STATISTICAL ADAPTIVE STOCHASTIC OPTIMIZATION

Anonymous authors
Paper under double-blind review

ABSTRACT

We investigate statistical methods for automatically scheduling the learning rate (step size) in stochastic optimization. First, we consider a broad family of stochastic optimization methods with constant hyperparameters (including the learning rate and various forms of momentum) and derive a general necessary condition for the resulting dynamics to be stationary. Based on this condition, we develop a simple online statistical test to detect (non-)stationarity and use it to automatically drop the learning rate by a constant factor whenever stationarity is detected. Unlike in prior work, our stationarity condition and statistical test apply to different algorithms without modification. Finally, we propose a smoothed stochastic line-search method that can be used to warm up the optimization process before the statistical test can be applied effectively. This removes the expensive trial and error for setting a good initial learning rate. The combined method is highly autonomous and it attains state-of-the-art training and testing performance in our experiments on several deep learning tasks.

1 INTRODUCTION

We study adaptive stochastic optimization methods in the context of large-scale machine learning. Specifically, we consider the stochastic optimization problem

\[
\min_{x \in \mathbb{R}^p} \quad F(x) \equiv \mathbb{E}_\xi [f_\xi(x)],
\]

where \(\xi\) is a random variable representing data sampled from some (unknown) probability distribution, \(x \in \mathbb{R}^p\) represents the parameters of the machine learning model (e.g., the weight matrices in a neural network), and \(f_\xi\) is the loss function associated with data \(\xi\). The objective function \(F\) is the expectation of \(f_\xi\) over the distribution of the data.

Many stochastic optimization methods for solving problem (1) can be written in the form of

\[
x_{k+1} = x_k - \alpha_k d_k,
\]

where \(d_k\) is a (stochastic) search direction and \(\alpha_k > 0\) is the step size or learning rate. In the classical stochastic gradient descent (SGD) method,

\[
d_k = g_k \equiv \nabla f_\xi_k(x_k),
\]

where \(\xi_k\) is a training example (or a mini-batch of examples) randomly generated at iteration \(k\). This method traces back to the seminal work of Robbins & Monro (1951), and it has become very popular in machine learning (e.g., Bottou [1998], Goodfellow et al. [2016]). Many modifications of SGD aim to improve its theoretical and practical performance. For example, Gupal & Bazhenov (1972) studied a stochastic analog of the heavy-ball method (Polyak [1964]), where

\[
d_k = (1 - \beta_k) g_k + \beta_k d_{k-1},
\]

and \(\beta_k \in [0, 1)\) is the momentum coefficient. Sutskever et al. [2013] proposed to use a stochastic variant of Nesterov’s accelerated gradient method [Nesterov [2004]], where

\[
d_k = \nabla f_\xi_k(x_k - \alpha_k \beta_k d_{k-1}) + \beta_k d_{k-1}.
\]

Other recent variants include, e.g., Jain et al. [2018] and Ma & Yarats [2019].

Theoretical conditions for the asymptotic convergence of SGD are well studied, and they mostly focus on polynomially decaying learning rates of the form \(\alpha_k = a/(b + k)^c\) for some \(a, b > 0\)
and $1/2 < c \leq 1$ (e.g., Robbins & Monro, 1951; Polyak & Juditsky, 1992). Similar conditions for the stochastic heavy-ball methods are also established (e.g., Gupal & Bazhenov, 1972; Polyak, 1977). However, these learning rate schedules still require significant hyperparameter tuning efforts in modern machine learning practice.

Adaptive rules for adjusting the learning rate and other parameters on the fly have been developed in both the stochastic optimization literature (e.g., Kesten, 1958; Mirzakhmedov & Uryasev, 1983; Ruszczyński & Syski, 1983; 1984; 1986a;b; Delyon & Juditsky, 1993) and by the machine learning community (e.g., Jacobs, 1988; Sutton, 1992; Schraudolph, 1999; Mahmood et al., 2012; Baydin et al., 2018). More recently, adaptive algorithms that use diagonal scaling—replacing $\alpha_k$ in (2) with an adjustable diagonal matrix—have become very popular (Duchi et al., 2011; Tieleman & Hinton, 2012; Kingma & Ba, 2014, e.g.). Despite these advances, costly hand-tuning efforts are still needed to obtain good (generalization) performance in practice (Wilson et al., 2017).

In this paper, we focus on statistical methods for automatically scheduling the learning rate. This line of work goes back to Kesten (1958), who used the change of sign of the inner products of consecutive stochastic gradients as a statistical indicator of slow progress and as a trigger to decrease the learning rate (see extensions in Delyon & Juditsky, 1993). Pflug (1983; 1988) considered the dynamics of SGD with constant step size for minimizing convex quadratic functions and derived necessary conditions for the resulting process to be stationary. Pflug (1983; 1990) also devised a sequential statistical test to detect stationarity; when the test fires, the step size is decreased by a constant factor. Ruszczyński & Syski (1983) used online statistical tests to check if the present step size and momentum constants satisfy certain optimality conditions and adjust them if the conditions are not satisfied. However, Lang et al. (2019) found that these methods have limited success in machine learning applications due to their reliance on quadratic approximation and/or strong assumptions on the noise models.

