Project: Test Accuracy of Using PyTorch and Our Matlab/Octave Implementation
Goal

- 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.

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

Of course both run the same number of epochs.

Present your findings and thoughts.
Let’s check the default settings of our own implementation
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
Pre-processing I

- 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).
Pre-processing II

For every pixel in image $Z^{1,i}$, we have

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

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

$$b = 0$$
Parameters for Stochastic Gradient I

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

For

$$\begin{align*}
v & \leftarrow \alpha v - \eta \left( \frac{\theta}{C} + \frac{1}{|S|} \nabla_{\theta} \sum_{i:i \in S} \xi(\theta; y^i, Z^{1,i}) \right) \\
\theta & \leftarrow \theta + v
\end{align*}$$
We run 1,000 epochs and use the model obtained in the end for prediction.

Initial learning rates:

- MNIST: 0.001
- CIFAR10: 0.003
Other Parameters

- Regularization parameter: $0.01/l$, where $l$ is the number of training data.
- Thus the formulation is

$$\frac{1}{2C}\theta^T\theta + \frac{1}{l}\sum_{i=1}^{l}\xi(z^{L+1,i}(\theta); y^i, Z^{1,i})$$

did PyTorch support regularization?
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.
