Test Accuracy of Using PyTorch and Matlab Implementation

CS 269: Optimization Methods for Deep Learning, Project 4

Zhiping Xiao (Patricia), Haoran Wang *

1 Introduction

The goal of this project is to make test accuracy of using PyTorch and of our Matlab code [1] be the same or similar, by strictly restricting the settings.

As we've discussed in project 3, using different settings can cause difference in performance, and thus comparison between Matlab and Pytorch version is not guaranteed to be fair. Modification is needed so as to make the results comparable.

This experiment should shed light on why checking the reproducibility of a paper is always so difficult.

2 Experiment Settings

The running time of training the full MNIST or CIFAR10 sets is too long, especially when we evaluate 10 runs under different seeds and check the mean accuracy, 500 epochs each. Therefore, we use the 5000-size subsets provided, while the test set remains unchanged. To guarantee the two settings are comparable, we chose to load data in PyTorch from *.mat* file. In order to achieve this we used the library function *loadmat* from *scipy*, and defined our own dataset direved from *Dataset* class of *pytorch*. One thing to notify at this stage is that data type matters a lot. We need to specify the data type as *uint8*, otherwise it'll be troublesome visualizing the inputs.

Min-max normalization is done by applying customized normalizer. Besides, for cases divided by zero, we define the result 0. Note that min-max normalization is applied to each image separately, while the zero-centering is applied to all channels together. We could do zero-centering right after normalization.

As for the initialization, what we are required to use in PyTorch, proposed by Kaiming He back in 2015, is already adopted into the PyTorch library. [2] For detailed reference see the *discussion online*, as well as the *documentation*. This initialization is used for the convolutional layers' *and* the linear layer's weights initialization, while the corresponding bias terms are initialized as 0.

The regularization term, controlled by the parameter *weight_decay* in PyTorch, is by-default zero, which means no regularization if we do not specify it.

However, it is important to notice that the

*Contact us at *{patriciaxiao, hwan252}@g.ucla.edu* for more details on this project.

[1] `https://github.com/cjlin1/simpleNN`

[2] It is named *kaiming_uniform_* under *torch.nn.init*.

term *param.decay* in Matlab is totally different with our *weight_decay* in PyTorch. It is the learning rate decay (by default not considered), instead of the weight decay. Therefore, we need to modify the Matlab code.

$$\mathcal{L} = \frac{1}{2C}\theta^T\theta + \frac{1}{I}\sum_{i=1}^{I}\xi(Z^{L+1,i}(\theta); y_i, Z^{1,i})$$

$$\mathcal{L}_{after} = \frac{1}{I}\sum_{i=1}^{I}\xi(Z^{L+1,i}(\theta); y_i, Z^{1,i})$$

where ξ in this case is simply the MSE loss. We find in *lossgrad_subset.m* that, in the current implementation the *loss* is calculated as $\|Z - Y\|_F^2$, and that is the loss without regularization. The regularization term is introduced in *sgd.m*, line 18.

Running time is not our concern this time, so we simply canceled the threads limitation for faster speed. Number of threads should not affect the accuracy.

Other parameters are exactly the same as specified in the requirements. We consider SGD + MSE only, learning rate on MNIST dataset is 0.001 and learning rate on CIFAR10 is 0.003. Momentum $\alpha = 0.9$ is fixed. Mini-batch size is fixed to be 128, as is specified in previous projects.

One thing that is pretty tricky here is that, the CIFAR10 dataset used by PyTorch and Matlab are *dramatically different from each other*. Although they have the same labels and the labels are in the same order by default, the content of the figures are different. The Matlab CIFAR10 images are somehow pre-processed into 3×3 blurred grids. Therefore, if we use the Matlab training set, we should use the Matlab test set to match it up so as to avoid bugs, otherwise the model will always perform terribly on test set.

PACKAGE	ACCURACY
Matlab	always 0.9768 (avg: 0.9768)
PyTorch	0.9788 0.9801 0.9795 0.9803
	0.9827 0.9784 0.9785 0.9790
	0.9792 0.9810 (avg: 0.9797)

Table 1: Final (500 epochs) Prediction Accuracy on **MNIST** Test Set

PACKAGE	ACCURACY
Matlab	always 0.4490 (avg: 0.4490)
PyTorch *ep #100*	0.4302 0.4345 0.4275 0.4302
	0.4371 0.4226 0.4266 0.4248
	0.4319 0.4173 (avg: 0.4283)
PyTorch *final*	0.3824 0.3833 0.3762 0.3732
	0.3870 0.3855 0.3885 0.3921
	0.3872 0.3717 (avg: 0.3827)

Table 2: Final (500 epochs) Test Accuracy, and Epoch #100 on **CIFAR10**

3 Results and Discussion

The test accuracy of 10 run with 500 epochs each on MNIST are as shown in Table 1, and the results on CIFAR10 are as shown in Table 2. The Matlab result is pretty stable, besides, the loss of each epoch using our Matlab code remains the same among the 10 runs. Only PyTorch version involves randomness.

