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 $\boldsymbol{b}^m$, so we aim to calculate

$$\frac{\partial f}{\partial W^m} = \frac{1}{C} W^m + \frac{1}{l} \sum_{i=1}^{l} \frac{\partial \xi_i}{\partial W^m}, \tag{1}$$

$$\frac{\partial f}{\partial \boldsymbol{b}^m} = \frac{1}{C} \boldsymbol{b}^m + \frac{1}{l} \sum_{i=1}^{l} \frac{\partial \xi_i}{\partial \boldsymbol{b}^m}. \tag{2}$$

- 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

$$S^{m,i} = W^m \underbrace{\text{mat}(P_\phi^m P_{\text{pad}}^m \text{vec}(Z^{m,i}))_{h^m h^m d^m \times a_{\text{conv}}^m b_{\text{conv}}^m}}_{\phi(\text{pad}(Z^{m,i}))} +$$

$$\boldsymbol{b}^m \mathbb{1}_{a_{\text{conv}}^m b_{\text{conv}}^m}^T$$

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

Vector Form II

- We have

$$
\begin{aligned}
& \text{vec}(S^{m,i}) \\
=& \text{vec}(W^m \phi(\text{pad}(Z^{m,i}))) + \text{vec}(\boldsymbol{b}^m \mathbb{1}^T_{a^m_{\text{conv}} b^m_{\text{conv}}}) \\
=& \left(\mathcal{I}_{a^m_{\text{conv}} b^m_{\text{conv}}} \otimes W^m\right) \text{vec}(\phi(\text{pad}(Z^{m,i}))) + \\
& \left(\mathbb{1}_{a^m_{\text{conv}} b^m_{\text{conv}}} \otimes \mathcal{I}_{d^{m+1}}\right) \boldsymbol{b}^m \qquad (4) \\
=& \left(\phi(\text{pad}(Z^{m,i}))^T \otimes \mathcal{I}_{d^{m+1}}\right) \text{vec}(W^m) + \\
& \left(\mathbb{1}_{a^m_{\text{conv}} b^m_{\text{conv}}} \otimes \mathcal{I}_{d^{m+1}}\right) \boldsymbol{b}^m, \qquad (5)
\end{aligned}
$$

Vector Form III

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

$$\mathcal{I}_{a^m_{\text{conv}} b^m_{\text{conv}}}$$

is an

$$a^m_{\text{conv}} b^m_{\text{conv}} \times a^m_{\text{conv}} b^m_{\text{conv}}$$

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

$$\begin{aligned} \text{vec}(AB) &= (\mathcal{I} \otimes A)\text{vec}(B) &(6)\\ &= (B^T \otimes \mathcal{I})\text{vec}(A), &(7) \end{aligned}$$

Vector Form IV

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

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

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,

$$
\begin{aligned}
&\boldsymbol{s}^{m,i} \\
=& W^m \boldsymbol{z}^{m,i} + \boldsymbol{b}^m \\
=& \left(\mathcal{I}_1 \otimes W^m\right) \boldsymbol{z}^{m,i} + \left(\mathbb{1}_1 \otimes \mathcal{I}_{n_{m+1}}\right) \boldsymbol{b}^m \qquad (8) \\
=& \left((\boldsymbol{z}^{m,i})^T \otimes \mathcal{I}_{n_{m+1}}\right) \text{vec}(W^m) + \left(\mathbb{1}_1 \otimes \mathcal{I}_{n_{m+1}}\right) \boldsymbol{b}^m, \qquad (9)
\end{aligned}
$$

  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

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

  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.