# ON THE INSUFFICIENCY OF EXISTING MOMENTUM SCHEMES FOR STOCHASTIC OPTIMIZATION

**Anonymous authors**
Paper under double-blind review

## ABSTRACT

Momentum based stochastic gradient methods such as heavy ball (HB) and Nesterov's accelerated gradient descent (NAG) method are widely used in practice for training deep networks and other supervised learning models, as they often provide significant improvements over stochastic gradient descent (SGD). In general, "fast gradient" methods have provable improvements over gradient descent only for the deterministic case, where the gradients are exact. In the stochastic case, the popular explanations for their wide applicability is that when these fast gradient methods are applied in the stochastic case, they partially mimic their *exact* gradient counterparts, resulting in some practical gain. This work provides a counterpoint to this belief by *proving* that there are simple problem instances where these methods *cannot* outperform SGD despite the best setting of its parameters. These negative problem instances are, in an informal sense, generic; they do not look like carefully constructed pathological instances. These results suggest (along with empirical evidence) that HB or NAG's practical performance gains are a by-product of minibatching.

Furthermore, this work provides a viable (and provable) alternative, which, on the same set of problem instances, significantly improves over HB, NAG, and SGD's performance. This algorithm, denoted as ASGD, is a simple to implement stochastic algorithm, based on a relatively less popular version of Nesterov's AGD. Extensive empirical results in this paper show that ASGD has performance gains over HB, NAG, and SGD.

## 1 INTRODUCTION

First order optimization methods, which access a function (to be optimized) through its gradient or an unbiased approximation of its gradient, are the workhorses for modern large scale optimization problems, which include training the current state-of-the-art deep neural networks. Gradient descent (Cauchy, 1847) is the simplest first order method that is used heavily in practice. However, it is known that for the class of smooth convex functions as well as some simple non-smooth problems (Nesterov, 2012a)), gradient descent is suboptimal (Nesterov, 2004) and there exists a class of algorithms called fast gradient/momentum based methods which achieve optimal convergence guarantees. The heavy ball method (Polyak, 1964) and Nesterov's accelerated gradient descent (Nesterov, 1983) are two of the most popular methods in this category.

On the other hand, training deep neural networks on large scale datasets have been possible through the use of Stochastic Gradient Descent (SGD) (Robbins & Monro, 1951), which samples a random subset of training data to compute gradient estimates that are then used to optimize the objective function. The advantages of SGD for large scale optimization and the related issues of tradeoffs between computational and statistical efficiency was highlighted in Bottou & Bousquet (2007).

The above mentioned theoretical advantages of fast gradient methods (Polyak, 1964; Nesterov, 1983) (albeit for smooth convex problems) coupled with cheap to compute stochastic gradient estimates led to the influential work of Sutskever et al. (2013), which demonstrated the empirical advantages possessed by SGD when augmented with the momentum machinery. This work has led to the wide spread adoption of momentum methods for training deep neural networks; so much so that, in the context of neural network training, gradient descent often refers to momentum methods.

But, there is a subtle difference between classical momentum methods and their implementation in practice – classical momentum methods work in the exact first order oracle model (Nesterov, 2004), i.e., they employ *exact gradients* (computed on the full training dataset), while in practice (Sutskever et al., 2013), they are implemented with *stochastic gradients* (estimated from a randomly sampled subset/mini-batch of the training data). This leads to a natural question:

*Are momentum methods optimal even in the* **stochastic first order oracle** *(SFO) model, where we access stochastic gradients computed on a small constant sized minibatches (or a batchsize of* $1$*?)*

Even disregarding the question of optimality of momentum methods in the SFO model, it is not even known if momentum methods (say, Polyak (1964); Nesterov (1983)) provide *any provable* improvement over SGD in this model. While these are open questions, a recent effort of Jain et al. (2017) showed that improving upon SGD (in the stochastic first order oracle) is rather subtle as there exists problem instances in SFO model where it is not possible to improve upon SGD, even information theoretically. Jain et al. (2017) studied a variant of Nesterov's accelerated gradient updates (Nesterov, 2012b) for stochastic linear regression and show that their method improves upon SGD wherever it is information theoretically admissible. Through out this paper, we refer to the algorithm of (Jain et al., 2017) as Accelerated Stochastic Gradient Method (ASGD) while we refer to a stochastic version of the most widespread form of Nesterov's method (Nesterov, 1983) as NAG; HB denotes a stochastic version of the heavy ball method. Critically, while the result of Jain et al. (2017) shows that ASGD improves on SGD in any information-theoretically admissible regime, it is still not known whether HB and NAG can achieve a similar performance gain.

A key contribution of this work is to show that HB *does not* provide similar performance gain to ASGD even when it is informationally-theoretically admissible. That is, we provide a problem instance where it is indeed possible to improve upon SGD (and in fact ASGD achieves this improvement), but HB *cannot* achieve any improvement over SGD. We validate this claim empirically as well. In fact, we provide empirical evidence to the claim that NAG also do not achieve any improvement over SGD for several problems where ASGD can still achieve better rates of convergence.

This raises a question about why HB and NAG provide better performance than SGD in practice (Sutskever et al., 2013), especially for training deep networks. Our conclusion (that is well supported by our theoretical result) is that HB and NAG's improved performance is attributed to mini-batching and hence, these methods will often struggle to improve over SGD with small constant batch sizes. Note that this result indicates that there is a natural tension between the gains offered by momentum methods with larger batches and the superior generalization properties offered by training with small mini-batches (Keskar et al., 2016), which is a regime that may not be amenable for HB/NAG to improve upon SGD. This is in stark contrast to methods like ASGD, which are designed to improve over SGD across small or large mini-batch sizes. In fact, based on our experiments, we observe that on the task of training deep residual networks (He et al., 2016a) on the cifar-10 dataset, we note that ASGD offers noticeable improvements by achieving $5-7\%$ better test error over HB and NAG even with commonly used batch sizes like $128$ during the initial stages of the optimization.

### 1.1 Contributions

The contributions of this paper are as follows.

1. In Section 3, we prove that HB is *not optimal* in the SFO model. In particular, there exist linear regression problems for which the performance of HB (with *any* step size and momentum) is either the same or worse than that of SGD while ASGD improves upon both of them.
2. Experiments on several different linear regression problems suggest that the suboptimality of HB in the SFO model is not restricted to special cases – it is rather widespread. Empirically, the same holds true for NAG as well (Section 5).
3. The above observations suggest that the only reason for the superiority of momentum methods in practice is minibatching, which reduces the variance in stochastic gradients and moves closer to exact first order oracle. This conclusion is supported by empirical evidence through training deep residual networks on cifar-10, with a batch size of $8$ (see Section 5.3).
4. We present an intuitive and easier to tune version of ASGD (see Section 4) and show that ASGD can provide significantly faster convergence to a reasonable accuracy than SGD, HB, NAG, while still providing asymptotically optimal accuracy.

**Algorithm 1** HB: Heavy ball with a SFO

**Require:** Initial $w_0$, stepsize $\delta$, momentum $\alpha$

1: $w_{-1} \leftarrow w_0$; $t \leftarrow 0$ /*Set $w_{-1}$ to $w_0$*/
2: **while** $w_t$ not converged **do**
3: $w_{t+1} \leftarrow w_t - \delta \cdot \widehat{\nabla} f_t(w_t) + \alpha \cdot (w_t - w_{t-1})$ /*Sum of stochastic gradient step and momentum*/
4: $t \leftarrow t + 1$

**Ensure:** $w_t$ /*Return the last iterate*/

**Algorithm 2** NAG: Nesterov's AGD with a SFO

**Require:** Initial $w_0$, stepsize $\delta$, momentum $\alpha$

1: $v_0 \leftarrow w_0$; $t \leftarrow 0$ /*Set $v_0$ to $w_0$*/
2: **while** $w_t$ not converged **do**
3: $v_{t+1} \leftarrow w_t - \delta \cdot \widehat{\nabla} f_t(w_t)$ /*SGD step*/
4: $w_{t+1} = (1+\alpha) v_{t+1} - \alpha v_t$ /*Sum of SGD step and previous iterate*/
5: $t \leftarrow t + 1$

**Ensure:** $w_t$ /*Return the last iterate*/

Hence, the take-home message of this paper is: *HB and NAG are not optimal in the SFO model. The only reason for the superiority of momentum methods in practice is minibatching. ASGD provides a distinct advantage in training deep networks over SGD, HB and NAG.*

## 2 Notation

We denote matrices by bold-face capital letters and vectors by lower-case letters. $f(w) = 1/n \sum_i f_i(w)$ denotes the function to optimize w.r.t. $w$. $\nabla f(w)$ denote gradient of $f$ at $w$ while $\widehat{\nabla} f_t(w)$ denote a stochastic gradient of $f$. That is, $\widehat{\nabla} f_t(w_t) = \nabla f_{i_t}(w)$ where $i_t$ is sampled uniformly at random from $[1, \ldots, n]$. For linear regression, $f_i(w) = (y - \langle w, x_i \rangle)^2$ where $y$ is the target variable and $x \in \Re^d$ is the covariate, and $\widehat{\nabla} f_t(w_t) = -(y_t - \langle w_t, x_t \rangle) x_t$. In this case, $\mathbf{H} = \mathbb{E}\left[x x^\top\right]$ denotes the Hessian of $f$ and $\kappa = \frac{\lambda_1(\mathbf{H})}{\lambda_d(\mathbf{H})}$ denotes it's condition number.

Algorithm 1 provides a pseudo-code of HB method (Polyak, 1964). $w_t - w_{t-1}$ is the momentum term and $\alpha$ denotes the momentum parameter. Next iterate $w_{t+1}$ is obtained by a linear combination of the SGD update and the momentum term. Algorithm 2 provides pseudo-code of a stochastic version of the most commonly used form of Nesterov's accelerated gradient descent (Nesterov, 1983)

## 3 Suboptimality of Heavy Ball Method

In this section, we show that there exist linear regression problems where the performance of HB (Algorithm 1) is no better than that of SGD, while ASGD significantly improves upon this performance.Let us now describe the problem instance.