Most recently, Yaida (2018) derived fluctuation-dissipation relations (necessary conditions) that characterized the stationary behavior of stochastic gradient methods with constant learning rate and momentum. These relations hold exactly for any stationary state (regardless of the loss function or the noise model), so they can be very effective at detecting stationarity and can serve as a reliable indicator of when to decrease the learning rate. Lang et al. (2019) devised a more rigorous statistical test for Yaida’s conditions and obtained robust and competitive performance on common deep learning tasks.

Our contributions in this paper are threefold:

- We consider a broad family of stochastic optimization methods with constant hyperparameters (including the learning rate and various forms of momentum) and derive a more general necessary condition (than Yaida’s) for the associated learning process to be stationary. Yaida (2018) established a “master equation,” from which different stationarity conditions can be derived for different variants of stochastic gradient methods (some of them may not admit a usable analytical form). We derive a simple “master condition” that works for different methods without any change and holds exactly for any stationary process. Because of its conceptual and analytical simplicity, it greatly simplifies implementation and deployment in software packages.

- We develop a simple statistical test for checking stationarity based on our “master condition.” It is a simple confidence interval test (checking if it contains zero), as opposed to the “equivalence test” for Yaida’s condition used by Lang et al. (2019). This simple test is as robust as the equivalence test and avoids the use of an additional hyperparameter.

- A major challenge for almost all adaptive stochastic gradient methods is how to set the initial learning rate appropriately without expensive trial and error. We propose a smoothed stochastic line-search method that can be used to warm up the optimization process. Starting from any safe, small learning rate, this method automatically increases it to a suitably large value for fast initial convergence. Afterwards, the statistical test for stationarity can be applied to gradually reduce the learning rate and obtain finer convergence.

Combining the three components above, we obtain a highly autonomous algorithm called SALSA (Stochastic Approximation with Line-search and Statistical Adaptation). We conduct extensive experiments on several deep learning tasks to test its training and testing performance as well as its robustness to changes in the hyperparameter settings. SALSA matches the best performance of hand-tuned methods across different deep learning tasks using default hyperparameters.
2 Necessary Conditions for Stationarity

We consider general stochastic optimization methods in the form of (2) with a constant learning rate:

\[ x^{k+1} = x^k - \alpha d^k. \]  

(6)

We assume that the stochastic search direction \( d^k \) is generated with time-homogeneous dynamics. In particular, any additional hyperparameters involved must be constant (not depending on \( k \)). As an example, we consider the search directions generated by the family of Quasi-Hyperbolic Momentum (QHM) methods (Ma & Yarats [2019]):

\begin{align*}
    h^k &= (1 - \beta) g^k + \beta h^{k-1}, \\
    d^k &= (1 - \nu) g^k + \nu h^k,
\end{align*}

(7)

where \( 0 \leq \beta < 1 \) and \( 0 \leq \nu \leq 1 \). With \( \beta = 0 \) or \( \nu = 0 \), it recovers the SGD direction (5). With \( \nu = 1 \), it recovers the stochastic heavy-ball direction (4). With \( 0 < \beta = \nu < 1 \), it is equivalent to the direction with Nesterov momentum given in (5) (Sutskever et al. [2013]; Gitman et al. [2019]). We assume that the dynamics of (6), e.g., driven by the stochastic gradients \( g^k \) through (7), is stable. Stability regions of the hyperparameters in QHM are characterized by (Gitman et al. [2019]).

In addition, we assume that the stochastic process \( \{x^k\} \), as \( k \to \infty \), becomes stationary. Recall that \( \{x^k\} \) is (strongly) stationary if the joint distribution of any subset of the sequence is invariant with respect to simultaneous shifts in the time index (e.g., [Dembo 2013] Section 3.2.3). As a direct consequence, for any deterministic function \( \phi : \mathbb{R}^p \to \mathbb{R} \), we have

\[ \mathbb{E}_\pi [\phi(x^{k+1})] = \mathbb{E}_\pi [\phi(x^k)], \quad \forall k, \]  

(8)

where \( \pi \) denotes the stationary distribution of \( \{x^k\} \). If the test function \( \phi \) is smooth, then we use Taylor expansion to obtain

\[ \phi(x^{k+1}) = \phi(x^k - \alpha d^k) = \phi(x^k) - \alpha \nabla \phi(x^k, d^k) + \frac{\alpha^2}{2} \nabla^2 \phi(x^k)(d^k, d^k) + O(\alpha^3). \]

Taking expectations on both sides of the above equality and applying (8), we have

\[ \mathbb{E}_\pi [\nabla \phi(x^k, d^k) - \frac{\alpha}{2} \nabla^2 \phi(x^k)(d^k, d^k)] = O(\alpha^2). \]

(9)

For an arbitrary smooth function \( \phi \), it is very hard in practice to characterize or approximate the \( O(\alpha^2) \) term on the right-hand side. In addition, computing the Hessian-vector product \( \nabla^2 \phi(x^k)(d^k) \) can be very costly. Therefore, we only consider simple quadratic functions for which the \( O(\alpha^2) \) term in (9) vanishes. In particular, the choice of \( \phi(x) = (1/2)!\|x\|^2 \) results in

\[ \mathbb{E}_\pi [\langle x^k, d^k \rangle - \frac{\alpha}{2} \|d^k\|^2] = 0. \]

(10)

This condition holds exactly for any stochastic optimization method of the form (6), if it reaches stationarity. Indeed, weak stationarity is sufficient since \( \phi \) is a quadratic function of the state (e.g., [Dembo 2013] Section 3.2.3). Beyond stationarity, it requires no specific assumption on the loss function or noise model for the stochastic gradients. Moreover, it can be applied to different methods without any change. We call this the master condition for stationarity.

