Gradient Calculation

Chih-Jen Lin
National Taiwan University
Last updated: May 9, 2019
Outline

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

1. Introduction
2. Gradient Calculation
3. Computational Complexity
4. Discussion
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.
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
Consider two layers $m$ and $m+1$. The variables between them are $W^m$ and $b^m$, so we aim to calculate

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

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

Note that (1) is in a matrix form.
Following past developments such as Vedaldi and Lenc (2015), it is easier to transform them to a vector form for the derivation.
For the convolutional layers, recall that

\[ S^{m,i} = W^m \text{mat}(P^m_{\phi} P^m_{\text{pad}} \text{vec}(Z^{m,i})))_{h^m h^m d^m \times a^m_{\text{conv}} b^m_{\text{conv}}} + \phi(\text{pad}(Z^{m,i})) \]

\[ b^m 1^T \]

\[ a^m_{\text{conv}} b^m_{\text{conv}} \]

\[ Z^{m+1,i} = \text{mat}(P^m_{\text{pool}} \text{vec}(\sigma(S^{m,i})))_{d^{m+1} \times a^{m+1} b^{m+1}}, \quad (3) \]
Gradient Calculation

**Vector Form II**

- We have

\[
\text{vec}(S_{m,i}^m) = \text{vec}(W^m \phi(\text{pad}(Z_{m,i}^m))) + \text{vec}(b_m^m \mathbf{1}_{a_{\text{conv}}^m}^T b_{\text{conv}}^m) \\
= (\mathcal{I}_{a_{\text{conv}}^m} b_{\text{conv}}^m \otimes W^m) \text{vec}(\phi(\text{pad}(Z_{m,i}^m))) + (\mathbf{1}_{a_{\text{conv}}^m} b_{\text{conv}}^m \otimes \mathcal{I}_{d_{m+1}^m}) b^m
\]

(4)

\[
= (\phi(\text{pad}(Z_{m,i}^m))^T \otimes \mathcal{I}_{d_{m+1}^m}) \text{vec}(W^m) + (\mathbf{1}_{a_{\text{conv}}^m} b_{\text{conv}}^m \otimes \mathcal{I}_{d_{m+1}^m}) b^m
\]

(5)
Gradient Calculation

Vector Form III

where $\mathcal{I}$ is an identity matrix, and (4) and (5) are respectively from

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

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

\[ A \in \mathbb{R}^{m \times n} \]

then

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

a much bigger matrix
For the fully-connected layers, 

\[ s^{m,i} = W^m z^{m,i} + b^m \]

\[ = (I_1 \otimes W^m) z^{m,i} + (1_1 \otimes I_{n_{m+1}}) b^m \]

\[ = ((z^{m,i})^T \otimes I_{n_{m+1}}) \text{vec}(W^m) + (1_1 \otimes I_{n_{m+1}}) b^m, \] (10)

\[ (11) \]

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
  \[ \phi(\text{pad}(z_{m,i}^m)) = \mathcal{I}_{nm}z_{m,i}^m, \quad L^c < m \leq L + 1, \]
  then (5) and (11) are in the same form.
- Thus we can derive the gradient of convolutional and fully-connected layers together
Gradient Calculation

For convolutional layers, from (5),

\[
\frac{\partial \xi_i}{\partial \text{vec}(W^m)^T} = \frac{\partial \xi_i}{\partial \text{vec}(S^{m,i})^T} \frac{\partial \text{vec}(S^{m,i})}{\partial \text{vec}(W^m)^T}
\]

\[
= \frac{\partial \xi_i}{\partial \text{vec}(S^{m,i})^T} \left( \phi(\text{pad}(Z^{m,i}))^T \bigotimes \mathcal{I}_{d^{m+1}} \right)
\]

\[
= \text{vec} \left( \frac{\partial \xi_i}{\partial S^{m,i}} \phi(\text{pad}(Z^{m,i}))^T \right)^T
\]  \hspace{1cm} (12)

where (12) is from (9).

