Using Statistics to Automate Stochastic Optimization

Hunter Lang*
hunterlang1@gmail.com

Pengchuan Zhang
Microsoft Research AI
penzhan@microsoft.com

Lin Xiao
Microsoft Research AI
lin.xiao@microsoft.com

Abstract

Despite the development of numerous adaptive optimizers, tuning the learning rate of stochastic gradient methods remains a major roadblock to obtaining good practical performance in machine learning. Rather than changing the learning rate at each iteration, we propose an approach that automates the most common hand-tuning heuristic: use a constant learning rate until "progress stops", then drop. We design an explicit statistical test that determines when the dynamics of stochastic gradient descent reach a stationary distribution. This test can be performed easily during training, and when it fires, we decrease the learning rate by a constant multiplicative factor. Our experiments on several deep learning tasks demonstrate that this statistical adaptive stochastic approximation (SASA) method can automatically find good learning rate schedules and match the performance of hand-tuned methods using default settings of its parameters. The statistical testing helps to control the variance of this procedure and improves its robustness.

1 Introduction

Stochastic approximation methods, including stochastic gradient descent (SGD) and its many variants, serve as the workhorses of machine learning with big data. Many tasks in machine learning can be formulated as the stochastic optimization problem:

$$\text{minimize}_{x\in\mathbf{R}^n} \;\; F(x) \triangleq \mathbf{E}_\xi\big[f(x,\xi)\big],$$

where ξ is a random variable representing data sampled from some (unknown) probability distribution, $x \in \mathbf{R}^n$ represents the parameters of the model (e.g., the weight matrices in a neural network), and f is a loss function. In this paper, we focus on the following variant of SGD with *momentum*,

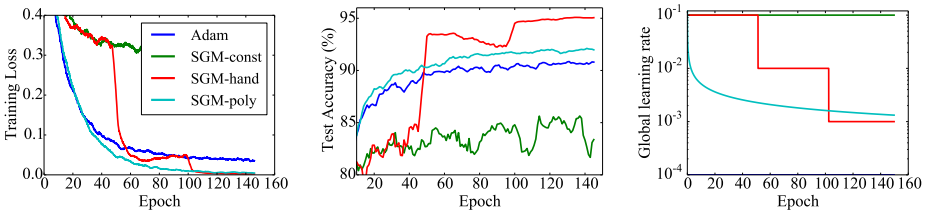
$$\begin{aligned} d^{k+1} &= (1-\beta_k)g^k + \beta_k d^k, \\ x^{k+1} &= x^k - \alpha_k d^{k+1}, \end{aligned} \tag{1}$$

where $g^k = \nabla_x f(x^k, \xi^k)$ is a stochastic gradient, $\alpha_k > 0$ is the learning rate, and $\beta_k \in [0, 1)$ is the momentum coefficient. This approach can be viewed as an extension of the heavy-ball method (Polyak, 1964) to the stochastic setting.[1] To distinguish it from the classical SGD, we refer to the method (1) as SGM (Stochastic Gradient with Momentum).

Theoretical conditions on the convergence of stochastic approximation methods are well established (see, e.g., Wasan, 1969; Kushner and Yin, 2003, and references therein). Unfortunately, these asymptotic conditions are insufficient in practice. For example, the classical rule $\alpha_k = a/(k+b)^c$ where $a, b > 0$ and $1/2 < c \leq 1$, often gives poor performance even when a, b, and c are hand-tuned. Additionally, despite the advent of numerous adaptive variants of SGD and SGM (e.g., Duchi et al., 2011; Tieleman and Hinton, 2012; Kingma and Ba, 2014, and other variants), achieving good performance in practice often still requires considerable hand-tuning (Wilson et al., 2017).

*Work done while HL was at Microsoft Research AI.

[1] For fixed values of α and β, this "normalized" update formula is equivalent to the more common updates $d^{k+1} = g^k + \beta d^k$, $x^{k+1} = x^k - \alpha' d^{k+1}$ with the reparametrization $\alpha = \alpha'/(1-\beta)$.



(a) training loss (b) test accuracy (c) learning rate

Figure 1: Smoothed training loss, test accuracy, and (global) learning rate schedule for an 18-layer ResNet model (He et al., 2016) trained on the CIFAR-10 dataset using four different methods (with constant momentum $\beta = 0.9$). Adam: $\alpha = 0.0001$; SGM-const: $\alpha = 1.0$; SGM-poly: $a = 1.0, b = 1, c = 0.5$; SGM-hand: $\alpha = 1.0$, drop by 10 every 50 epochs.

Figure 1 shows the training loss and test accuracy of a typical deep learning task using four different methods: SGM with constant step size (SGM-const), SGM with diminishing $O(1/k)$ step size (SGM-poly), Adam (Kingma and Ba, 2014), and hand-tuned SGM with learning rate scheduling (SGM-hand). For the last method, we decrease the step size by a multiplicative factor after a suitably long number of epochs ("constant-and-cut"). The relative performance depicted in Figure 1 is typical of many tasks in deep learning. In particular, SGM with a large momentum and a constant-and-cut step-size schedule often achieves the best performance. Many former and current state-of-the-art results use constant-and-cut schedules during training, such as those in image classification (Huang et al., 2018), object detection (Szegedy et al., 2015), machine translation (Gehring et al., 2017), and speech recognition (Amodei et al., 2016). Additionally, some recent theoretical evidence indicates that in some (strongly convex) scenarios, the constant-and-cut scheme has better finite-time last-iterate convergence performance than other methods (Ge et al., 2019).

Algorithm 1: General SASA method

Input: $\{x^0, \alpha_0, M, \beta, \zeta\}$
for $j \in \{0, 1, \ldots\}$ **do**
  for $k \in \{jM, \ldots, (j+1)M - 1\}$ **do**
    Sample ξ^k.
    Compute $g^k = \nabla_x f(x^k, \xi^k)$.
    $d^{k+1} = (1-\beta)g^k + \beta d^k$
    $x^{k+1} = x^k - \alpha d^{k+1}$
    // collect statistics
  end
  if *test(statistics)* **then**
    $\alpha \leftarrow \zeta\alpha$
    // reset statistics
  end
end

Inspired by the success of the "constant-and-cut" scheduling approach, we develop an algorithm that can automatically decide when to drop α. Most common heuristics try to identify when "training progress has stalled." We formalize stalled progress as when the SGM dynamics in (1), with constant values of α and β, reach a stationary distribution. The existence of such a distribution seems to match well with many empirical results (e.g., Figure 1), though it may not exist in general. Since SGM generates a rich set of information as it runs (i.e. $\{x^0, g^0, \ldots, x^k, g^k\}$), a natural approach is to collect some statistics from this information and perform certain tests on them to decide whether the process (1) has reached a stationary distribution. We call this general method SASA: statistical adaptive stochastic approximation.

Algorithm 1 summarizes the general SASA method. It performs the SGM updates (1) in phases of M iterations, in each iteration potentially computing some additional statistics. After M iterations are complete, the algorithm performs a statistical test to decide whether to drop the learning rate by a factor $\zeta < 1$. Dropping α after a fixed number of epochs and dropping α based on the loss of a held-out validation set correspond to heuristic versions of Algorithm 1. In the rest of this work, we detail how to perform the "test" procedure and evaluate SASA on a wide range of deep learning tasks.