Yaida (2018) focused on state perturbation along \( g^k = \nabla f_{\xi_k}(x^k) \) to obtain his “master equation”

\[ \mathbb{E}_\pi [\phi(x^k - \alpha g^k)] = \mathbb{E}_\pi [\phi(x^k)], \quad \forall k. \]

This equation can be combined with the actual state update equation (6) to obtain more specific stationarity conditions for different algorithms. For example, for the stochastic heavy-ball method with \( d^k = (1 - \beta) g^k + \beta h^{k-1} \), the master equation leads to (Yaida 2018)

\[ \mathbb{E}_\pi [\langle x^k, g^k \rangle - \frac{\alpha}{2} \|d^k\|^2] = 0. \]

(11)

It is a simple exercise to show that this is equivalent to our master condition (10). For the QHM update (7), a more complex stationarity condition may also be derived, but it will still be equivalent to (10). In practice, the single, simple master condition (10) is much more preferred.

If \( \{x^k\} \) starts with a nonstationary distribution and converges to a stationary state, we have

\[ \lim_{k \to \infty} \mathbb{E} [\langle x^k, d^k \rangle - \frac{1}{2} \|d^k\|^2] = 0. \]

(12)

In the next section, we devise a simple statistical test to determine if the master condition fails to hold. In this case, the learning process has not stalled, and we can continue with the same step size. If we fail to detect non-stationarity (i.e., the dynamics may be approximately stationary), we would like to reduce the learning rate \( \alpha \) to allow for finer convergence.
Algorithm 1: SASA+: SASA with master condition and simple statistical testing

1. \textbf{input:} $x^0 \in \mathbb{R}^p$, $\alpha_0 > 0$, $N_{\text{min}} > 0$, $K_{\text{test}} > 0$, $\delta \in (0, 1)$, $\theta \in (0, 1)$, $\tau \in (0, 1)$
2. $\alpha \leftarrow \alpha_0$
3. $k_0 \leftarrow 0$
4. \textbf{for} $k = 0, \ldots, T - 1$ \textbf{do}
5. \quad Randomly sample $\zeta_k$ and compute stochastic search direction $d^k$ (e.g., using QHM)
6. \quad $x^{k+1} \leftarrow x^k - \alpha d^k$
7. \quad $\Delta_k \leftarrow \langle x^k, d^k \rangle - \frac{\alpha}{2} \|d^k\|^2$
8. \quad $N \leftarrow \lceil \theta (k - k_0) \rceil$
9. \quad \textbf{if} $N > N_{\text{min}}$ \textbf{and} $k \mod K_{\text{test}} == 0$ \textbf{then}
10. \quad \quad $(\hat{\mu}_N, \hat{\sigma}_N) \leftarrow$ sample mean and BM/OLBM variance of $\{\Delta_k - N+1, \ldots, \Delta_k\}$
11. \quad \quad \textbf{if} $0 \in \hat{\mu}_N \pm t^*_1 - \delta / 2 \frac{\hat{\sigma}_N}{\sqrt{N}}$ \textbf{then}
12. \quad \quad \quad $\alpha \leftarrow \tau \alpha$
13. \quad \quad \quad $k_o \leftarrow k$
14. \quad \textbf{end}
15. \textbf{end}
16. \textbf{end}
17. \textbf{output:} $x^T$ (or the average of last epoch)

Table 1: List of hyperparameters of Algorithm 1

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Range</th>
<th>Explanation</th>
<th>Default value</th>
</tr>
</thead>
<tbody>
<tr>
<td>$N_{\text{min}}$</td>
<td>$\mathbb{Z}_+$</td>
<td>minimum number of samples for testing</td>
<td>min{1000, $[n/b]$}</td>
</tr>
<tr>
<td>$K_{\text{test}}$</td>
<td>$\mathbb{Z}_+$</td>
<td>period to perform statistical test</td>
<td>min{100, $[n/b]$}</td>
</tr>
<tr>
<td>$\delta$</td>
<td>$(0, 1)$</td>
<td>$(1 - \delta)$-confidence interval</td>
<td>0.01 (99% confidence)</td>
</tr>
<tr>
<td>$\theta$</td>
<td>$(0, 1)$</td>
<td>fraction of recent samples to keep (after reset)</td>
<td>1/4</td>
</tr>
<tr>
<td>$\tau$</td>
<td>$(0, 1)$</td>
<td>learning rate drop factor</td>
<td>1/10</td>
</tr>
</tbody>
</table>

3. Statistical tests of (non-)stationarity

In order to test if the master condition \cite{10} holds approximately, we collect the simple statistics

$$\Delta_k \triangleq \langle x^k, d^k \rangle - \frac{\alpha}{2} \|d^k\|^2 = \langle x^{k+1}, d^k \rangle + \frac{\alpha}{2} \|d^k\|^2.$$ 

Here the second expression for $\Delta_k$ is obtained using the direct substitution $x^{k+1} = x^k - \alpha d^k$, which can be more convenient to implement if $\Delta_k$ is collected after the state updates to $x^{k+1}$.

