On the Insufficiency of Existing Momentum Schemes for Stochastic Optimization

Difan Zhang, Chen Tian & Yixiao Fan **School of Electronics and Computer Science**
University of Southampton
Southampton, SO17 1BJ, UK
{dz1n18,ct2n18,yf1n18}@soton.ac.uk

Abstract

An accelerated stochastic gradient descent method is proposed in an ICLR paper, which has higher performance than HB, NAG, and SGD. It has been proved by the experiments that it can converge quickly regardless of the batch sizes. This report mainly focuses on reproducing the experiments in the original paper, and proves that the performance of AccSGD is indeed better than other momentumbased stochastic gradient descent methods.

1 Introduction

Stochastic first order oracle (SFO) models, which access stochastic gradients computed on a small constant sized mini batches, provide a significant performance on large scale optimization problems. Among SFO models, gradient descent is the simplest well-known method that is heavily used. It is believed that the momentum based stochastic gradient methods including heavy ball (HB) and Nesterov's accelerated gradient descent (NAG) provide significant improvements over stochastic gradient descent (SGD) for the deterministic gradients. In the stochastic case, both HB and NAG only provide partially better performances than SGD while in the rest of the practices, their performances are as same as SGD. The core idea of the paper Kidambi et al. (2018) is to prove that even if the parameters are set to be optimum, HB and NAG can not outperform SGD in some simple instances, which is caused by the construction of these methods. The practical performances are actually gained by mini-batching. In addition, a Nesterov's AGD based alternative algorithm, which has significant improvements over HB, NAG, and SGD, named ASGD, is proposed and experimented in the paper Kidambi et al. (2018). In this report, the experiments mentioned in the paper are implemented and the analysis of the results will be provided at the end of the report.

2 Target Questions Contributions

2.1 Questions

Three questions are proposed in the paper and will be addressed in the experiments:

1. Is the suboptimality of HB restricted to specific distributions or does it hold for more general distributions as well? Is the same true of NAG?
2. What is the reason for the superiority of HB and NAG in practice? Is it because momentum methods have better performance that SGD for stochastic gradients or due to mini-batching? Does this superiority hold even for small mini batches?
3. How does the performance of ASGD compare to that of SGD, HB and NAG, especially while training deep networks?

2.2 Contributions

Based on the experiments, the contributions given by the paper contain four aspects:

1. HB is proved to be suboptimal in the SFO model. The performance is either the same or worse than that of SGD while ASGD improves upon both of them.
2. The suboptimality of HB in the SFO model is widespread and the same holds true for NAG.
3. The only reason for the superiority of momentum methods in practice is minibatching.
4. ASGD can provide significantly faster convergence to a reasonable accuracy than SGD, HB, NAG, while still providing asymptotically optimal accuracy.

3 Implementation

3.1 Introduction of ASGD

Accelerated stochastic gradient descent (ASGD) is designed to improve over SGD for minibatch sizes. Algorithm 1 illustrates how ASGD works. The loss function $f(w) = 1/n \sum_i f_i(w)$ w.r.t weight w. $\widehat{\bigtriangledown} f_t(w)$ is defined as a stochastic gradient of f, i.e. $\widehat{\bigtriangledown} f_t(w_t) = \bigtriangledown f_{i_t}(w)$ where i_t is sampled uniformly at random from [1,...,n]. $\mathbf{H} = E[xx^T]$ is the hessian matrix of f and $k = \frac{\lambda_l(H)}{\lambda_d(H)}$ is the condition number of $\mathbf{H}$. The algorithm and source code of ASGD is described in Jain et al. (2017) and available on Github rahulkidambi (2018). In the following experiments, we use the source code to implement ASGD optimiser. Both SGD and NAG optimisers are implemented by *Pytorch* library *torch.optim* while for HB's case, we create our own code.