1.1 Related Work and Contributions

The idea of using statistical testing to augment stochastic optimization methods goes back at least to Pflug (1983), who derived a stationary condition for the dynamics of SGD on quadratic functions

and designed a heuristic test to determine when the dynamics had reached stationary. He used this test to schedule a fixed-factor learning rate drop. Chee and Toulis (2018) recently re-investigated Pflug's method for general convex functions. Pflug's stationary condition relies heavily on a quadratic approximation to F and limiting noise assumptions, as do several other recent works that derive a stationary condition (e.g., Mandt et al., 2017; Chee and Toulis, 2018). Additionally, Pflug's test assumes no correlation exists between consecutive samples in the optimization trajectory. Neither is true in practice, which we show in Appendix C.2 can lead to poor predictivity of this condition.

Yaida (2018) derived a very general stationary condition that does not depend on any assumption about the underlying function F and applies under general noise conditions regardless of the size of α. Like Pflug (1983), Yaida (2018) used this condition to determine when to decrease α, and showed good performance compared to hand-tuned SGM for two deep learning tasks with small models. However, Yaida's method does not account for the variance of the terms involved in the test, which we show can cause large variance in the learning rate schedules in some cases. This variance can in turn cause poor empirical performance.

In this work, we show how to more rigorously perform statistical hypothesis testing on samples collected from the dynamics of SGM. We combine this statistical testing with Yaida's stationary condition to develop an adaptive "constant-and-cut" optimizer (SASA) that we show is more robust than present methods. Finally, we conduct large-scale experiments on a variety of deep learning tasks to demonstrate that SASA is competitive with the best hand-tuned and validation-tuned methods without requiring additional tuning.

2 Stationary Conditions

To design a statistical test that fires when SGM reaches a stationary distribution, we first need to derive a condition that holds at stationarity and consists of terms that we can estimate during training. To do so, we analyze the long-run behavior of SGM with constant learning rate and momentum parameter:

$$\begin{aligned} d^{k+1} &= (1-\beta)g^k + \beta d^k, \\ x^{k+1} &= x^k - \alpha d^{k+1}, \end{aligned} \tag{2}$$

where $\alpha > 0$ and $0 \leq \beta < 1$. This process starts with $d^0 = 0$ and arbitrary x^0. Since α and β are constant, the sequence $\{x^k\}$ does not converge to a local minimum, but the distribution of $\{x^k\}$ may converge to a stationary distribution. Letting $\mathcal{F}_k$ denote the σ-algebra defined by the history of the process (2) up to time k, i.e., $\mathcal{F}_k = \sigma(d^0, \ldots, d^k; x^0, \ldots, x^k)$, we denote by $\mathbf{E}_k[\cdot] := \mathbf{E}[\cdot|\mathcal{F}_k]$ the expectation conditioned on that history. Assuming that g^k is Markovian and unbiased, i.e.,

$$\mathbf{P}[g^k|\mathcal{F}_k] = \mathbf{P}[g^k|d^k, x^k], \quad \mathbf{E}[g^k|d^k, x^k] = \nabla F(x^k), \tag{3}$$

then the SGM dynamics (2) form a homogeneous[2] Markov chain with continuous state $(d^k, x^k, g^k) \in \mathbf{R}^{3n}$. These assumptions are always satisfied when $g^k = \nabla_x f(x^k, \xi^k)$ for an i.i.d. sample ξ^k. We further assume that the SGM process converges to a stationary distribution, denoted as $\pi(d, x, g)$[3]. With this notation, we need a relationship $\mathbf{E}_\pi[X] = \mathbf{E}_\pi[Y]$ for certain functions X and Y of (x^k, d^k, g^k) that we can compute during training. Then, if we assume the Markov chain is ergodic, we have that:

$$\bar{z}_N = \frac{1}{N}\sum_{i=0}^{N-1} z_i = \frac{1}{N}\sum_{i=0}^{N-1} \Big(X(x^i, d^i, g^i) - Y(x^i, d^i, g^i) \Big) \to 0. \tag{4}$$

Then we can check the magnitude of the time-average $\bar{z}_N$ to see how close the dynamics are to reaching their stationary distribution. Next, we consider two different stationary conditions.

2.1 Pflug's condition

Assuming $F(x) = (1/2)x^T Ax$, where A is positive definite with maximum eigenvalue L, and that the stochastic gradient g^k satisfies $g^k = \nabla F(x^k) + r^k$, with $\mathbf{E}[r^k] = 0$ and r^k independent of x^k, Pflug

[2] "Homogeneous" means that the transition kernel is time independent.

[3] As stated in Section 1, this need not be true in general, but seems to often be the case in practice.

(1983) derived a stationary condition for the SGD dynamics. His condition can be extended to the SGM dynamics. For appropriate α and β, the generalized Pflug stationary condition says

$$\mathbf{E}_\pi\big[\langle g, d\rangle\big] = -\frac{\alpha(1-\beta)}{2(1+\beta)}\mathbf{tr}(A\Sigma_r) + O(\alpha^2), \tag{5}$$

where Σ_r is the covariance of the noise r. One can estimate the left-hand-side during training by computing the inner product $\langle g^k, d^k\rangle$ in each iteration. Pflug (1983) also designed a clever estimator for the right-hand-side, so it is possible to compute estimators for both sides of (5).

The Taylor expansion in α used to derive (5) means that the relationship may only be accurate for small α, but α is typically large in the first phase of training. This, together with the other assumptions required for Pflug's condition, are too strong to make the condition (5) useful in practice.

2.2 Yaida's condition

Yaida (2018) showed that as long as the stationary distribution π exists, the following relationship holds *exactly:*

$$\mathbf{E}_\pi[\langle x, \nabla F(x)\rangle] = \frac{\alpha}{2}\frac{1+\beta}{1-\beta}\mathbf{E}_\pi[\langle d, d\rangle]$$

In particular, this holds for general functions F and arbitrary values of α. Because the stochastic gradients g^k are unbiased, one can further show that:

$$\mathbf{E}_\pi[\langle x, g\rangle] = \frac{\alpha}{2}\frac{1+\beta}{1-\beta}\mathbf{E}_\pi[\langle d, d\rangle]. \tag{6}$$

We can estimate both sides of (6) by computing $\langle x^k, g^k\rangle$ and $\langle d^k, d^k\rangle = ||d^k||^2$ at each iteration and updating the running mean $\bar{z}_N$ with their difference. That is, we let

$$z_k = \langle x^k, g^k\rangle - \frac{\alpha}{2}\frac{1+\beta}{1-\beta}\langle d^k, d^k\rangle \qquad \bar{z}_N = \frac{1}{N}\sum_{k=B}^{N+B-1} z_k. \tag{7}$$

Here B is the number of samples discarded as part of a "burn-in" phase to reduce bias that might be caused by starting far away from the stationary distribution; we typically take $B = N/2$, so that we use the most recent $N/2$ samples.

Yaida's condition has two key advantages over Pflug's: it holds with no approximation for arbitrary functions F and any learning rate α, and both sides can be estimated with negligible cost. In Appendix C.2, we show in Figure 14 that even on a strongly convex function, the error term in (5) is large, whereas $\bar{z}_N$ in (7) quickly converges to zero. Given these advantages, in the next section, we focus on how to test (6), i.e., that $\bar{z}_N$ defined in (7) is approximately zero.

3 Testing for Stationarity

By the Markov chain law of large numbers, we know that $\bar{z}_N \to 0$ as N grows, but there are multiple ways to determine whether $\bar{z}_N$ is "close enough" to zero that we should drop the learning rate.

Deterministic test. If in addition to $\bar{z}_N$ in (7), we keep track of

$$\bar{v}_N = \frac{1}{N}\sum_{i=B}^{N+B-1}\frac{\alpha}{2}\frac{1+\beta}{1-\beta}\langle d^i, d^i\rangle, \tag{8}$$

A natural idea is to test

$$|\bar{z}_N| < \delta\bar{v}_N \quad \text{or equivalently} \quad |\bar{z}_N/\bar{v}_N| < \delta \tag{9}$$

to detect stationarity, where $\delta > 0$ is an error tolerance. The $\bar{v}_N$ term is introduced to make the error term δ relative to the scale of $\bar{z}$ and $\bar{v}$ ($\bar{v}_N$ is always nonnegative). If $\bar{z}_N$ satisfies (9), then the dynamics (2) are "close" to stationarity. This is precisely the method used by Yaida (2018).

