Comparison of Modern Variants of Stochastic Gradient Descent (Application Mode)

Shivam Kumar (170668)   Prateek Varshney (170494)

***Abstract*—Momentum based stochastic gradient methods are widely used for training supervised learning models, and are considered to be an improvement over gradient descent only algorithms. Though they have guaranteed improvement over gradient descent when gradients are exact, there is no theoretical guarantee for the same in erstwhile cases. In this paper, we mainly focus on reproducing and extending the results of "On the Insufficiency of Existing Momentum Schemes for Stochastic Optimization" published in ICLR conference 2018. We first establish experimentally that there exist simple stochastic problem instances where momentum based methods do not outperform Stochastic Gradient Descent (SGD). We then establish that the Accelerated Stochastic Gradient Descent (ASGD) can converge more quickly than Heavy Ball (HB), Nesterov's Accelerated Gradient Descent (NAG) and SGD, irrespective of Batch Sizes.**

***Index Terms*—Stochastic Gradient Descent (SGD), Heavy Ball (HB), Nesterov's Accelerated Gradient Descent (NAG), Accelerated Stochastic Gradient Descent (ASGD), Fast Gradient Methods, Exact Gradients, Minibatch, Deep Neural Networks**

I. Introduction

Even though First Order Optimization, gradient descent [1] being the simplest such method, are the defacto methods for large scale optimization problems; it has been well established that gradient descent is suboptimal [2] for the class of smooth convex functions along with some simple non-smooth problems. However, there are momentum-based methods such as The heavy ball method [3] and Nesterov's accelerated gradient descent [4] achieve optimal convergence guarantee. However, momentum-based methods employ exact gradients (computed on the full training dataset) and have not been proven to provide improvement over Stochastic Gradient Descent (SGD) [5] in Stochastic first-order oracle (SFO) [6] model, where we access stochastic gradients computed on a small sized mini batches and in the extreme case a batch size of 1; yet there continues to be widespread adoption of momentum methods for training deep neural nets, due to their observable practical gains.

In this paper, we give empirical evidence that in the SFO model, both HB and NAG only partially improve upon the performance of vanilla SGD, while their performance is similar to SGD in rest of the practices. The key idea of Kidambi et al. [7] is to prove and demonstrate empirically that there exist problem instances where it is informationally-theoretically possible to improve upon SGD's performance. However, HB and NAG fail to achieve it even when their hyperparameters are optimal. While ASGD is able to achieve better rates of convergence in the same class of problems. This also indicates that the performance gain of momentum-based methods over SGD in practise is due to mini-batching and not the algorithms themselves. In this paper, we aim to reproduce the results given in Section 5 of the original paper [8]. Moreover, we extend the scope by comparing the performance of each algorithms against Adam [9] (and not Semi-Stochastic Gradient Descent which we had planned as an optional). The authors had not provided any source besides the ASGD Algorithm . For our simulations, we use the PyTorch 'optim' [10] built-in package for SGD, HB, NAG and Adam and the code given on the GitHub repository [11] for the original paper for ASGD.

II. Algorithm

Algorithm 1: Accelerated SGD [8]

```
Input: Initial $\omega_0$ , short step $\delta$, long step
       hyperparameter $\kappa \geq 1$, statistical advantage
       hyperparameter $\xi \leq \sqrt{\kappa}$
1 $\bar{\omega}_0 \leftarrow \omega_0; t \leftarrow 0$          // Set running average to $\omega_0$
2 $\alpha \leftarrow 1 - \frac{0.7^2 \cdot \xi}{\kappa}$            // Set momentum value
3 while $\omega_t$ not converged do
4     $\bar{\omega}_{t+1} \leftarrow \alpha \cdot \bar{\omega}_t + (1-\alpha) \cdot \left(\omega_t - \frac{\kappa \cdot \delta}{0.7} \cdot \hat{\nabla} f_t(\omega_t)\right)$
      // Update the running average as a weighted
      average of previous running average and a
      long step gradient
5     $\omega_{t+1} \leftarrow$
      $\frac{0.7}{0.7+(1-\alpha)} \cdot \left(\omega_t - \delta \cdot \hat{\nabla} f_t(\omega_t)\right) + \frac{1-\alpha}{0.7+(1-\alpha)} \cdot \bar{\omega}_{t+1}$
      // Update the iterate as weighted average
      of current running average and short step
      gradient
6     $t \leftarrow t+1$
Output: $\omega_t$                // Return the last iterate
```