Algorithm 1 Accelerated stochastic gradient descent - ASGD

```
Input: Initial $w_0$, short step $\delta$, long step parameter $k \geq 1$, statistical advantage parameter $\xi \leq \sqrt{k}$
1: Output: $w_t$
2: $\overline{w}_0 \leftarrow w_0\, t \leftarrow 0$    ▷ Set running average to $w_0$
3: $\alpha \leftarrow 1 - \frac{\xi}{k}$    ▷ Set momentum value
4: while $w_t$ not converged do
5:     $\overline{w}_{t+1} \leftarrow \alpha \cdot \overline{w}_t + (1 - \alpha) \cdot \left(w_t - \frac{k \cdot \delta}{0.7} \cdot \widehat{\bigtriangledown} f_t(w_t)\right)$    ▷ Update the running average as a weighted average of previous running average and a long step gradient
6:     $w_{t+1} \leftarrow \frac{0.7}{0.7+(1-\alpha)} \cdot \left(w_t - \delta \cdot \widehat{\bigtriangledown} f_t(w_t)\right) + \frac{1-\alpha}{0.7+(1-\alpha) \cdot w_{t+1}}$    ▷ Update the iterate as weighted average of current running average and short step
7:     $t \leftarrow t + 1$
8: end while
```

3.2 Experiment Details

3.2.1 Experiment A: Linear Regression

In this experiment, two 2-dimension distributions: *Discrete* and *Gaussian* are generated for the simple linear regression model. For each distribution, we vary condition number k from $2^4, 2^5, ..., 2^{12}$ and for each trial, we run a total of $t = 5k$ iterations. The convergent condition is that after $2.5k$ iterations, all the errors are less than the starting error. To find out the optimal learning rate for ASGD/ SGD and momentum for NAG/HB, we conduct a 10×10 grid in $[0, 1] \times [0, 1]$ for grid search and choose the parameters that yield the minumum error (at the t^{th} iteration) from a subset containing 100 trials. The convergence performance is defined as: rate = $(\log(f(w_0)) - \log(f(w_t)))/t$.

3.3 Experiment B: Deep Residual Networks for CIFAR-10

A deep residual network (dubbed preresnet-44) for images classification on cifa-10 data set is experimented in this section. Unlike the previous experiments, this section focuses on learning rate decay. The scheme is defined as decaying learning rate by a certain factor based on grid search every three epochs if the validation error does not decrease by at least a certain amount.

To test the effect of minibatch sizes, we reproduce the experiments following the instructions in the paper with constrained combinations of parameters, which is conducting two batch sizes: 128 and

16 (instead of 8 in the paper) with two epoches: 60 and 20 respectively. The loss function is chosen as cross entropy as before (zero-one error is not evaluated), and the learning rate decay scheme is combined with the grid search method.

The source code of the network structure is available on github by D-X-Y (2018) and the AccSGD optimiser used here is the same as that in the previous section. The training and testing processes(with a proportion of around 70% among the whole experiment) are finished by our own code.

3.4 Experiment C: Deep Autoencoders for MNIST

In the last experiment, we planned to apply the MNIST dataset a 9 Restricted Boltzmann machine(RBM) layers Deep Believe Network(DBN). The whole network contains 784 1000 500 250 30 250 500 1000 784 nodes for the respective layer. Besides, the learning rate and momentum value were set to 0.1 and 0.9, the epoch was set to 10. However, the algorithm consumes a lot of resources, which lead the Colab ran out the memory and suspend the task. Therefore, we finally decided to switch to a ”512 hidden units” VAE autoencoder which introduced in class. The VAE encoder contains 3 fully-connected layers with the linear activation function and the decoder has 2 fully-connected layers with sigmoid activation. Moreover, the dataset was switched from MNIST to MNIST Fashion, the batch size and epoch are 256 and 30.

As the evidence, the source code of VAE and incomplete DBN are available on our GitHub repoRep (2018). The reference code of DBN can be found at DBN (2018).