Fix $w^* \in \mathbb{R}^2$ and let $(\mathbf{x}, \mathbf{y}) \sim \mathcal{D}$ be the following distribution over $\mathbb{R}^2$:

$$x = \begin{cases} \sigma_1 \cdot z \cdot e_1 \text{ w.p. } 0.5 \\ \sigma_2 \cdot z \cdot e_2 \text{ w.p. } 0.5, \end{cases} \qquad \text{and} \qquad y = \langle w^*, x \rangle,$$

where $e_1, e_2 \in \mathbb{R}^2$ are canonical basis vectors, $\sigma_1 > \sigma_2 > 0$. Let $z$ be a random variable such that $\mathbb{E}\left[z^2\right] = 2$ and $\mathbb{E}\left[z^4\right] = 2c \geq 4$. Hence, we have: $\mathbb{E}\left[(x^{(i)})^2\right] = \sigma_i^2, \mathbb{E}\left[(x^{(i)})^4\right] = c\sigma_i^4$, for $i = 1, 2$. Now, our goal is to minimize:

$$f(w) \stackrel{\text{def}}{=} \mathbb{E}\left[(\langle w^*, x \rangle - y)^2\right], \text{ Hessian } \mathbf{H} \stackrel{\text{def}}{=} \mathbb{E}\left[x x^\top\right] = \begin{bmatrix} \sigma_1^2 & 0 \\ 0 & \sigma_2^2 \end{bmatrix}.$$

Let $\kappa$ and $\tilde{\kappa}$ denote the *computational* and *statistical* condition numbers – see Jain et al. (2017) for definitions. For the problem above, we have $\kappa = \frac{2c\sigma_1^2}{\sigma_2^2}$ and $\tilde{\kappa} = 2c$. Then we obtain following convergence rates for SGD and ASGD when applied to the above given problem instance:

**Corollary 1** (of Theorem 1 of Jain et al. (2016))**.** *Let $w_t^{SGD}$ be the $t^{th}$ iterate of SGD on the above problem with starting point $w_0$ and stepsize $\frac{1}{c\sigma_1^2}$. The error of $w_t^{SGD}$ can be bounded as,*

$$\mathbb{E}\left[f\left(w_t^{SGD}\right)\right] - f\left(w_*\right) \leq \exp\left(\frac{-t}{\kappa}\right)\left(f\left(w_0\right) - f\left(w_*\right)\right).$$

On the other hand, ASGD achieves the following superior rate.

**Algorithm 3** Accelerated stochastic gradient descent – ASGD

**Input:** Initial $w_0$, short step $\delta$, long step parameter $\kappa \geq 1$, statistical advantage parameter $\xi \leq \sqrt{\kappa}$

1: $\bar{w}_0 \leftarrow w_0$; $t \leftarrow 0$ /*Set running average to $w_0$*/

2: $\alpha \leftarrow 1 - \frac{\xi}{\kappa}$ /*Set momentum value*/

3: **while** $w_t$ not converged **do**

4: $\bar{w}_{t+1} \leftarrow \alpha \cdot \bar{w}_t + (1-\alpha) \cdot \left( w_t - \frac{\kappa \cdot \delta}{0.7} \cdot \widehat{\nabla} f_t(w_t) \right)$ /*Update the running average as a weighted average of previous running average and a long step gradient */

5: $w_{t+1} \leftarrow \frac{0.7}{0.7+(1-\alpha)} \cdot \left( w_t - \delta \cdot \widehat{\nabla} f_t(w_t) \right) + \frac{1-\alpha}{0.7+(1-\alpha)} \cdot \bar{w}_{t+1}$ /*Update the iterate as weighted average of current running average and short step gradient*/

6: $t \leftarrow t+1$

**Output:** $w_t$ /*Return the last iterate*/

**Corollary 2** (of Theorem 1 of Jain et al. (2017))**.** *Let $w_t^{ASGD}$ be the $t^{th}$ iterate of ASGD on the above problem with starting point $w_0$ and appropriate parameters. The error of $w_t^{ASGD}$ can be bounded as,*

$$\mathbb{E}\left[f\left(w_t^{ASGD}\right)\right] - f\left(w_*\right) \leq \text{poly}(\kappa) \exp\left(\frac{-t}{\sqrt{\kappa\tilde{\kappa}}}\right)\left(f\left(w_0\right) - f\left(w_*\right)\right).$$

Note that $\tilde{\kappa} = 2c$ is a constant while $\kappa = \frac{2c\sigma_1^2}{\sigma_2^2}$ can be arbitrarily large. Hence, ASGD improves upon rate of SGD by a factor of $\sqrt{\kappa}$. The following proposition, which is the main result of this section, establishes that HB (Algorithm 1) cannot provide a similar improvement. In fact, we show that despite selecting best parameters for HB, it's convergence rate is exactly same as that SGD (Corollary 1), up to constants.

**Proposition 3.** *Let $w_t^{HB}$ be the $t^{th}$ iterate of HB (Algorithm 1) on the above problem with starting point $w_0$. For* any *choice of stepsize $\delta$ and momentum $\alpha \in [0,1]$, $\exists T$ large enough such that $\forall t \geq T$, we have,*

$$\mathbb{E}\left[f\left(w_t^{HB}\right)\right] - f\left(w_*\right) \geq C(\kappa,\delta,\alpha) \cdot \exp\left(\frac{-500t}{\kappa}\right)\left(f\left(w_0\right) - f\left(w_*\right)\right),$$

*where $C(\kappa,\delta,\alpha)$ depends on $\kappa, \delta$ and $\alpha$ (but not on $t$).*

That is, to obtain $\widehat{w}$ s.t. $\|\widehat{w} - w^*\| \leq \epsilon$, HB requires $\Omega(\kappa \log \frac{1}{\epsilon})$ samples and iterations. On the other hand, ASGD can obtain $\epsilon$-approximation to $w^*$ in $\mathcal{O}(\sqrt{\kappa} \log \frac{1}{\epsilon})$ iterations. While we do not prove it theoretically, we observe empirically that for the same problem instance, NAG also obtains nearly same rate as HB and SGD.

## 4 Algorithm

In this section, we will present and explain an intuitive version of ASGD. Algorithm 3 presents the pseudocode. The algorithm takes three inputs: short step $\delta$, long step parameter $\kappa$ and statistical advantage parameter $\xi$. The short step $\delta$ is precisely the same as the step size in SGD, HB or NAG. For convex problems, this scales inversely with the smoothness of the function. The long step parameter $\kappa$ is intended to give an estimate of the ratio of the largest and smallest curvatures of the function; for convex functions, this is just the condition number. The statistical advantage parameter $\xi$ captures trade off between statistical and computational condition numbers – in the deterministic case, $\xi = \sqrt{\kappa}$ and ASGD is equivalent to NAG, while in the high stochasticity regime, $\xi$ is much smaller. The algorithm itself maintains two iterates: descent iterate $w_t$ and a running average $\bar{w}_t$. The running average is a weighted average of the previous average and a long gradient step from the descent iterate, while the descent iterate is updated as a convex combination of short gradient step from the descent iterate and the running average. The idea is that since the algorithm takes a long step as well as short step and an appropriate average of both of them, it can make progress on different directions at a similar pace. Appendix B shows the equivalence between Algorithm 3 and ASGD as proposed in Jain et al. (2017). Note that the constant 0.7 appearing in Algorithm 3 has no special significance. Jain et al. (2017) require it to be smaller than $\sqrt{1/6}$ but any constant smaller than 1 seems to work in experiments.

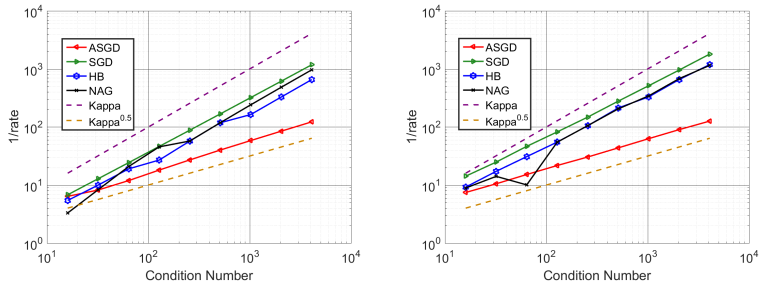


Figure 1: Plot of 1/rate (refer equation (1)) vs condition number ($\kappa$) for various methods for the linear regression problem. Discrete distribution in the left, Gaussian to the right.

## 5 EXPERIMENTS

We now present our experimental results exploring performance of SGD, HB, NAG and ASGD. Our experiments are geared towards answering the following questions:

- Even for linear regression, is the suboptimality of HB restricted to specific distributions given in Section 3 or does it hold for more general distributions as well? Is the same true of NAG?
- 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 minibatching? Does this superiority hold even for small minibatches?
- How does the performance of ASGD compare to that of SGD, HB and NAG, especially while training deep networks?

Section 5.1 and parts of Section 5.2 address the first two questions. Section 5.2 and 5.3 address Question 2 partially and the last question. We use Matlab to conduct experiments presented in Section 5.1 and use PyTorch pyt for our deep networks related experiments.

### 5.1 LINEAR REGRESSION

In this section, we will present results on performance of the four optimization methods (SGD, HB, NAG, and ASGD) for linear regression problems. We consider two different class of linear regression problems, both of them in two dimensions. Given $\kappa$ which stands for condition number, we consider the following two distributions:

**Discrete**: $x = e_1$ w.p. $0.5$ and $x = \frac{2}{\kappa} \cdot e_2$ with $0.5$; $e_i$ is the $i^{th}$ standard basis vector.

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

We fix a randomly generated $w^* \in \mathbb{R}^2$ and for both the distributions above, we let $y = \langle w^*, x \rangle$. We vary $\kappa$ from $\{2^4, 2^5, ..., 2^{12}\}$ and for each $\kappa$ in this set, we run 100 independent runs of all four methods, each for a total of $t = 5\kappa$ iterations. We define that the algorithm converges if there is no error in the second half (i.e. after $2.5\kappa$ updates) that exceeds the starting error - this is reasonable since we expect *geometric convergence* of the initial error.