However, because $\bar{z}_N$ is a random variable, there is some potential for error in this procedure due to its variance, which is unaccounted for by (9). Especially when we aim to make a critical decision based on the outcome of this test (i.e., dropping the learning rate), it seems important to more directly account for this variance. To do so, we can appeal to statistical hypothesis testing.

I.i.d. t-test. The simplest approach to accounting for the variance in $\bar{z}_N$ is to assume each sample z_i is drawn i.i.d. from the same distribution. Then by the central limit theorem, we have that $\sqrt{N}\bar{z}_N \to \mathcal{N}(0, \sigma_z^2)$, and moreover $\hat{\sigma}_N^2 = \frac{1}{N-1}\sum_{i=1}^N (z_i - \bar{z}_N)^2 \approx \sigma_z^2$ for large N. So we can estimate the variance of $\bar{z}_N$'s sampling distribution using the sample variance of the z_i's. Using this variance estimate, we can form the $(1-\gamma)$ confidence interval

$$\bar{z}_N \pm t^*_{1-\gamma/2}\frac{\hat{\sigma}_N}{\sqrt{N}},$$

where $t^*_{1-\gamma/2}$ is the $(1-\gamma/2)$ quantile of the Student's t-distribution with $N-1$ degrees of freedom. Then we can check whether

$$\left[\bar{z}_N - t^*_{1-\gamma/2}\frac{\hat{\sigma}_N}{\sqrt{N}},\ \bar{z}_N + t^*_{1-\gamma/2}\frac{\hat{\sigma}_N}{\sqrt{N}}\right] \in \left(-\delta\bar{v}_N,\ \delta\bar{v}_N\right). \tag{10}$$

If so, we can be confident that $\bar{z}_N$ is close to zero. The method of Pflug (1983, Algorithm 4.2) is also a kind of i.i.d. test that tries to account for the variance of $\bar{z}_N$, but in a more heuristic way than (10). The procedure (10) can be thought of as a relative *equivalence test* in statistical hypothesis testing (e.g. Streiner, 2003). When $\hat{\sigma}_N = 0$ (no variance) or $\gamma = 1$ ($t^*_{1-\gamma/2} = 0$, no confidence), this recovers (9).

Unfortunately, in our case, samples z_i evaluated at nearby points are highly correlated (due to the underlying Markov dynamics), which makes this procedure inappropriate. To deal with correlated samples, we appeal to a stronger Markov chain result than the Markov chain law of large numbers (4).

Markov chain t-test Under suitable conditions, Markov chains admit the following analogue of the central limit theorem:

Theorem 1 (Markov Chain CLT (informal); (Jones et al., 2006))**.** *Let $X = \{X_0, X_1, \ldots\}$ be a Harris ergodic Markov chain with state space $\mathcal{X}$, and with stationary distribution π, that satisfies any one of a number of additional ergodicity criteria (see Jones et al. (2006), page 6). For suitable functions $z : \mathcal{X} \to \mathbb{R}$, we have that:*

$$\sqrt{N}\left(\bar{z}_N - \mathbf{E}_\pi z\right) \to \mathcal{N}(0, \sigma_z^2),$$

where $\bar{z}_N = \frac{1}{N}\sum_{i=0}^{N-1} z(X_i)$ is the running mean over time of $z(X_i)$, and $\sigma_z^2 \triangleq var_\pi z$ in general due to correlations in the Markov chain.

This shows that in the presence of correlation, the sample variance is not the correct estimator for the variance of $\bar{z}_N$'s sampling distribution. In light of Theorem 1, if we are given a consistent estimator $\hat{\sigma}_N^2 \to \sigma_z^2$, we can properly perform the test (10). All that remains is to construct such an estimator.

Batch mean variance estimator. Methods for estimating the asymptotic variance of the history average estimator, e.g., $\bar{z}_N$ in (7), on a Markov chain are well-studied in the MCMC (Markov chain Monte Carlo) literature. They can be used to set a stopping time for an MCMC simulation and to determine the simulation's random error (Jones et al., 2006). We present one of the simplest estimators for σ_z^2, the *batch means* estimator.

Given N samples $\{z_i\}$, divide them into b batches each of size m, and compute the batch means: $\bar{z}^j = \frac{1}{m}\sum_{i=jm}^{(j+1)m-1} z_i$ for each batch j. Then let

$$\hat{\sigma}_N^2 = \frac{m}{b-1}\sum_{j=0}^{b-1}(\bar{z}^j - \bar{z}_N)^2. \tag{11}$$

Here $\hat{\sigma}_N^2$ is simply the variance of the batch means around the full mean $\bar{z}_N$. When used in the test (10), it has $b-1$ degrees of freedom. Intuitively, when b and m are both large enough, these batch means are roughly independent because of the mixing of the Markov chain, so their unbiased sample variance gives a good estimator of σ_z^2. Jones et al. (2006) survey the formal conditions under which $\hat{\sigma}_N^2$ is a strongly consistent estimator of σ_z^2, and suggest taking $b = m = \sqrt{N}$ (the theoretically correct sizes of b and m depend on the mixing of the Markov chain). Flegal and Jones (2010) prove strong consistency for a related method called *overlapping batch means* (OLBM) that has better asymptotic variance. The OLBM estimator is similar to (11), but uses $n - b + 1$ overlapping batches of size b and has $n - b$ degrees of freedom.

Algorithm 2: SASA

Input: $\{x^0, \alpha_0, M, \beta, \delta, \gamma, \zeta\}$

zQ = HalfQueue()
vQ = HalfQueue()
for $j \in \{0, 1, 2, \ldots\}$ **do**
  for $k \in \{jM, \ldots, (j+1)M - 1\}$ **do**
    Sample ξ^k and compute $g^k = \nabla_x f(x^k, \xi^k)$
    $d^{k+1} = (1-\beta)g^k + \beta d^k$
    $x^{k+1} = x^k - \alpha d^{k+1}$
    zQ.push($\langle x^k, g^k \rangle - \frac{\alpha}{2}\frac{1+\beta}{1-\beta}||d^{k+1}||^2$)
    vQ.push($\frac{\alpha}{2}\frac{1+\beta}{1-\beta}||d^{k+1}||^2$)
  end
  if $test(zQ, vQ, \delta, \gamma)$ **then**
    $\alpha \leftarrow \zeta\alpha$
    zQ.reset()
    vQ.reset()
  end
end

Algorithm 3: Test

Input: $\{zQ, vQ, \delta, \gamma\}$
Output: boolean (whether to drop)

$\bar{z}_N = \frac{1}{zQ.N}\sum_i zQ[i]$
$\bar{v}_N = \frac{1}{vQ.N}\sum_i vQ[i]$
$m = b = \sqrt{zQ.N}$
for $i \in \{0, \ldots, b-1\}$ **do**
  $\bar{z}^i = \frac{1}{m}\sum_{t=im}^{(i+1)m-1} zQ[t]$
end
$\hat{\sigma}_N^2 = \frac{m}{b-1}\sum_{i=0}^{b-1}(\bar{z}^i - \bar{z}_N)^2$.
$L = \bar{z}_N - t^*_{1-\gamma/2}\frac{\hat{\sigma}_N}{\sqrt{zQ.N}}$
$U = \bar{z}_N + t^*_{1-\gamma/2}\frac{\hat{\sigma}_N}{\sqrt{zQ.N}}$
return $[L, U] \in (-\delta\bar{v}_N, \delta\bar{v}_N)$

3.1 Statistical adaptive stochastic approximation (SASA)

Finally, we turn the above analysis into an adaptive algorithm for detecting stationarity of SGM and decreasing α, and discuss implementation details. Algorithm 2 describes our full SASA algorithm.

