After checking formulations for gradient calculation we would like to get into implementation details.

Take the following operation as an example

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

It’s a matrix-matrix product.

We all know that a three-level for loop does the job.

Does that mean we can easily write an efficient implementation?

The answer is no.
We want to use optimized code written by experts.

To illustrate this point, we check a video about optimized BLAS (Basic Linear Algebra Subprograms) borrowed from the course “numerical methods”.

In particular, we discuss the implementation of matrix-matrix multiplications.
The discussion on fast matrix-matrix products roughly explains why GPU is used for deep learning.

GPU is efficient for such operations.

Note that we did not touch multi-core implementations, though parallelization is possible.

Anyway, the conclusion is that for some operations, using code written by experts is more efficient than our own implementation.

How about other operations besides matrix-matrix products?
If they can also be done by calling others’ efficient implementation, then a simple and efficient CNN implementation can be done.

The MATLAB implementation in simpleNN is a good experimental environment for us to study this.

We will explain details and use it in our subsequent projects.
In the earlier discussion, we check each individual data.

However, for practical implementations, all (or some) instances must be considered together for memory and computational efficiency.

Recall we do **mini-batch** stochastic gradient

In our discussion we use $l$ to denote the number of data instances in calculating the gradient (or the sub-gradient)
In our MATLAB implementation, we store \( Z^{m,i}, \forall i = 1, \ldots, l \) as the following matrix.

\[
\begin{bmatrix}
Z^{m,1} & Z^{m,2} & \cdots & Z^{m,l}
\end{bmatrix} \in \mathbb{R}^{d_m \times a_m b_m l}.
\] (1)

Similarly, we store

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

as

\[
\begin{bmatrix}
\frac{\partial \xi_1}{\partial S^{m,1}} & \cdots & \frac{\partial \xi_l}{\partial S^{m,l}}
\end{bmatrix} \in \mathbb{R}^{d^{m+1} \times a_{conv} b_{conv} l}.
\] (2)
Storage III

- We will explain our decision.
- Note that (1)-(2) are only the main setting to store these matrices because for some operations they may need to be re-shaped.
Operations of a Convolutional Layer I

- Recall for gradient we have operations

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

(3)

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

(4)

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

(5)
Based on the way discussed to store variables, we will discuss two operations in detail:

- Generation of $\phi(\text{pad}(Z^{m,i}))$
- Vector $\times P^m_\phi$