Comparing Various Stochastic Gradient Methods by PyTorch

CS 269: Optimization Methods for Deep Learning, Project 2

Zhiping Xiao (Patricia) [*], Student ID 604775684

1 Introduction

The goal of this project is to check the use of several different stochastic gradient implementations in PyTorch for training CNN models. This is an extension of project 1.

I am hereby testing the performance of the PyTorch CNN implementation using different optimizer, whose settings are to be specified in section 2.

2 Experiment Settings

This time, since we are required to use CPU only, and that we have to run 50 epochs to evaluate the performance, I chose to run it on a desktop computer with OS Windows 10, 64bit, with 8^{th} Generation Intel Core i7-8700 6-Core Processor (12MB Cache, up to 4.6 GHz). Both with/without normalization are tried [1], and both MNIST and CIFAR10 are used. The architecture of the CNN is the same as project 1 [2]. For both datasets, I'm testing 4 cases of optimizer respectively: (1) standard SGD, (2) SGD with momentum 0.9, (3) Adagrad, (4) Adam.

Loss function is MSE loss by default. Other requirements are as listed in Table 1.

settings	learning rate	batch size
values	0.01	128

Table 1: Fixed Parameters

We are required to examine the relation between accuracy and the accumulated number of epochs.

3 Results and Discussion

Generally speaking, similar with project 1, running on MNIST is faster than running on CIFAR10, about $80 \sim 85$ and $90 \sim 95$ seconds respectively, with normalization. Without normalization, the running time is typically 3 to 5 seconds faster per epoch (typically, around 80 and 90 seconds, respectively). CIFAR10 running time reflects the difference between optimizers. Whether to choose MSE-loss or cross-entropy loss as the loss function has little influence on the time complexity. Momentum has little influence on time either. *Adagrad* and *standard SGD* have similar time complexity, *Adam* is slightly faster, about $1 \sim 2$ seconds faster each epoch than the rest.

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

[1]CIFAR10 uses transforms.Normalize((0.5, 0.5, 0.5), (0.5, 0.5, 0.5)), MNIST uses transforms.Normalize((0.1307,), (0.3081,)), for normalization.

[2]For the detailed hyper-parameters see `https://www.csie.ntu.edu.tw/~cjlin/courses/optdl2019/slides/proj_simple_sg_pytorch.pdf`

In most of the other cases the model converges as expected, except that once it isn't working with MSE-loss and Adam together, but works well with cross-entropy loss. This problem vanished when I re-run it. This might infer to the existence of local optimal points with MSE + Adam settings.

Since using MSE loss is required by this project's spec [3], I'm still comparing the MSE-loss version among all different optimizers, using the reasonable results where it is not trapped in local optimal.

As is shown in Figure 1, all optimizers work well except some specific cases (special case not plotted, email me for discussion), which we've briefly discussed. And no matter we add momentum, or we change the optimizer to Adagrad, the performance is generally better than using the standard SGD method. Normalization is beneficial for improving testing accuracy and accelerating convergence for standard SGD, but probably not as useful for other optimizers.

With a more complex dataset CIFAR10, as is shown in Figure 2, the difference among optimizers is more significant. In general, it seems that for the given settings specified before, Adagrad converges faster, but standard SGD with momentum, normalized, seems to have slightly-higher accuracy in the end. Standard SGD benefits a lot from doing normalization, but benefits even more from adding momentum. Adam converges fast but the accuracy fluctuates a lot, all other optimizers have smoother curves. Seems that doing normalization doesn't help a lot in improving Adagrad performance, at least in our case, it converges even faster without normalization, although it seems that in the end normalization might lead it to a higher accuracy.

Randomness is observed after several runs, especially with Adam and Adagrad. Those advanced techniques have good performance anyway, but sometimes one setting could beat another unexpectedly by chance. Randomness has larger influence than normalization to them.

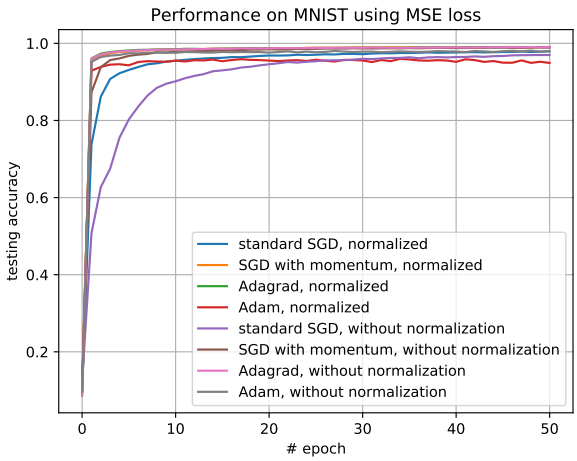

Figure 1: The performance of different optimizers on MNIST dataset.

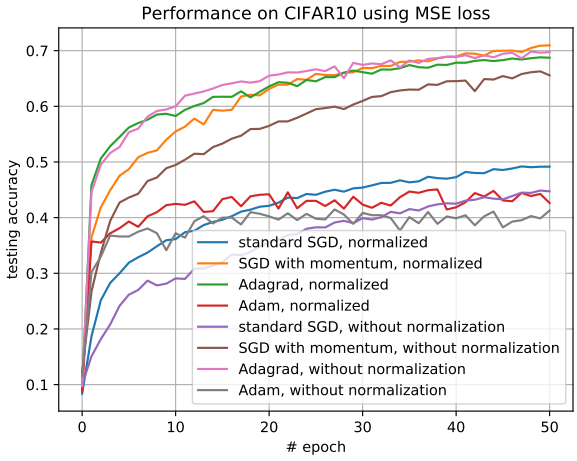

Figure 2: The performance of different optimizers on CIFAR10 dataset.

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