Gradient Calculation

Chih-Jen Lin
National Taiwan University
Last updated: January 9, 2020
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

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

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

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

\[ b^m \mathbb{1}_{a^m_{\text{conv}} b^m_{\text{conv}}} \]

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

Vector Form II

We have

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

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

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

(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)

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

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

then

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

a much bigger matrix
For the fully-connected layers, 

\[ s^{m,i} \]

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

\[ = (I_1 \otimes W^m) z^{m,i} + (1_1 \otimes I_{n_{m+1}}) b^m \]  \hspace{1cm} (10)

\[ = ((z^{m,i})^T \otimes I_{n_{m+1}}) \text{vec}(W^m) + (1_1 \otimes I_{n_{m+1}}) b^m, \]  \hspace{1cm} (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
  \[ \phi(\text{pad}(z_{m,i})) = I_{nm}z_{m,i}, \ L^c < m \leq L + 1, \]
  then (5) and (11) are in the same form.
- Thus we can derive the gradient of convolutional and fully-connected layers together
For convolutional layers, from (5),

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

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

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

(12)

where (12) is from (9).

We applied chain rule here.
Note that we define

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

where \( x \) and \( y \) are column vectors.
Thus if \( y = Ax \) then

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

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

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

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), $\frac{\partial \xi}{\partial S_{m,i}}$ is also needed.

We will show that it can be obtained by a backward process.
Gradient Calculation

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 $\frac{\partial \xi_i}{\partial S^m,i}$ 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 $\frac{\partial \xi_i}{\partial S^{m,i}}$ II

- Assume the RELU activation function is used

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

- Note that

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

is a squared diagonal matrix
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 $\frac{\partial \xi_i}{\partial S^{m,i}}$ IV

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 $\frac{\partial \xi_i}{\partial S^{m,i}}$ V

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

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

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

(16)

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

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

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

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

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

\[
= \frac{\partial \xi_i}{\partial \text{vec}(S^{m,i})^T} \left( I_{a_{\text{conv}}} b_{\text{conv}}^m \otimes W^m \right) P^m P^m_{\text{pad}} \tag{17}
\]

\[
= \text{vec} \left( (W^m)^T \frac{\partial \xi_i}{\partial S^{m,i}} \right)^T P^m P^m_{\text{pad}} \tag{18}
\]
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).

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

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

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

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

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

\[ Z_{m,i} \] is available from the forward process.

Therefore
\[ Z_{m,i}, \forall m \]

are stored.
But we also need $S^{m,i}$ for

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

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

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

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

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

- $Z^{m+1,i}$ is a “smaller matrix” than $S^{m,i}$
- That is, (16) is a “reverse mapping” of the pooling operation
- In (16),
  \[
  \frac{\partial \xi_i}{\partial \text{vec}(Z^{m+1,i})^T} \times P_{\text{pool}}^{m,i} \tag{24}
  \]
  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

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

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

If using (16),

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

\[ = \begin{bmatrix} 0 & 0 & 0 & 0 & 0 & \mathbf{v}_1 & 0 & \mathbf{v}_2 & 0 & 0 & 0 & 0 & 0 & \mathbf{v}_3 & \mathbf{v}_4 & 0 \end{bmatrix} \odot \]

\[ \begin{bmatrix} 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 \end{bmatrix} \]
If using (23),

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

\[
= (v^T \odot [1 1 1 1])P_{\text{pool}}
\]

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

So they are the same

In the derivation we used the properties of

- RELU activation function and
- max pooling
Gradient Calculation

Notes on Practical Implementations IX

to get

\[ Z^{m+1,i} \] component \( > 0 \) or not
\[ \iff \] 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

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

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

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

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

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

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

- 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 = model.weightm’ * 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_padm + a*b*[0:S_k-1]);
Gradient Calculation

Summary of Operations VII

% 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:

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

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

\[ Z^{m+1,i} = \text{mat}(P^m_{\text{pool}} \text{vec}(\sigma(S^{m,i}))) \]
\[ O(l \times h^m h^m d^{m+1} a^m_{\text{conv}} b^m_{\text{conv}}) \]
Complexity III

- Backward:

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

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

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

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

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

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

\[
\text{vec}(\cdot) P_{\phi}^m : \mathcal{O}(l \times h^m h^m d^{m+1} a_{\text{conv}}^m b_{\text{conv}}^m)
\]
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?
Recall we have

\[ Z^{m+1,i} = \text{mat}(P^m_i \text{vec}(\sigma(S^m_i)))_{d^{m+1} \times a^{m+1} b^{m+1}}, \]

We note that

\[ P^m_i \]

is not a constant 0/1 matrix

It depends on \( \sigma(S^m_i) \) to decide the positions of 0 and 1.
Thus like the RELU activation function, max pooling is another place to cause that $f(\theta)$ 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

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

\[ \nabla f(A) = \begin{bmatrix} 1 \\ 0 \\ 0 \\ 0 \end{bmatrix} \]

at \( A = \begin{bmatrix} A_{11} & A_{12} \\ A_{21} & A_{22} \end{bmatrix} \)

This explains why we can use \( P_{pool}^{m,j} \) in function and gradient evaluations.