Unlike ASGD and SGD, we do not know optimal learning rate and momentum parameters for NAG and HB in the stochastic gradient model. So, we perform a *grid search* over the values of the learning rate and momentum parameters. In particular, we lay a $10 \times 10$grid in $[0, 1] \times [0, 1]$ for learning rate and momentum and run NAG and HB. Then, for each grid point, we consider the subset of 100 trials that converged and computed the final error using these. Finally, the parameters that yield the minimal error are chosen for NAG and HB, and these numbers are reported. We measure convergence performance of a method using:

$$\text{rate} = \frac{\log(f(w_0)) - \log(f(w_t))}{t}, \tag{1}$$

We compute the rate (1) for all the algorithms with varying condition number $\kappa$. Given a rate vs $\kappa$ plot for a method, we compute it's *slope* (denoted as $\gamma$) using linear regression. Table 1 presents the estimated slopes (i.e. $\gamma$) for various methods for both the discrete and the Gaussian case. The slope

| Algorithm | Slope – discrete | Slope – Gaussian |
|---|---|---|
| SGD | 0.9302 | 0.8745 |
| HB | 0.8522 | 0.8769 |
| NAG | 0.98 | 0.9494 |
| ASGD | 0.5480 | 0.5127 |

Table 1: Slopes (i.e. $\gamma$) obtained by fitting a line to the curves in Figure 1. A value of $\gamma$ indicates that the error decays at a rate of $\exp\left(\frac{-t}{\kappa^{\gamma}}\right)$. A smaller value of $\gamma$ indicates a faster rate of error decay.

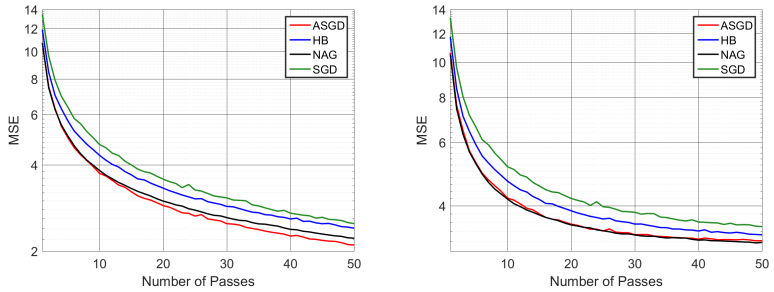


Figure 2: Training loss (left) and test loss (right) while training deep autoencoder for mnist with minibatch size 8. Clearly, ASGD matches performance of NAG and outperforms SGD on the test data. HB also outperforms SGD.

values clearly show that the rate of SGD, HB and NAG have a nearly linear dependence on $\kappa$ while that of ASGD seems to scale linearly with $\sqrt{\kappa}$.

### 5.2 Deep Autoencoders for MNIST

In this section, we present experimental results on training deep autoencoders for the mnist dataset, and we closely follow the setup of Hinton & Salakhutdinov (2006). This problem is a standard benchmark for evaluating the performance of different optimization algorithms e.g., Martens (2010); Sutskever et al. (2013); Martens & Grosse (2015); Reddi et al. (2017). The network architecture follows previous work (Hinton & Salakhutdinov, 2006) and is represented as $784-1000-500-250-30-250-500-1000-784$ with the first and last 784 nodes representing the input and output respectively. All hidden/output nodes employ sigmoid activations except for the layer with 30 nodes which employs linear activations and we use MSE loss. Initialization follows the scheme of Martens (2010), also employed in Sutskever et al. (2013); Martens & Grosse (2015). We perform training with two minibatch sizes $-1$ and 8. The runs with minibatch size of 1 were run for 30 epochs while the runs with minibatch size of 8 were run for 50 epochs. For each of SGD, HB, NAG and ASGD, a grid search over learning rate, momentum and long step parameter (whichever is applicable) was done and best parameters were chosen based on achieving the smallest training error in the same protocol followed by say, Sutskever et al. (2013). The grid was extended whenever the best parameter fell at the edge of a grid. For the parameters chosen by grid search, we perform 10 runs with different seeds and averaged the results. The results are presented in Figures 2 and 3. Note that the final loss values reported here are suboptimal compared to those in published literature e.g., Sutskever et al. (2013); while Sutskever et al. (2013) report results after 750000 updates with a large batch size of 200 (which implies a total of $750000 \times 200 = 150$M gradient evaluations), whereas, our results are after 1.8M updates of SGD with a batch size 1 (which is just 1.8M gradient evaluations).

**Effect of minibatch sizes**: While HB and NAG decay the loss faster compared to SGD for a minibatch size of 8 (Figure 2), this superior decay rate does not hold for a minibatch size of 1 (Figure 3). This supports our intuitions from the stochastic linear regression setting, where we demonstrate that HB and NAG are suboptimal in the stochastic first order oracle model.

**Comparison of ASGD with momentum methods**: While ASGD performs slightly better than NAG for batch size 8 in the training error (Figure 2), ASGD decays the error at a faster rate compared to all the three other methods for a batch size of 1 (Figure 3).

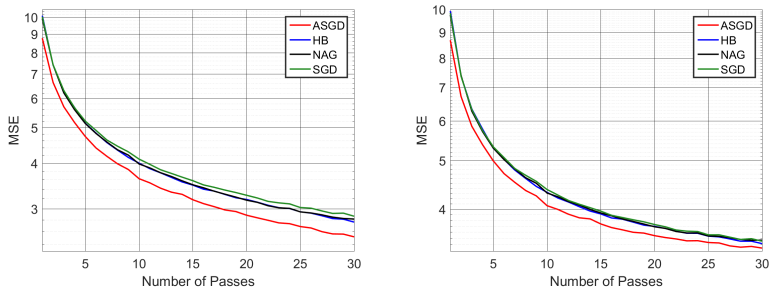


Figure 3: Training loss (left) and test loss (right) while training deep autoencoder for mnist with minibatch size 1. Interestingly, SGD, HB and NAG, all decrease the loss at a similar rate, while ASGD decays at a faster rate.

### 5.3 Deep Residual Networks for CIFAR-10

In this section, we will present experimental results on training deep residual networks (He et al., 2016b) with pre-activation blocks as introduced in He et al. (2016a) for classifying images in cifar-10 (Krizhevsky & Hinton, 2009); the network we use has 44 layers (dubbed preresnet-44). The code for these experiments was downloaded from pre. One of the most distinct characteristics of this experiment compared to our previous experiments is learning rate decay. We use a validation set based decay scheme, wherein, after every 3 epochs, we decay the learning rate by a certain factor (which we grid search on) if the validation zero one error does not decrease by at least a certain amount (precise numbers are provided in the appendix since they vary across batch sizes). Due to space constraints, we present only a subset of training error plots. Please see Appendix C.3 for some more plots on training errors.

**Effect of minibatch sizes**: Our first experiment tries to understand how the performance of HB and NAG compare with that of SGD and how it varies with minibatch sizes. Figure 4 presents the test zero one error for minibatch sizes of 8 and 128. While training with batch size 8 was done for 40 epochs, with batch size 128, it was done for 120 epochs. We perform a grid search over all parameters for each of these algorithms. See Appendix C.3 for details on the grid search parameters. We observe that final error achieved by SGD, HB and NAG are all very close for both batch sizes. While NAG exhibits a superior rate of convergence compared to SGD and HB for batch size 128, this superior rate of convergence disappears for a batch size of 8.

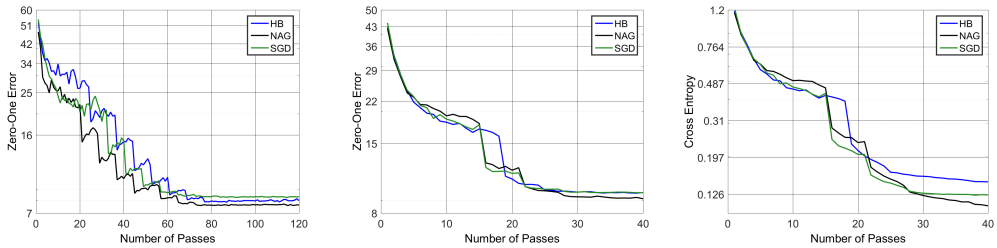


Figure 4: Test zero one loss for batch size 128 (left), batch size 8 (center) and training function value for batch size 8 (right) for SGD, HB and NAG.

**Comparison of ASGD with momentum methods**: The next experiment tries to understand how ASGD compares with HB and NAG. The errors achieved by various methods when we do grid search over all parameters are presented in Table 2. Note that the final test errors for batch size 128 are better than those for batch size 8 since the former was run for 120 epochs while the latter was run only for 40 epochs (due to time constraints).

While the final error achieved by ASGD is similar to that of all other methods, we are more interested in understanding whether ASGD has a superior convergence speed. In order to do this experiment however, we need to address the issue of different learning rates used by various algorithms and different places where they decay learning rate. So, for each of HB and NAG, we choose the learning

| Algorithm | Final test error – batch size 128 | Final test error – batch size 8 |
|---|---|---|
| SGD | $8.32 \pm 0.21$ | $9.57 \pm 0.18$ |
| HB | $7.98 \pm 0.19$ | $9.28 \pm 0.25$ |
| NAG | $7.63 \pm 0.18$ | $9.07 \pm 0.18$ |
| ASGD | $\mathbf{7.23} \pm \mathbf{0.22}$ | $\mathbf{8.52} \pm \mathbf{0.16}$ |

Table 2: Final test errors achieved by various methods for batch sizes of 128 and 8. The hyperparameters have been chosen by grid search.