In general, our result show that after restricting the settings (as well as the dataset) to be the same, Matlab and PyTorch package have similar test accuracy.

However, it is obvious that the performance on CIFAR10 is different. We have our PyTorch model settings carefully synchronized with the Matlab implementation, but the difference is still there.

We suppose that it ends up in this situation is mostly due to overfitting, yet partly due to

subtle settings such as that the loss functions of the two packages could not be exactly the same. Looking into the Matlab code we found that the equivalent version of its loss function (*Frobenius Norm squared*) should be MSELoss with *sum* reduction (normalized by dividing the batch size) instead of with *mean* reduction. However, in practice, PyTorch MSELoss with sum reduction is not well-handled and it suffers *exploding gradients* a lot, thus we gave in and also logged the PyTorch test accuracy at around epoch # 100, which is sufficient in proving the ability and potential of PyTorch model. There are many standard ways out yet none of them is used since the settings are fixed. As for the accuracy difference, we naturally suspect that some differences are introduced by the two different versions of loss.

MNIST safely avoided the issue since it is simpler and thus subset size 5000 is likely to be representative enough. We find that the reason why our PyTorch MSELoss with sum reduction fails is that the batch size is too big. Matlab package avoided this issue by dividing the batch size.

Xin Jiang and *Kewei Cheng* from another group discussed over it with us, they found that reducing the learning rate will avoid the accuracy dropping. Xin suspects that it is underfitting. But we believed that the case could be: their learning rate was so small that the model was underfitting. Further experiment is needed for a concrete and convincing explanation.

Using the mini dataset, on the other hand, significantly saves our time. Recall that last time we have MNIST running time approximately 200 seconds per epoch, and CIFAR10 running time approximately 250 seconds; this time it is approximately 22 seconds on MNIST and 25 seconds / epoch on

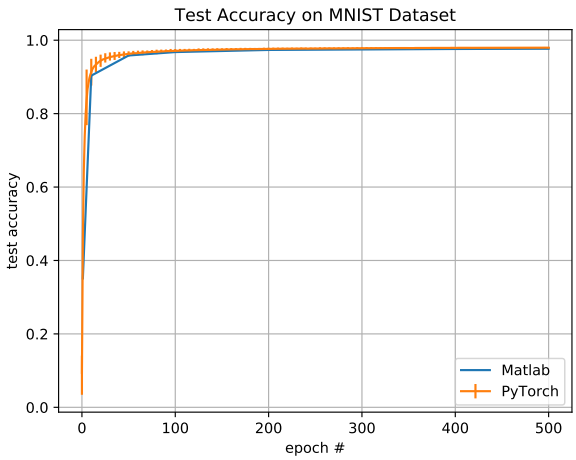

Figure 1: *Test Accuracy on MNIST, with PyTorch plotted as error bar.*

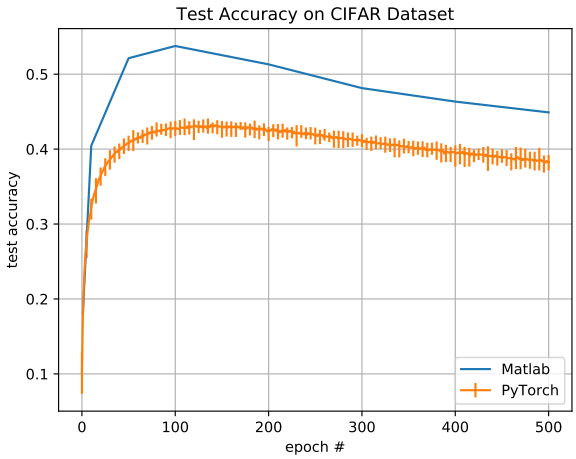

Figure 2: *Test Accuracy on CIFAR10, with PyTorch plotted as error bar.*

CIFAR10. Using PyTorch, the running time per epoch is around 9 and 10 seconds with single core respectively, comparing with last time's 100 s / epoch and 110 s / epoch, we can say that simply switching to the mini-sized dataset makes the model around 10 times faster, on Matlab or PyTorch.

The test accuracy plotted by epoch is as shown in Figure 1 and Figure 2 respectively. We used error plot to show the randomness of PyTorch results. It is obvious that the general trend is similar, while the performance is different due to some subtle differences in subtle details, dependencies or languages.