To diminish the effect of "initialization bias" due to starting outside of the stationary distribution, we only keep track of the latter half of samples z_i and v_i. That is, if N total iterations of SGM have been run, the "HalfQueues" zQ and vQ contain the most recent $N/2$ values of z_i and v_i—these queues "pop" every other time they "push." If we decrease the learning rate, we empty the queues; otherwise, we keep collecting more samples. To compute the batch mean estimator, we need $O(N)$ space, but in deep learning the total number of training iterations (the worst case size of these queues) is usually small compared to the number of parameters of the model. Collection of the samples z_i and v_i only requires two more inner products per iteration than SGM.

The "test" algorithm follows the Markov chain t-test procedure discussed above. Lines 1-2 compute the running means $\bar{z}_N$ and $\bar{v}_N$; lines 3-7 compute the variance estimator $\hat{\sigma}_N^2$ according to (11), and lines 8-10 determine whether the corresponding confidence interval for $\bar{z}_N$ is within the acceptable interval $(-\delta\bar{v}_N, \delta\bar{v}_N)$. Like the sample collection, the test procedure is computationally efficient: the batch mean and overlapping batch mean estimators can both be computed with a 1D convolution.

For all experiments, we use default values $\delta = 0.02$ and $\gamma = 0.2$. In equivalence testing, γ is typically taken larger than usual to increase the power of the test (Streiner, 2003). We discuss the apparent multiple testing problem of this sequential testing procedure in Appendix D.

4 Experiments

To evaluate the performance of SASA, we run Algorithm 2 on several models from deep learning. We compare SASA to *tuned* versions of Adam and SGM. Many adaptive optimizers do not compare to SGM with hand-tuned step size scheduling, (e.g., Schaul et al., 2013; Zhang and Mitliagkas, 2017; Baydin et al., 2018), and instead compare to SGM with a fixed α or to SGM with tuned polynomial decay. As detailed in Section 1, tuned constant-and-cut schedules are typically a stronger baseline.

Throughout this section, we *do not tune the SASA parameters* δ, γ, M, instead using the default settings of $\delta = 0.02$ and $\gamma = 0.2$, and setting M = one epoch (we test the statistics once per epoch). In each experiment, we use the same α_0 and ζ as for the best SGM baseline. We stress that SASA is not fully automatic: it requires choices of α_0 and ζ, but we show in Appendix A that SASA achieves good performance for different values of ζ. We use weight decay in every experiment—without weight decay, there are simple examples where the process (2) does not converge to a stationary

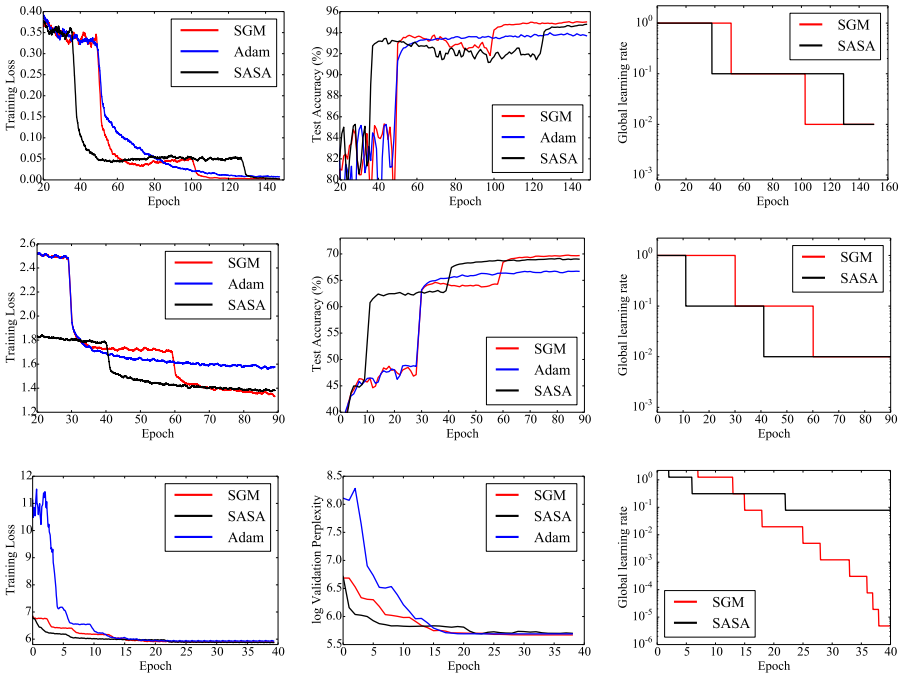


Figure 2: Training loss, test accuracy, and learning rate schedule for SASA, SGM, and Adam on different datasets. Top: ResNet18 on CIFAR-10. Middle: ResNet18 on ImageNet. Bottom: RNN model on WikiText-2. In all cases, starting with the same α_0, SASA achieves similar performance to the best hand-tuned or validation-tuned SGM result.

distribution, such as with logistic regression on separable data. While weight decay does not guarantee convergence to a stationary distribution, it at least rules out this simple case. Finally, we conduct an experiment on CIFAR-10 that shows directly accounting for the variance of the test statistic, as in (10), improves the robustness of this procedure compared to (9).

For hand-tuned SGM (SGM-hand), we searched over "constant-and-cut" schemes for each experiment by tuning α_0, the drop frequency, and the drop amount ζ with grid search. In all experiments, SASA and SGM use a constant $\beta = 0.9$. For Adam, we tuned the initial global learning rate as in Wilson et al. (2017) and used $\beta_1 = 0.9$, $\beta_2 = 0.999$. We also allowed Adam to have access to a "warmup" phase to prevent it from decreasing the learning rate too quickly. To "warm up" Adam, we initialize it with the parameters obtained after running SGM with constant α_0 for a tuned number of iterations. While the warmup phase improves Adam's performance, it still does not match SASA or SGM on the tasks we tried. Appendix A contains a full list of the hyperparameters used in each experiment, additional results for object detection, sensitivity analysis for δ and γ, plots of the evolution of the statistic $\bar{z}_N$ over training, and plots of the different estimators for the variance σ_z^2.

CIFAR-10. We trained an 18-layer ResNet model (He et al., 2016) on CIFAR-10 (Krizhevsky and Hinton, 2009) with random cropping and random horizontal flipping for data augmentation and weight decay 0.0005. Row 1 of Figure 2 compares the best performance of each method. Here SGM-hand uses $\alpha_0 = 1.0$ and $\beta = 0.9$ and drops α by a factor of 10 ($\zeta = 0.1$) every 50 epochs. SASA uses $\gamma = 0.2$ and $\delta = 0.02$, as always. Adam has a tuned global learning rate $\alpha_0 = 0.0001$ and a tuned "warmup" phase of 50 epochs, but is unable to match SASA and tuned SGM.

ImageNet. Unlike CIFAR-10, reaching a good performance level on ImageNet (Deng et al., 2009) seems to require more gradual annealing. Even when tuned and allowed to have a long warmup phase, Adam failed to match the generalization performance of SGM. On the other hand, SASA was able to match the performance of hand-tuned SGM using the default values of its parameters. We again used an 18-layer ResNet model with random cropping, random flipping, normalization, and weight decay 0.0001. Row 2 of Figure 2 shows the performance of the different optimizers.