Algorithm 2: HB: Heavy ball with a SFO

```
Input: Initial $\omega_0$, stepsize $\delta$, momentum $\alpha$
1 $v_0 \leftarrow \omega_0; t \leftarrow 0$                 // Set $v_0$ to $\omega_0$
2 while $\omega_t$ not converged do
3     $v_{t+1} \leftarrow \alpha * v_t + \hat{\nabla} f_t(\omega_t)$
4     $\omega_{t+1} \leftarrow \omega_t - \delta \cdot v_{t+1}$
5     $t \leftarrow t+1$
Output: $\omega_t$                // Return the last iterate
```

Algorithm 3: NAG: Nesterov's AGD with a SFO

```
Input: Initial $\omega_0$ , stepsize $\delta$, momentum $\alpha$
1 $v_0 \leftarrow \omega_0; t \leftarrow 0$                    // Set $v_0$ to $\omega_0$
2 while $\omega_t$ not converged do
3     $v_{t+1} \leftarrow \alpha \cdot v_t + \hat{\nabla} f_t(\omega_t - \delta * \alpha * v_t)$
4     $\omega_{t+1} \leftarrow \omega_t - \delta * v_{t+1}$
5     $t \leftarrow t + 1$
Output: $\omega_t$                    // Return the last iterate
```

III. Experimental Setup

A. Linear Regression

In this experiment, we compared the performance of the four optimization methods (i.e. SGD, HB, NAG, and ASGD) for two different classes of 2 Dimensional distributions, i.e., Discrete distribution and Gaussian distribution, generated as:

- **Discrete:**

$$a = \begin{cases} e_1 & \text{with probability } 0.5 \\ \frac{2}{\kappa} e_2 & \text{with probability } 0.5 \end{cases}$$

- **Gaussian:** $a \epsilon \mathbb{R}^2$ is distributed as a Gaussian random vector with covariance matrix $\begin{bmatrix} 1 & 0 \\ 0 & 1/\kappa \end{bmatrix}$.

where κ is the condition number and e_i is the i^{th} standard basis vector. Therefore, our dataset comprises of (b, a) pairs where $b =< w^*, a >$ and w is a fixed randomly generated matrix s.t. $w \epsilon \mathbb{R}^2$. For each distribution, we vary the condition number κ as $\{2^4, 2^5, 2^6, \ldots, 2^{12}\}$ and for each κ we run a total of $t = 5\kappa$ iterations. This constitutes one simulation. We run each simulation 10 times and take the mean as our final result. An algorithm is said to converge when after half of the total iterations, i.e. 2.5κ, no error exceeds the initial/starting error. To choose the optimal hyperparameters for each of the algorithms, we perform a grid search over a 10×10 grid in [0,1]×[0,1] and choose the hyperparameters that yield the minimum error for a subset of 100 trials that converged. An algorithm's convergence performance is defined as:

$$rate = \frac{\log(f(w_0)) - \log(f(w_t))}{t}$$

B. Deep Autoencoders

The experimental setup for this section as per the original paper is to train a deep auto-encoder model on MNIST dataset using the different optimization methods. This problem is used as a standard benchmark for the evaluation of various optimization algorithms. The paper suggests an architecture of $784 - 1000 - 500 - 250 - 30 - 250 - 500 - 1000 - 784$ nodes [12]. Since the dimension of MNIST images are 28×28 so the first and last layers of the architecture representing the input and output each consists of 784 nodes. Also the layer with 30 nodes follows linear activation and Mean Square Error loss is used whereas sigmoid activations are used for all the remaining output or hidden nodes. The architecture is trained with mini batch sizes of 1 and 8 which run for 30 and 50 epochs respectively. However, we found that this architecture required a high computational resource for training in the case of Batch size = 1, which we were lacking. So we decided to removal the layers with 1000 nodes from the earlier architecture. The architecture of the now modified auto encoder model used to train on MNIST dataset was $784-500-250-30-250-500-784$. Also the size of dataset used is reduced to one-sixth of the original MNIST dataset by means of random sampling. A grid search is done over the momentum, learning rate and long step hyperparameter for each of HB, NAG, SGD, Adam and ASGD whereby best hyperparameters is chosen based on achieving the smallest training error [13].

