0, & \text{otherwise.} \end{cases}$$

Let "$\circ$" denote the Hadamard product (also known as elementwise product.) Then

$$\frac{\partial Q}{\partial \mathbf{Z}} = \mathbf{A} \circ \frac{\partial Q}{\partial \mathbf{X}'} \in \mathbb{R}^{b \times d_{\text{out}}}. \tag{2.1}$$

From $\mathbf{Z}$ to $\mathbf{X}$. Second, compute $\frac{\partial Q}{\partial \mathbf{X}} \in \mathbb{R}^{b \times d_{\text{in}}}$. Let $\mathbf{x}_{i:} \in \mathbb{R}^{1 \times d_{\text{in}}}$ and $\mathbf{z}_{i:} \in \mathbb{R}^{1 \times d_{\text{out}}}$ be the i-th rows of $\mathbf{X}$ and $\mathbf{Z}$, respectively, for $i = 1$ to b.[2] Thus $\mathbf{x}_{i:}^T \in \mathbb{R}^{d_{\text{in}} \times 1}$ and $\mathbf{z}_{i:}^T \in \mathbb{R}^{d_{\text{out}} \times 1}$ are the i-th column of $\mathbf{X}^T \in \mathbb{R}^{d_{\text{in}} \times b}$ and $\mathbf{Z}^T \in \mathbb{R}^{d_{\text{out}} \times b}$, respectively. It follows from the chain rule that

$$\frac{\partial Q}{\partial \mathbf{x}_{i:}^T} = \sum_{j=1}^{b} \frac{\partial \mathbf{z}_{j:}^T}{\partial \mathbf{x}_{i:}^T} \frac{\partial Q}{\partial \mathbf{z}_{j:}^T} = \frac{\partial \mathbf{z}_{i:}^T}{\partial \mathbf{x}_{i:}^T} \frac{\partial Q}{\partial \mathbf{z}_{i:}^T} + \sum_{i \neq j} \frac{\partial \mathbf{z}_{j:}^T}{\partial \mathbf{x}_{i:}^T} \frac{\partial Q}{\partial \mathbf{z}_{j:}^T}.$$

If $i \neq j$, $\mathbf{z}_{j:}$ will not depend on $\mathbf{x}_{i:}$, and thus $\frac{\partial \mathbf{z}_{j:}^T}{\partial \mathbf{x}_{i:}^T}$ is the all-zero matrix. It follows that

$$\frac{\partial Q}{\partial \mathbf{x}_{i:}^T} = \underbrace{\frac{\partial \mathbf{z}_{i:}^T}{\partial \mathbf{x}_{i:}^T}}_{d_{\text{in}} \times d_{\text{out}}} \underbrace{\frac{\partial Q}{\partial \mathbf{z}_{i:}^T}}_{d_{\text{out}} \times 1} \in \mathbb{R}^{d_{\text{in}} \times 1}.$$

Since $\mathbf{z}_{i:} = \mathbf{x}_{i:} \mathbf{W}^T$, we have $\mathbf{z}_{i:}^T = \mathbf{W} \mathbf{x}_{i:}^T$, and thus $\frac{\partial \mathbf{z}_{i:}^T}{\partial \mathbf{x}_{i:}^T} = \mathbf{W}^T \in \mathbb{R}^{d_{\text{in}} \times d_{\text{out}}}$. It follows that

$$\frac{\partial Q}{\partial \mathbf{x}_{i:}^T} = \mathbf{W}^T \cdot \frac{\partial Q}{\partial \mathbf{z}_{i:}^T} \in \mathbb{R}^{d_{\text{in}} \times 1}.$$

Since $\mathbf{x}_{i:}^T$ is the i-th column of $\mathbf{X}^T \in \mathbb{R}^{d_{\text{in}} \times b}$,

$$\frac{\partial Q}{\partial \mathbf{X}^T} = \left[\frac{\partial Q}{\partial \mathbf{x}_{1:}^T}, \cdots, \frac{\partial Q}{\partial \mathbf{x}_{b:}^T}\right] = \mathbf{W}^T \cdot \left[\frac{\partial Q}{\partial \mathbf{z}_{1:}^T}, \cdots, \frac{\partial Q}{\partial \mathbf{z}_{b:}^T}\right] = \underbrace{\mathbf{W}^T}_{d_{\text{in}} \times d_{\text{out}}} \cdot \underbrace{\frac{\partial Q}{\partial \mathbf{Z}^T}}_{d_{\text{out}} \times b} \in \mathbb{R}^{d_{\text{in}} \times b}.$$

Hence

$$\frac{\partial Q}{\partial \mathbf{X}} = \left(\frac{\partial Q}{\partial \mathbf{X}^T}\right)^T = \frac{\partial Q}{\partial \mathbf{Z}} \cdot \mathbf{W} \in \mathbb{R}^{b \times d_{\text{in}}}. \tag{2.2}$$

[1] $[\mathsf{ReLU}(\mathbf{Z})]_{ij} = \max\{z_{ij}, 0\}$.