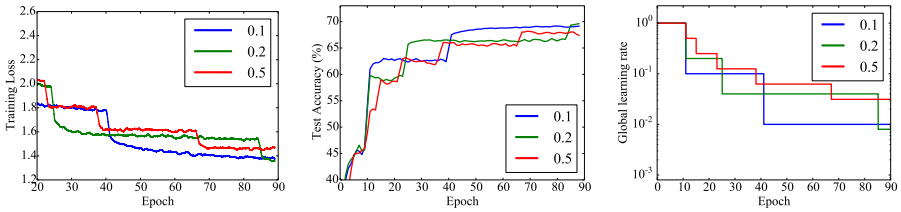


Figure 3: Smoothed training loss, test accuracy and learning rate schedule for ResNet18 trained on ImageNet using SASA with different values of ζ. SASA automatically adapts the drop frequency.

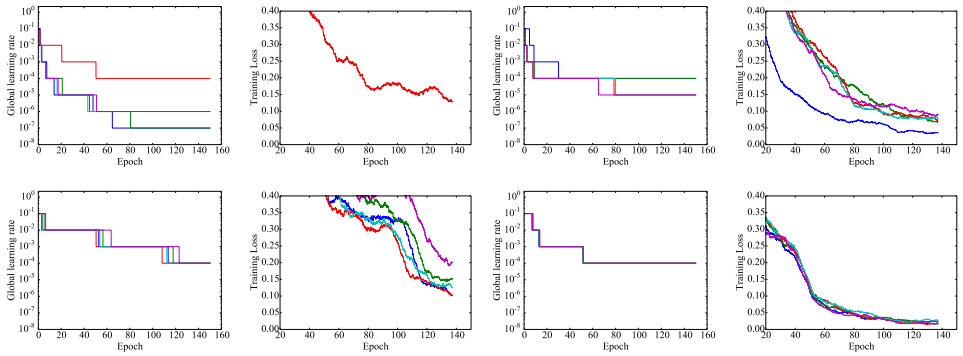


Figure 4: Variance in learning rate schedule and training loss for the two tests (9) (top row) and (10) (bottom row) with *fixed* sample size N, across five independent runs. With the same number of samples and the same value of δ (0.02), the test (9) is much more sensitive to the level of noise.

RNN. We also evaluate SASA on a language modeling task using an RNN. In particular, we train the PyTorch word-level language model example (2019) on the Wikitext-2 dataset (Merity et al., 2016). We compare against SGM and Adam with (global) learning rate tuned using a validation set. These baselines drop the learning rate α by a factor of 4 when the validation loss stops improving. Row 3 of Figure 2 shows that *without using the validation set*, SASA is competitive with these baselines.

Adaptation to the drop factor. At first glance, the choice of the drop factor ζ seems critical. However, Figure 3 shows that SASA automatically adapts to different values of ζ. When ζ is larger, so α decreases slower, the dynamics converge more quickly to the stationary distribution, so the overall rate of decrease stays roughly constant across different values of ζ. Aside from the different choices of ζ, all other hyperparameters were the same as in the ImageNet experiment of Figure 2.

Variance. Figure 4 shows the variance in learning rate schedule and training loss for the two tests in (9) (top row) and (10) (bottom row) with a fixed testing frequency $M = 400$ iterations, across five independent runs. The model is ResNet18 trained on CIFAR-10 using the same procedure as in the previous CIFAR experiment, but with different batch sizes. The left two columns use batch size four, and the right two use batch size eight. With the same number of samples and the same value of $\delta = 0.02$, the test (9) is much more sensitive to the level of noise in these small-batch examples. When the batch size is four, only one of the training runs using the test (9) achieves training loss on the same scale as the others. Appendix B contains additional discussion comparing these two tests.

5 Conclusion

We provide a theoretically grounded statistical procedure for automatically determining when to decrease the learning rate α in constant-and-cut methods. On the tasks we tried, SASA was competitive with the best hand-tuned schedules for SGM, and it came close to the performance of SGM and Adam when they were tuned using a validation set. The statistical testing procedure controls the variance of the method and makes it more robust than other more heuristic tests. Our experiments across several different tasks and datasets did not require any adjustment to the parameters γ, δ, or M.

We believe these practical results indicate that automatic "constant-and-cut" algorithms are a promising direction for future research in adaptive optimization. We used a simple statistical test to check Yaida's stationary condition (6). However, there may be better tests that more properly control the false discovery rate (Blanchard and Roquain, 2009; Lindquist and Mejia, 2015), or more sophisticated conditions that also account for non-stationary dynamics like overfitting or limit cycles (Yaida, 2018). Such techniques could make the SASA approach more broadly useful.

References

Pytorch word language model. `https://github.com/pytorch/examples/tree/master/word_language_model`, 2019.

Dario Amodei, Sundaram Ananthanarayanan, Rishita Anubhai, Jingliang Bai, Eric Battenberg, Carl Case, Jared Casper, Bryan Catanzaro, Qiang Cheng, Guoliang Chen, et al. Deep speech 2: End-to-end speech recognition in english and mandarin. In *International conference on machine learning*, pages 173–182, 2016.

Atilim Günes Baydin, Robert Cornish, David Martínez Rubio, Mark Schmidt, and Frank Wood. Online learning rate adaptation with hypergradient descent. In *Proceedings of the Sixth International Conference on Learning Representations (ICLR)*, Vancouver, Canada, 2018.

Yoav Benjamini and Yosef Hochberg. Controlling the false discovery rate: a practical and powerful approach to multiple testing. *Journal of the Royal statistical society: series B (Methodological)*, 57(1):289–300, 1995.

Gilles Blanchard and Étienne Roquain. Adaptive false discovery rate control under independence and dependence. *Journal of Machine Learning Research*, 10(Dec):2837–2871, 2009.

Jerry Chee and Panos Toulis. Convergence diagnostics for stochastic gradient descent with constant learning rate. In *International Conference on Artificial Intelligence and Statistics*, pages 1476–1485, 2018.

J. Deng, W. Dong, R. Socher, L.-J. Li, K. Li, and L. Fei-Fei. ImageNet: A Large-Scale Hierarchical Image Database. In *CVPR09*, 2009.

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

James M Flegal and Galin L Jones. Batch means and spectral variance estimators in markov chain monte carlo. *The Annals of Statistics*, 38(2):1034–1070, 2010.

Rong Ge, Sham M Kakade, Rahul Kidambi, and Praneeth Netrapalli. The step decay schedule: A near optimal, geometrically decaying learning rate procedure. *arXiv preprint arXiv:1904.12838*, 2019.

Jonas Gehring, Michael Auli, David Grangier, Denis Yarats, and Yann N Dauphin. Convolutional sequence to sequence learning. In *Proceedings of the 34th International Conference on Machine Learning-Volume 70*, pages 1243–1252. JMLR. org, 2017.

Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Deep residual networks for image recognition. In *Proceedgins of the 29th IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 770–778, 2016.

Kaiming He, Georgia Gkioxari, Piotr Dollár, and Ross Girshick. Mask r-cnn. In *Proceedings of the IEEE international conference on computer vision*, pages 2961–2969, 2017.

Yanping Huang, Yonglong Cheng, Dehao Chen, HyoukJoong Lee, Jiquan Ngiam, Quoc V Le, and Zhifeng Chen. Gpipe: Efficient training of giant neural networks using pipeline parallelism. *arXiv preprint arXiv:1811.06965*, 2018.

Galin L Jones, Murali Haran, Brian S Caffo, and Ronald Neath. Fixed-width output analysis for markov chain monte carlo. *Journal of the American Statistical Association*, 101(476):1537–1547, 2006.