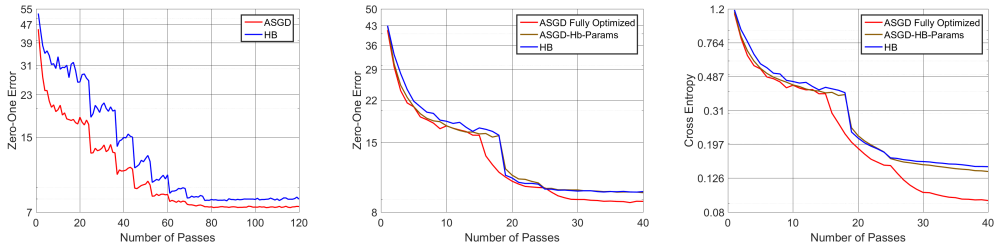


Figure 5: Test zero one loss for batch size 128 (left), batch size 8 (center) and training function value for batch size 8 (right) for ASGD compared to HB. In the above plots, both ASGD and ASGD-Hb-Params refer to ASGD run with the learning rate and decay schedule of HB. ASGD-Fully-Optimized refers to ASGD where learning rate and decay schedule were also selected by grid search.

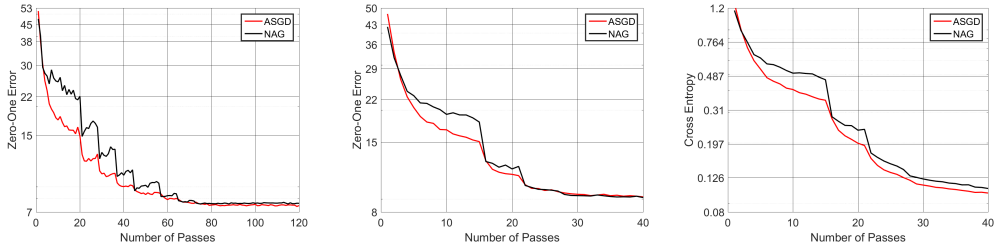


Figure 6: Test zero one loss for batch size 128 (left), batch size 8 (center) and training function value for batch size 8 (right) for ASGD compared to NAG. In the above plots, ASGD was run with the learning rate and decay schedule of NAG. Other parameters were selected by grid search.

rate and decay factors by grid search, use these values for ASGD and do grid search only over long step parameter $\kappa$ and momentum $\alpha$ for ASGD. The results are presented in Figures 5 and 6. For batch size 128, ASGD decays error at a faster rate compared to both HB and NAG. For batch size 8, while we see a superior convergence of ASGD compared to NAG, we do not see this superiority over HB. The reason for this turns out to be that the learning rate for HB, which we also use for ASGD, turns out to be quite suboptimal for ASGD. So, for batch size 8, we also compare fully optimized (i.e., grid search over learning rate as well) ASGD with HB. The superiority of ASGD over HB is clear from this comparison. These results suggest that ASGD decays error at a faster rate compared to HB and NAG across different batch sizes.

## 6 Related Work

**First order oracle methods**: The primary method in this family is Gradient Descent (GD) (Cauchy, 1847). As mentioned previously, GD is suboptimal for smooth convex optimization (Nesterov, 2004), and this is addressed using momentum methods such as the Heavy Ball method (Polyak, 1964) (for quadratics), and Nesterov's Accelerated gradient descent (Nesterov, 1983).

**Stochastic first order methods**: The simplest method employing the SFO is Stochastic Gradient Descent (Robbins & Monro, 1951)(SGD); the effectiveness of SGD has been immense, and its applicability goes well beyond optimizing convex objectives. Accelerating SGD is a tricky proposition

given the instability of fast gradient methods in dealing with noise, as evidenced by several negative results which consider both statistical (Proakis, 1974; Polyak, 1987; Roy & Shynk, 1990) and adversarial errors (Devolder et al., 2014). A result of Jain et al. (2017) developed the first provably accelerated SGD method for linear regression inspired by a method of Nesterov (2012b). Other schemes such as Ghadimi & Lan (2012; 2013); Dieuleveut et al. (2016), which indicate acceleration is possible with noisy gradients do not hold in the SFO model satisfied by algorithms that are run in practice (see Jain et al. (2017) for more details).

While HB (Polyak, 1964) and NAG (Nesterov, 1983) are known to be effective in case of exact first order oracle, for the SFO, the theoretical performance of HB and NAG is not well understood. Polyak (1987) describes HB to be rather brittle when provided with noisy gradient estimates.

**Accelerated and Fast Methods for finite-sums:** There have been exciting recent developments pertaining to faster methods for finite-sums (also referred to as offline stochastic optimization): in particular, methods such as SDCA (Shalev-Shwartz & Zhang, 2012), SAG (Roux et al., 2012), SVRG (Johnson & Zhang, 2013), SAGA (Defazio et al., 2014) offer superior linear convergence rates for strongly convex finite-sums, dramatically improving over SGD's sub-linear rates. Furthermore, these methods are also augmented with the corresponding accelerated variants (Frostig et al., 2015a; Lin et al., 2015; Defazio, 2016; Allen-Zhu, 2016) which are known to improve rates of convergence over the unaccelerated methods. However, note that all these methods require storing the entire training set in memory and taking multiple passes over the same for guaranteed progress. Most of these methods typically require computing a batch gradient or large memory requirements (typically $\mathcal{O}(|\text{ training data points}|)$). For several deep learning problems, data augmentation is deemed necessary for achieving good performance; this implies that computing quantities such as batch gradient over this entire augmented dataset is often infeasible. Such requirements are mitigated by the use of simple streaming methods such as SGD, HB, NAG, ASGD. For other distinctions between the offline and online stochastic methods, refer to, Frostig et al. (2015b).

**Practical methods for training deep networks**: Momentum based methods employed with stochastic gradients (Sutskever et al., 2013) have become standard and very popular in practice. These schemes tend to outperform standard SGD on several important practical problems. As previously mentioned, we attribute this improvement to effect of minibatching rather than improvement by HB or NAG when working with stochastic gradients. Other schemes such as Adagrad (Duchi et al., 2011), RMSProp (Tieleman & Hinton, 2012), Adam (Kingma & Ba, 2014) represent an important and useful class of algorithms. The advantages offered by these methods are orthogonal to the advantages offered by fast gradient methods; it is an important direction to explore augmenting these methods with ASGD.

Recently, Chaudhari et al. (2017) proposed Entropy-SGD, a variant of SGD that adds a local strong convexity term to the objective function to improve generalization. However, currently we do not understand convergence rates or generalization performance of the technique rigorously, even for convex problems. The paper proposes to use SGD to optimize the altered objective function and mentions that one can use the HB method as well. Naturally, we can use the ASGD method as well to optimize the proposed objective function in the paper.

## 7 Conclusions and Future Directions

In this paper, we show that the performance gain of HB over SGD in stochastic setting is attributable to minibatching rather than the algorithm's ability to *accelerate* with stochastic gradients. Concretely, we provide a formal proof that for several *easy* problem instances, HB does not outperform SGD despite large condition number of the problem; we observe this trend for NAG in our experiments. In contrast, ASGD (Jain et al., 2017) provides significant improvement over SGD for the same problem instances. We observe similar trend when training a resnet on cifar-10 and an autoencoder on mnist. This work motivates several directions such as understanding the behavior of ASGD on other domains such as NLP, combining ASGD with adagrad (Duchi et al., 2011), adam (Kingma & Ba, 2014); note that ASGD can be combined with schemes such as path normalized SGD (Neyshabur et al., 2015), entropic SGD (Chaudhari et al., 2017) (both of which are indicated in their papers) and possibly developing automatic tuning schemes similar to Yellowfin (Zhang et al., 2017).

## REFERENCES


Preresnet-44 for cifar-10. https://github.com/D-X-Y/ResNeXt-DenseNet. Accessed: 2017-10-25.

Pytorch. https://github.com/pytorch. Accessed: 2017-10-25.

Zeyuan Allen-Zhu. Katyusha: The first direct acceleration of stochastic gradient methods. *CoRR*, abs/1603.05953, 2016.

Léon Bottou and Olivier Bousquet. The tradeoffs of large scale learning. In *NIPS 20*, 2007.

Louis Augustin Cauchy. Méthode générale pour la résolution des systémes d'équations simultanees. *C. R. Acad. Sci. Paris*, 1847.

Pratik Chaudhari, Anna Choromanska, Stefano Soatto, Yann LeCun, Carlo Baldassi, Christian Borgs, Jennifer Chayes, Levent Sagun, and Riccardo Zecchina. Entropy-sgd: Biasing gradient descent into wide valleys. *CoRR*, abs/1611.01838, 2017.

Aaron Defazio. A simple practical accelerated method for finite sums. *Advances in Neural Information Processing Systems 29 (NIPS 2016)*, 2016.

Aaron Defazio, Francis R. Bach, and Simon Lacoste-Julien. SAGA: A fast incremental gradient method with support for non-strongly convex composite objectives. In *NIPS 27*, 2014.

Olivier Devolder, Franccois Glineur, and Yurii E. Nesterov. First-order methods of smooth convex optimization with inexact oracle. *Mathematical Programming*, 146:37–75, 2014.

Aymeric Dieuleveut, Nicolas Flammarion, and Francis R. Bach. Harder, better, faster, stronger convergence rates for least-squares regression. *CoRR*, abs/1602.05419, 2016.

John C. Duchi, Elad Hazan, and Yoram Singer. Adaptive subgradient methods for online learning and stochastic optimization. *Journal of Machine Learning Research*, 12:2121–2159, 2011.

Roy Frostig, Rong Ge, Sham Kakade, and Aaron Sidford. Un-regularizing: approximate proximal point and faster stochastic algorithms for empirical risk minimization. In *ICML*, 2015a.

Roy Frostig, Rong Ge, Sham M. Kakade, and Aaron Sidford. Competing with the empirical risk minimizer in a single pass. In *COLT*, 2015b.

Saeed Ghadimi and Guanghui Lan. Optimal stochastic approximation algorithms for strongly convex stochastic composite optimization i: A generic algorithmic framework. *SIAM Journal on Optimization*, 2012.

Saeed Ghadimi and Guanghui Lan. Optimal stochastic approximation algorithms for strongly convex stochastic composite optimization, ii: shrinking procedures and optimal algorithms. *SIAM Journal on Optimization*, 2013.

Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Identity mappings in deep residual networks. In *ECCV (4)*, Lecture Notes in Computer Science, pp. 630–645. Springer, 2016a.

Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Deep residual learning for image recognition. In *CVPR*, pp. 770–778, 2016b.

Geoffrey E Hinton and Ruslan R Salakhutdinov. Reducing the dimensionality of data with neural networks. *science*, 313(5786):504–507, 2006.

Prateek Jain, Sham M Kakade, Rahul Kidambi, Praneeth Netrapalli, and Aaron Sidford. Parallelizing stochastic approximation through mini-batching and tail-averaging. *arXiv preprint arXiv:1610.03774*, 2016.

Prateek Jain, Sham M Kakade, Rahul Kidambi, Praneeth Netrapalli, and Aaron Sidford. Accelerating stochastic gradient descent. *arXiv preprint arXiv:1704.08227*, 2017.

Rie Johnson and Tong Zhang. Accelerating stochastic gradient descent using predictive variance reduction. In *NIPS 26*, 2013.

Nitish Shirish Keskar, Dheevatsa Mudigere, Jorge Nocedal, Mikhail Smelyanskiy, and Ping Tak Peter Tang. On large-batch training for deep learning: Generalization gap and sharp minima. *CoRR*, abs/1609.04836, 2016.

Diederik P. Kingma and Jimmy Ba. Adam: A method for stochastic optimization. *CoRR*, abs/1412.6980, 2014.

Alex Krizhevsky and Geoffrey Hinton. Learning multiple layers of features from tiny images. 2009.

Hongzhou Lin, Julien Mairal, and Zaïd Harchaoui. A universal catalyst for first-order optimization. In *NIPS*, 2015.

James Martens. Deep learning via hessian-free optimization. In *International conference on machine learning*, 2010.

James Martens and Roger Grosse. Optimizing neural networks with kronecker-factored approximate curvature. In *International conference on machine learning*, 2015.

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.

Yurii Nesterov. Gradient methods for minimizing composite functions. *Mathematical Programming Series B*, 2012a.

Yurii E. Nesterov. *Introductory lectures on convex optimization: A basic course*, volume 87 of *Applied Optimization*. Kluwer Academic Publishers, 2004.

Yurii E. Nesterov. Efficiency of coordinate descent methods on huge-scale optimization problems. *SIAM Journal on Optimization*, 22(2):341–362, 2012b.

Behnam Neyshabur, Ruslan Salakhutdinov, and Nathan Srebro. Path-sgd: Path-normalized optimization in deep neural networks. *CoRR*, abs/1506.02617, 2015.

Boris T Polyak. Some methods of speeding up the convergence of iteration methods. *USSR Computational Mathematics and Mathematical Physics*, 4(5):1–17, 1964.

Boris T. Polyak. *Introduction to Optimization*. Optimization Software, 1987.

John G. Proakis. Channel identification for high speed digital communications. *IEEE Transactions on Automatic Control*, 1974.

Sashank Reddi, Manzil Zaheer, Suvrit Sra, Barnabas Poczos, Francis Bach, Ruslan Salakhutdinov, and Alexander Smola. A generic approach for escaping saddle points. *arXiv preprint arXiv:1709.01434*, 2017.

Herbert Robbins and Sutton Monro. A stochastic approximation method. *The Annals of Mathematical Statistics*, vol. 22, 1951.

Nicolas Le Roux, Mark Schmidt, and Francis R. Bach. A stochastic gradient method with an exponential convergence rate for strongly-convex optimization with finite training sets. In *NIPS 25*, 2012.

Sumit Roy and John J. Shynk. Analysis of the momentum lms algorithm. *IEEE Transactions on Acoustics, Speech and Signal Processing*, 1990.

Shai Shalev-Shwartz and Tong Zhang. Stochastic dual coordinate ascent methods for regularized loss minimization. *CoRR*, abs/1209.1873, 2012.

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.

Tijmen Tieleman and Geoffrey Hinton. Lecture 6.5-rmsprop: Divide the gradient by a running average of its recent magnitude. *COURSERA: Neural networks for machine learning*, 2012.

Jian Zhang, Ioannis Mitliagkas, and Christopher R. Yellowfin and the art of momentum tuning. *CoRR*, abs/1706.03471, 2017.

# A SUBOPTIMALITY OF HB: PROOF OF PROPOSITION 3

Before proceeding to the proof, we introduce some additional notation. Let $\boldsymbol{\theta}_{t+1}^{(j)}$ denote the concatenated and centered estimates in the $j^{\text{th}}$ direction for $j = 1, 2$.

$$\boldsymbol{\theta}_{t+1}^{(j)} \stackrel{\text{def}}{=} \begin{bmatrix} \mathbf{w}_{t+1}^{(j)} - (w^*)^{(j)} \\ \mathbf{w}_{t}^{(j)} - (w^*)^{(j)} \end{bmatrix}, \quad j = 1, 2.$$

Since the distribution over $x$ is such that the coordinates are decoupled, we see that $\boldsymbol{\theta}_{t+1}^{(j)}$ can be written in terms of $\boldsymbol{\theta}_{t}^{(j)}$ as:

$$\boldsymbol{\theta}_{t+1}^{(j)} = \widehat{\mathbf{A}}_{t+1}^{(j)} \boldsymbol{\theta}_{t}^{(j)}, \text{ with } \widehat{\mathbf{A}}_{t+1}^{(j)} = \begin{bmatrix} 1 + \alpha - \delta(x_{t+1}^{(j)})^2 & -\alpha \\ 1 & 0 \end{bmatrix}.$$

Let $\boldsymbol{\Phi}_{t+1}^{(j)} \stackrel{\text{def}}{=} \mathbb{E}\left[\boldsymbol{\theta}_{t+1}^{(j)} \otimes \boldsymbol{\theta}_{t+1}^{(j)}\right]$ denote the covariance matrix of $\boldsymbol{\theta}_{t+1}^{(j)}$. We have $\boldsymbol{\Phi}_{t+1}^{(j)} = \mathcal{B}^{(j)} \boldsymbol{\Phi}_{t}^{(j)}$ with, $\mathcal{B}^{(j)}$ defined as

$$\mathcal{B}^{(j)} \stackrel{\text{def}}{=} \begin{bmatrix} \mathbb{E}\left[(1+\alpha-\delta(x^{(j)})^2)^2\right] & \mathbb{E}\left[-\alpha(1+\alpha-\delta(x^{(j)})^2)\right] & \mathbb{E}\left[-\alpha(1+\alpha-\delta(x^{(j)})^2\right] & \alpha^2 \\ \mathbb{E}\left[(1+\alpha-\delta(x^{(j)})^2)\right] & 0 & -\alpha & 0 \\ \mathbb{E}\left[(1+\alpha-\delta(x^{(j)})^2)\right] & -\alpha & 0 & 0 \\ 1 & 0 & 0 & 0 \end{bmatrix}$$
$$= \begin{bmatrix} (1+\alpha-\delta\sigma_j^2)^2 + (c-1)(\delta\sigma_j^2)^2 & -\alpha(1+\alpha-\delta\sigma_j^2) & -\alpha(1+\alpha-\delta\sigma_j^2) & \alpha^2 \\ (1+\alpha-\delta\sigma_j^2) & 0 & -\alpha & 0 \\ (1+\alpha-\delta\sigma_j^2) & -\alpha & 0 & 0 \\ 1 & 0 & 0 & 0 \end{bmatrix}.$$

We prove Proposition 3 by showing that for any choice of stepsize and momentum, either of the two holds:

- $\mathcal{B}^{(1)}$ has an eigenvalue larger than 1, or,
- the largest eigenvalue of $\mathcal{B}^{(2)}$ is greater than $1 - \frac{500}{\kappa}$.

This is formalized in the following two lemmas.

**Lemma 4.** *If the stepsize $\delta$ is such that $\delta\sigma_1^2 \geq \frac{2(1-\alpha^2)}{c+(c-2)\alpha}$, then $\mathcal{B}^{(1)}$ has an eigenvalue $\geq 1$.*

**Lemma 5.** *If the stepsize $\delta$ is such that $\delta\sigma_1^2 < \frac{2(1-\alpha^2)}{c+(c-2)\alpha}$, then $\mathcal{B}^{(2)}$ has an eigenvalue of magnitude $\geq 1 - \frac{500}{\kappa}$.*

Given this notation, we can now consider the $j^{th}$ dimension without the superscripts; when needed, they will be made clear in the exposition. Denoting $x \stackrel{\text{def}}{=} \delta\sigma^2$ and $t \stackrel{\text{def}}{=} 1 + \alpha - x$, we have:

$$\mathcal{B} = \begin{bmatrix} t^2 + (c-1)x^2 & -\alpha t & -\alpha t & \alpha^2 \\ t & 0 & -\alpha & 0 \\ t & -\alpha & 0 & 0 \\ 1 & 0 & 0 & 0 \end{bmatrix}$$

## A.1 PROOF

The analysis goes via computation of the characteristic polynomial of $\mathcal{B}$ and evaluating at different values to obtain bounds on its roots.

**Lemma 6.** *The characteristic polynomial of $\mathcal{B}$ is:*

$$D(z) = z^4 - (t^2 + (c-1)x^2)z^3 + (2\alpha t^2 - 2\alpha^2)z^2 + (-t^2 + (c-1)x^2)\alpha^2 z + \alpha^4.$$

*Proof.* We first begin by writing out the expression for the determinant:

$$Det(\mathcal{B} - z\mathcal{I}) = \begin{vmatrix} t^2 + (c-1)x^2 - z & -\alpha t & -\alpha t & \alpha^2 \\ t & -z & -\alpha & 0 \\ t & -\alpha & -z & 0 \\ 1 & 0 & 0 & -z \end{vmatrix}.$$

expanding along the first column, we have:

$$\begin{aligned} Det(\mathcal{B} - z\mathcal{I}) &= (t^2 + (c-1)x^2 - z)(\alpha^2 z - z^3) - t(-\alpha t z^2 + \alpha^2 t z) + t(-\alpha t(\alpha z) + z \cdot \alpha t z) - (z \cdot \alpha^2 z - \alpha^4) \\ &= (t^2 + (c-1)x^2 - z)(\alpha^2 z - z^3) - 2t(\alpha^2 t z - \alpha t z^2) - (\alpha^2 z^2 - \alpha^4). \end{aligned}$$

Expanding the terms yields the expression in the lemma. □

The next corollary follows by some simple arithmetic manipulations.