[2] To calculate gradient in the standard way, we must use column vectors.

From Z to W. Third, compute $\frac{\partial Q}{\partial \mathbf{W}} \in \mathbb{R}^{d_{\text{out}} \times d_{\text{in}}}$. Let $\mathbf{z}_{:j} \in \mathbb{R}^{b\times 1}$ be the j-th columns of $\mathbf{Z}$ and $\mathbf{w}_{j:} \in \mathbb{R}^{1\times d_{\text{in}}}$ be the j-th row of $\mathbf{W}$, for $j = 1$ to d_{out}. Thus $\mathbf{w}_{j:}^T \in \mathbb{R}^{d_{\text{in}}\times 1}$ is the j-th column of $\mathbf{W}^T \in \mathbb{R}^{d_{\text{in}}\times d_{\text{out}}}$. It follows from the chain rule that

$$\frac{\partial Q}{\partial \mathbf{w}_{j:}^T} = \sum_{i=1}^{d_{\text{out}}} \frac{\partial \mathbf{z}_{:i}}{\partial \mathbf{w}_{j:}^T} \frac{\partial Q}{\partial \mathbf{z}_{:i}} = \frac{\partial \mathbf{z}_{:j}}{\partial \mathbf{w}_{j:}^T} \frac{\partial Q}{\partial \mathbf{z}_{:j}} + \sum_{i\neq j} \frac{\partial \mathbf{z}_{:i}}{\partial \mathbf{w}_{j:}^T} \frac{\partial Q}{\partial \mathbf{z}_{:i}} = \underbrace{\frac{\partial \mathbf{z}_{:j}}{\partial \mathbf{w}_{j:}^T}}_{d_{\text{in}}\times b} \underbrace{\frac{\partial Q}{\partial \mathbf{z}_{:j}}}_{b\times 1} \in \mathbb{R}^{d_{\text{in}}\times 1}.$$

Since $\mathbf{z}_{:j} = \mathbf{X}\mathbf{w}_{j:}^T \in \mathbb{R}^{b\times 1}$, we can show that $\frac{\partial \mathbf{z}_{:j}}{\partial \mathbf{w}_{j:}^T} = \mathbf{X}^T \in \mathbb{R}^{d_{\text{in}}\times b}$. Thus

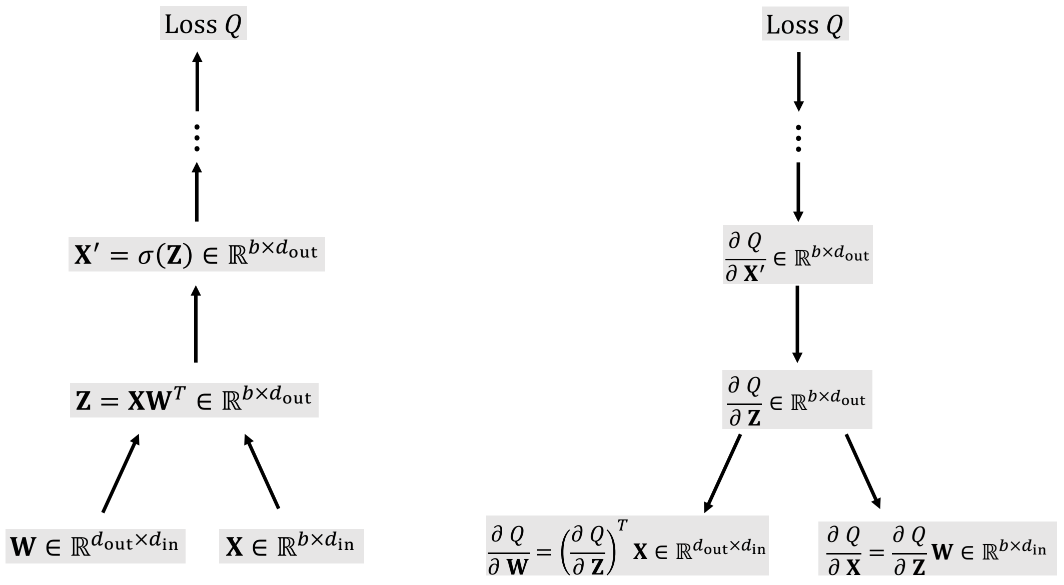
$$\frac{\partial Q}{\partial \mathbf{w}_{j:}^T} = \mathbf{X}^T \cdot \frac{\partial Q}{\partial \mathbf{z}_{:j}} \in \mathbb{R}^{d_{\text{in}}\times 1}.$$