Diederik P Kingma and Jimmy Ba. Adam: A method for stochastic optimization. *arXiv preprint arXiv:1412.6980*, 2014.

Alex Krizhevsky and Geoffrey Hinton. Learning multiple layers of features from tiny images. Technical report, Citeseer, 2009.

Harold J. Kushner and G. George Yin. *Stochastic Approximation and Recursive Algorithms and Applications*. Springer, 2nd edition, 2003.

Tsung-Yi Lin, Michael Maire, Serge Belongie, James Hays, Pietro Perona, Deva Ramanan, Piotr Dollár, and C Lawrence Zitnick. Microsoft coco: Common objects in context. In *European conference on computer vision*, pages 740–755. Springer, 2014.

Tsung-Yi Lin, Piotr Dollár, Ross Girshick, Kaiming He, Bharath Hariharan, and Serge Belongie. Feature pyramid networks for object detection. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 2117–2125, 2017.

Martin A Lindquist and Amanda Mejia. Zen and the art of multiple comparisons. *Psychosomatic medicine*, 77(2):114, 2015.

Stephan Mandt, Matthew D Hoffman, and David M Blei. Stochastic gradient descent as approximate bayesian inference. *The Journal of Machine Learning Research*, 18(1):4873–4907, 2017.

Francisco Massa and Ross Girshick. maskrcnn-benchmark: Fast, modular reference implementation of Instance Segmentation and Object Detection algorithms in PyTorch. `https://github.com/facebookresearch/maskrcnn-benchmark`, 2018. Accessed: [Insert date here].

John H McDonald. *Handbook of biological statistics*, volume 2. 2009.

Stephen Merity, Caiming Xiong, James Bradbury, and Richard Socher. Pointer sentinel mixture models. *arXiv preprint arXiv:1609.07843*, 2016.

Georg Ch. Pflug. On the determination of the step size in stochastic quasigradient methods. Collaborative Paper CP-83-025, International Institute for Applied Systems Analysis (IIASA), Laxenburg, Austria, 1983.

Georg Ch. Pflug. Non-asymptotic confidence bounds for stochastic approximation algorithms with constant step size. *Monatshefte für Mathematik*, 110:297–314, 1990.

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

Tom Schaul, Sixin Zhang, and Yann LeCun. No more pesky learning rates. In *International Conference on Machine Learning*, pages 343–351, 2013.

David L Streiner. Unicorns do exist: A tutorial on "proving" the null hypothesis. *The Canadian Journal of Psychiatry*, 48(11):756–761, 2003.

Christian Szegedy, Wei Liu, Yangqing Jia, Pierre Sermanet, Scott Reed, Dragomir Anguelov, Dumitru Erhan, Vincent Vanhoucke, and Andrew Rabinovich. Going deeper with convolutions. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 1–9, 2015.

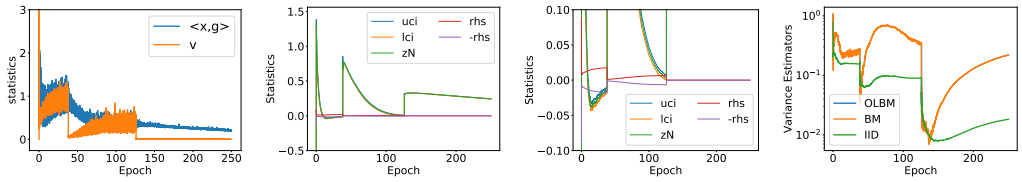
Tijmen Tieleman and Geoffrey Hinton. Lecture 6.5-rmsprop: Divide the gradient by a running average of its recent magnitude. *COURSERA: Neural networks for machine learning*, 4(2):26–31, 2012.

M. T. Wasan. *Stochastic Approximation*. Cambridge University Press, 1969.

Ashia C Wilson, Rebecca Roelofs, Mitchell Stern, Nati Srebro, and Benjamin Recht. The marginal value of adaptive gradient methods in machine learning. In *Advances in Neural Information Processing Systems*, pages 4148–4158, 2017.

Sho Yaida. Fluctuation-dissipation relations for stochastic gradient descent. *arXiv preprint arXiv:1810.00004*, 2018.

Jian Zhang and Ioannis Mitliagkas. Yellowfin and the art of momentum tuning. *arXiv preprint arXiv:1706.03471*, 2017.



(a) (b) (c) (d)

Figure 5: Evolution of the different statistics for SASA over the course of training ResNet18 on CIFAR-10 using the default parameters $\delta = 0.02, \gamma = 0.2, \zeta = 0.1$. Panel (a) shows the raw data for both sides of condition (6). That is, it shows the values of $\langle x^k, g^k \rangle$ and $\frac{\alpha}{2}\frac{1+\beta}{1-\beta}\langle d^k, d^k \rangle$ at each iteration. Panel (1) shows $\bar{z}_N$ with its lower and upper confidence interval $[lci, uci]$ and the "right hand side" (rhs) $(-\delta\bar{v}_N, \delta\bar{v}_N)$ (see Eqn. (10)). Panel (c) shows a zoomed-in version of (b) to show the drop points in more detail. Panel (d) depicts the different variance estimators (i.i.d., batch means, overlapping batch means) over the course of training. The i.i.d. variance (green) is a poor estimate of σ_z^2.

A Details and Additional Experiments

In this section, we provide more experimental details and discussion, and examine the sensitivity of SASA's performance with respect to its parameters.

A.1 CIFAR-10 experiments in Section 4

For the CIFAR-10 experiment, we set SGM to use $\alpha_0 = 1.0$ and $\zeta = 0.1$, and drop every 50 epochs. We used the same values of α_0 and ζ for SASA. For Adam, we used a warmup phase of 50 epochs with $\alpha = 1.0$, and then set $\alpha_0 = 0.0001$, the optimal value in our grid search of $\{0.00001, 0.0001, 0.01, 0.1\}$. The weight decay parameter for all models was set to 0.0005. All methods used batch size 128.

Evolution of statistics. Figure 5 shows the evolution of SASA's different statistics over the course of training the ResNet18 model on CIFAR-10 using the default parameter settings $\delta = 0.02, \gamma = 0.2, \zeta = 0.1$. In each phase, the running average of the difference between the statistics, $\bar{z}_N$, decays toward zero. The learning rate α drops once $\bar{z}_N$ and its confidence interval are contained in $(-\delta\bar{v}_N, \delta\bar{v}_N)$; see Eqn (10). After the drop, the statistics increase in value and enter another phase of convergence. The batch means variance estimator (BM) and overlapping batch means variance estimator (OLBM) give very similar estimates of the variance, while the i.i.d. variance estimator, as expected, gives quite different values.

Sensitivity analysis. We perturb the relative equivalence threshold δ, the confidence level γ and the decay rate ζ around their default values $(0.2, 0.02, 0.1)$ and repeat the CIFAR-10 experiment from the previous section, using the same values for the other hyperparameters. In Figure 6, the top row shows the performance for fixed $(\gamma, \zeta) = (0.2, 0.1)$ and changing δ. The middle row shows the performance for fixed $(\delta, \zeta) = (0.02, 0.1)$ and changing γ. The bottom row shows the performance for fixed $(\delta, \gamma) = (0.02, 0.2)$ and changing ζ. Increases in both δ and γ tend to cause the algorithm to drop sooner; this behavior is intuitive from the testing procedure (10). For values of the parameters close to the defaults, SASA still obtains good performance. The value of ζ does not influence the final performance, as long as the learning rate finally decays to the same level.

