Gradient Calculation

Chih-Jen Lin
National Taiwan University
Last updated: May 25, 2020

Outline

1. Introduction
2. Gradient Calculation
3. Computational Complexity
4. Discussion

Outline

1 Introduction

2 Gradient Calculation

3 Computational Complexity

4 Discussion

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
- Challenge: can we clearly describe it in a simple way?
- That's what we would like to try here

Outline

1 Introduction

2 Gradient Calculation

3 Computational Complexity

4 Discussion

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

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

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

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

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, and (4) and (5) are respectively from

$$
\begin{aligned}
\text{vec}(AB) &= (\mathcal{I} \otimes A)\text{vec}(B) && (6)\\
&= (B^T \otimes \mathcal{I})\text{vec}(A), && (7)\\
\text{vec}(AB)^T &= \text{vec}(B)^T(\mathcal{I} \otimes A^T) && (8)\\
&= \text{vec}(A)^T(B \otimes \mathcal{I}) && (9)
\end{aligned}
$$

- Here $\otimes$ is the Kronecker product.

Vector Form IV

- 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,

$$\begin{aligned} & \boldsymbol{s}^{m,i} \\ = & W^m \boldsymbol{z}^{m,i} + \boldsymbol{b}^m \\ = & \left(\mathcal{I}_1 \otimes W^m\right) \boldsymbol{z}^{m,i} + \left(\mathbb{1}_1 \otimes \mathcal{I}_{n_{m+1}}\right) \boldsymbol{b}^m \qquad (10) \\ = & \left(\left(\boldsymbol{z}^{m,i}\right)^T \otimes \mathcal{I}_{n_{m+1}}\right) \text{vec}(W^m) + \left(\mathbb{1}_1 \otimes \mathcal{I}_{n_{m+1}}\right) \boldsymbol{b}^m, \qquad (11) \end{aligned}$$

where (10) and (11) are from (6) and (7), respectively.

Vector Form VI

- An advantage of using (4) and (10) is that they are in the same form.
- 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 (11) are in the same form.
- Thus we can derive the gradient of convolutional and fully-connected layers together

Gradient Calculation I

- For convolutional layers, from (5),

$$
\begin{aligned}
\frac{\partial \xi_i}{\partial \text{vec}(W^m)^T} &= \frac{\partial \xi_i}{\partial \text{vec}(S^{m,i})^T} \frac{\partial \text{vec}(S^{m,i})}{\partial \text{vec}(W^m)^T} \\
=& \frac{\partial \xi_i}{\partial \text{vec}(S^{m,i})^T} \left(\phi(\text{pad}(Z^{m,i}))^T \otimes \mathcal{I}_{d^{m+1}} \right) \\
=& \text{vec} \left(\frac{\partial \xi_i}{\partial S^{m,i}} \phi(\text{pad}(Z^{m,i}))^T \right)^T \qquad (12)
\end{aligned}
$$

  where (12) is from (9).

- We applied chain rule here

Gradient Calculation II

- Note that we define

$$\frac{\partial \boldsymbol{y}}{\partial (\boldsymbol{x})^T} = \begin{bmatrix} \frac{\partial y_1}{\partial x_1} & \cdots & \frac{\partial y_1}{\partial x_{|\mathrm{x}|}} \\ \vdots & \ddots & \vdots \\ \frac{\partial y_{|\mathrm{y}|}}{\partial x_1} & \cdots & \frac{\partial y_{|\mathrm{y}|}}{\partial x_{|\mathrm{x}|}} \end{bmatrix}, \tag{13}$$

  where $\boldsymbol{x}$ and $\boldsymbol{y}$ are column vectors, and $|\boldsymbol{x}|$, $|\boldsymbol{y}|$ are their lengths.