Note that $\mathbf{w}_{j:}^T$ is the j-th column of $\mathbf{W}^T \in \mathbb{R}^{d_{\text{in}}\times d_{\text{out}}}$. It follows that

$$\frac{\partial Q}{\partial \mathbf{W}^T} = \left[\frac{\partial Q}{\partial \mathbf{w}_{1:}^T}, \cdots, \frac{\partial Q}{\partial \mathbf{w}_{d_{\text{out}}:}^T}\right] = \mathbf{X}^T\left[\frac{\partial Q}{\partial \mathbf{z}_{:1}}, \cdots, \frac{\partial Q}{\partial \mathbf{z}_{:d_{\text{out}}}}\right] = \underbrace{\mathbf{X}^T}_{d_{\text{in}}\times b} \underbrace{\frac{\partial Q}{\partial \mathbf{Z}}}_{b\times d_{\text{out}}} \in \mathbb{R}^{d_{\text{in}}\times d_{\text{out}}}.$$

Hence,

$$\frac{\partial Q}{\partial \mathbf{W}} = \left(\frac{\partial Q}{\partial \mathbf{W}^T}\right)^T = \left(\frac{\partial Q}{\partial \mathbf{Z}}\right)^T \mathbf{X} \in \mathbb{R}^{d_{\text{out}}\times d_{\text{in}}}. \tag{2.3}$$



Loss Q
$\mathbf{X}' = \sigma(\mathbf{Z}) \in \mathbb{R}^{b\times d_{\text{out}}}$
$\mathbf{Z} = \mathbf{X}\mathbf{W}^T \in \mathbb{R}^{b\times d_{\text{out}}}$
$\mathbf{W} \in \mathbb{R}^{d_{\text{out}}\times d_{\text{in}}}$
$\mathbf{X} \in \mathbb{R}^{b\times d_{\text{in}}}$
Loss Q
$\frac{\partial Q}{\partial \mathbf{X}'} \in \mathbb{R}^{b\times d_{\text{out}}}$
$\frac{\partial Q}{\partial \mathbf{Z}} \in \mathbb{R}^{b\times d_{\text{out}}}$
$\frac{\partial Q}{\partial \mathbf{W}} = \left(\frac{\partial Q}{\partial \mathbf{Z}}\right)^T \mathbf{X} \in \mathbb{R}^{d_{\text{out}}\times d_{\text{in}}}$
$\frac{\partial Q}{\partial \mathbf{X}} = \frac{\partial Q}{\partial \mathbf{Z}} \mathbf{W} \in \mathbb{R}^{b\times d_{\text{in}}}$

Figure 1: Differential for one FC Layer.

We summarize the gradients flow in Figure 1. Given $\frac{\partial Q}{\partial \mathbf{X}'}$, we can compute first $\frac{\partial Q}{\partial \mathbf{Z}}$ according to (2.1) and then $\frac{\partial Q}{\partial \mathbf{X}}$ according to (2.2) and $\frac{\partial Q}{\partial \mathbf{W}}$ according to (2.3).

3 Fully-Connected (FC) Neural Network

An FC neural network is composed of multiple FC layers. Suppose the FC network has L (> 1) layers; the l-th layer is parameterized by weight matrix $\mathbf{W}^{(l)}$. The l-th layer takes matrix $\mathbf{X}^{(l)}$ as input, computes $\mathbf{Z}^{(l)} = \mathbf{X}^{(l)}\mathbf{W}^{(l)^T}$, and outputs $\mathbf{X}^{(l+1)} = \sigma(\mathbf{Z}^{(l)})$. The dependence can be depicted as

$$\text{input} \longrightarrow \cdots \longrightarrow \underbrace{\left.\begin{matrix}\mathbf{X}^{(l)}\\ \mathbf{W}^{(l)}\end{matrix}\right\} \longrightarrow \mathbf{Z}^{(l)} \longrightarrow \mathbf{X}^{(l+1)}}_{\text{the } l\text{-th layer}} \longrightarrow \cdots \longrightarrow \mathbf{X}^{(L+1)}(\text{i.e. output}) \longrightarrow \text{loss}.$$

Note that for $\mathbf{X}^{(1)}, \cdots, \mathbf{X}^{(L+1)}$ all have b rows, where b is the batch size; but they can have different numbers of columns.[3]

The L-th layer is called the output layer. The output of the L-th layer, denote $\mathbf{X}^{(L+1)} \in \mathbb{R}^{b\times m}$, is the prediction the neural network makes for the input $\mathbf{X}^{(1)}$. Let $\mathbf{Y} \in \mathbb{R}^{b\times m}$ be the labels of this batch of samples.[4] We need to define a loss function Q that measures the difference between the prediction and the ground truth (labels). For example, the loss function can be

$$Q\left(\mathbf{X}^{(1)}, \mathbf{Y};\, \mathbf{W}^{(1)}, \cdots, \mathbf{W}^{(L)}\right) \;=\; \frac{1}{2}\left\|\mathbf{X}^{(L+1)} - \mathbf{Y}\right\|_F^2.$$

See Figure 2(left) for the structure of the FC neural network (with $\mathbf{Z}$'s abbreviated.)