C. Deep Residual Networks

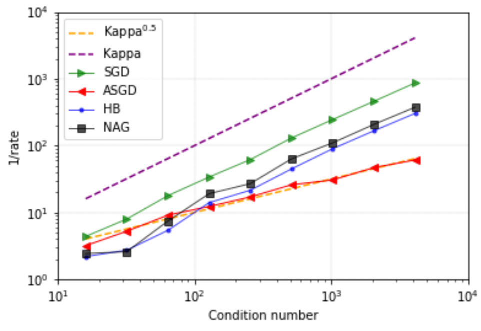
In this experiment, we compared the performance of our optimization algorithms by training deep residual networks [14] for classifying CIFAR-10 dataset images. The particular Resnet architecture which we experimented on had 44 layers (henceforth called PreResnet-44) [15], source code for which is available on preresnet(2017). Unlike our previous experiments, here we employed a dynamically changing hyperparameters, which we tuned at training time with the number of passes and based on changes in specific metrics (such as validation loss or accuracy) which are evaluated after every few (fixed number of) epochs. To verify the effect of mini batching in the performance of NAG, HB and Adam as compared to SGD and ASGD, we perform the training of the two different instances of the model differing with respect to two different batch sizes: 120 and 8 with number of epochs as 120 and 20 respectively. We employed the loss function as Cross Entropy and experimented with two different learning rates: scheduled and decayed.

IV. Simulations Analysis and Key Insights

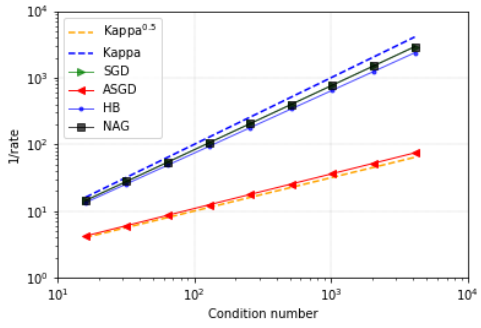
A. Linear Regression

Following are our key insights obtained from simulations:

1) In the original paper, the authors had provided the optimal hyperparameter settings for both SGD and ASGD, and had recommended to perform Grid Search only for finding the optimal learning rate and momentum values for Heavy Ball and NAG. However in our experience, we found to the contrary that the given learning rates for both SGD and ASGD failed to meet the convergence criteria as defined by the authors.
2) Given the lack of optimal learning rates for SGD and ASGD, we performed Grid Search to obtain the optimal hyperparameter settings for them also, along with the previously decided Grid Search for Heavy Ball and NAG. Though we were able to obtain hyperparameters which minimized the error value for each algorithm using Grid Search; this led to divergence in the convergence performance metric. Note that at the end of each iteration t, the loss value $f(w_t)$ decreases, and optimal hyperparameters obtained from grid search often led to



(a) 1/rate vs κ as obtained



(b) 1/rate vs κ on using a best fit line

Fig. 1: 1/rate vs κ for Gaussian Distribution

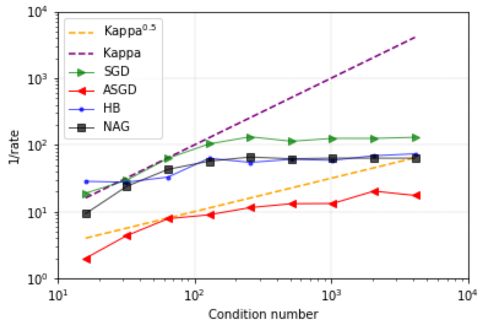


Fig. 2: 1/rate vs κ for Discrete Distribution as obtained

the loss value becoming zero. As a consequence, the value of convergence performance diverged.