Gradient Calculation III

- Thus if

$$\boldsymbol{y} = A\boldsymbol{x}$$

then

$$\frac{\partial \boldsymbol{y}}{\partial (\boldsymbol{x})^T} = \begin{bmatrix} A_{11} & A_{12} & \cdots \\ A_{21} & & \\ \vdots & & \end{bmatrix} = A$$

Gradient Calculation IV

- Similarly

$$
\begin{aligned}
\frac{\partial \xi_i}{\partial (\boldsymbol{b}^m)^T} =& \frac{\partial \xi_i}{\partial \text{vec}(S^{m,i})^T} \frac{\partial \text{vec}(S^{m,i})}{\partial (\boldsymbol{b}^m)^T} \\
=& \frac{\partial \xi_i}{\partial \text{vec}(S^{m,i})^T} \left(\mathbb{1}_{a^m_{\text{conv}} b^m_{\text{conv}}} \otimes \mathcal{I}_{d^{m+1}} \right) \\
=& \text{vec} \left(\frac{\partial \xi_i}{\partial S^{m,i}} \mathbb{1}_{a^m_{\text{conv}} b^m_{\text{conv}}} \right)^T, \qquad (14)
\end{aligned}
$$

where (14) is from (9).

Gradient Calculation V

- To calculate (12), $\phi(\text{pad}(Z^{m,i}))$ has been available from the forward process of calculating the function value.
- In (12) and (14), $\partial\xi_i/\partial S^{m,i}$ is also needed
- We will show that it can be obtained by a backward process.

Calculation of $\partial \xi_i / \partial S^{m,i}$ I

- What we will do is to assume that $\partial \xi_i / \partial Z^{m+1,i}$ is available
- Then we show details of calculating

$$\frac{\partial \xi_i}{\partial S^{m,i}} \text{ and } \frac{\partial \xi_i}{\partial Z^{m,i}}$$

  for layer m.
- Thus a back propagation process

Calculation of $\partial \xi_i / \partial S^{m,i}$ II

- We have the following workflow.

$$
\begin{aligned}
Z^{m,i} &\leftarrow \text{padding} \leftarrow \text{convolution} \leftarrow \sigma(S^{m,i}) \\
&\leftarrow \text{pooling} \leftarrow Z^{m+1,i}.
\end{aligned} \tag{15}
$$

- Assume the RELU activation function is used

$$
\frac{\partial \xi_i}{\partial \text{vec}(S^{m,i})^T} = \frac{\partial \xi_i}{\partial \text{vec}(\sigma(S^{m,i}))^T} \frac{\partial \text{vec}(\sigma(S^{m,i}))}{\partial \text{vec}(S^{m,i})^T}
$$

Calculation of $\partial \xi_i / \partial S^{m,i}$ III

- Note that
$$\frac{\partial \text{vec}(\sigma(S^{m,i}))}{\partial \text{vec}(S^{m,i})^T}$$
is a squared diagonal matrix of
$$|\text{vec}(S^{m,i})| \times |\text{vec}(S^{m,i})|$$

- Recall that we assume
$$\sigma'(x) = \begin{cases} 1 & \text{if } x > 0 \\ 0 & \text{otherwise} \end{cases}$$
though $\sigma(x)$ is not differentiable at $x = 0$

Calculation of $\partial \xi_i / \partial S^{m,i}$ IV

- We can define

$$I[S^{m,i}]_{(p,q)} = \begin{cases} 1 & \text{if } S^{m,i}_{(p,q)} > 0, \\ 0 & \text{otherwise,} \end{cases}$$

and have

$$\frac{\partial \xi_i}{\partial \text{vec}(S^{m,i})^T} = \frac{\partial \xi_i}{\partial \text{vec}(\sigma(S^{m,i}))^T} \odot \text{vec}(I[S^{m,i}])^T$$

where $\odot$ is Hadamard product (i.e., element-wise products)

Calculation of $\partial \xi_i / \partial S^{m,i}$ V

- Q: can we extend this to other scalar activation functions?
- Yes, the general form is

$$\frac{\partial \xi_i}{\partial \text{vec}(S^{m,i})^T} = \frac{\partial \xi_i}{\partial \text{vec}(\sigma(S^{m,i}))^T} \odot \text{vec}(\sigma'(S^{m,i}))^T$$

- Next,

Calculation of $\partial \xi_i / \partial S^{m,i}$ VI

$$
\begin{aligned}
&\frac{\partial \xi_i}{\partial \mathrm{vec}(S^{m,i})^T} \\
=&\frac{\partial \xi_i}{\partial \mathrm{vec}(Z^{m+1,i})^T} \frac{\partial \mathrm{vec}(Z^{m+1,i})}{\partial \mathrm{vec}(\sigma(S^{m,i}))^T} \frac{\partial \mathrm{vec}(\sigma(S^{m,i}))}{\partial \mathrm{vec}(S^{m,i})^T} \\
=&\left(\frac{\partial \xi_i}{\partial \mathrm{vec}(Z^{m+1,i})^T} \frac{\partial \mathrm{vec}(Z^{m+1,i})}{\partial \mathrm{vec}(\sigma(S^{m,i}))^T} \right) \odot \mathrm{vec}(I[S^{m,i}])^T \\
=&\left(\frac{\partial \xi_i}{\partial \mathrm{vec}(Z^{m+1,i})^T} P_{\mathrm{pool}}^{m,i} \right) \odot \mathrm{vec}(I[S^{m,i}])^T \qquad (16)
\end{aligned}
$$