4 Computing Gradients via BackPropagation

Let Q be the loss function parameterized by $\mathbf{W}^{(1)}, \mathbf{W}^{(2)}, \cdots, \mathbf{W}^{(L)}$, where $\mathbf{W}^{(l)}$ is the weight of the l-th layer. We seek to minimize Q w.r.t. to the weights, so we need the gradients:

$$\frac{\partial Q}{\partial \mathbf{W}^{(1)}},\; \frac{\partial Q}{\partial \mathbf{W}^{(2)}},\; \cdots,\; \frac{\partial Q}{\partial \mathbf{W}^{(L)}}.$$

With the gradients, we can update the weights $\mathbf{W}^{(1)}, \mathbf{W}^{(2)}, \cdots, \mathbf{W}^{(L)}$, e.g., by (stochastic) gradient descent:

$$\mathbf{W}^{(l)} \longleftarrow \mathbf{W}^{(l)} - \alpha \frac{\partial Q}{\partial \mathbf{W}^{(l)}}, \qquad \text{for } l = 1, \cdots, L, \tag{4.1}$$

where α (> 0) is called step size or learning rate. See my lecture note *Logistic Regression* for gradient-based algorithms.

Since Q is complicated, directly computing $\frac{\partial Q}{\partial \mathbf{W}^{(l)}}$ for any layer is difficult. The best practice is BackPropagation, that is, using the chain rule to let gradients flow from the top to the bottom. See the illustration in Figure 2(right).

- The loss function is a simple function of the output, $\mathbf{X}^{(L+1)}$. So it is easy to compute the derivative $\frac{\partial Q}{\partial \mathbf{X}^{(L+1)}}$.

[3] In the case of MNIST hand-written digit classification, the inputs are 784 ($= 28 \times 28$) dimensional vectors, and the outputs are a 10-dimensional vectors (for there are 10 classes). So $\mathbf{X}^{(1)}$ has 784 columns, and $\mathbf{X}^{(L+1)}$ has 10 columns.

[4] In the case of housing price prediction, the labels (housing price) are scalars, so $m = 1$. In the case of hand-written digit classification, there are ten classes, and the labels are one-hot encode (10-dimensional vectors); thus $m = 10$.