**Corollary 7.** *Substituting $z = 1 - \tau$ in the characteristic equation of Lemma 6, we have:*

$$\begin{aligned} D(1-\tau) &= \tau^4 + \tau^3(-4 + t^2 + (c-1)x^2) + \tau^2(6 - 3t^2 - 3(c-1)x^2 - 2\alpha^2 + 2\alpha t^2) \\ &+ \tau(-4 + 3t^2 + 3(c-1)x^2 + 4\alpha^2 - 4\alpha t^2 - (c-1)x^2\alpha^2 + t^2\alpha^2) \\ &+ (1 - t^2 - (c-1)x^2 - 2\alpha^2 + 2\alpha t^2 + (c-1)x^2\alpha^2 - t^2\alpha^2 + \alpha^4) \\ &= \tau^4 + \tau^3[-(3+\alpha)(1-\alpha) - 2x(1+\alpha) + cx^2] \\ &+ \tau^2[(3 - 4\alpha - \alpha^2 + 2\alpha^3) - 2x(1+\alpha)(2\alpha - 3) + x^2(2\alpha - 3c)] \\ &+ \tau[-(1-\alpha)^2(1-\alpha^2) - 2x(3-\alpha)(1-\alpha^2) + x^2(3c - 4\alpha + (2-c)\alpha^2)] \\ &+ x(1-\alpha)[2(1-\alpha^2) - x(c + (c-2)\alpha)]. \end{aligned} \tag{2}$$

*Proof of Lemma 4.* The first observation necessary to prove the lemma is that the characteristic polynomial $D(z)$ approaches $\infty$ as $z \to \infty$, i.e., $\lim_{z\to\infty} D(z) = +\infty$.

Next, we evaluate the characteristic polynomial at 1, i.e. compute $D(1)$. This follows in a straightforward manner from corollary (7) by substituting $\tau = 0$ in equation (2), and this yields,

$$D(1) = (1-\alpha)x \cdot \Big(2(1-\alpha^2) - x(1-\alpha) - (c-1)x(1+\alpha)\Big).$$

As $\alpha < 1$, $x = \delta\sigma^2 > 0$, we have the following by setting $D(1) \leq 0$ and solving for $x$:

$$x \geq \frac{2(1-\alpha^2)}{c + (c-2)\alpha}.$$

Since $D(1) \leq 0$ and $D(z) \geq 0$ as $z \to \infty$, there exists a root of $D(\cdot)$ which is $\geq 1$. □

**Remark 8.** *The above characterization is striking in the sense that for any $c > 1$, increasing the momentum parameter $\alpha$ naturally requires the reduction in the step size $\delta$ to permit the convergence of the algorithm, which is not observed when fast gradient methods are employed in deterministic optimization. For instance, in the case of deterministic optimization, setting $c = 1$ yields $\delta\sigma_1^2 < 2(1+\alpha)$. On the other hand, when employing the stochastic heavy ball method with $x^{(j)} = 2\sigma_j^2$, we have the condition that $c = 2$, and this implies, $\delta\sigma_1^2 < \frac{2(1-\alpha^2)}{2} = 1 - \alpha^2$.*

We now prove Lemma 5. We first consider the large momentum setting.

**Lemma 9.** *When the momentum parameter $\alpha$ is set such that $1 - 450/\kappa \leq \alpha \leq 1$, $\mathcal{B}$ has an eigenvalue of magnitude $\geq 1 - \frac{450}{\kappa}$.*

*Proof.* This follows easily from the fact that $\det(\mathcal{B}) = \alpha^4 = \prod_{j=1}^4 \lambda_j(\mathcal{B}) \leq (\lambda_{\max}(\mathcal{B}))^4$, thus implying $1 - 450/\kappa \leq \alpha \leq |\lambda_{\max}(\mathcal{B})|$. □

**Remark 10.** *Note that the above lemma holds for any value of the learning rate $\delta$, and holds for every eigen direction of $\mathbf{H}$. Thus, for "large" values of momentum, the behavior of stochastic heavy ball does degenerate to the behavior of stochastic gradient descent.*

We now consider the setting where momentum is bounded away from 1.

**Corollary 11.** *Consider $\mathcal{B}^{(2)}$, by substituting $\tau = l/\kappa$, $x = \delta\lambda_{\min} = c(\delta\sigma_1^2)/\kappa$ in equation (2) and accumulating terms in varying powers of $1/\kappa$, we obtain:*

$$
\begin{aligned}
G(l) \stackrel{def}{=} & \frac{c^3(\delta\sigma_1^2)^2 l^3}{\kappa^5} + \frac{l^4 - 2c(\delta\sigma_1^2)l^3(1+\alpha) + (2\alpha - 3c)c^2(\delta\sigma_1^2)^2 l^2}{\kappa^4} \\
& + \frac{-(3+\alpha)(1-\alpha)l^3 - 2(1+\alpha)(2\alpha-3)c(\delta\sigma_1^2)l^2 + (3c - 4\alpha + (2-c)\alpha^2)c^2(\delta\sigma_1^2)^2 l}{\kappa^3} \\
& + \frac{(3 - 4\alpha - \alpha^2 + 2\alpha^3)l^2 - 2c(\delta\sigma_1^2)l(3-\alpha)(1-\alpha^2) - c^2(\delta\sigma_1^2)^2(1-\alpha)(c + (c-2)\alpha)}{\kappa^2} \\
& + \frac{-(1-\alpha)^2(1-\alpha^2)l + 2c(\delta\sigma_1^2)(1-\alpha)(1-\alpha^2)}{\kappa}
\end{aligned} \tag{3}
$$

**Lemma 12.** *Let $2 < c < 3000$, $0 \le \alpha \le 1 - \frac{450}{\kappa}$, $l = 1 + \frac{2c(\delta\sigma_1^2)}{1-\alpha}$. Then, $G(l) \le 0$.*

*Proof.* Since $(\delta\sigma_1^2) \le \frac{2(1-\alpha^2)}{c+(c-2)\alpha}$, this implies $\frac{(\delta\sigma_1^2)}{1-\alpha} \le \frac{2(1+\alpha)}{c+(c-2)\alpha} \le \frac{4}{c}$, thus implying, $1 \le l \le 9$.

Substituting the value of $l$ in equation (3), the coefficient of $\mathcal{O}(1/\kappa)$ is $-(1-\alpha)^3(1+\alpha)$.

We will bound this term along with $(3 - 4\alpha - \alpha^2 + 2\alpha^3)l^2/\kappa^2 = (1-\alpha)^2(3+2\alpha)l^2/\kappa^2$ to obtain:

$$
\begin{aligned}
\frac{-(1-\alpha)^3(1+\alpha)}{\kappa} + \frac{(1-\alpha)^2(3+2\alpha)l^2}{\kappa^2} &\le \frac{-(1-\alpha)^3(1+\alpha)}{\kappa} + \frac{405(1-\alpha)^2}{\kappa^2} \\
&\le \frac{(1-\alpha)^2}{\kappa}\left(\frac{405}{\kappa} - (1-\alpha^2)\right) \\
&\le \frac{(1-\alpha)^2}{\kappa}\left(\frac{405}{\kappa} - (1-\alpha)\right) \le -\frac{45 \cdot 450^2}{\kappa^4},
\end{aligned}
$$

where, we use the fact that $\alpha < 1$, $l \le 9$. The natural implication of this bound is that the terms that are lower order, such as $\mathcal{O}(1/\kappa^4)$ and $\mathcal{O}(1/\kappa^5)$ will be negative owing to the large constant above. Let us verify that this is indeed the case by considering the terms having powers of $\mathcal{O}(1/\kappa^4)$ and $\mathcal{O}(1/\kappa^5)$ from equation (3):

$$
\begin{aligned}
& \frac{c^3(\delta\sigma_1^2)^2 l^3}{\kappa^5} + \frac{l^4 - 2c(\delta\sigma_1^2)l^3(1+\alpha) + (2\alpha - 3c)c^2(\delta\sigma_1^2)^2 l^2}{\kappa^4} - \frac{45 \cdot 450^2}{\kappa^4} \\
& \le \frac{c^3(\delta\sigma_1^2)^2 l^3}{\kappa^5} + \frac{l^4}{\kappa^4} - \frac{45 \cdot 450^2}{\kappa^4} \\
& \le \frac{cl^3}{\kappa^5} + \frac{(9^4 - (45 \cdot 450^2))}{\kappa^4} \le \frac{9^3 c + 9^4 - (45 \cdot 450^2)}{\kappa^4}
\end{aligned}
$$

The expression above evaluates to $\le 0$ given an upperbound on the value of $c$. The expression above follows from the fact that $l \le 9, \kappa \ge 1$.