In the language of statistical hypothesis testing (e.g., \cite{Lehmann}), we make as our \textit{null hypothesis} that the dynamics \cite{qhm} have reached a stationary distribution $\pi$. If we have $N$ samples $\{\Delta_k\}$, we know from equation \cite{10} and the Markov chain CLT (see its application in \cite{Jones}) that as $N \to \infty$, under the null hypothesis the mean statistic $\hat{\Delta}$ follows a normal distribution with mean 0 and variance $\sigma_N^2 / \sqrt{N}$. Our \textit{alternative hypothesis} is that the dynamics \cite{qhm} have \textit{not} reached stationarity. To test these hypotheses, we adopt the classical confidence interval test. More specifically, we use the most recent $N$ samples $\{\Delta_{k-N-1}, \ldots, \Delta_k\}$ to compute the sample mean $\hat{\mu}_N$ and a variance estimator $\hat{\sigma}_N^2$ for $\sigma_N^2$. Then we can form the $(1 - \delta)$-confidence interval

$$\hat{\mu}_N - \omega_N, \hat{\mu}_N + \omega_N$$

with half width

$$\omega_N = t^*_{1 - \delta / 2} \frac{\hat{\sigma}_N}{\sqrt{N}}, \quad (13)$$

where $t^*_{1 - \delta / 2}$ is the $(1 - \delta / 2)$ quantile of the Student’s $t$-distribution with degrees of freedom corresponding to that in the variance estimator $\hat{\sigma}_N^2$. Because the sequence $\{\Delta_{k-N+1}, \ldots, \Delta_k\}$ are highly correlated due to the underlying Markov dynamics, the classical formula for the sample variance (obtained by assuming i.i.d. samples) does not work for $\hat{\sigma}_N^2$. We need to use more sophisticated \textit{batch mean} (BM) or \textit{overlapping batch mean} (OLBM) variance estimators developed in the Markov chain Monte Carlo literature (e.g., \cite{Jones} \cite{Flegal}). See \cite{Lang} for detailed explanation. We also list the formulas for computing BM and OLBM in Appendix A.
During each step \( k \), SSLS uses the classical Armijo line-search to find an appropriate step size \( \eta_k \) for the randomly chosen function \( f_{\xi_k} \) (initialized by \( \alpha_k^{-1} \)), then sets the overall learning rate to be

\[
\alpha_k = (1 - \gamma)\alpha_k^{-1} + \gamma\eta_k,
\]

where \( \gamma \in [0, 1] \). When \( \gamma = 1 \), SSLS reduces to the stochastic Armijo line-search used by Vaswani et al. (2019). A good choice is to set \( \gamma = b/n \) where \( n \) is the total number of training examples and \( b \) is the mini-batch size. Suppose \( \rho_{inc} = 2 \) and \( \eta_k = 2\alpha_k^{-1} \) is accepted at step \( k \), then

\[
\alpha_k = (1 - \gamma)\alpha_k^{-1} + \gamma(2\alpha_k^{-1}) = (1 + \gamma)\alpha_k^{-1}.
\]

If this happens at every iteration over one epoch (of \([n/b]\) iterations) and \( n \gg b \), then

\[
\alpha_{k+\lceil n/b \rceil} = (1 + b/n)^{\lceil n/b \rceil}\alpha_k \approx e \cdot \alpha_k.
\]  

Therefore, the most aggressive growth of the learning rate is by a factor of \( e \) over one epoch. Such a growing factor is reasonable for line search in deterministic optimization (e.g., Nesterov 2013).
Algorithm 3: SALSA: SASA+ with warmup by SSLS

1. **input:** $x^0 \in \mathbb{R}^p$, number of steps for warmup $T_{wu}$ and total number of steps $T > T_{wu}$

2. for $k = 0, \ldots, T_{wu}$ do
   3. Run SSLS (Algorithm 2)
   4. end

5. for $k = T_{wu} + 1, \ldots, T$ do
   6. Run SASA+ (Algorithm 1)
   7. end

8. **output:** $x^T$

Setting $\gamma = \sqrt{b/n}$ leads to maximum growth of $e^2$ over one epoch. A similar smoothing effect holds for decreasing the learning rate as well. The smoothing scheme allows us to use standard increasing and decreasing factors, such as $\rho_{inc} = 2$ and $\rho_{dec} = 1/2$. Without the smoothing scheme, Vaswani et al. (2019) set $\rho_{inc} = 2^{b/n}$ to restrict dramatic growth of $\alpha_k = \eta_k$ (equivalent to $\gamma = 1$ in SSLS). However, the decreasing of $\alpha_k = \eta_k$ can be excessive and premature, even with $\rho_{dec} = 0.9$.