A.2 ImageNet experiments in Section 4

For the ImageNet experiment, we again used $\alpha_0 = 1.0$ and $\zeta = 0.1$ for SGM and SASA, and dropped the SGM learning rate every 30 epochs. We let Adam have a warmup phase of 30 epochs, initializing it with the parameters obtained from running SGM with $\alpha = 1.0$. After this phase, we used $\alpha_0 = 0.0001$, the optimal value from a grid $\{0.00001, 0.0001, 0.001, 0.01\}$. The weight decay for all models was set to 0.0001. All methods used batch size 256.

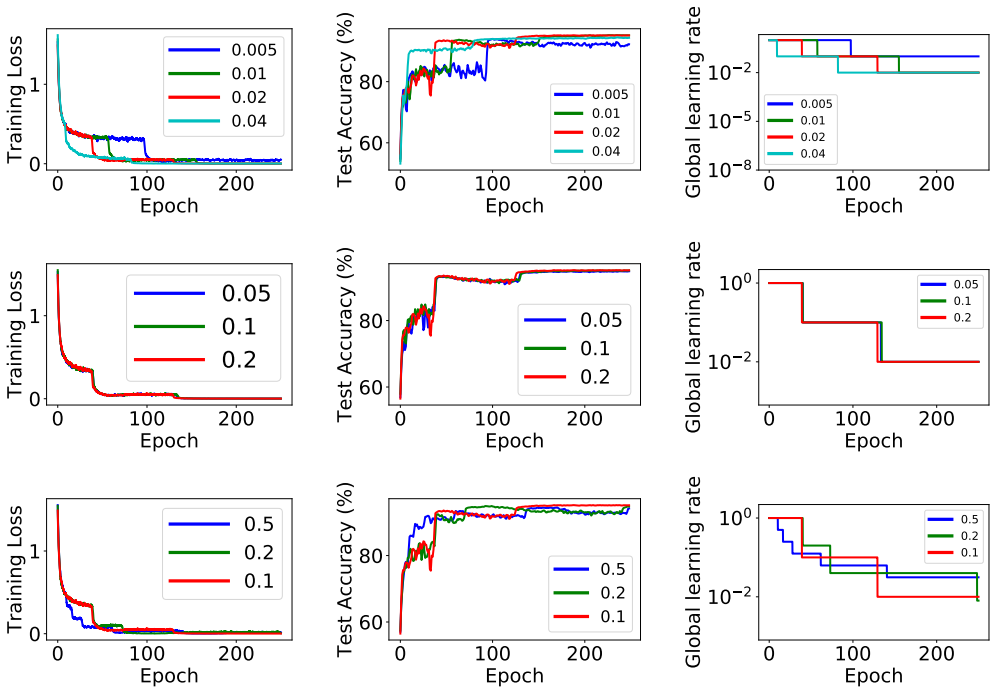


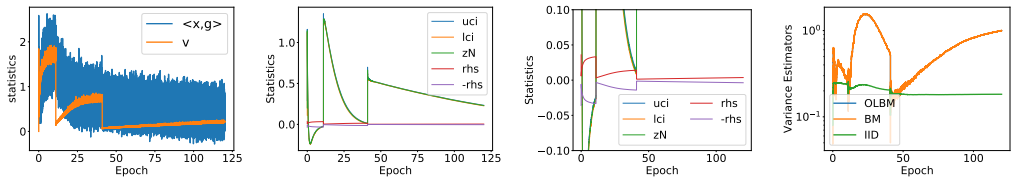
Figure 6: Training loss, test accuracy, and learning rate schedule for SASA using different values of γ, δ and ζ around the defaults 0.2, 0.02 and 0.1. The model is ResNet18 trained on CIFAR-10, with the procedure the same as in Section 4. Top row: performance for fixed $\gamma = 0.2, \zeta = 0.1$, and $\delta \in \{0.005, 0.01, 0.02, 0.04\}$. Middle row: performance for fixed $\delta = 0.02, \zeta = 0.1$, and $\gamma \in \{0.05, 0.1, 0.2\}$. Bottom row: performance for fixed $\gamma = 0.2, \delta = 0.02$, and $\zeta \in \{0.5, 0.2, 0.1\}$. Qualitatively, increasing δ and increasing γ both cause the algorithm to drop sooner. The value of ζ does not influence the final performance, as long as the learning rate finally decays to the same level.

Evolution of statistics. Figure 7 shows the evolution of SASA's different statistics over the course of training the ResNet18 model on CIFAR-10, under the default parameter setting $\delta = 0.02, \gamma = 0.2, \zeta = 0.1$. In each phase, z and v get close two easy other as predicted by (6). Together with its confidence interval, the statistics $\bar{z}_N$ decay toward zero. The learning rate is dropped as long as the confidence interval is contained in $(-\delta\bar{v}_N, \delta\bar{v}_N)$; see Eqn (10). The batch mean variance estimator (bm) and overlapping batch mean variance estimator (olbm) give very close variance estimates, while the i.i.d. variance estimator is clearly much different from the batch mean and overlapping batch mean estimators.

Sensitivity analysis. We perturb the relative equivalence threshold δ, the confidence level γ and the decay rate ζ around their default values $(0.2, 0.02, 0.1)$ and repeat the CIFAR-10 experiment from the previous section, using the same values for the other hyperparameters. In Figure 8, the top row shows the performance for fixed $(\gamma, \zeta) = (0.2, 0.1)$ and changing δ. The middle row shows the performance for fixed $(\delta, \zeta) = (0.02, 0.1)$ and changing γ. The bottom row shows the performance for fixed $(\delta, \gamma) = (0.02, 0.2)$ and changing ζ. Increases in both δ and γ tend to cause the algorithm to drop sooner; this behavior is intuitive from the testing procedure (10). For values of the parameters close to the defaults, SASA still obtains good performance. The value of ζ does not influence the final performance, as long as the learning rate finally decays to the same level.

A.3 RNN experiments in Section 4

For the RNN experiment, we trained the PyTorch word-level language model example (2019) with 600 hidden units, 600-dimensional embeddings, dropout 0.65, and tied weights. All optimizers also used gradient clipping with 2.0 as the threshold and weight decay 0.0005. We set α_0 and ζ for SGM and SASA to be 2.0 and 0.25, respectively. Because Adam was tuned in this example using the



(a) (b) (c) (d)

Figure 7: Evolution of the different statistics for SASA over the course of training ResNet18 on ImageNet using the default parameters $\delta = 0.02, \gamma = 0.2, \zeta = 0.1$. Panel (a) shows the raw data for both sides of condition (6). That is, it shows the values of $\langle x^k, g^k \rangle$ and $\frac{\alpha}{2}\frac{1+\beta}{1-\beta}\langle d^k, d^k \rangle$ at each iteration. Panel (b) shows $\bar{z}_N$ with its lower and upper confidence interval $[lci, uci]$ and the "right hand side" (rhs) $(-\delta\bar{v}_N, \delta\bar{v}_N)$ (see Eqn. (10)). Panel (c) shows a zoomed-in version of (b) to show the drop points in more detail. Panel (d) depicts the different variance estimators (i.i.d., batch means, overlapping batch means) over the course of training. The i.i.d. variance (green) is a poor estimate of σ_z^2.