$$f(w_t) \to 0 \implies \log(f(w_t)) \to -\infty \implies rate \to \infty$$

This leads to breaks in the plot of $1/rate$ vs κ

3) To mitigate this; while performing grid search we only picked those hyperparameters which met the convergence criteria and did not lead to a zero error value $f(w_t)$. This is also evident in our code.
4) In practise, Grid Search did not yield the optimum set of hyperparameters exactly. This is because each hyperparameter can have a wide range of values in [0, 1] for 100 trials, making it computationally expensive to perform a fine search over the range of values.
5) Since we only had 9 (κ, rate) points, one for each value of κ, we obtained the values of slopes by fitting the points along a straight line and adjusting the intercept. The intercepts were inaccurate to begin with since we had not picked the optimal hyperparameter values due to the aforementioned reasons.

TABLE I: Best Fit Line Slopes

	SGD	HB	NAG	ASGD
Gaussian	0.95978	0.93446	0.95946	**0.51848**

Figures 1a and 2 illustrate the relation between reciprocal of convergence performance and condition number κ on a logarithmic scale, as obtained originally for Gaussian and Discrete Distributions respectively for the different optimization algorithms. Figure 1b illustrates the slope values after plotting the best fit line for the Gaussian case. The chosen hyperparameters, their convergence rates and slopes were documented as seen in Table I. Two reference lines were also plotted (slope = 1 for κ and slope = $\sqrt{1/\kappa}$ for $\kappa^{0.5}$ to contrast the performance of each algorithm. If a plotted line is closer to κ, then the relation between $\log \kappa$ and $\log(1/rate)$ is likely to be linear; else if a plotted line is closer to the line $\kappa^{0.5}$ then the relation between $\log \kappa$ and $\log(1/rate)$ is likely to be square root (i.e. $\sqrt{\kappa}$). Clearly, ASGD is closer to the $\kappa^{0.5}$ line while the other algorithms are closer to the κ. Therefore, ASGD can provide significant improvement over other algorithms while Heavy Ball and NAG perform similar to SGD, and are therefore not the optimal choice in most cases. Our experiments provide an empirical basis to the conclusions made in the original paper and are consistent with contribution (1) and (2) of the same.

Note that we obtained surprising results for the Discrete Distribution case as opposed to those mentioned in the paper. We believe this can be explained as follows:

1) The value of w^* is fixed beforehand, and by definition a can take only one of two value (say a_1 and a_2) depending on the Binomial Distribution Sampling. This implies that b can only take one of two values since $b =< w^*, a >$ (say b_1 and b_2 respectively). Hence our entire dataset is populated with only two types data points (a_1, b_1) and (a_2, b_2).
2) Therefore, the aim of our Linear Neural Network is to essentially associate label b_1 with a_1 and b_2 with a_2, i.e., there is no diversity either in the input values nor in the output values. Note that this was not the case in Gaussian Distribution where $a \epsilon$ Gaussian Distribution $\implies b =< w^*, a > \epsilon$ Gaussian Distribution also.
3) Any decent optimization method should attain this within a constant number of iterations, since the only other point of consideration will be the actual skewdness of the data set (a function of the random sampling).
4) As evident from the plot for Discrete Distribution, this is seemed to be achieved at Condition Number = 10^2 mark (even though the total iterations for each κ are defined as $5 * \kappa$), after which the the loss function converges approximately, and hence $rate \to constant$ irrespective

of $kappa$ after 10^2 mark, resulting in line approximately parallel to the X-axis.

5) However, even then, it is clearly evident that ASGD achieves this convergence much faster than the other algorithms, irrespective of the value of κ.
6) Since a Best Fit Line method assigns an equal weightage to each point, the best fit line for each algorithm would have a significant contribution of 0 in its slope. Hence, approximate slopes are no longer a suitable criteria to compare their convergence power for the Discrete Case.