Following the theoretical framework of Vaswani et al. (2019), it may be possible to show that SSLS has similar convergence properties as stochastic Armijo line-search under the smooth and interpolation assumptions, which we leave as a future research project. In this paper, we are mainly interested in investigating its performance in practice. In particular, we use it on deep neural network models with ReLU activations. Here the loss functions are non-smooth, and classical theoretical foundations for line-search do not carry over. Nevertheless, we found SSLS to have robust performance in all of our experiments. In order to handle the case of potential non-descent directions in the non-smooth case, we exit the line search after a maximum of $m$ tries by adding the condition $\eta_k > \rho_{dec}^m \eta_k$ for staying in the line-search loop. By default, we set $m = 10$ together with $\rho_{dec} = 1/2$.

Finally, we combine SASA+ with SSLS to form Algorithm 3 which we call SALSA (Stochastic Approximation with Line-search and Statistical Adaption). Without prior knowledge on the loss function and training dataset, we start with a very small learning rate and use SSLS to gradually increase it to be around a stationary value that is (automatically) customized to the problem. The after $T_{wu}$ steps, we switch to SASA+ to gradually reduce the learning rate to settle down to a (hopefully good) local minimum. We typically set the number of warm-up steps $T_{wu}$ to be equivalent to a few tens of epochs. According to the calculation in (14), using 30 epochs would allow a potential growth factor up to a reasonable fraction of $e^{30}$, which is sufficient for most applications. We could avoid a warmup phase by running SASA+ and SSLS together until the first SASA+ drop, which could only come after SSLS had reached a relatively stable value for $\alpha_k$, allowing the dynamics to approach stationarity. However, we use a fixed warmup phase $T_{wu}$ in our experiments for simplicity.

**Computational cost of SSLS.** When we fix the number of training epochs, the wall-clock time of Smoothed Stochastic Line Search (SSLS) is about 1.5 times of that of the plain stochastic optimization algorithm. This 0.5 computational overhead is due to 2 function evaluations in each line search step on average. Notice that the function evaluation is on the same minibatch and line search only needs the function value without gradient. In SALSA, we switch from SSLS to SALSA at one-third of the total training epochs. Therefore, the total computational overhead caused by using SSLS is only 0.15 of the original total training time.

## 5 Experiments

We evaluate the performance of SASA+ (Algorithm 1), SSLS (Algorithm 2) and SALSA (Algorithm 3) by conducting several experiments on two common deep learning datasets. We compare SALSA to the following baselines: SGD with a hand-tuned constant-and-cut learning rate schedule and Adam with a tuned warmup phase (e.g., Wilson et al., 2017; Lang et al., 2019). We also compare SASA+ to the original SASA from Lang et al. (2019). We use weight decay in all experiments.

We use the default values in Table 1 for hyperparameters.
CIFAR-10  We trained an 18-layer ResNet model [He et al., 2016] on CIFAR-10 (Krizhevsky & Hinton, 2009) with random cropping and random horizontal flipping for data augmentation and weight decay 0.0005. Row 1 of Figure [1] compares the best performance of each method. Here SGM-hand uses $\alpha_0 = 1.0$ and $\beta = 0.9$ and drops $\alpha$ by a factor of 10 every 50 epochs. Adam has a tuned global learning rate $\alpha_0$ and a “warmup” phase of 50 epochs. With only a tuned $\alpha_0 = 1e^{-4}$ (the optimal value in our grid search of $\{1e^{-5}, 1e^{-4}, 1e^{-3}, 1e^{-2}\}$), Adam is only able to reach around 93% test accuracy for this model. On the other hand, both SASA+ and SALSA are able to reach similar performance with the hand-tuned SGD. While other methods starts from an oracle maximal learning rate 1.0, SALSA starts from a small initial learning rate 0.01. The SSL in the first stage of SALSA is able to gradually grow the learning rate to a maximal value and then switches to SASA+ in the second stage.

Figure [2] shows the evolution of SASA+’s different statistics over the course of training the ResNet18 model on CIFAR-10 using the default parameters in Table [4]. Between two jumps, the statistics $\Delta_k$ decays toward zero. As long as its confidence interval [13] does not overlap with 0, we are confident that the process is not stationary yet and keep exploring with the current learning rate. Otherwise, we decrease the learning rate and enter another cycle of approaching stationarity.

We give experiments that demonstrate the sensitivity of SASA+ to values around its defaults in Figure [5]. The top row shows that SASA+ is robust to the choice of the dropping factor $\tau$. The larger the $\tau$ is, the longer the process will stay between two drops, which can be also seen in Figure [2]. The middle row shows the effect of statistical parameter $\delta$. Smaller $\delta$ leads to wider confidence interval, and makes easier and thus earlier to fire the statistical test. The bottom row shows the effect of $\theta$, the fractions of samples to keep after reset. Smaller $\theta$ values lead to more frequent dropping, but do not seem to impact the final performance.