We applied chain rule here.
Note that we define

\[
\frac{\partial \mathbf{y}}{\partial (\mathbf{x})^T} = \begin{bmatrix}
\frac{\partial y_1}{\partial x_1} & \cdots & \frac{\partial y_1}{\partial x_{|\mathbf{x}|}} \\
\vdots & \ddots & \vdots \\
\frac{\partial y_{|\mathbf{y}|}}{\partial x_1} & \cdots & \frac{\partial y_{|\mathbf{y}|}}{\partial x_{|\mathbf{x}|}}
\end{bmatrix},
\]  

(13)

where \( \mathbf{x} \) and \( \mathbf{y} \) are column vectors.
Thus if

\[ y = Ax \]

then

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

\[
\frac{\partial \xi_i}{\partial (b^m)^T} = \frac{\partial \xi_i}{\partial \text{vec}(S^{m,i})^T} \frac{\partial \text{vec}(S^{m,i})}{\partial (b^m)^T} = \frac{\partial \xi_i}{\partial \text{vec}(S^{m,i})^T} \left( \mathbb{1} a_{\text{conv}}^m b_{\text{conv}}^m \otimes I_{d^{m+1}} \right)
\]

\[
= \text{vec} \left( \frac{\partial \xi_i}{\partial S^{m,i}} \mathbb{1} a_{\text{conv}}^m b_{\text{conv}}^m \right)^T,
\]

where (14) is from (9).
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 $\frac{\partial \xi_i}{\partial S_{m,i}}$

- What we will do is to assume that $\frac{\partial \xi_i}{\partial Z_{m+1,i}}$ 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.
- We have the following workflow.

  $$Z_{m,i} \leftarrow \text{padding} \leftarrow \text{convolution} \leftarrow \sigma(S_{m,i}) \leftarrow \text{pooling} \leftarrow Z_{m+1,i}. \quad (15)$$
Calculation of $\partial \xi_i / \partial S^{m,i}$

- Assume the RELU activation function is used

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

- Note that

\[
\frac{\partial \text{vec}(\sigma(S^{m,i}))}{\partial \text{vec}(S^{m,i})^T}
\]

is a squared diagonal matrix
Calculation of $\partial \xi_i / \partial S^{m,i}$

- Recall that we assume
  \[\sigma'(x) = \begin{cases} 
  1 & \text{if } x > 0 \\
  0 & \text{otherwise}
\end{cases}\]

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

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

- Q: can we extend this to other 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 $\frac{\partial \xi_i}{\partial S^{m,i}}$

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

$$= \frac{\partial \xi_i}{\partial \text{vec}(Z^{m+1,i})^T} \frac{\partial \text{vec}(Z^{m+1,i})}{\partial \text{vec}(\sigma(S^{m,i}))^T} \frac{\partial \text{vec}(\sigma(S^{m,i}))}{\partial \text{vec}(S^{m,i})^T}$$

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

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

Note that (16) is from (3)
Calculation of $\partial \xi_i / \partial S^{m,i}$ VI

- If a general 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 \( \frac{\partial \xi_i}{\partial S_{m,i}} \) VII

\[
\frac{\partial \xi_i}{\partial \text{vec}(Z_{m,i})^T} = \frac{\partial \xi_i}{\partial \text{vec}(S_{m,i})^T} \frac{\partial \text{vec}(S_{m,i})}{\partial \text{vec}(\phi(\text{pad}(Z_{m,i})))^T} \frac{\partial \text{vec}(\phi(\text{pad}(Z_{m,i})))}{\partial \text{vec}(\text{pad}(Z_{m,i}))^T} \frac{\partial \text{vec}(\text{pad}(Z_{m,i}))}{\partial \text{vec}(Z_{m,i})^T} \\
= \frac{\partial \xi_i}{\partial \text{vec}(S_{m,i})^T} \left( \mathcal{I} a_{\text{conv}} \otimes b_{\text{conv}} \otimes W^m \right) P_{\phi}^m P_{\text{pad}}^m \\
= \text{vec} \left( (W^m)^T \frac{\partial \xi_i}{\partial S_{m,i}} \right)^T P_{\phi}^m P_{\text{pad}}^m,
\]

(17)
where (18) is from (8).
For fully-connected layers, by the same form in (10), (11), (4) and (5), we immediately get results from (12), (14), (16) and (18).