B. Deep Autoencoders

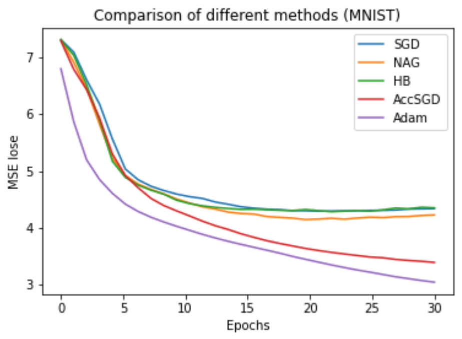


Fig. 3: Plot for MSE loss of deep autoencoder trained with mini batch size of 8. NAG performs better than SGD. The error decay rate of Adam is largest.

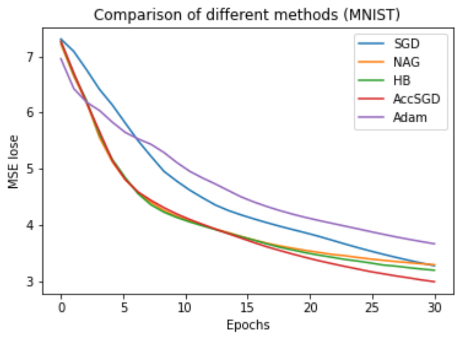


Fig. 4: MSE loss of deep autoencoder trained with batch size 1. The error of ASGD decays faster than other methods (which have similar decay rates with Adam being the slowest).

TABLE II: MSE Loss Convergence Value

Batch size	SGD	HB	NAG	ASGD	Adam
8	4.34133	4.35470	4.22998	3.39756	**3.05110**
1	3.28540	3.20536	3.30243	**3.00185**	3.67435

Following are our key insights obtained from simulations:

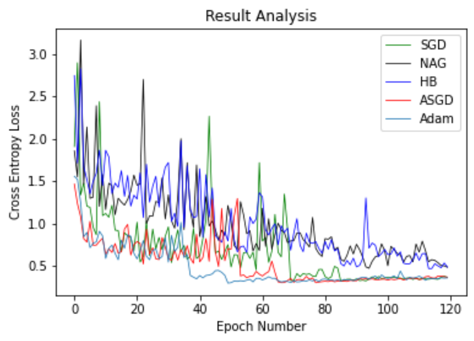
1) Some additional changes were made regarding the original experimental setup for Deep Autoencoder along with those mentioned previously. We performed 30 epochs in contrast to original setup for training of mini batches of size 8 since the MSE loss was found to converge till the number of runs reached to 30 (Fig 3).
2) For tuning the learning rate and momentum hyperparameters (whichever is applicable) we performed grid search on the given list of hyperparameters in the appendix of the original paper but found that MSE loss did not converge within that subset. This inconsistency arose due to the difference in the architecture used by us and the original paper. The smaller size of dataset used, also led to it. So we changed the list of learning rate hyperparameters set on which grid search worked.
3) We chose a set consisting of smaller learning rates following the intuition that our model requires smaller learning rates to learn due to the smaller size of dataset used by us. We decided to vary the learning rate hyperparameter from the original one, by a factor of 100, after which we were able to achieve convergence.
4) For changing the settings and estimating the range of the learning rate hyperparameters, we plotted all the the MSE loss curves for that particular algorithm by running them for 10 epochs to get an early estimation. On the basis of the plots attained we changed the range of the probable hyperparameter set (used for grid search) depending on whether they converged at a very high MSE loss (suggesting a smaller learning rate) or the graph diverged (suggesting learning rate is too high), until we achieved a set where we could observe a convergence in the plots.
5) We devised a strategy of tuning the hyperparameters, in accordance with intuition behind binary search algorithm, where we first obtain the two best hyperparameters from the initial grid search and then run another grid search for the values lying between the two values obtained. The optimal hyperparameter value obtained from this nested search is used for the final settings.
6) The MSE loss where the various gradient methods converged, is higher than that where those algorithm converged in the original paper. The reason can be explained by the fact pertaining to the simplicity of the model's architecture along with smaller dataset used.