Figure [3] shows the dynamics of SSLS and SALSA. In the upper row, SLSS increases the learning rate in the initial phase, and then reaches a stable learning rate once the increases and decrease balance out. With enough momentum (QHM $\beta = \nu = 0.9$), the SSLS can further decrease the learning rate automatically and achieve good training and testing performance. However, without momentum (SGD), the stable learning rate gets too large, and its optimization performance is bad, but the process is still stable and the learning rate does not blow up. In the lower row, SALSA switches from SSLS to SASA+ when it reaches the maximal learning rate. Then even in the case without momentum, SASA+ still works and finally gets as good performance as best hand-tuning SGD. This shows that the SSLS is able to approximate the best hand-tuned maximal learning rate, and the combination of SSLS and SASA+ is necessary to get final good optimization performance.

ImageNet  We also test our algorithms on the large scale ImageNet dataset [Deng et al., 2009]. We use the ResNet18 architecture, random cropping and random horizontal flipping for data augmentation and weight decay 0.0001. Row 1 of Figure [5] compares the performance of the different optimizers. Even with a tuned $\alpha_0 = 1e^{-4}$ (the optimal value in our grid search of $\{1e^{-5}, 1e^{-4}, 1e^{-3}, 1e^{-2}\}$) and allowed to have a long warmup phase (30 epochs), Adam failed to match the generalization performance of SGM. On the other hand, both SASA+ and SALSA were able to match the performance of hand-tuned SGM using the default values of its parameters. Although both SASA+ and SALSA are using the NAG (equivalent to QHM with $\beta = \nu = 0.9$),
Figure 2: Statistics used in SASA+. Upper row: drop ratio \( \tau = 1/2 \); lower row: \( \tau = 1/10 \). The left column shows the instantaneous value of \( \Delta_k \). The middle column shows the confidence interval of \( E[\Delta_k] \) constructed by SASA+, which should contain 0 with high probability if the process is stationary. The right column shows the variance estimated by different methods, where BM and OLBM takes into the consideration of correlation in Markov chains and are more accurate.

Figure 3: Sensitivity analysis of SASA+ on CIFAR10, using \( \beta = 0.9 \) and \( \nu = 1 \). The training loss, test accuracy, and learning rate schedule for SASA+ using different values of \( \tau \) (top row), \( \delta \) (middle row), and \( \theta \) (bottom row) around the default values are shown.

they start with different initial learning rates and have quite different dynamics. SASA+(Nag) starts with an initial learning rate of 1.0, while the SALSA(Nag) starts with a small learning rate 0.1 and automatically determines the appropriate largest learning rate by using SSLS. As a consequence, SASA+(Nag) reaches a stationary state much faster. Nevertheless, their final performances are nearly the same, with SALSA(Nag) even slightly higher.
Row 2 of Figure 5 shows that SASA+ is simple and general enough to work with different stochastic optimization methods with constant hyperparameters, i.e., SGM with $\beta = 0.9$, NAG, and the recommended QHM setting ($\beta = 0.999$ and $\nu = 0.7$). The dotted curves show the performance of these methods with the best hand-tuned constant-and-cut learning rate schedule. The solid lines show the automatic tuning by SASA+, which learns faster and finally achieves similar performance.

Figure 6 shows the sensitivity analysis of SSLS. From the left and middle panels, we can see that the SSLS always converges to a large learning rate (1.806 in this case), which is comparable with the best human-chosen value of 1.0. The right panel shows that the larger the sufficient descent constant $c$ is, the smaller the maximal learning rate is and the slower SSLS reaches to its maximal learning rate. This is intuitive because larger $c$ puts more limits on the SSLS to grow the learning rate. Moreover, within 60 epochs, SSLS never decreases the learning rate again once it achieves a large stable learning rate, so it is necessary to switch to SASA+ after SSLS reaches a stable learning rate.

6 Conclusions

We presented SASA+, a simpler yet more powerful variant of the statistical adaptive stochastic approximation (SASA) method proposed by [Lang et al., 2019]. SASA+ is equipped with a single statistical test for (non-)stationarity that works for a broad family of stochastic optimization methods without modification. This greatly simplifies its implementation and deployment in software packages. While SASA+ focuses on how to automatically reducing the learning rate to obtain better asymptotic convergence, we also propose a smoothed stochastic line-search method (SSLS) to warm up the optimization process, thus removing the burden of expensive trial and error for setting a good initial learning rate. The combined algorithm, SALSA, is highly autonomous and robust over variations of its hyperparameters. Using the same default setting, SALSA obtained state-of-the-art performance on several common deep learning models that is competitive with the best hand-tuned optimizers.

References

Figure 5: Upper: Comparison of different optimizers on ImageNet. All SASA runs use SGD with momentum 0.9, while SASA+ and SALSA use NAG (QHM with $\beta = \nu = 0.9$). SALSA starts from a small initial learning rate 0.1 while other methods start from an oracle maximal learning rate 1.0. SALSA witches from SSLS to SASA+ at epoch 40. Lower: the first three curves are stochastic optimization algorithms with hand-tuning learning rate, i.e., decrease every 30 epochs as it is shown in the lower right panel. The last three curves are SASA+ combined with these 3 algorithms. SASA+ automatically adapt their learning rate (see the lower right panel), achieves comparable and even slightly higher testing accuracy (see the lower middle panel).

