Project: Test Accuracy of Using PyTorch and Our Matlab/Octave Implementation
Making test accuracy of using PyTorch and our code be the same or similar
In our timing comparison you may have noticed that even by using the simplest stochastic gradient steps, the two packages (PyTorch and ours) give different test accuracy.

Many things can cause the difference.

For example,

- data pre-processing
- initial solution
- learning rate
- mini-batch selection
- and others
Now we see why checking the reproducibility of a paper is so difficult.

For this project, let’s modify the PyTorch settings to be the same as those for Matlab/Octave (see details below).

The goal is to have that the two packages give same (or very similar) test accuracy (or even the objective value).

Sometimes this is easy but sometimes it’s tremendously difficult.

Present your findings and thoughts.
Data Sets I

- The running time of training the full MNIST or CIFAR10 sets is too long
- Thus we have prepared a 5,000-instance subset for this project
- Please download them from the course page
- Test sets are not changed. You can use files from the previous project
- But the above is for Matlab! You need to generate corresponding files for PyTorch
- Label issue: our code assumes 1, ..., 10 but PyTorch may assume something else
Let’s check the default settings of our own implementation.

This means we need to use experiment.m for the previous project.
Network Architecture

- See the config file of the previous project
- It should be the same as the one for the first project, where PyTorch was used
- Therefore, this part should be all set though you want to double check
Min-max normalization.
For each pixel of every image $Z^{1,i}$, we have

$$Z_{a,b,d}^{1,i} \leftarrow \frac{Z_{a,b,d}^{1,i} - \min}{\max - \min},$$

where max/min is the maximum/minimum value of all pixels in $Z^{1,i}$.

Zero-centering.
This is commonly applied before training CNN (Krizhevsky et al., 2012; Zeiler and Fergus, 2014).
For every pixel in image $Z_{1,i}^1$, we have

$$Z_{a,b,d}^{1,i} \leftarrow Z_{a,b,d}^{1,i} - \text{mean}(Z_{a,b,d}^{1,:})$$

where $\text{mean}(Z_{a,b,d}^{1,:})$ is the per-pixel mean value across all the training images.

- How to do this in PyTorch?
- One way is that we check if PyTorch supports the same pre-processing procedures
- Another is that we use Python (e.g., NumPy) to do pre-processing before calling PyTorch
We follow He et al. (2015) to set the weight values by multiplying random values from the \( \mathcal{N}(0, 1) \) distribution and

\[
\sqrt{\frac{2}{n_{in}^m}}, \text{ where } n_{in}^m = \begin{cases} 
  d^m \times h^m \times h^m & \text{if } m \leq L^c, \\
  n_m & \text{otherwise}.
\end{cases}
\]

For the bias, let’s have the initial value be

\[ b = 0 \]
- Mini-batch size: we use 128
- We consider the algorithm with momentum
- We use $\alpha = 0.9$

$$v \leftarrow \alpha v - \eta \left( \frac{\theta}{C} + \frac{1}{|S|} \nabla_\theta \sum_{i:i \in S} \xi(\theta; y^i, Z^{1,i}) \right)$$

$$\theta \leftarrow \theta + v$$
Parameters for Stochastic Gradient II

- We run 500 epochs and use the model obtained in the end for prediction
- Initial learning rates:
  - MNIST: 0.001
  - CIFAR10: 0.003
It seems PyTorch uses something called “weight_decay” for the regularization term.

To avoid any issue caused by this, let’s not use regularization for both packages.

That is, the objective function is simply

$$\frac{1}{l} \sum_{i=1}^{l} \xi(z^{L+1,i}(\theta); y^i, Z^{1,i})$$

It’s easy to modify our code: in the gradient calculation the first term of $\theta$ is removed.
This is probably the only Matlab change you must do

For PyTorch we can simply set “weight_decay” to be zero
Random Seed I

- Even with exactly the same algorithm, results may still be different due to the sequence of random numbers used.
- Note that random numbers are used in generating the initial weights, and selecting the mini batch.
- To handle this situation, let’s do 10 runs under different seeds and check the mean accuracy.
- We hope to achieve that the mean accuracy values of the two implementations (ours and PyTorch) are very similar.
Presentations I

- I think we have time to run only two more projects
- We might need to do 10 presentations per project?