While contrasting the variants regarding the batch size, the ASGD outperforms the other three methods on mini-batch size of 1. For Figure 3, in case of mini-batch size of 8, NAG and HB performs better than than the SGD algorithm whereas for batch size of 1 (Fig 4), all the the three performs almost similarly. The error decay rate of ASGD is comparable to Adam in case of mini-batch size of 8 (Fig 3), but ASGD decays the MSE loss at faster rate in case of 1 mini-batch size (Fig 4). Also, the SGD decays with least rate in all the cases. Adam decays the MSE error with fastest rate in case of mini-batch size being 8 whereas this decay rate is least when mini-batch size is 1.

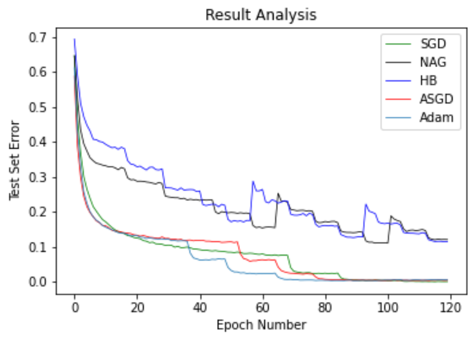
C. Deep Residual Networks

Following are our key insights obtained from simulations:

1) Since Training PreResnet for 120 epochs was already computationally expensive, given our lack of sufficient computational resources, performing grid search to obtain optimal hyperparameter settings for each algorithm

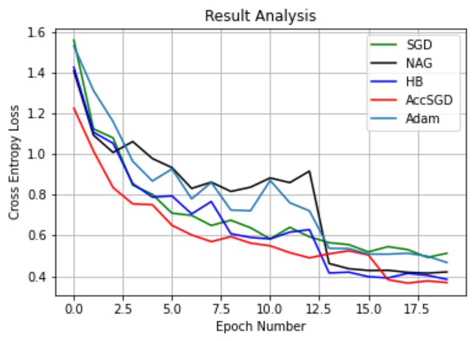


(a) Validation Loss vs Number of Epochs

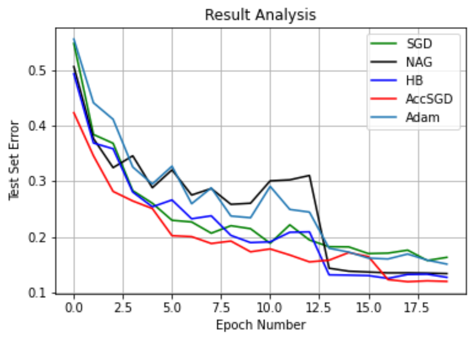


(b) Test Error vs Number of Epochs

Fig. 5: Plots for fixed hyperparameter schedule-128 batch size



(a) Validation Loss vs Number of Epochs



(b) Test Error vs Number of Epochs

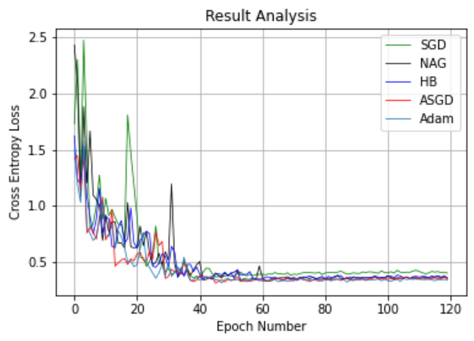
Fig. 6: Plots for decayed hyperparameter schedule-8 batch size

