

Introduction I

- Many deep learning courses have contents like
 - fully-connected networks
 - its optimization problem
 - its gradient (back propagation)
 - ...
 - other types of networks (e.g., CNN)
 - ...
- If I am a student of such courses, after seeing the significant differences of CNN from fully-connected networks, I wonder how the back propagation can be done

Introduction II

- The problem is that back propagation for CNN seems to be very complicated
- So fewer people talk about details
- Here we try to give a clear explanation

Gradient I

- Consider two layers m and $m + 1$. The variables between them are W^m and $\mathbf{b}^m$, so we aim to calculate

$$\frac{\partial f}{\partial W^m} = \frac{1}{C} W^m + \frac{1}{I} \sum_{i=1}^I \frac{\partial \xi_i}{\partial W^m}, \quad (1)$$

$$\frac{\partial f}{\partial \mathbf{b}^m} = \frac{1}{C} \mathbf{b}^m + \frac{1}{I} \sum_{i=1}^I \frac{\partial \xi_i}{\partial \mathbf{b}^m}. \quad (2)$$

- Note that (1) is in a **matrix** form

Gradient II

- Following past developments such as Vedaldi and Lenc (2015), it is easier to transform them to a **vector** form for the derivation.

Vector Form I

- For the convolutional layers, recall that

$$S^{m,i} = W^m \underbrace{\text{mat}(P_{\phi}^m P_{\text{pad}}^m \text{vec}(Z^{m,i}))}_{\phi(\text{pad}(Z^{m,i}))} + b^m \mathbb{1}_{a_{\text{conv}}^m b_{\text{conv}}^m}^T$$

$$Z^{m+1,i} = \text{mat}(P_{\text{pool}}^{m,i} \text{vec}(\sigma(S^{m,i})))_{d^{m+1} \times a^{m+1} b^{m+1}}, \quad (3)$$

Vector Form II

- We have

$$\begin{aligned} & \text{vec}(S^{m,i}) \\ &= \text{vec}(W^m \phi(\text{pad}(Z^{m,i}))) + \text{vec}(\mathbf{b}^m \mathbb{1}_{a_{\text{conv}}^m b_{\text{conv}}^m}^T) \\ &= (\mathcal{I}_{a_{\text{conv}}^m b_{\text{conv}}^m} \otimes W^m) \text{vec}(\phi(\text{pad}(Z^{m,i}))) + \\ & \quad (\mathbb{1}_{a_{\text{conv}}^m b_{\text{conv}}^m} \otimes \mathcal{I}_{d^{m+1}}) \mathbf{b}^m \end{aligned} \tag{4}$$

$$\begin{aligned} &= (\phi(\text{pad}(Z^{m,i}))^T \otimes \mathcal{I}_{d^{m+1}}) \text{vec}(W^m) + \\ & \quad (\mathbb{1}_{a_{\text{conv}}^m b_{\text{conv}}^m} \otimes \mathcal{I}_{d^{m+1}}) \mathbf{b}^m, \end{aligned} \tag{5}$$

Vector Form III

where $\mathcal{I}$ is an identity matrix. For example,

$$\mathcal{I}_{a_{\text{conv}}^m b_{\text{conv}}^m}$$

is an

$$a_{\text{conv}}^m b_{\text{conv}}^m \times a_{\text{conv}}^m b_{\text{conv}}^m$$

identity matrix. Eqs. (4) and (5) are respectively from

$$\text{vec}(AB) = (\mathcal{I} \otimes A) \text{vec}(B) \quad (6)$$

$$= (B^T \otimes \mathcal{I}) \text{vec}(A), \quad (7)$$

Vector Form IV

- Here $\otimes$ is the Kronecker product.
- What's the Kronecker product? If

$$A \in \mathbf{R}^{m \times n}$$

then

$$A \otimes B = \begin{bmatrix} a_{11}B & \cdots & a_{1n}B \\ & \vdots & \\ a_{m1}B & \cdots & a_{mn}B \end{bmatrix},$$

a much bigger matrix

Vector Form V

- For the fully-connected layers,

$$\begin{aligned} & \mathbf{s}^{m,i} \\ &= \mathbf{W}^m \mathbf{z}^{m,i} + \mathbf{b}^m \\ &= (\mathcal{I}_1 \otimes \mathbf{W}^m) \mathbf{z}^{m,i} + (\mathbb{1}_1 \otimes \mathcal{I}_{n_{m+1}}) \mathbf{b}^m \end{aligned} \quad (8)$$

$$= ((\mathbf{z}^{m,i})^T \otimes \mathcal{I}_{n_{m+1}}) \text{vec}(\mathbf{W}^m) + (\mathbb{1}_1 \otimes \mathcal{I}_{n_{m+1}}) \mathbf{b}^m, \quad (9)$$

where (8) and (9) are from (6) and (7), respectively.

- An advantage of using (4) and (8) is that they are in the same form.

Vector Form VI

- Further, if for fully-connected layers we define

$$\phi(\text{pad}(\mathbf{z}^{m,i})) = \mathcal{I}_{n_m} \mathbf{z}^{m,i}, \quad L^c < m \leq L + 1,$$

then (5) and (9) are in the same form.

- Thus we can derive the gradient of convolutional and fully-connected layers together

References I

- A. Vedaldi and K. Lenc. MatConvNet: Convolutional neural networks for matlab. In *Proceedings of the 23rd ACM International Conference on Multimedia*, pages 689–692, 2015.