4 Result Analysis and Findings

4.1 Linear Regression

Figure 1(a) and 1(b) illustrate the correlation between converge rate and condition number k with different optimisers. In practical, the performance of grid search did not give us an optimal result exactly as described in the paper. The reason is that even though we have applied the fine grid search method to find the parameters, each parameter has a wide range of values in [0,1] for 100 trials, which is a challenge to the computational power. Finally, the chosen parameters and results(slope) given by a convergence rate vs k is shown in Table 1. The slopes are estimated based on least-squares by Moore-Penrose Pseudoinverse while it is estimated by linear regression in the paper. Two reference lines (slope=1 for kappa and slope=$\sqrt{1/k}$ for kappa$^{0.5}$) are drawn to compare the optimisers performances. Theoretically, ASGD is the closest to the line with slope=$\sqrt{1/k}$ while the others are closer to slope=1, which have been proved in Gaussian's case. The discrete case illustrates that NAG could outperform ASGD in some cases (has the lowest slope in discrete case) while HB is as worse as SGD, i.e. HB is not optimal in the SFO model. To sum up, ASGD could provide significant improvement over other algorithms (the slope in Gaussian's case is much less than others), while NAG can provide suboptimality and HB is not the optimal choice in most cases. The conclusions of experiment A are consistent with the contribution(1) and (2) in the paper Kidambi et al. (2018).

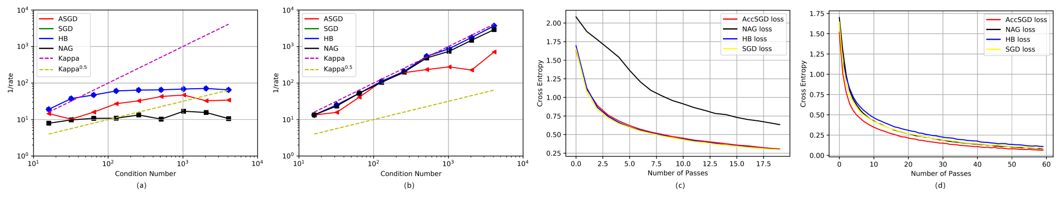

Figure 1: (a)(b): Plot of 1/rate vs condition number (k) for various methods for the linear regression problem. (a): Discrete distribution, (b): Gaussian. (c)(d): Test cross entropy loss for batch size 16 (c) and batch size 128(d) for ASGD compared to NAG, HB, and SGD. In the above plots, ASGD-Fully-Optimized refers to ASGD where learning rate and momentum were selected by grid search.

Table 1: Parameters and Results of Linear Regression with Different Optimisers

Experiment	Distribution	Parameters	SGD	AccSGD	NAG	HB
Linear Regression	Gaussian	Learning Rate	0.8	0.8	0.8	0.8
		Momentum	-	-	0.2	0.2
		Slope	0.8836	**0.1466**	0.706	0.898
	Discrete	Learning Rate	0.02	0.02	0.02	0.02
		Momentum	-	-	0.8	0.8
		Slope	6.10E-03	3.78E-03	**4.98E-04**	6.11E-03

4.2 ResNet44 FOR CIFAR-10

The experiment parameters and results are shown in Table2 and Figure 1(c) and 1(d). It can be concluded that for batch size=128, the performance of AccSGD including the convergence rate and loss is superior to SGD, HB, and NAG. In the case of batch size=16, both HB and SGD have slightly better performance over AccSGD. The results prove the second and third contributions of the paper Kidambi et al. (2018), which are: 1. Both HB and NAG are suboptimal in the SFO model. 2. The superiority of NAG and HB is caused by minibatching (size=16 in our case) only as the small batch reduces the variance in stochastic gradients, which increases the efficiency of convergence.