TABLE III: Decayed Hyperparameter schedule metrics

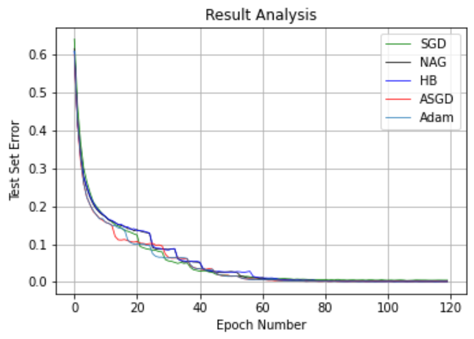
Algorithm	Batch/Epoch	Final Validation Loss	Final Test Error
SGD	128/80	0.40805 ±0.00778	0.09779 ±0.00076
	8/20	0.52085 ±0.019727	0.16665 ±0.00710
HB	128/80	0.36592 ±0.00813	0.08501 ±0.00068
	8/20	0.39928 ±0.01137	0.12905 ±0.00327
NAG	128/80	0.36007 ±0.00798	0.08652 ±0.0007
	8/20	0.42158 ±0.00508	0.13450 ±0.00055
ASGD	128/80	0.3509 ±0.00856	0.08216 ±0.00093
	8/20	**0.37443 ±0.00656**	**0.12043 ±0.001420**
Adam	128/120	**0.34208 ±0.007**	**0.08162 ±0.00074**
	8/20	0.49724 ±0.01765	0.15937 ±0.00649

was no longer feasible. This led us to create two different hyperparameter tuning schemes for training our models: Fixed/Decayed Hyperparameter Schedule.

2) Figure 5 shows the results of our Fixed Hyperparameter Schedule Scheme on 128 batch size instance. We store a learning rate set of {0.27,0.09,0.03,0.01} and momentum set {0.97,0.95,0.9,0.8,0.5} with the initial learning rate and momentum set to the first elements of the sets respectively. Then every 4th epoch, if the validation loss did not reduce by more than 1%, we move onto the next momentum value, and if the momentum set has been exhausted (or if momentum is not applicable for an algorithm like SGD), we shift to the next learning rate in the set (and reset the momentum to the first element).
3) Note that there are a lot of oscillations in the validation loss as well as test error in the case of momentum based methods like HB and NAG, while SGD, ASGD and Adam (which tune only the learning rate have much smoother converging plots). We deduced that this was because HB and NAG exhausted all possible momentum values before moving onto the next lower learning rate (see Point 2), i.e. as compared to SGD, ASGD and Adam, higher learning rates were being employed for significantly more time in HB and NAG (when varying momentum values) leading to oversteppings and hence oscillations. Therefore, we needed to devise a scheme where we could reduce our hyperparameter values to finer levels as we come nearer to the convergence mark. However, it is still quite evident that ASGD outperforms all other algorithms (except Adam).
4) Using the above insights, we devised a new hyperparameter tuning scheme based on the concept of decay. We fixed our initial learning rate and momentum, and after every fixed number of epochs, if our chosen metric does not change by more than a certain threshold, we reduced the learning rate by a constant factor. For batch size 128, we fixed the initial learning rate to 0.27 and momentum to 0.05 while we performed a grid search in the case of batch size 8.
5) Figure 6 shows our results obtained on 8 Batch Size PreResnet 44 Instance. By performing grid search on a set of recommended values (refer paper), we set the learning rate and the momentum value for each algorithm. Then after every 3 epochs, if the validation error did not reduce by more than 1%, we reduced the learning rate by a factor of 5. The minimal learning rate



(a) Validation Loss vs Number of Epochs



(b) Test Error vs Number of Epochs

Fig. 7: Decayed hyperparameter schedule plots-128 batch size

was fixed to $6.25 * 10^{-5}$, so that our progress did not curtail due to too much decay. Clearly, we obtained a much smoother plot than from our previous scheme.

6) Encouraged by the visible improvements, we employed the same decayed hyperparameter scheme to the 128 batch size problem instance. We set the learning rate to 0.27 and the momentum value to 0.5 for each algorithm. Then after every 4 epochs, if the validation error did not reduce by more than 0.2%, we reduced the learning rate by a factor of 2 (decay). The minimal learning rate was fixed to $1 * 10^{-3}$. Note that we did not employ Grid Search here because 1) It was computationally infeasible given our limited resources, 2) We found that for each algorithm, by the completion of 120 epochs each value in the set $\{0.27, 0.135, ..., 0.002109375, 0.001054687, 0.001\}$ was being used as learning rate at some point, with the optimal learning rates being used for a large number of epochs; approximately mimicking a grid search. Figure 7 plots our obtained results for the same.