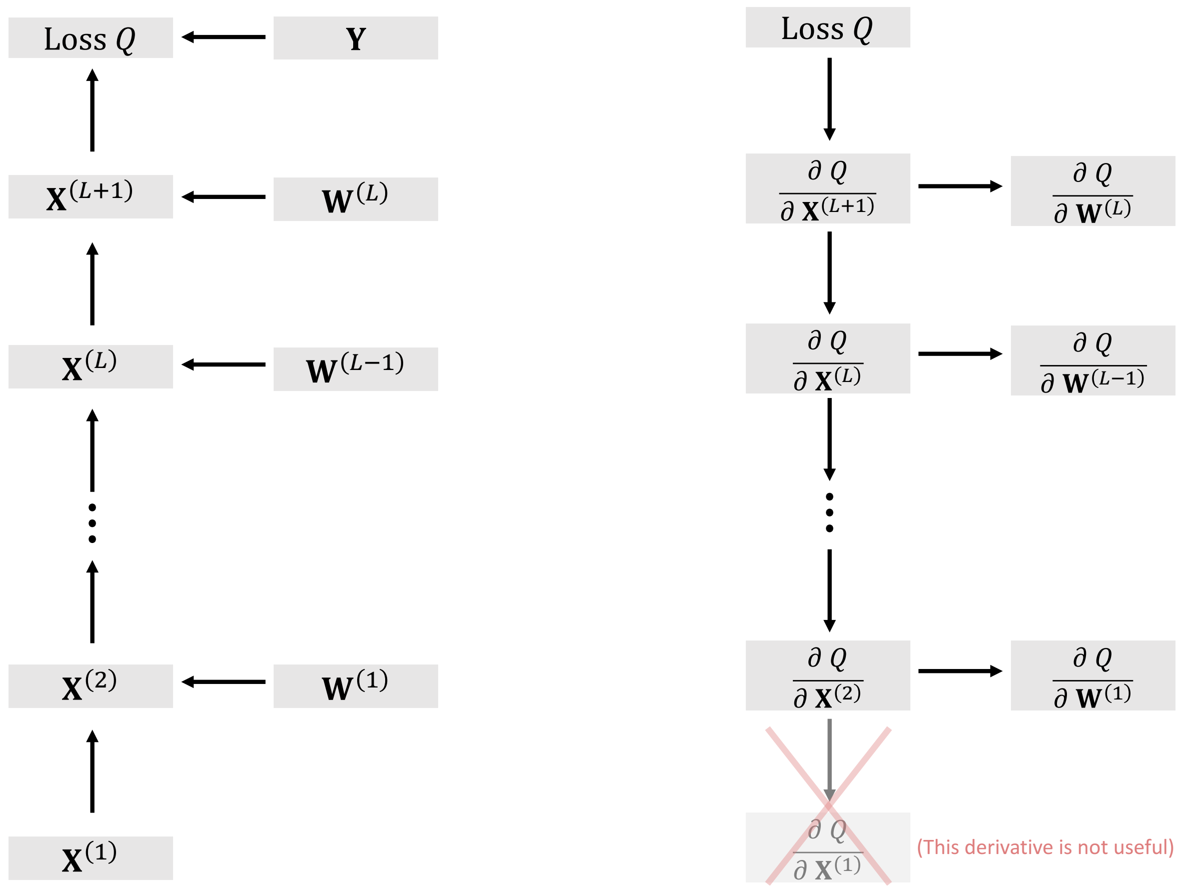

Loss Q
$\mathbf{Y}$
$\mathbf{X}^{(L+1)}$
$\mathbf{W}^{(L)}$
$\mathbf{X}^{(L)}$
$\mathbf{W}^{(L-1)}$
$\mathbf{X}^{(2)}$
$\mathbf{W}^{(1)}$
$\mathbf{X}^{(1)}$
Loss Q
$\frac{\partial Q}{\partial \mathbf{X}^{(L+1)}}$
$\frac{\partial Q}{\partial \mathbf{W}^{(L)}}$
$\frac{\partial Q}{\partial \mathbf{X}^{(L)}}$
$\frac{\partial Q}{\partial \mathbf{W}^{(L-1)}}$
$\frac{\partial Q}{\partial \mathbf{X}^{(2)}}$
$\frac{\partial Q}{\partial \mathbf{W}^{(1)}}$
$\frac{\partial Q}{\partial \mathbf{X}^{(1)}}$
(This derivative is not useful)

Figure 2: The forward and backward pass for computing gradients.

- After knowing $\frac{\partial Q}{\partial \mathbf{X}^{(L+1)}}$, we can use the equations (chain rule) in Section 2 to compute $\frac{\partial Q}{\partial \mathbf{W}^{(L)}}$ and $\frac{\partial Q}{\partial \mathbf{X}^{(L)}}$. We record $\frac{\partial Q}{\partial \mathbf{W}^{(L)}}$ and pass $\frac{\partial Q}{\partial \mathbf{X}^{(L)}}$ to the $(L-1)$-th layer.
- We repeat the above step from the $(L-1)$-th layer all the way down to the first (bottom) layer.

In this way, we obtained $\frac{\partial Q}{\partial \mathbf{W}^{(L)}}, \cdots, \frac{\partial Q}{\partial \mathbf{W}^{(2)}}, \frac{\partial Q}{\partial \mathbf{W}^{(1)}}$ one by one. We will use them to update $\mathbf{W}^{(L)}$, e.g., according (4.1).

5 Expressing Convolution as Matrix Multiplication

The rest of this paper extends what we have done for FC layers to convolutional layers. In this section, we express convolution as matrix multiplication to reveal the connection between FC layers and convolutional layers. In the next section, we will derive the gradients for convolution. Differentiation for convolutional layer is the same as FC layer except for the unfold and fold operations; understanding unfold and fold will be the key to comprehend this and the next section.

Tensor convolution. Let $\mathbf{T}$ be a $d_1 \times d_2 \times d_3$ input tensor and $\mathbf{K}$ be a $k_1 \times k_2 \times d_3$ kernel (aka filter) tensor. The convolution $\mathbf{T} * \mathbf{K}$ outputs a $(d_1 - k_1 + 1) \times (d_2 - k_2 + 1)$ matrix, denote $\mathbf{C}$.

- What decides the output shape of convolution? It is the number of $k_1 \times k_2 \times d_3$ patches in $\mathbf{T}$. Tensor $\mathbf{T}$ has $(d_1 - k_1 + 1) \times (d_2 - k_2 + 1)$ such patches. In Figure 3, since $d_1 = 4$, $d_2 = 3$, and $k_1 = k_2 = 2$, there are totally $3 \times 2 = 6$ patches.