- Note that (16) is from (3)

Calculation of $\partial \xi_i / \partial S^{m,i}$ VII

- If a general scalar activation function is considered, (16) is changed to

$$\frac{\partial \xi_i}{\partial \text{vec}(S^{m,i})^T} = \left(\frac{\partial \xi_i}{\partial \text{vec}(Z^{m+1,i})^T} P_{\text{pool}}^{m,i} \right) \odot \text{vec}(\sigma'(S^{m,i}))^T$$

- In the end we calculate $\partial \xi_i / \partial Z^{m,i}$ and pass it to the previous layer.

Calculation of $\partial \xi_i / \partial S^{m,i}$ VIII

$$
\begin{aligned}
&\frac{\partial \xi_i}{\partial \text{vec}(Z^{m,i})^T} \\
=&\frac{\partial \xi_i}{\partial \text{vec}(S^{m,i})^T} \frac{\partial \text{vec}(S^{m,i})}{\partial \text{vec}(\phi(\text{pad}(Z^{m,i})))^T} \frac{\partial \text{vec}(\phi(\text{pad}(Z^{m,i})))}{\partial \text{vec}(\text{pad}(Z^{m,i}))^T} \\
&\frac{\partial \text{vec}(\text{pad}(Z^{m,i}))}{\partial \text{vec}(Z^{m,i})^T} \\
=&\frac{\partial \xi_i}{\partial \text{vec}(S^{m,i})^T} \left(\mathcal{I}_{a^m_{\text{conv}} b^m_{\text{conv}}} \otimes W^m \right) P^m_\phi P^m_{\text{pad}} && (17) \\
=&\text{vec}\left((W^m)^T \frac{\partial \xi_i}{\partial S^{m,i}} \right)^T P^m_\phi P^m_{\text{pad}}, && (18)
\end{aligned}
$$

Calculation of $\partial \xi_i / \partial S^{m,i}$ IX

where (17) is from (4) and (18) is from (8).

Fully-connected Layers I

- For fully-connected layers, by the same form in (10), (11), (4) and (5), we immediately get results from (12), (14), (16) and (18).

$$\frac{\partial \xi_i}{\partial \text{vec}(W^m)^T} = \text{vec}\left(\frac{\partial \xi_i}{\partial \boldsymbol{s}^{m,i}}(\boldsymbol{z}^{m,i})^T\right)^T \quad (19)$$

$$\frac{\partial \xi_i}{\partial (\boldsymbol{b}^m)^T} = \frac{\partial \xi_i}{\partial (\boldsymbol{s}^{m,i})^T} \quad (20)$$

Fully-connected Layers II

$$\frac{\partial \xi_i}{\partial (\boldsymbol{z}^{m,i})^T} = \left((W^m)^T \frac{\partial \xi_i}{\partial (\boldsymbol{s}^{m,i})} \right)^T \mathcal{I}_{n_m}$$

$$= \left((W^m)^T \frac{\partial \xi_i}{\partial (\boldsymbol{s}^{m,i})} \right)^T, \qquad (21)$$

where

$$\frac{\partial \xi_i}{\partial (\boldsymbol{s}^{m,i})^T} = \frac{\partial \xi_i}{\partial (\boldsymbol{z}^{m+1,i})^T} \odot I[\boldsymbol{s}^{m,i}]^T. \qquad (22)$$

- Finally, we check the initial values of the backward process.