Table III documents our final results. In case of batch size 128, the final metric value was taken to be the mean of last 40 epochs, while in batch size 8 it was the mean of last 4 epochs. For batch size 8, ASGD attains a lower test error and a higher convergence rate than all the other algorithms including Adam. Surprisingly, Adam seems to fare much worse than all the algorithms except SGD. In case of batch size 128, ASGD convergences faster as well achieves less error than all other algorithms, outperformed only by Adam by a small margin. We believe an extensive grid search will enable ASGD to perform atleast as well as Adam if not better in case of larger batch sizes. This provides empirical evidence for the fact that minibatching stabilises the variance in momentum based methods, and greatly improves the performance of NAG and HB as compared to SGD.

V. Conclusions and Recommendations

In this report, we have provided empirical evidence for the fact that Momentum based optimization methods such as HB and NAG enjoy practical gains over SGD in deep learning applications majorly due to minibatching. Moreover, these methods are sub optimal in case of SFO problem instances. On the other hand, ASGD improves significantly over them for the same problem instances. We observe this gain on simple problems such as Linear Regression as well as Deep Learning problems such as Autoencoders on MNIST and Resnet on CIFAR10. While ASGD performs similar to Adam on large mini-batch sizes, ASGD outperforms Adam when using small mini-batch sizes. Based on our experiments, we highly recommend the use of ASGD over pre-existing variants of Stochastic based optimization methods.

References

[1] L. A. Cauchy. Méthode générale pour la résolution des systèmes d'équations simultanees. C. R. Acad. Sci.Paris, 1847. Paris, 1847.
[2] Yurii E. Nesterov. Introductory lectures on convex optimization: A basic course, volume 87 of Applied Optimization. Kluwer Academic Publishers, 2004.
[3] Boris T Polyak. Some methods of speeding up the convergence of iteration methods. USSR Computational Mathematics and Mathematical Physics, 4(5):1–17, 1964.
[4] Yurii Nesterov. A method of solving a convex programming problem with convergence rate o (1/k2). In Soviet Mathematics Doklady, volume 27, pp. 372–376, 1983.
[5] Herbert Robbins and Sutton Monro. A stochastic approximation method. The Annals of Mathematical Statistics, vol.22, 1951.
[6] Yurii E. Nesterov.Introductory lectures on convex optimization: A basic course, volume 87 of Applied Optimization Kluwer Academic Publishers, 2004.
[7] Prateek Jain, Sham M Kakade, Rahul Kidambi, Praneeth Netrapalli, and Aaron Sidford. Accelerating stochastic gradient descent. arXiv preprint arXiv:1704.08227, 2017.
[8] Rahul Kidambi, Praneeth Netrapalli, Prateek Jain, and Sham M. Kakade. On the insufficiency of existing momentum schemes for Stochastic Optimization
[9] Diederik P. Kingma, Jimmy Lei Ba ADAM: A METHOD FOR STOCHASTIC OPTIMIZATION
[10] Pytorch https://github.com/pytorch.
[11] AccSGD https://github.com/rahulkidambi/AccSGD
[12] Geoffrey E Hinton and Ruslan R Salakhutdinov. Reducing the dimensionality of data with neural networks. science, 313 (5786):504–507, 2006.
[13] Ilya Sutskever, James Martens, George Dahl, and Geoffrey Hinton. On the importance of initialization and momentum in deep learning. In International conference on machine learning, pp. 1139–1147, 2013. pp. 740–741, August 1987 [Digests 9th Annual Conf. Magnetics Japan, p. 301, 1982].
[14] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Deep residual learning for image recognition. In CVPR, pp. 770–778, 2016b
[15] Preresnet-44 for cifar-10. ResNeXt-DenseNet
[16] Insufficiency momentum schemes for Stochastic Optimization
[17] Report on the Insufficiency of Existing Momentum Schemes for Stochastic Optimization