- What are the entries of matrix $\mathbf{C}$? Let scalar $c_{ij} \in \mathbb{R}$ be the (i,j)-th entry of $\mathbf{C}$ and tensor $\mathbf{P}_{ij} \in \mathbb{R}^{k_1 \times k_2 \times d_3}$ be the (i,j)-th patch of $\mathbf{T}$. Then

$$c_{ij} \; = \; \big\langle \mathbf{K}, \, \mathbf{P}_{ij} \big\rangle \; = \; \big\langle \mathsf{vec}(\mathbf{K}), \, \mathsf{vec}(\mathbf{P}_{ij}) \big\rangle. \tag{5.1}$$

  Here, $\mathsf{vec}(\mathbf{K})$ means reshaping tensor $\mathbf{K}$ to a $k_1 k_2 d_3 \times 1$ vector, and $\langle \cdot, \cdot \rangle$ denotes vector/matrix/tensor inner product.

Unfolding. Based on the above discussions, we know that the $d_1 \times d_2 \times d_3$ tensor $\mathbf{T}$ can be converted to $(d_1 - k_1 + 1) \times (d_2 - k_2 + 1)$ patches; each patch is a $k_1 \times k_2 \times d_3$ tensor (the same to the kernel $\mathbf{K}$). Converting the order-3 tensor $\mathbf{T}$ to the order-5 tensor $\overline{\mathbf{X}}$ is called unfolding. The shape of $\overline{\mathbf{X}}$ is

$$(d_1 - k_1 + 1) \times (d_2 - k_2 + 1) \times k_1 \times k_2 \times d_3;$$

$\overline{\mathbf{X}}$ consists of the patches $\{\mathbf{P}_{ij}\}$:

$$\overline{\mathbf{X}}\,[i, \, j, \, :, \, :, \, :] \; = \; \mathbf{P}_{ij} \; \in \; \mathbb{R}^{k_1 \times k_2 \times d_3}.$$

Note that unfold is supported by software systems like PyTorch. Then, we reshape the order-5 tensor $\overline{\mathbf{X}}$ to the

$$(d_1 - k_1 + 1)(d_2 - k_2 + 1) \; \times \; (k_1 k_2 d_3)$$

matrix, denote $\mathbf{X}$. In sum, the procedure is

$$\mathbf{T} \text{ (order-3 tensor)} \xrightarrow{\text{unfold}} \overline{\mathbf{X}} \text{ (order-5 tensor)} \xrightarrow{\text{reshape}} \mathbf{X} \text{ (matrix)}.$$

Figure 3 illustrates this procedure.

Convolution as matrix-vector multiplication. Let matrix $\mathbf{X}$ be the outcome after unfolding and reshaping. Let $\mathbf{w} = \mathsf{vec}(\mathbf{K}) \in \mathbb{R}^{k_1 k_2 d_3 \times 1}$ be the vectorization of $\mathbf{K} \in \mathbb{R}^{k_1 \times k_2 \times d_3}$. Then, compute the vector

$$\mathbf{z} \; = \; \mathbf{X} \cdot \mathsf{vec}(\mathbf{K}) \; \in \; \mathbb{R}^{(d_1 - k_1 + 1)(d_2 - k_2 + 1) \times 1}. \tag{5.2}$$

Recall that the $(d_1 - k_1 + 1) \times (d_2 - k_2 + 1)$ matrix $\mathbf{C}$ is the outcome of convolution. By comparing (5.1) and (5.2), we find that

$$\mathbf{z} \; = \; \mathsf{vec}(\mathbf{C}) \qquad \text{and} \qquad \mathbf{C} \; = \; \mathsf{reshape}\Big(\mathbf{z}, \; \big((d_1 - k_1 + 1), (d_2 - k_2 + 1)\big)\Big).$$

To summarize, the forward pass of tensor convolution can be expressed in the following two equivalent forms:

$$\left.\begin{array}{r} \mathbf{T} \text{ (input tensor)} \\ \mathbf{K} \text{ (kernel tensor)} \end{array}\right\} \xrightarrow{\text{convolution}} \mathbf{C} \text{ (output matrix)}$$

and

$$\left.\begin{array}{l}\mathbf{T} \xrightarrow{\text{unfold}} \overline{\mathbf{X}} \xrightarrow{\text{reshape}} \mathbf{X} \\ \qquad\quad \mathbf{K} \xrightarrow{\text{vectorize}} \mathbf{w}\end{array}\right\} \xrightarrow{\text{multiply}} \mathbf{z} \xrightarrow{\text{reshape}} \mathbf{C}. \tag{5.3}$$

In the next section, we will use the latter to perform backpropagation.