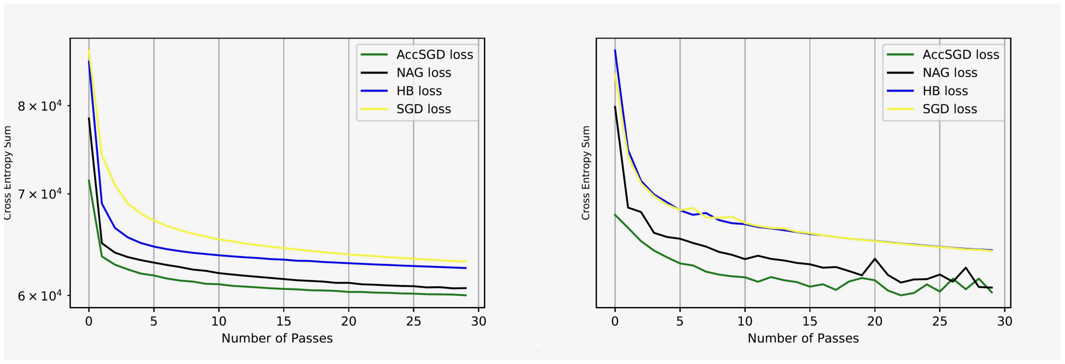

Figure 2: Training loss (left) and test loss (right) of deep autoencoder for mnist with batch size=256.

Table 2: Parameters/ Results of ResNet44 and Autoencoder with Different Optimisers

Experiment	Neural Network	Batch/ Epoch	Unit Time	Total Time	Loss			
					SGD	HB	NAG	ASGD
Deep Residual Neural Network	ResNet44	128/60	1h	4hr	0.0878	0.1095	0.0825	**0.0629**
		16/20	1.5hr	6hr	**0.2970**	0.3003	0.6343	0.3073
Deep Autoencoder	VAE	256/30	15sec	0	269	266	261	**258**
	DBN	8/10	10hr	-	-	-	-	-

4.3 DEEP AUTO ENCODERS FOR MNIST

The VAE network was used to present different optimizer's performance. To highlight the differences in loss values, the sum of cross-entropy error from each epoch is shown on the Y-axis. As a result, the AccSGD has the best convergence performance no matter in training or testing scene. On the contrary, HB and SGD are both presented a misadventure trend. NAG showed consistent results similar to the previous tests.

In the Table.2, due to the differences of dataset size and the structural complexity, autoencoder shows a faster training speed. However, DNNs have incomparable precision advantage.

5 Conclusion

To conclude, this report focuses on the comparison of different optimisers on SFO models and reflects to the questions proposed at the beginning of the report. Firstly, under certain conditions, there exists better performance of NAG or SGD but not much difference between AccSGD and the optimal models. The average performance of AccSGD is better than that of HB and NAG, no matter in linear regression, autoencoder or ResNet44 training processes. Secondly, when the value of Batch_Size is as small as 1, the performance of HB and NAG may be better than that of AccSGD. Finally, the average performance of AccSGD in deep network structure is significantly better than that of SGD, HB and NAG.

References

restricted-boltzmann-machine-deep-belief-network-deep-boltzmann-machine-in-pytorch, 2018. URL `https://github.com/wmingwei/restricted-boltzmann-machine\-deep-belief-network-deep-boltzmann-machine-in-pytorch`.

Insufficiency-momentum-schemes-for-stochastic-optimization, 2018. URL `https://github.com/COMP6248-Reproducability-Challenge/Insufficiency-momentum-schemes-for-Stochastic-Optimization`.

D-X-Y. Resnext resnet pytorch implementation, 2018. URL `https://github.com/D-X-Y/ResNeXt-DenseNet`.

Prateek Jain, Sham Kakade, Rahul Kidambi, Praneeth Netrapalli, and Aaron Sidford. Accelerating stochastic gradient descent. *CoRR*, abs/1704.08227, 2017.

Rahul Kidambi, Praneeth Netrapalli, Prateek Jain, and Sham Kakade. On the insufficiency of existing momentum schemes for stochastic optimization. In *International Conference on Learning Representations*, 2018.

rahulkidambi. Accsgd, 2018. URL `https://github.com/rahulkidambi/AccSGD`.