Fully-connected Layers III

- Assume that the squared loss is used and in the last layer we have an identity activation function
- Then

$$\frac{\partial \xi_i}{\partial \boldsymbol{z}^{L+1,i}} = 2(\boldsymbol{z}^{L+1,i} - \boldsymbol{y}^i), \text{ and } \frac{\partial \xi_i}{\partial \boldsymbol{s}^{L,i}} = \frac{\partial \xi_i}{\partial \boldsymbol{z}^{L+1,i}}.$$

Notes on Practical Implementations I

- Recall we said that in

$$\frac{\partial \xi_i}{\partial W^m} = \frac{\partial \xi_i}{\partial S^{m,i}} \phi(\text{pad}(Z^{m,i}))^T,$$

  $Z^{m,i}$ is available from the forward process
- Therefore

$$Z^{m,i}, \forall m$$

  are stored.

Notes on Practical Implementations II

- But we also need $S^{m,i}$ for

$$\frac{\partial \xi_i}{\partial \text{vec}(S^{m,i})^T} = \left(\frac{\partial \xi_i}{\partial \text{vec}(Z^{m+1,i})^T} P_{\text{pool}}^{m,i} \right) \odot \text{vec}(I[S^{m,i}])^T \tag{23}$$

- Do we need to store both $Z^{m,i}$ and $S^{m,i}$?

Notes on Practical Implementations III

- We can avoid storing $S^{m,i}, \forall m$ by replacing (23) with

$$
\begin{aligned}
&\frac{\partial \xi_i}{\partial \text{vec}(S^{m,i})^T} \\
=& \left(\frac{\partial \xi_i}{\partial \text{vec}(Z^{m+1,i})^T} \odot \text{vec}(I[Z^{m+1,i}])^T \right) P_{\text{pool}}^{m,i}.
\end{aligned} \tag{24}
$$

- Why? Let's look at the relation between $Z^{m+1,i}$ and $S^{m,i}$

$$
Z^{m+1,i} = \text{mat}(P_{\text{pool}}^{m,i} \text{vec}(\sigma(S^{m,i})))
$$

Notes on Practical Implementations IV

- $Z^{m+1,i}$ is a "smaller matrix" than $S^{m,i}$
- That is, (23) is a "reverse mapping" of the pooling operation
- In (23),

$$\frac{\partial \xi_i}{\partial \text{vec}(Z^{m+1,i})^T} \times P_{\text{pool}}^{m,i} \tag{25}$$

  generates a large zero vector and puts values of $\partial \xi_i / \partial \text{vec}(Z^{m+1,i})^T$ into positions selected earlier in the max pooling operation.
- Then, element-wise multiplications of (25) and $I[S^{m,i}]^T$ are conducted.

Notes on Practical Implementations V

- Positions not selected in the max pooling procedure are zeros after (25)
- They are still zeros after the Hadamard product between (25) and $I[S^{m,i}]^T$
- Thus, (23) and (24) give the same results.
- An illustration using our earlier example. This illustration was generated with the help of Cheng-Hung Liu in my group

Notes on Practical Implementations VI

- Recall an earlier pooling example is

$$\text{image B} \left[\begin{array}{cc|cc} 3 & 2 & 3 & 6 \\ 4 & 5 & 4 & 9 \\ \hline 2 & 1 & 2 & 6 \\ 3 & 4 & 3 & 2 \end{array}\right] \rightarrow \begin{bmatrix} 5 & 9 \\ 4 & 6 \end{bmatrix}$$

- The corresponding pooling matrix is

$$P_{\text{pool}} = \begin{bmatrix} 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 \end{bmatrix}$$

Notes on Practical Implementations VII

- We have that

$$P_{\text{pool}}\text{vec}(\text{image}) = \begin{bmatrix} 5 \\ 4 \\ 9 \\ 6 \end{bmatrix} = \text{vec}(\begin{bmatrix} 5 & 9 \\ 4 & 6 \end{bmatrix})$$

- If using (23),

$$\begin{aligned} &\boldsymbol{v}^T P_{\text{pool}} \odot \text{vec}(I[S^m])^T \\ = &\begin{bmatrix} 0 & 0 & 0 & 0 & 0 & v_1 & 0 & v_2 & 0 & 0 & 0 & 0 & 0 & v_3 & v_4 & 0 \end{bmatrix} \\ &\odot \\ &\begin{bmatrix} 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 \end{bmatrix} \end{aligned}$$