Figure 6: Sensitivity analysis of SSLS around $\alpha_0 = 0.01, \gamma = b/n = 0.0002, c = 0.1$, with SGD $\beta = 0.9, \nu = 1$ on ImageNet.


### A MCMC VARIANCE ESTIMATORS

Several estimators for the asymptotic variance of the history-average estimator of a Markov chain have appeared in work on Markov Chain Monte Carlo (MCMC). Jones et al. (2006) gives a nice example of such results. Here we simply list two common variance estimators, and we refer the reader to that work and Lang et al. (2019) (from which we borrow notation) for appropriate context and formality.

**Batch means variance estimator.** Let $\bar{z}_N$ be the history average estimator with $N$ samples, that is, given samples $\{z_i\}$ from a Markov chain, $\bar{z}_N = \frac{1}{N} \sum_{i=0}^{N} z_i$. Now form $p$ batches from the $N$ samples, each of size $q$. Compute the “batch means” $\bar{z}_j = \frac{1}{q} \sum_{i=jq}^{(j+1)q-1} z_i$ for each batch $j$. Then compute the batch means estimator using:

$$\hat{\sigma}_N^2 = \frac{q}{p-1} \sum_{j=0}^{p-1} (\bar{z}_j - \bar{z}_N)^2. \quad (15)$$

The estimator is just the variance of the batch means around the history average $\bar{z}_N$. This statistic has $p-1$ degrees of freedom. As in Jones et al. (2006) and Lang et al. (2019), we take $p = q = \sqrt{N}$ when using this estimator.

**Overlapping batch means variance estimator.** The overlapping batch means (OLBM) estimator Flegal & Jones (2010) has better asymptotic variance than the batch means estimator. The OLBM estimator is conceptually the same, but it uses $N - p + 1$ overlapping batches of size $p$ (rather than disjoint batches) and has $N - p$ degrees of freedom. It can be computed as:

$$\hat{\sigma}_N^2 = \frac{Np}{(N-p)(N-p+1)} \sum_{j=0}^{N-p} (\bar{z}_N - \frac{1}{p} \sum_{i=1}^{p} z_{j+i})^2. \quad (16)$$

### B THE STATISTICAL TESTS IN SASA AND SASA+

As mentioned in our contributions, there are two main differences between SASA from Lang et al. (2019) and SASA+. First, SASA+ has a much more general “master stationary condition” (10), so it can be applied to any stochastic optimization method with constant hyperparameters (e.g., SGD, stochastic heavy-ball, NAG, QHM, etc.), while the stationary condition in Yaida (2018) and SASA proposed in Lang et al. (2019) (see Equation (11)) only applies to the stochastic heavy-ball method. Second, the statistical tests used in SASA and SASA+ are quite different, as we elaborate below. The difference starts from a conceptual change from SASA to SASA+. In SASA, one wants to **confidently** detect stationarity. If stationarity is detected, one decreases the learning rate, and otherwise keeps it the same. In SASA+, one wants to **confidently** detect non-stationarity. If non-stationarity is detected, one keeps the learning rate the same, and otherwise decreases. This conceptual change leads to a simpler and more rigorous statistical test in SASA+.

#### B.1 EQUIVALENCE TEST IN SASA

To confidently detect stationarity, SASA has to set non-stationarity as null hypothesis and stationarity as the alternative hypothesis. If one confidently rejects the null (non-stationarity), then one can be confident that the process is stationary. Instead of detecting stationarity, SASA simplifies to only detect the **necessary but not sufficient** stationarity condition (11). Formally, the test in SASA is

$$H_0 : E[\Delta] \neq 0 \text{ vs. } H_1 : E[\Delta] = 0, \quad (17)$$

where samples of $\Delta$, i.e., $\Delta_k \triangleq \langle x^k, d^k \rangle - \frac{\alpha}{2} \|d^k\|^2$, are collected along the training process. This kind of test is called an equivalence test in statistics, see, e.g., Streiner (2003). There is no power\(^1\) in the equivalence test (17), i.e., one cannot confidently reject the null hypothesis and prove stationarity.

\(^1\)In statistical hypothesis testing, power is the ability to reject the null hypothesis when it is false.
at all! Intuitively, even when the process is stationary, with only a finite number of (noisy) samples \( \{ \Delta_k \}_{k=1}^N \), the sample mean \( \Delta_N \neq 0 \) (with probability one) is always more likely to be the true mean than the singleton 0. In other words, one cannot deny that the process is probably infinitely close to stationary but still non-stationary.