\[
\frac{\partial \xi_i}{\partial \text{vec}(W^m)^T} = \text{vec} \left( \frac{\partial \xi_i}{\partial s_{m,i}} (z_{m,i}^T)^T \right)^T
\]  
(19)

\[
\frac{\partial \xi_i}{\partial (b^m)^T} = \frac{\partial \xi_i}{\partial (s_{m,i})^T}
\]  
(20)
Fully-connected Layers II

\[
\frac{\partial \xi_i}{\partial (z^{m,i})^T} = \left( (W^m)^T \frac{\partial \xi_i}{\partial (s^{m,i})} \right)^T I_{nm}
\]

\[
= \left( (W^m)^T \frac{\partial \xi_i}{\partial (s^{m,i})} \right)^T, \quad (21)
\]

where

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

Finally, we check the initial values of the backward process.
Assume that the squared loss is used and in the last layer we have an identity activation function.

Then

\[
\frac{\partial \xi_i}{\partial z^{L+1,i}} = 2(z^{L+1,i} - y^i), \quad \text{and} \quad \frac{\partial \xi_i}{\partial s^{L,i}} = \frac{\partial \xi_i}{\partial z^{L+1,i}}.
\]
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.
But we also need $S^{m,i}$ for

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

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 (16) with

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

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

$$Z^{m+1,i} = \text{mat}(P_{\text{pool}}^{m,i} \text{vec}(\sigma(S^{m,i})))$$
\( Z^{m+1,i} \) is a “smaller matrix” than \( S^{m,i} \)

That is, (16) is a “reverse mapping” of the pooling operation

In (16),

\[
\frac{\partial \xi_i}{\partial \text{vec}(Z^{m+1,i})^T} \times P^{m,i}_{\text{pool}}
\]

(24)

generates a large zero vector and puts values of \( \frac{\partial \xi_i}{\partial \text{vec}(Z^{m+1,i})^T} \) into positions selected earlier in the max pooling operation.

Then, element-wise multiplications of (24) and \( I[S^{m,i}]^T \) are conducted.
Positions not selected in the max pooling procedure are zeros after (24).

They are still zeros after the Hadamard product between (24) and $I[S^m,i]^T$

Thus, (16) and (23) give the same results.

An illustration using our earlier example. This illustration was generated with the help of Cheng-Hung Liu in my group.
Recall an earlier pooling example is

\[
\begin{bmatrix}
3 & 2 & 3 & 6 \\
4 & 5 & 4 & 9 \\
2 & 1 & 2 & 6 \\
3 & 4 & 3 & 2
\end{bmatrix}
\rightarrow
\begin{bmatrix}
5 & 9 \\
4 & 6
\end{bmatrix}
\]

The corresponding pooling matrix is

\[
P_{\text{pool}} = \begin{bmatrix}
0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1
\end{bmatrix}
\]
We have that

\[
P_{\text{pool}} \text{vec(image)} = \begin{bmatrix} 5 \\ 4 \\ 9 \\ 6 \end{bmatrix} = \text{vec}\left(\begin{bmatrix} 5 & 9 \\ 4 & 6 \end{bmatrix}\right)
\]

If using (16),

\[
\mathbf{v}^T P_{\text{pool}} \odot \text{vec}(I[S^m])^T
\]

\[
= \begin{bmatrix} 0 & 0 & 0 & 0 & 0 & \nu_1 & 0 & \nu_2 & 0 & 0 & 0 & 0 & 0 & \nu_3 & \nu_4 & 0 \end{bmatrix} \odot
\]

\[
\begin{bmatrix} 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 \end{bmatrix}
\]

\[
= \begin{bmatrix} 0 & 0 & 0 & 0 & 0 & \nu_1 & 0 & \nu_2 & 0 & 0 & 0 & 0 & 0 & \nu_3 & \nu_4 & 0 \end{bmatrix}
\]
If using (23),

\[
(\mathbf{v}^T \odot \text{vec}(I[Z^{m+1}]))^T \mathbf{P}_{\text{pool}}
\]

\[
= (\mathbf{v}^T \odot [1 \ 1 \ 1 \ 1]) \mathbf{P}_{\text{pool}}
\]

\[
= \begin{bmatrix}
0 & 0 & 0 & 0 & 0 & 0 & v_1 & 0 & v_2 & 0 & 0 & 0 & 0 & 0 & v_3 & v_4 & 0
\end{bmatrix}
\]