Notes on Practical Implementations VIII

- If using (24),

$$
\begin{aligned}
&(\boldsymbol{v}^T \odot \text{vec}(I[Z^{m+1}])^T) P_{\text{pool}} \\
=&(\boldsymbol{v}^T \odot \begin{bmatrix} 1 & 1 & 1 & 1 \end{bmatrix}) P_{\text{pool}} \\
=&\begin{bmatrix} 0 & 0 & 0 & 0 & 0 & v_1 & 0 & v_2 & 0 & 0 & 0 & 0 & 0 & v_3 & v_4 & 0 \end{bmatrix}
\end{aligned}
$$

- So they are the same
- In the derivation we used the properties of
  - RELU activation function and
  - max pooling

Notes on Practical Implementations IX

to get

a $Z^{m+1,i}$ component > 0 or not

$\Leftrightarrow$the corresponding $\sigma'(S^{m,i})$ component > 0 or not

- For general cases we might not be able to avoid storing $\sigma'(S^{m,i})$?
- We may go back to this issue later in discussing the implementation issues

Summary of Operations I

- We show convolutional layers only and the bias term is omitted
- Operations in order

$$\frac{\partial \xi_i}{\partial \text{vec}(S^{m,i})^T} = \left(\frac{\partial \xi_i}{\partial \text{vec}(Z^{m+1,i})^T} \odot \text{vec}(I[Z^{m+1,i}])^T \right) P_{\text{pool}}^{m,i}. \quad (26)$$

$$\frac{\partial \xi_i}{\partial W^m} = \frac{\partial \xi_i}{\partial S^{m,i}} \phi(\text{pad}(Z^{m,i}))^T \quad (27)$$

Summary of Operations II

$$\frac{\partial \xi_i}{\partial \text{vec}(Z^{m,i})^T} = \text{vec}\left((W^m)^T \frac{\partial \xi_i}{\partial S^{m,i}} \right)^T P_\phi^m P_{\text{pad}}^m, \tag{28}$$

- Note that after (26), we change

$$\text{a vector } \frac{\partial \xi_i}{\partial \text{vec}(S^{m,i})^T} \text{ to a matrix } \frac{\partial \xi_i}{\partial S^{m,i}}$$

  because in (27) and (28), matrix form is needed
- In (26), information of the next layer is used.

Summary of Operations III

- Instead we can do

$$\frac{\partial \xi_i}{\partial \text{vec}(Z^{m,i})^T} \odot \text{vec}(I[Z^{m,i}])^T$$

  in the end of the current layer

  This becomes the information passed to the previous layer
- Then only information in the current layer is used

Summary of Operations IV

- Finally an implementation for one convolutional layer:

$$\Delta \leftarrow \text{mat}(\text{vec}(\Delta)^T P_{\text{pool}}^{m,i})$$

$$\frac{\partial \xi_i}{\partial W^m} = \Delta \cdot \phi(\text{pad}(Z^{m,i}))^T$$

$$\Delta \leftarrow \text{vec}\left(\left(W^m\right)^T \Delta\right)^T P_\phi^m P_{\text{pad}}^m$$

$$\Delta \leftarrow \Delta \odot I[Z^{m,i}]$$

- A sample segment of code

Summary of Operations V

```
for m = LC : -1 : 1
   if model.wd_subimage_pool(m) > 1
      dXidS = reshape(vTP(param, model, net, m,
                       dXidS, 'pool_gradient'),
                       model.ch_input(m+1), []);
   end
   phiZ = padding_and_phiZ(model, net, m);
   net.dlossdW{m} = dXidS*phiZ';
   net.dlossdb{m} = dXidS*ones(model.wd_conv(m)*
                    model.ht_conv(m)*S_k, 1);
```

Summary of Operations VI

```
if m > 1
   V = model.weight{m}' * dXidS;
   dXidS = reshape(vTP(param, model, net, m,
                       V, 'phi_gradient'),
                       model.ch_input(m), []);

   % vTP_pad
   a = model.ht_pad(m); b = model.wd_pad(m);
   dXidS = dXidS(:, net.idx_pad{m} +
                     a*b*[0:S_k-1]);
```

Summary of Operations VII

```
        % activation function
        dXidS = dXidS.*(net.Z{m} > 0);
    end
end
```

Storing $\phi(\text{pad}(Z^{m,i}))$

- From the above summary, we see that

  $$\phi(\text{pad}(Z^{m,i}))$$

  is calculated twice in both forward and backward processes
- If this expansion is expensive, we can store it
- But memory is a concern as this is a huge matrix
- So this setting trades space for time
- It's more suitable for CPU environments