Next, consider the terms involving $\mathcal{O}(1/\kappa^3)$ and $\mathcal{O}(1/\kappa^2)$, in particular,

$$
\begin{aligned}
& \frac{(3c - 4\alpha + (2-c)\alpha^2)c^2(\delta\sigma_1^2)^2 l}{\kappa^3} - \frac{c^2(\delta\sigma_1^2)^2(1-\alpha)(c + (c-2)\alpha)}{\kappa^2} \\
& \le \frac{c^2(\delta\sigma_1^2)^2}{\kappa^2}\big(\frac{l(3c+2)}{\kappa} - (1-\alpha)(c + (c-2)\alpha)\big) \\
& \le \frac{c^2(\delta\sigma_1^2)^2}{\kappa^2}\big(\frac{5cl}{\kappa} - (1-\alpha)(c + (c-2)\alpha)\big) \\
& \le \frac{c^2(\delta\sigma_1^2)^2}{\kappa^2}\big(\frac{5cl}{\kappa} - (1-\alpha)c\big) \\
& \le \frac{c^3(\delta\sigma_1^2)^2}{\kappa^2}\big(\frac{5l}{\kappa} - \frac{450}{\kappa}\big) \\
& \le \frac{c^3(\delta\sigma_1^2)^2}{\kappa^2} \cdot \frac{-405}{\kappa} \le 0.
\end{aligned}
$$

Next,

$$\begin{aligned}
&\frac{-2(1+\alpha)(2\alpha-3)c(\delta\sigma_1^2)l^2}{\kappa^3} - \frac{2c(\delta\sigma_1^2)l(3-\alpha)(1-\alpha^2)}{\kappa^2} \\
&\leq \frac{2(1+\alpha)c(\delta\sigma_1^2)l}{\kappa^2}\left(\frac{-(2\alpha-3)l}{\kappa} - (3-\alpha)(1-\alpha)\right) \\
&\leq \frac{2(1+\alpha)c(\delta\sigma_1^2)l}{\kappa^2}\left(\frac{3l}{\kappa} - 2(1-\alpha)\right) \\
&\leq \frac{2(1+\alpha)c(\delta\sigma_1^2)l}{\kappa^2}\left(\frac{3l}{\kappa} - \frac{2\cdot 450}{\kappa}\right) \\
&\leq \frac{2(1+\alpha)c(\delta\sigma_1^2)l}{\kappa^2}\left(\frac{3\cdot 27}{\kappa} - \frac{2\cdot 450}{\kappa}\right) \leq 0.
\end{aligned}$$

In both these cases, we used the fact that $\alpha \leq 1 - \frac{450}{\kappa}$ implying $-(1-\alpha) \leq \frac{-450}{\kappa}$. Finally, other remaining terms are negative. □

We are now ready to prove Lemma 5.

*Proof of Lemma 5.* Combining Lemmas 9 and 12, we see that no matter what stepsize and momentum we choose, $\mathcal{B}^{(j)}$ has an eigenvalue of magnitude at least $1 - \frac{500}{\kappa}$ for some $j \in \{1, 2\}$. This proves the lemma. □

## B EQUIVALENCE OF ALGORITHM 3 AND ASGD

We begin by writing out the updates of ASGD as written out in Jain et al. (2017), which starts with two iterates $a_0$ and $d_0$, and from time $t = 0, 1, ...T-1$ implements the following updates:

$$b_t = \alpha_1 a_t + (1-\alpha_1)d_t \tag{4}$$

$$a_{t+1} = b_t - \delta_1 \hat{\nabla} f_{t+1}(b_t) \tag{5}$$

$$c_t = \beta_1 b_t + (1-\beta_1)d_t \tag{6}$$

$$d_{t+1} = c_t - \gamma_1 \hat{\nabla} f_{t+1}(b_t). \tag{7}$$

Next, we specify the step sizes $\beta_1 = c_3^2/\sqrt{\kappa\tilde{\kappa}}$, $\alpha_1 = c_3/(c_3+\beta)$, $\gamma_1 = \beta/(c_3\lambda_{\min})$ and $\delta_1 = 1/R^2$, where $\kappa = R^2/\lambda_{\min}$. Note that the step sizes in the paper of Jain et al. (2017) with $c_1$ in their paper set to 1 yields the step sizes above. Now, substituting equation 6 in equation 7 and substituting the value of $\gamma_1$, we have:

$$\begin{aligned}
d_{t+1} &= \beta_1\left(b_t - \frac{1}{c_3\lambda_{\min}}\hat{\nabla} f_{t+1}(b_t)\right) + (1-\beta_1)d_t \\
&= \beta_1\left(b_t - \frac{\delta\kappa}{c_3}\hat{\nabla} f_{t+1}(b_t)\right) + (1-\beta_1)d_t.
\end{aligned} \tag{8}$$

We see that $d_{t+1}$ is precisely the update of the running average $\bar{w}_{t+1}$ in the ASGD method employed in this paper.

We now update $b_t$ to become $b_{t+1}$ and this can be done by writing out equation 4 at $t+1$, i.e:

$$\begin{aligned}
b_{t+1} &= \alpha_1 a_{t+1} + (1-\alpha_1)d_{t+1} \\
&= \alpha_1\left(b_t - \delta_1\hat{\nabla} f_{t+1}(b_t)\right) + (1-\alpha_1)d_{t+1}.
\end{aligned} \tag{9}$$

By substituting the value of $\alpha_1$ we note that this is indeed the update of the iterate as a convex combination of the current running average and a short gradient step as written in this paper. In this paper, we set $c_3$ to be equal to 0.7, and any constant less than 1 works. In terms of variables, we note that $\alpha$ in this paper's algorithm description maps to $1-\beta_1$.

# C MORE DETAILS ON EXPERIMENTS

In this section, we will present more details on our experimental setup.

## C.1 LINEAR REGRESSION

In this section, we will present some more results on our experiments on the linear regression problem. Just as in Appendix A, it is indeed possible to compute the expected error of all the algorithms among SGD, HB, NAG and ASGD, by tracking certain covariance matrices which evolve as linear systems. For SGD, for instance, denoting $\mathbf{\Phi}_t^{SGD} \stackrel{\text{def}}{=} \mathbb{E}\left[\left(\mathbf{w}_t^{SGD} - w^*\right) \otimes \left(\mathbf{w}_t^{SGD} - w^*\right)\right]$, we see that $\mathbf{\Phi}_{t+1}^{SGD} = \mathcal{B} \circ \mathbf{\Phi}_t^{SGD}$, where $\mathcal{B}$ is a linear operator acting on $d \times d$ matrices such that $\mathcal{B} \circ M \stackrel{\text{def}}{=} M - \delta HM - \delta MH + \delta^2 \mathbb{E}\left[\langle x, Mx\rangle\, xx^\top\right]$. Similarly, HB, NAG and ASGD also have corresponding operators (see Appendix A for more details on the operator corresponding to HB). The largest magnitude of the eigenvalues of these matrices indicate the rate of decay achieved by the particular algorithm – smaller it is compared to 1, faster the decay.

We now detail the range of parameters explored for these results: the condition number $\kappa$ was varied from $\{2^4, 2^5, .., 2^{28}\}$ for all the optimization methods and for both the discrete and gaussian problem. For each of these experiments, we draw $1000$ samples and compute the empirical estimate of the fourth moment tensor. For NAG and HB, we did a very fine grid search by sampling $50$ values in the interval $(0, 1]$ for both the learning rate and the momentum parameter and chose the parameter setting that yielded the smallest $\lambda_{\max}(\mathcal{B})$ that is less than $1$ (so that it falls in the range of convergence of the algorithm). As for SGD and ASGD, we employed a learning rate of $1/3$ for the Gaussian case and a step size of $0.9$ for the discrete case. The statistical advantage parameter of ASGD was chosen to be $\sqrt{3\kappa/2}$ for the Gaussian case and $\sqrt{2\kappa/3}$ for the Discrete case, and the a long step parameters of $3\kappa$ and $2\kappa$ were chosen for the Gaussian and Discrete case respectively. The reason it appears as if we choose a parameter above the theoretically maximal allowed value of the advantage parameter is because the definition of $\kappa$ is different in this case. The $\kappa$ we speak about for this experiment is $\lambda_{\max}/\lambda_{\min}$ unlike the condition number for the stochastic optimization problem. In a manner similar to actually running the algorithms (the results of whose are presented in the main paper), we also note that we can compute the rate as in equation 1 and join all these rates using a curve and estimate its slope (in the $\log$ scale). This result is indicated in table 3.

Figure 7 presents these results, where for each method, we did grid search over all parameters and chose parameters that give smallest $\lambda_{\max}$. We see the same pattern as in Figure 1 from actual runs – SGD,HB and NAG all have linear dependence on condition number $\kappa$, while ASGD has a dependence of $\sqrt{\kappa}$.

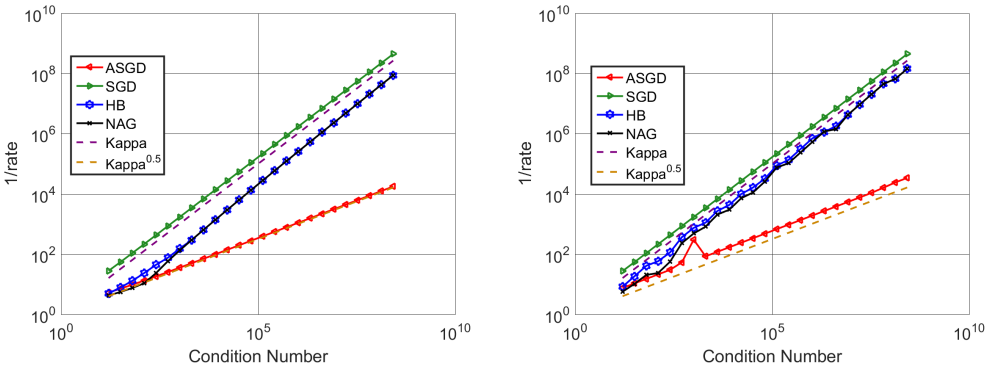


Figure 7: Expected rate of error decay (equation 1) vs condition number for various methods for the linear regression problem. Left is for discrete distribution and right is for Gaussian distribution.

| Algorithm | Slope – discrete | Slope – Gaussian |
|---|---|---|
| SGD | 0.9990 | 0.9995 |
| HB | 1.0340 | 0.9989 |
| NAG | 1.0627 | 1.0416 |
| ASGD | 0.4923 | 0.4906 |

Table 3: Slopes (i.e. $\gamma$) obtained by fitting a line to the curves in Figure 7. A value of $\gamma$ indicates that the error decays at a rate of $\exp\left(\frac{-t}{\kappa^{\gamma}}\right)$. A smaller value of $\gamma$ indicates a faster rate of error decay.