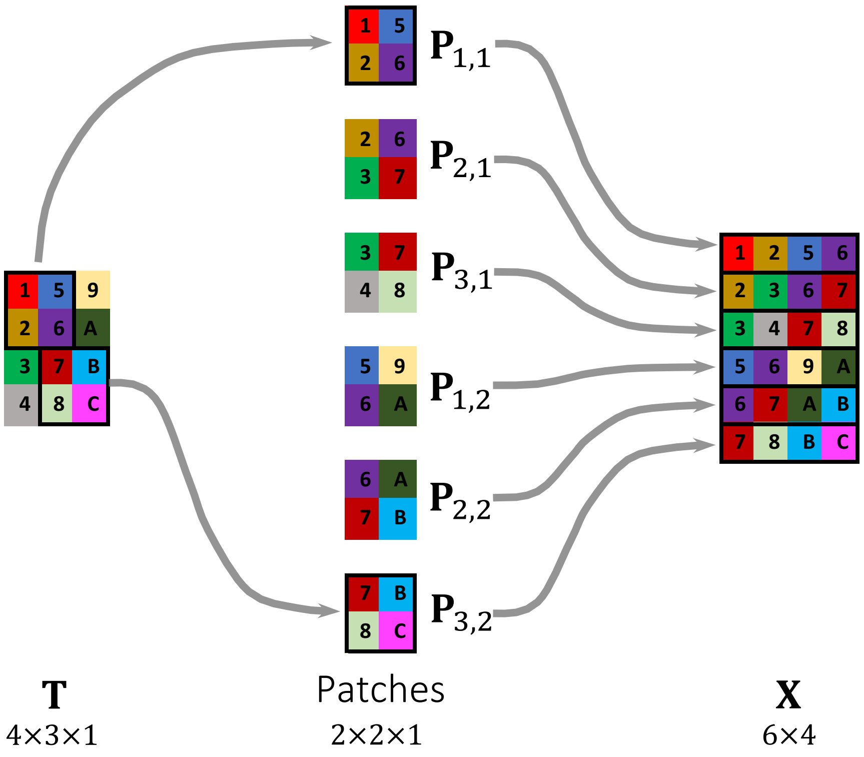


Figure 3: Illustrating patching and unfolding. Here, $d_1 = 4$, $d_2 = 3$, $d_3 = 1$, and $k_1 = k_2 = 2$. Thus, there are $(d_1 - k_1 + 1) \times (d_2 - k_2 + 1) = 6$ patches, and each patch is $k_1 \times k_2 = 2 \times 2$.

6 Differentiation for Convolution

By representing convolution as the procedure (6.1), the backpropagation will be easier to derive. During the backpropagation, we receive $\frac{\partial Q}{\partial \mathbf{C}}$ and propagate it to $\mathbf{K}$ and $\mathbf{T}$. The backpropagation has the following steps:

$$\left.\begin{array}{l}\frac{\partial Q}{\partial \mathbf{T}} \xleftarrow{\text{fold}} \frac{\partial Q}{\partial \overline{\mathbf{X}}} \xleftarrow{\text{reshape}} \frac{\partial Q}{\partial \mathbf{X}} \\ \qquad\quad \frac{\partial Q}{\partial \mathbf{K}} \xleftarrow{\text{reshape}} \frac{\partial Q}{\partial \mathbf{w}}\end{array}\right\} \xleftarrow{\text{multiply}} \frac{\partial Q}{\partial \mathbf{z}} \xleftarrow{\text{vectorize}} \frac{\partial Q}{\partial \mathbf{C}}. \tag{6.1}$$

We describe the steps one by one.

From C to z. This step is almost trivial. Since $\mathbf{z} = \text{vec}(\mathbf{C})$, we have

$$\frac{\partial Q}{\partial \mathbf{z}} = \text{vec}\left(\frac{\partial Q}{\partial \mathbf{C}}\right) \in \mathbb{R}^{(d_1-k_1+1)(d_2-k_2+1)\times 1}.$$

It just performs a vectorization.

From z to X and w. Recall from (5.2) that $\mathbf{z}$ is computed by the matrix-vector multiplication: $\mathbf{z} = \mathbf{X}\mathbf{w}$. Once we have $\frac{\partial Q}{\partial \mathbf{z}}$, we can propagate it to $\mathbf{X}$ and $\mathbf{w}$ by

$$\begin{aligned} \underbrace{\frac{\partial Q}{\partial \mathbf{X}}}_{(d_1-k_1+1)(d_2-k_2+1)\times(k_1k_2d_3)} &= \underbrace{\frac{\partial Q}{\partial \mathbf{z}}}_{(d_1-k_1+1)(d_2-k_2+1)\times 1} \underbrace{\mathbf{w}^T}_{1\times(k_1k_2d_3)}, \\ \underbrace{\frac{\partial Q}{\partial \mathbf{w}}}_{(k_1k_2d_3)\times 1} &= \underbrace{\mathbf{X}^T}_{(k_1k_2d_3)\times(d_1-k_1+1)(d_2-k_2+1)} \underbrace{\frac{\partial Q}{\partial \mathbf{z}}}_{(d_1-k_1+1)(d_2-k_2+1)\times 1}. \end{aligned}$$

The above equations follow from (2.2) and (2.3).