To gain power in the equivalence test, one needs to use domain knowledge to define an equivalence interval. Formally, the true test in SASA is

\[
H_0 : |E[\Delta]| > \zeta \nu \text{ vs. } H_1 : E[\Delta] \in [-\zeta \nu, \zeta \nu], \tag{18}
\]

where \( \zeta \nu \) is the equivalence interval. In English, instead of the usual null hypothesis of not-equal-to-zero in (17), now the null hypothesis is not-equal-to-zero by a margin \( \zeta \nu \). In this case, when \( E[\Delta] \)'s confidence interval is contained in the equivalence interval, i.e.,

\[
\left[ \Delta_N - t_{1-\delta/2} \frac{\hat{\sigma}_N}{\sqrt{N}}, \Delta_N + t_{1-\delta/2} \frac{\hat{\sigma}_N}{\sqrt{N}} \right] \subset [-\zeta \nu N, \zeta \nu N], \tag{19}
\]

we are confident to reject the null hypothesis and to prove/accept \( H_1 \) (the stationary condition is met within an error tolerance). This is exactly the test in SASA [Lang et al., 2019], see its Equation (10). However, this equivalence test requires estimation of \( \nu_N \) (that estimates the magnitude of \( \Delta \)) and an additional hyperparameter \( \zeta \) that controls the equivalence interval width. In SASA’s notation, the equivalence width \( \zeta \) is denoted by \( \delta \).

Moreover, SASA makes the unjustified assumption that under their null hypothesis \( (H_0 : |E[\Delta]| > \zeta \nu) \), the Markov central limit theorem holds. Under their \( H_0 \), the process is non-stationary, while the construction of \( E[\Delta] \)'s confidence interval in Eqn. (19) relies on the (asymptotic) stationarity of the Markov process. Therefore, despite the empirical success of SASA, its statistical test is intrinsically flawed because of this testing setup.

### B.2 STANDARD TEST IN SASA+

In SASA+, we want to confidently detect non-stationarity. If non-stationarity is detected, one keeps the learning rate the same, and otherwise one decreases it. This conceptual change naturally removes the complication of the equivalence test and the intrinsic flaw in SASA.

To confidently detect non-stationarity, SASA+ sets stationarity as the null hypothesis and non-stationarity as the alternative:

\[
H_0 : \text{The process is stationary} \text{ vs. } H_1 : \text{The process is not stationary.} \tag{20}
\]

The master stationary condition \( E[\Delta] = 0 \) is a necessary condition for stationary of the process, and thus confidently rejecting \( E[\Delta] = 0 \) is sufficient to confidently reject the null (stationarity) hypothesis. Moreover, under this null hypothesis, the Central Limit Theorem of Markov Processes exactly holds true (because the process is stationary), validating the construction of \( E[\Delta] \)'s confidence interval. Therefore, the intrinsic flaw in SASA is naturally solved in SASA+.

Now we describe the test in SASA+. When the confidence interval of \( E[\Delta] \) does not contain 0, i.e.,

\[
0 \not\in \left[ \Delta_N - t_{1-\delta/2} \frac{\hat{\sigma}_N}{\sqrt{N}}, \Delta_N + t_{1-\delta/2} \frac{\hat{\sigma}_N}{\sqrt{N}} \right], \tag{21}
\]

then we reject the null (stationary) hypothesis and keep the learning rate the same. Otherwise, we decrease the learning rate. Notice that there is no additional hyperparameter \( \zeta \).

### B.3 THE DIFFERENCE IN PRACTICE

Although SASA and SASA+ are conceptually quite different, they both use the same confidence interval (see (19) and (21)) for \( E[\Delta] \). In practice, their difference is illustrated in Figure 7. Their difference lies in Case (2) and (3). In Case (3), the process has not reached stationarity yet with high probability, according to the confidence interval of \( E[\Delta] \) (Red). SASA+’s test is confident to reject its (stationary) null hypothesis, so it keeps the learning rate the same. However, SASA decreases its learning rate because it is confident that the stationary condition holds true within its error tolerance (equivalence interval). In this case, SASA makes an error due to its relatively large equivalence
Figure 7: Statistical tests in SASA and SASA+. Case (1): both SASA and SASA+ keep the learning rate. Case (4): both SASA and SASA+ decrease the learning rate. Case (2): SASA keeps the learning rate while SASA+ decreases. Case (3): SASA decreases the learning rate while SASA+ keeps.

interval. In Case (2), SASA+’s test is not confident to reject its (stationary) null hypothesis and so it decreases its learning rate. On the contrary, SASA’s test is not confident to claim that the stationary condition holds true within its error tolerance and thus keeps its learning rate. In this sense, SASA+ is more aggressive than SASA in decreasing learning rate.

In numerical experiments on the CIFAR10 and ImageNet datasets, the (width of) the equivalence interval $\zeta_\nu$ is typically much smaller than the (width of) the confidence interval of $E[\Delta]$, see Figure 5(a), 7(a) and 9(a) in [Lang et al., 2019]. Notice that in those figures, the yellow curve is $\nu$ instead of $\zeta_\nu$, and thus the equivalence interval is even smaller. Therefore, Case (3) happens very rarely in those experiments, and this explains the reason why SASA does not make obvious mistakes in decreasing the learning rate. In practice, Case (2) sometimes happens, and thus we can see that SASA+ seems to be slightly more aggressive at decreasing the learning rate than SASA. For example in Figure 1, the black curve (SASA+) is faster at decreasing the learning rate than the green curve (SASA).