Some General Comments

- You should analyze your results
- Some just present tables or figures, but we want more insight
About Project Submission

- The default setting in the class system is that you can submit the report only once.
- Otherwise it’s called a draft.
- This is very inconvenient for you.
- I have learned how to allow unlimited submissions.
- We will do that from now on.
You should identify the major operations
Then check percentage of these operations
From the results you draw some conclusions
Let’s consider a good example (from P2 of Davis Cho’s report):
The Analysis II

Forward
Eq. 1: $\phi_{pad}(Z_{m,i}^m)$
Eq. 2: $W^m \phi(\cdot)$
Eq. 3: $Z_{m+1,i}^{m+1} = \text{mat}(P_{pool}^m \text{vec}(\sigma(S_{m,i}^m)))$

As can be seen, the vTP function from the backward process takes a majority of the run time of the whole program, as shown in figure 1c. The vTP function is used in both the first equation for pooling as well as the third equation mentioned in the slides. The next most time consuming portion would be the forward feeding equation 2, denoted as F Eq.2 in Figure 1c.

Given that the vTP function is a vector matrix multiplication function, indeed the prediction that the next most costly function is the bottleneck is true in Matlab optimizes matrix matrix operations well.
One possible conclusion from this is that by the complexity analysis

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

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

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

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

\[ \mathcal{O}(l \times h^m h^m d^{m+1} a_{\text{conv}}^m b_{\text{conv}}^m) \]
From this complexity analysis matrix-matrix products are the bottleneck.

Roughly,

\[ d^m \text{ times} \]

more than others at layer \( m \).

We use \( d^m = 32 \) or 64 in our experiments.

However, we see that matrix-matrix products take less than half (feedforward part).

Thus optimized BLAS is very effective.
- It is easier to optimize the computationally heavy part.
- But this also means that probably there is still room to improve other operations.
Why Checking Main Steps is Important

- We that some have feedforward ?? %
  max pooling ?? %
- But the problem is that max pooling is part of feedforward
- Thus you cannot say much from such a list of results
Matlab/Octave versus PyTorch I

- For a timing comparison, you should use the same environment
- We have in a report with
  - Matlab: Windows on computer A
  - PyTorch: Ubuntu on computer B
- This isn’t very appropriate
- About ours versus PyTorch, we see very different results
- Let’s consider time per epoch for MNIST
<table>
<thead>
<tr>
<th>Matlab</th>
<th>PyTorch</th>
<th>OS</th>
</tr>
</thead>
<tbody>
<tr>
<td>140</td>
<td>70</td>
<td>Ubuntu</td>
</tr>
<tr>
<td>2.5</td>
<td>2.8</td>
<td>Mac OS</td>
</tr>
<tr>
<td>238</td>
<td>272</td>
<td>windows</td>
</tr>
<tr>
<td>212</td>
<td>169</td>
<td>windows</td>
</tr>
<tr>
<td>206</td>
<td>124</td>
<td>Mac OS</td>
</tr>
<tr>
<td>125</td>
<td>70</td>
<td>Ubuntu</td>
</tr>
<tr>
<td>305</td>
<td>96</td>
<td>Mac OS</td>
</tr>
<tr>
<td>115</td>
<td>92</td>
<td>Mac OS</td>
</tr>
</tbody>
</table>

To make things simpler, I don’t list Octave time here.
It seems that on Linux machines timing results are more consistent.

In fact, when I designed this course I naively assume that it’s the environment we all will use.

In the area of numerical computation, you don’t use your own laptop for experiments.

Anyway, what happened on Windows or Mac OS needs more investigation.

Let’s focus on the Linux results for now.

For the MNIST set, we see that PyTorch is more than two times faster than our Matlab code.
Of course we ask why
A research question is how we will investigate this issue
One possibility is to check the running time of matrix-matrix products
Both use Intel MKL
Thus this part should take a similar amount of time
However, in our case the three matrix-matrix products (one in forward and two in backward) already take slightly more than half
Thus very likely their matrix-matrix products are faster than ours

How could this be possible?
While trying profiling on PyTorch we found it uses Intel MKL-DNN:
https://github.com/intel/mkl-dnn

We see their example file called cnn_training_fp32.c

From fp32 we see they use float

Therefore, an issue is PyTorch uses less floating-point precision than ours

The issue of float versus double
is always an issue in numerical computation

- Usually using float can significantly shorten the running time

- A preliminary check of changing PyTorch to use double almost double its running time

- More investigation is needed but certain this is an important factor to cause the difference

- How about changing Matlab to use float?