### C.2 AUTOENCODERS FOR MNIST

We begin by noting that the learning rates tend to vary as we vary batch sizes, which is something that is known in theory (Jain et al., 2016). Furthermore, we extend the grid especially whenever our best parameters of a baseline method tends to land at the edge of a grid. The parameter ranges explored by our grid search are:

**Batch Size** 1: (parameters chosen by running for 20 epochs)

- SGD: learning rate: $\{0.01, 0.01\sqrt{10}, 0.1, 0.1\sqrt{10}, 1, \sqrt{10}, 5, 10, 20, 10\sqrt{10}, 40, 60, 80, 100.$
- NAG/HB: learning rate: $\{0.01\sqrt{10}, 0.1, 0.1\sqrt{10}, 1, \sqrt{10}, 10\}$, momentum $\{0, 0.5, 0.75, 0.9, 0.95, 0.97\}$.
- ASGD: learning rate: $\{2.5, 5\}$, long step $\{100.0, 1000.0\}$, advantage parameter $\{2.5, 5.0, 10.0, 20.0\}$.

**Batch Size** 8: (parameters chosen by running for 50 epochs)

- SGD: learning rate: $\{0.001, 0.001\sqrt{10.0}, 0.01, 0.01\sqrt{10}, 0.1, 0.1\sqrt{10}, 1, \sqrt{10}, 5, 10$ $, 10\sqrt{10}, 40, 60, 80, 100, 120, 140\}$.
- NAG/HB: learning rate: $\{5.0, 10.0, 20.0, 10\sqrt{10}, 40, 60\}$, momentum $\{0, 0.25, 0.5, 0.75, 0.9, 0.95\}$.
- ASGD: learning rate $\{40, 60\}$. For a long step of 100, swept over advantage parameters of $\{1.5, 2, 2.5, 5, 10, 20\}$. For a long step of 1000, we swept over advantage parameters of $\{2.5, 5, 10\}$.

### C.3 DEEP RESIDUAL NETWORKS FOR CIFAR-10

In this section, we will provide more details on our experiments on cifar-10, as well as present some additional results. We used a weight decay of $0.0005$ in all our experiments. The grid search parameters we used for various algorithms are as follows. Note that the ranges in which parameters such as learning rate need to be searched differ based on batch size (Jain et al., 2016). Furthermore, we tend to extrapolate the grid search whenever a parameter (except for the learning rate decay factor) at the edge of the grid has been chosen; this is done so that we always tend to lie in the interior of the grid that we have searched on. Note that for the purposes of the grid search, we choose a hold out set from the training data and add it in to the training data after the parameters are chosen, for the final run.

**Batch Size** 8: Note: (i) parameters chosen by running for $40$ epochs and picking the grid search parameter that yields the smallest validation $0/1$ error. (ii) The validation set decay scheme that we use is that if the validation error does not decay by at least $1\%$ every three passes over the data, we cut the learning rate by a constant factor (which is grid searched as described below). The minimal learning rate to use is fixed to be $6.25 \times 10^{-5}$, so that we do not decay far too many times and curtail progress prematurely.

- SGD: learning rate: $\{0.0033, 0.01, 0.033, 0.1, 0.33\}$, learning rate decay factor $\{5, 10\}$.
- NAG/HB: learning rate: $\{0.001, 0.0033, 0.01, 0.033\}$, momentum $\{0.8, 0.9, 0.95, 0.97\}$, learning rate decay factor $\{5, 10\}$.

- ASGD: learning rate $\{0.01, 0.0330, 0.1\}$, long step $\{1000, 10000, 50000\}$, advantage parameter $\{5, 10\}$, learning rate decay factor $\{5, 10\}$.

**Batch Size** 128: Note: (i) parameters chosen by running for 120 epochs and picking the grid search parameter that yields the smallest validation $0/1$ error. (ii) The validation set decay scheme that we use is that if the validation error does not decay by at least $0.2\%$ every four passes over the data, we cut the learning rate by a constant factor (which is grid searched as described below). The minimal learning rate to use is fixed to be $1 \times 10^{-3}$, so that we do not decay far too many times and curtail progress prematurely.

- SGD: learning rate: $\{0.01, 0.03, 0.09, 0.27, 0.81\}$, learning rate decay factor $\{2, \sqrt{10}, 5\}$.
- NAG/HB: learning rate: $\{0.01, 0.03, 0.09, 0.27\}$, momentum $\{0.5, 0.8, 0.9, 0.95, 0.97\}$, learning rate decay factor $\{2, \sqrt{10}, 5\}$.
- ASGD: learning rate $\{0.01, 0.03, 0.09, 0.27\}$, long step $\{100, 1000, 10000\}$, advantage parameter $\{5, 10, 20\}$, learning rate decay factor $\{2, \sqrt{10}, 5\}$.

As a final remark, for any comparison across algorithms, such as, (i) ASGD vs. NAG, (ii) ASGD vs HB, we fix the starting learning rate, learning rate decay factor and decay schedule chosen by the best grid search run of NAG/HB respectively and perform a grid search over the long step and advantage parameter of ASGD. In a similar manner, when we compare (iii) SGD vs NAG or, (iv) SGD vs. HB, we choose the learning rate, learning rate decay factor and decay schedule of SGD and simply sweep over the momentum parameter of NAG or HB and choose the momentum that offers the best validation error.

We now present plots of training function value for different algorithms and batch sizes.

**Effect of minibatch sizes**: Figure 8 plots training function value for batch sizes of 128 and 8 for SGD, HB and NAG. We notice that in the initial stages of training, NAG obtains substantial improvements compared to SGD and HB for batch size 128 but not for batch size 8. Towards the end of training however, NAG starts decreasing the training function value rapidly for both the batch sizes. The reason for this phenomenon is not clear. Note however, that at this point, the test error has already stabilized and the algorithms are just overfitting to the data.

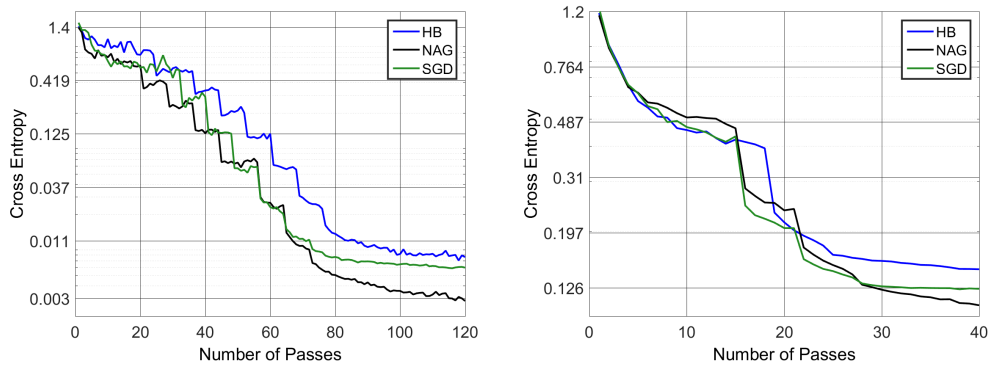


Figure 8: Training loss for batch sizes 128 and 8 respectively for SGD, HB and NAG.

**Comparison of ASGD with momentum methods**: We now present the training error plots for ASGD compared to HB and NAG in Figures 9 and 10 respectively. As mentioned earlier, in order to see a clear trend, we constrain the learning rate and decay schedule of ASGD to be the same as that of HB and NAG respectively, which themselves were learned using grid search. We see similar trends as in the validation error plots from Figures 5 and 6. Please see the figures and their captions for more details.

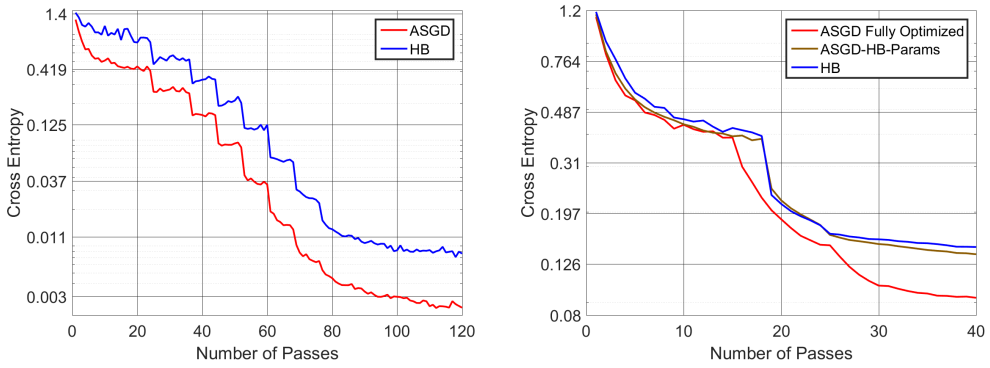


Figure 9: Training function value for ASGD compared to HB for batch sizes 128 and 8 respectively.

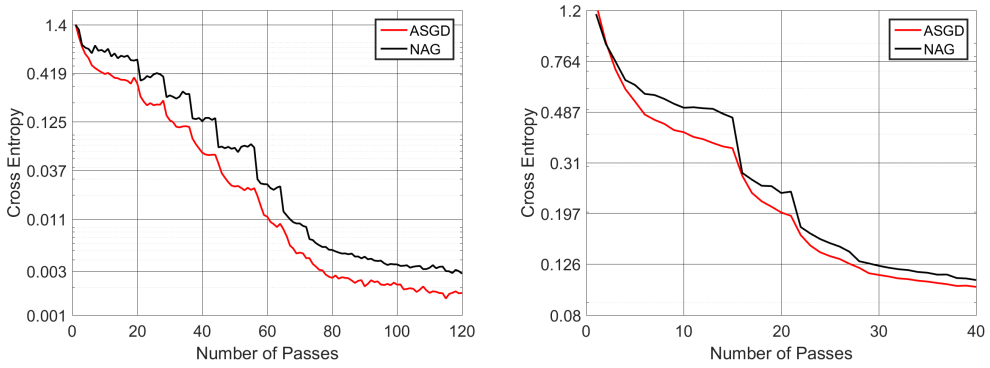


Figure 10: Training function value for ASGD compared to NAG for batch sizes 128 and 8 respectively.