So they are the same

In the derivation we used the properties of

- RELU activation function and
- max pooling
to get

\[ a \ Z^{m+1,i} \ \text{component} > 0 \ \text{or not} \]

\[ \iff \ \text{the corresponding} \ \sigma'(S^{m,i}) \ \text{component} > 0 \ \text{or not} \]

- 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

\[
\frac{\partial \xi_i}{\partial \text{vec}(S^{m,i})^T} = \left( \frac{\partial \xi_i}{\partial \text{vec}(Z^{m+1,i})^T} \otimes \text{vec}(I[Z^{m+1,i}])^T \right) P_{m,i}^{pool}. \tag{25}
\]

\[
\frac{\partial \xi_i}{\partial W^m} = \frac{\partial \xi_i}{\partial S^{m,i}} \phi(\text{pad}(Z^{m,i}))^T \tag{26}
\]
Gradient Calculation

Summary of Operations II

\[
\frac{\partial \xi_i}{\partial \text{vec}(Z^{m,i})^T} = \text{vec} \left( (W^m)^T \frac{\partial \xi_i}{\partial S^{m,i}} \right)^T P^m \phi P^m_{\text{pad}},
\]

(27)

- Note that after (25), we change a vector \( \frac{\partial \xi_i}{\partial \text{vec}(S^{m,i})^T} \) to a matrix \( \frac{\partial \xi_i}{\partial S^{m,i}} \) because in (26) and (27), matrix form is needed.
- In (25), information of the next layer is used.
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

Then only information in the current layer is used
Finally an implementation for one convolutional layer:

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

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

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

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

A sample segment of code
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.dlossdWm = dXidS*phiZ';
net.dlossdbm = dXidS*ones(model.wd_conv(m)*model.ht_conv(m)*S_k, 1);
if \( m > 1 \)

\[
V = \text{model.weight}m' * \text{dXidS};
\]

\[
d\text{XidS} = \text{reshape}(\text{vTP}(\text{param, model, net, m, V, 'phi_gradient'}),
\]

\[
\text{model.ch_input}(m), []);
\]

\%
\text{vTP_pad}
\]

\[
a = \text{model.ht_pad}(m); b = \text{model.wd} \_ \text{pad}(m);
\]

\[
d\text{XidS} = d\text{XidS}( :, \text{net.idx} \_ \text{padm} +
\]

\[
a*b*[0:S_k-1]);
\]
% activation function

dXidS = dXidS.*(net.Zm > 0);

end

end
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.
Forward:

\[
W^m \text{mat}(P^m \phi \ P^m \text{pad vec}(Z^{m,i})) = W^m \phi(\text{pad}(Z^{m,i}))
\]

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

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

\[
Z^{m+1,i} = \text{mat}(P^m_{\text{pool}} \text{vec}(\sigma(S^{m,i})))
\]

\[
\mathcal{O}(l \times h^m h^m d^{m+1} a^m_{\text{conv}} b^m_{\text{conv}})
\]
Complexity III

Backward:

\[ \Delta \leftarrow \text{mat}(\text{vec}(\Delta)^T \mathcal{P}_m) \]

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

\[ \frac{\partial \xi_i}{\partial W^m} = \Delta \phi(\text{pad}(Z_{m,i}^m))^T \]

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

\[ \Delta \leftarrow \text{vec} \left( (W^m)^T \Delta \right)^T \mathcal{P}_m^m \mathcal{P}_\phi^m \text{pad} \]

\[ (W^m)^T \Delta : \mathcal{O}(l \times h^m h^m d^m d^m+1 a_{\text{conv}}^m b_{\text{conv}}^m) \]

\[ \text{vec}(\cdot) \mathcal{P}_\phi^m : \mathcal{O}(l \times h^m h^m d^{m+1} a_{\text{conv}}^m b_{\text{conv}}^m) \]
We see that matrix-matrix products are the bottleneck.

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?