Outline

1 Introduction

2 Gradient Calculation

3 Computational Complexity

4 Discussion

Complexity I

- To see where the computational bottleneck is, it's important to check the complexity of major operations
- Assume l is the number of data (for the case of calculating the whole gradient)
- For stochastic gradient, l becomes the size of a mini-batch

Complexity II

- Forward:

$$
\begin{aligned}
& W^m \text{mat}(P_\phi^m P_{\text{pad}}^m \text{vec}(Z^{m,i})) \\
= & W^m \phi(\text{pad}(Z^{m,i}))
\end{aligned}
$$

$$\phi(\text{pad}(Z^{m,i})) : \mathcal{O}(l \times h^m h^m d^m a^m_{\text{conv}} b^m_{\text{conv}})$$

$$W^m \phi(\cdot) : \mathcal{O}(l \quad \times \quad d^{m+1} \quad h^m h^m d^m \quad a^m_{\text{conv}} b^m_{\text{conv}})$$

$$Z^{m+1,i} = \text{mat}(P_{\text{pool}}^{m,i} \text{vec}(\sigma(S^{m,i})))$$

$$
\begin{aligned}
& \mathcal{O}(l \times h^m h^m d^{m+1} a^{m+1} b^{m+1}) \\
= & \mathcal{O}(l \times d^{m+1} a^m_{\text{conv}} b^m_{\text{conv}})
\end{aligned}
$$

Complexity III

- Backward:

$$\Delta \leftarrow \text{mat}(\text{vec}(\Delta)^T P_{\text{pool}}^{m,i})$$

$$\mathcal{O}(l \times d^{m+1} a_{\text{conv}}^m b_{\text{conv}}^m)$$

$$\frac{\partial \xi_i}{\partial W^m} = \Delta \phi(\text{pad}(Z^{m,i}))^T$$

$$\mathcal{O}(l \quad \times \quad d^{m+1} \quad a_{\text{conv}}^m b_{\text{conv}}^m \quad h^m h^m d^m).$$

$$\Delta \leftarrow \text{vec}\left(\left(W^m\right)^T \Delta\right)^T P_\phi^m P_{\text{pad}}^m$$

Complexity IV

$(W^m)^T \Delta : \mathcal{O}(l \times h^m h^m d^m \quad d^{m+1} \quad a^m_{\text{conv}} b^m_{\text{conv}})$

$\text{vec}(\cdot) P^m_\phi : \mathcal{O}(l \times h^m h^m d^m a^m_{\text{conv}} b^m_{\text{conv}})$

Here we convert a matrix of

$$h^m h^m d^m \times a^m_{\text{conv}} b^m_{\text{conv}}$$

to a smaller matrix

$$d^m \times a^m_{\text{pad}} b^m_{\text{pad}}$$

- We see that matrix-matrix products are the bottleneck

Complexity V

- If so, why check others?
- The issue is that matrix-matrix products may be better optimized
- You will get first-hand experiences in doing projects

Outline

1 Introduction

2 Gradient Calculation

3 Computational Complexity

4 Discussion

Discussion I

- We tried to have a simple way to describe the gradient calculation for CNN
- Is the description good enough? Can we do better?

Discussion: Pooling and Differentiability I

- Recall we have

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

- We note that

$$P_{\text{pool}}^{m,i}$$

  is not a constant $0/1$ matrix
- It depends on $\sigma(S^{m,i})$ to decide the positions of 0 and 1.

Discussion: Pooling and Differentiability II

- Thus like the RELU activation function, max pooling is another place to cause that $f(\boldsymbol{\theta})$ is not differentiable
- However, it is almost differentiable around the current point
- Consider

$$f(A) = \max\left(\begin{bmatrix} A_{11} & A_{12} \\ A_{21} & A_{22} \end{bmatrix}\right)$$

and

$$A_{11} > A_{12}, A_{21}, A_{22}$$

Discussion: Pooling and Differentiability III

- Then

$$\nabla f(A) = \begin{bmatrix} 1 \\ 0 \\ 0 \\ 0 \end{bmatrix} \text{ at } A = \begin{bmatrix} A_{11} & A_{12} \\ A_{21} & A_{22} \end{bmatrix}$$

- This explains why we can use $P_{\text{pool}}^{m,i}$ in function and gradient evaluations

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.