Efficiency of Matlab SG Implementation

CS 269: Optimization Methods for Deep Learning, Project 3

Zhiping Xiao (Patricia) ,* Student ID 604775684

1 Introduction

The goal of this project is to conduct the running-time analysis of the Matlab/Octave SG implementation and how efficient it is in comparison with PyTorch implementation.

2 Environment

Octave is an open-sourced free alternative of Matlab, but would be much more troublesome installing statistics toolkit, except on Linux. In this case, since UCLA doesn't provide us free Matlab license, I used the license from my previous institution [1]. PyTorch implementation could simply go by the code implemented in previous projects from this course. As for Matlab/Octave:

1. Install Octave and Octave Statistics toolkit, or simply install Matlab. In my case I installed Matlab R2019a.

2. Clone/Download the Matlab code [2] and data [3] provided by Prof. Lin. Put all given *.config* files under *config* folder, all *.mat* files under *data* folder.

3. Figure out how to pass in options and verify that the code works. e.g. Opening them in Matlab IDE and type *"example_mnist("-s 2 -epoch_max 5")"* in the command window will make it run SG solver instead of the Newton one in default settings, and run 5 epochs instead of the default 500.

4. We care more about the time efficiency than the accuracy. As is stated in the project spec, we need *profile* as a detailed timer. [4] Matlab provides perfect support in *profile*, allowing us to evaluate the calls of almost every paragraph. By the way, if we only need the starting and ending time of the epochs we could use the tic-toc built-in timer.

I run the experiments on mac OS 10.13.3, processor 2.2 GHz Intel Core i7.

3 Settings and Data

The hyper-parameter settings are as shown in Table 1. No momentum or decay. As for loss function, only MSE (minimum square error) loss is used. Maximum epochs of

*Contact me at *patriciaxiao@g.ucla.edu* for more details on this project.

[1] https://www.mathworks.com/academia/tah-portal/berkeley-731130.html

[2] https://github.com/cjlin1/simpleNN.

[3] https://www.csie.ntu.edu.tw/~cjlin/courses/optdl2019/slides/proj_efficiency_matlab/

[4] https://www.mathworks.com/help/matlab/ref/profile.html

running is always fixed to be 5. Single-core is also specified.

settings	learning rate	batch size
values	0.01	128

Table 1: Fixed Parameters

I have tried to compare the results from Matlab to that of PyTorch, with or without normalization. Normalization parameters are the same as in previous projects.

We use both MNIST and CIFAR dataset.

As for the datasets, I've verified that the datasets used by Matlab are **exactly the same** with the ones we use in PyTorch. I did so by printing all labels and comparing them (range 0 to 9). It is not hard to figure out that they have the same values and are in the same order, for both training sets and testing sets. For example, MNIST training sets both goes $5, 0, 4, 1, 9, \ldots$, testing $7, 2, 1, 0, 4, \ldots$; CIFAR training set $6, 9, 9, 4, 1, \ldots$ and testing $3, 8, 8, 0, 6, \ldots$.

4 Results and Discussion

Although testing accuracy is not our major concern in this project, it always remains an important measurement of the code performance, thus I still list it out in Table 2.

dataset	model	norm	accuracy
MNIST	Matlab	yes	0.9561
	PyTorch	yes	0.9222
	PyTorch	no	0.7648
CIFAR10	Matlab	yes	0.4766
	PyTorch	yes	0.325
	PyTorch	no	0.267

Table 2: Testing Accuracy after 5 Epochs

What we care the most is the running time of each model, listed in Table 3.

dataset	model	norm	time (s)
MNIST	Matlab	yes	215.99
	PyTorch	yes	108.9851
	PyTorch	no	104.4568
CIFAR10	Matlab	yes	293.5902
	PyTorch	yes	111.1747
	PyTorch	no	107.2799

Table 3: Average Running Time per Epoch

It is said that there remains an issue of the above setting: PyTorch runs 2 threads and uses 50% CPU on each, then considering the impact of the automatically-paralleled settings, we should say that the Matlab implementation has approximately the same efficiency with PyTorch implementation. Another reason why Matlab version exceed twice the running time of PyTorch might be the overhead of handling some tricky issues.

Examining the detailed information from Matlab profiling, we found that the summary of MNIST and CIFAR10 result are similar with similar portion of time-consumption on each part, except that experiment on MNIST takes more time (109.307s) on *maxpooling* than on *padding and phiZ* (80.59s), while CIFAR10 test takes more time (173.950s) on *padding and phiZ* than on *maxpooling* (141.979s). The top-3 most time-consuming parts are *lossgrad_subset*, *vTP* calculation and *feedforward* process. *padding and phiZ* and *maxpooling* rank either 4 or 5 on the two datasets. The above finding might infer that latter on when we want to improve the algorithm's performance, we should focus more on these aspects.

Another advantage of the Matlab implementation is that it really converges fast. This tendency could be observed by simply viewing the final testing accuracy after limited epochs.