From w to K. Since $\mathbf{w} = \mathsf{vec}(\mathbf{K})$, the $k_1 \times k_2 \times d_3$ tensor $\mathbf{K}$ can be obtained by $\mathsf{reshaping}$ $\mathbf{w}$ to order-3 tensor. Thus,

$$\frac{\partial Q}{\partial \mathbf{K}} = \mathsf{reshape}\left(\frac{\partial Q}{\partial \mathbf{w}}, \left(k_1,\ k_2,\ d_3\right)\right)$$

From X to T. As illustrated in Figure 3, $\mathbf{X}$ is obtained by $\mathsf{unfolding}$ $\mathbf{T}$, and one entry of $\mathbf{T}$ is copied to multiple (at most k_1k_2) entries of $\mathbf{X}$. For example, in Figure 3, the green "A" entry is copied from $t_{2,3,1}$ to both $x_{4,4}$ and $x_{5,3}$.[5] Thus

$$t_{2,3,1} = x_{4,4} = x_{5,3}. \tag{6.2}$$

So the $(2,3,1)$-th entry of $\mathbf{T}$ influences Q via only two enties of $\mathbf{X}$, and thus

$$\frac{\partial x_{ij}}{\partial t_{2,3,1}} = \begin{cases} 1 & \text{if } (i,j) = (4,4) \text{ or } (5,3); \\ 0 & \text{otherwise.} \end{cases}$$

It follows that

$$\left[\frac{\partial Q}{\partial \mathbf{T}}\right]_{2,3,1} = \sum_{i,j} \frac{\partial x_{ij}}{\partial t_{2,3,1}} \frac{\partial Q}{\partial x_{ij}} = \frac{\partial Q}{\partial x_{4,4}} + \frac{\partial Q}{\partial x_{5,3}} = \left[\frac{\partial Q}{\partial \mathbf{X}}\right]_{4,4} + \left[\frac{\partial Q}{\partial \mathbf{X}}\right]_{5,3}. \tag{6.3}$$

Deep learning platforms like PyTorch provide the "fold" for aggregating the entries of $\frac{\partial Q}{\partial \mathbf{X}}$ to get $\frac{\partial Q}{\partial \mathbf{T}}$.[6] First, $\mathsf{reshape}$ $\frac{\partial Q}{\partial \mathbf{X}}$ to the $(d_1 - k_1 + 1) \times (d_2 - k_2 + 1) \times k_1 \times k_2 \times d_3$ order-5 tensor:

$$\frac{\partial Q}{\partial \overline{\mathbf{X}}} = \mathsf{reshape}\left(\frac{\partial Q}{\partial \mathbf{X}}, \left((d_1 - k_1 + 1),\ (d_2 - k_2 + 1),\ k_1,\ k_2,\ d_3\right)\right).$$

Then, apply "fold" to get the $d_1 \times d_2 \times d_3$ order-3 tensor $\frac{\partial Q}{\partial \mathbf{T}}$:

$$\frac{\partial Q}{\partial \mathbf{T}} = \mathsf{fold}\left(\frac{\partial Q}{\partial \overline{\mathbf{X}}}\right).$$

To summerize, the forward pass from $\mathbf{T}$ to $\mathbf{X}$ and the backward pass from $\mathbf{X}$ to $\mathbf{T}$ are

$$\mathbf{T} \xrightarrow{\mathsf{unfold}} \overline{\mathbf{X}} \xrightarrow{\mathsf{reshape}} \mathbf{X} \qquad \text{and} \qquad \frac{\partial Q}{\partial \mathbf{T}} \xleftarrow{\mathsf{fold}} \frac{\partial Q}{\partial \overline{\mathbf{X}}} \xleftarrow{\mathsf{reshape}} \frac{\partial Q}{\partial \mathbf{X}}.$$

[5] We let $x_{ij} = \mathbf{X}[i,j]$ denote the (i,j)-th entry of $\mathbf{X}$.

[6] The aggregation is according to the way the entries of $\mathbf{T}$ are copied to $\mathbf{X}$, e.g., $t_{2,3,1}$ is copied to $x_{4,4}$ and $x_{5,3}$.