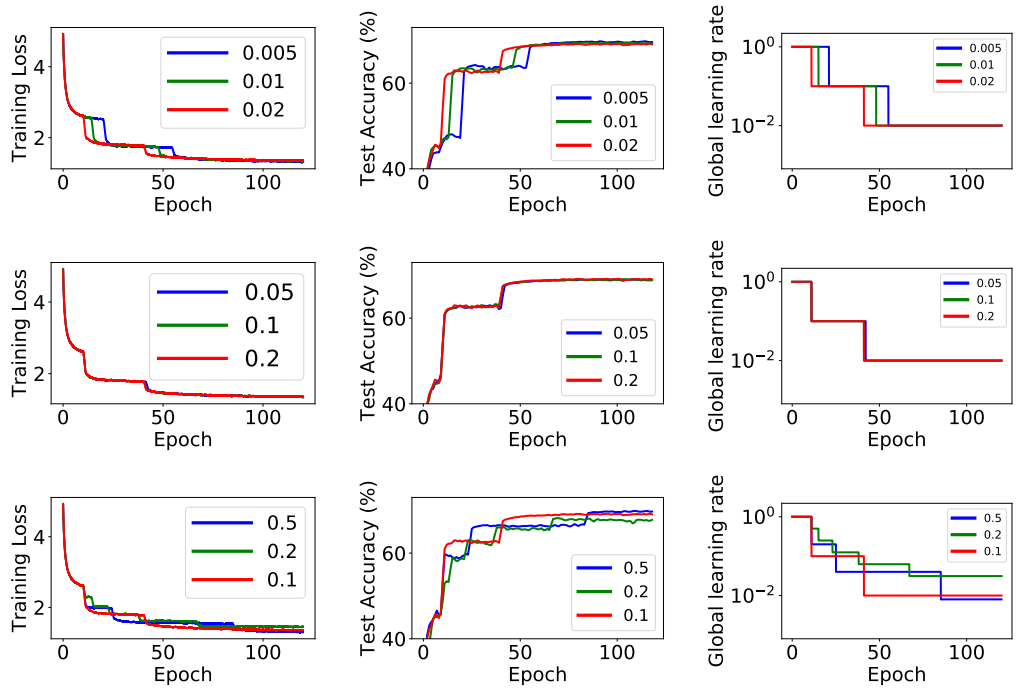


Figure 8: Training loss, test accuracy, and learning rate schedule for SASA using different values of γ, δ and ζ around the default 0.2, 0.02 and 0.1. The model is ResNet18 trained on ImageNet, as in 4. Top row: performance for fixed $\gamma = 0.2, \zeta = 0.1$, and $\delta \in \{0.005, 0.01, 0.02\}$. Middle row: performance for fixed $\delta = 0.02, \zeta = 0.1$, and $\gamma \in \{0.05, 0.1, 0.2\}$. Bottom row: performance for fixed $\gamma = 0.2, \delta = 0.02$, and $\zeta \in \{0.5, 0.2, 0.1\}$. Qualitatively, increasing δ and increasing γ both cause the algorithm to drop sooner. The value of ζ does not influence the final performance, as long as the learning rate finally decays to the same level.

validation set, we also used $\zeta = 0.25$ for Adam. The optimal α_0 for Adam was 0.5, chosen from the grid $\{0.1, 0.5, 1.0, 2.0, 3.0\}$.

A.4 Additional experiment: training logistic regression on the MNIST dataset

We train a logistic regression model on the MNIST dataset with weight decay 0.0005.

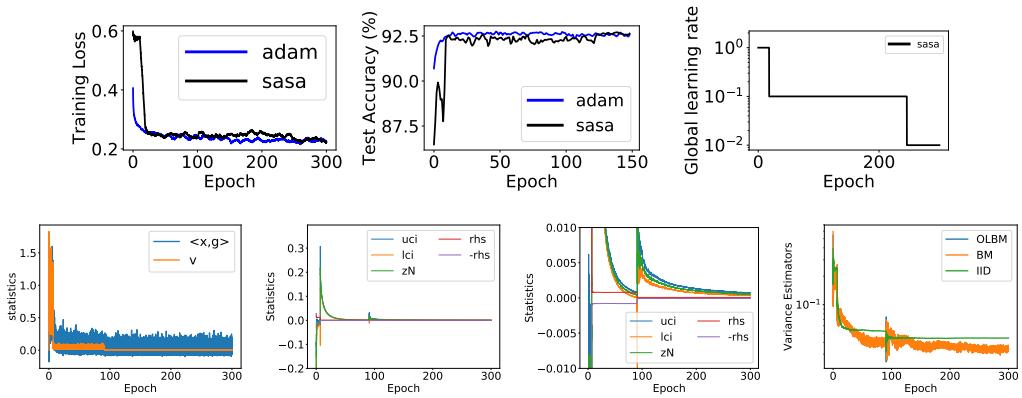


Figure 9: Top: training loss, test accuracy, and learning rate schedule for SASA and Adam for logistic regression on MNIST. Bottom: Evolution of the different statistics for SASA, as in Figures 5 and 7. SASA uses its default parameters $(\delta, \gamma, \zeta) = (0.02, 0.2, 0.1)$. Adam uses its default $(\beta_1, \beta_2) = (0.9, 0.999)$ but its initial learning rate $\alpha_0 = 0.00033$ is obtained from a grid search.

Default value performance. Figure 9 shows SASA's performance with default parameters. For this convex optimization problem, SASA and Adam achieve similar performance. SASA uses its default parameters $(\delta, \gamma, \zeta) = (0.02, 0.2, 0.1)$ and initial $\alpha_0 = 1.0$. Adam uses its default $(\beta_1, \beta_2) = (0.9, 0.999)$ and its initial learning rate $lr = 0.00033$ is obtained from a grid search over $\{0.01, 0.0033, 0.001, 0.00033, 0.0001\}$.

Sensitivity analysis. As with the experiments on CIFAR-10 and ImageNet, we perturb the relative equivalence threshold δ, the confidence level γ, and the decay rate ζ around their default values $(0.2, 0.02, 0.1)$. In Figure 10, the top row shows the performance for fixed $(\gamma, \zeta) = (0.2, 0.1)$ and changing δ. The middle row shows the performance for fixed $(\delta, \zeta) = (0.02, 0.1)$ and changing γ. The bottom row shows the performance for fixed $(\delta, \gamma) = (0.02, 0.2)$ and changing ζ. The results are the qualitatively the same as in Figures 6 and 8.

A.5 Additional experiment: training MaskRCNN on the COCO dataset

We train a Mask-RCNN model He et al. (2017) with a Feature Pyramid Network (FPN) Lin et al. (2017) as a backbone for both object detection and instance segmentation on the the COCO dataset Lin et al. (2014). The FPN backbone is based on the ResNet50, and the implementation is based on the MaskRCNN-benchmark repo Massa and Girshick (2018). In the recommend training setting, the model is trained for 90000 iterations with the SGM optimizer. The learning rate is scheduled to decay by 10 ($\zeta = 0.1$) at iteration 60000 and 80000. Readers can refer to `https://github.com/facebookresearch/maskrcnn-benchmark/blob/master/configs/e2e_mask_rcnn_R_50_FPN_1x.yaml` for a detailed experiment setup. This hyperparameter setting is carefully tuned to reach the reported performance: object detection mean average precision (bbox-AP) 37.8% and instance segmentation mean average precision (segm-AP) 34.2%; see `https://github.com/facebookresearch/maskrcnn-benchmark/blob/master/MODEL_ZOO.md`.

Default value performance. Figure 11 shows SASA's performance with default parameters. For this challenging task, SASA achieves a slightly better performance than the hand-tuned SGM optimizer *without any parameter tuning*. However, SASA with default parameters takes longer to achieve comparable performance, because SASA decides to decay the learning rate later than the hand-tuned SGM. Notice that SASA only decreases the learning rate once and already surpasses the performance of the hand-tuned SGM. We believe that if the learning rate is decreased again, the performance can be further improved. However, when the training reaches the maximum iteration 200000, the training loss is still constantly decreasing, so the dynamics have not reached a stationary distribution. This prevents SASA from decreasing its learning rate. Meanwhile, the model starts to overfit at this stage, which suggests that we should either decrease the learning rate or stop the

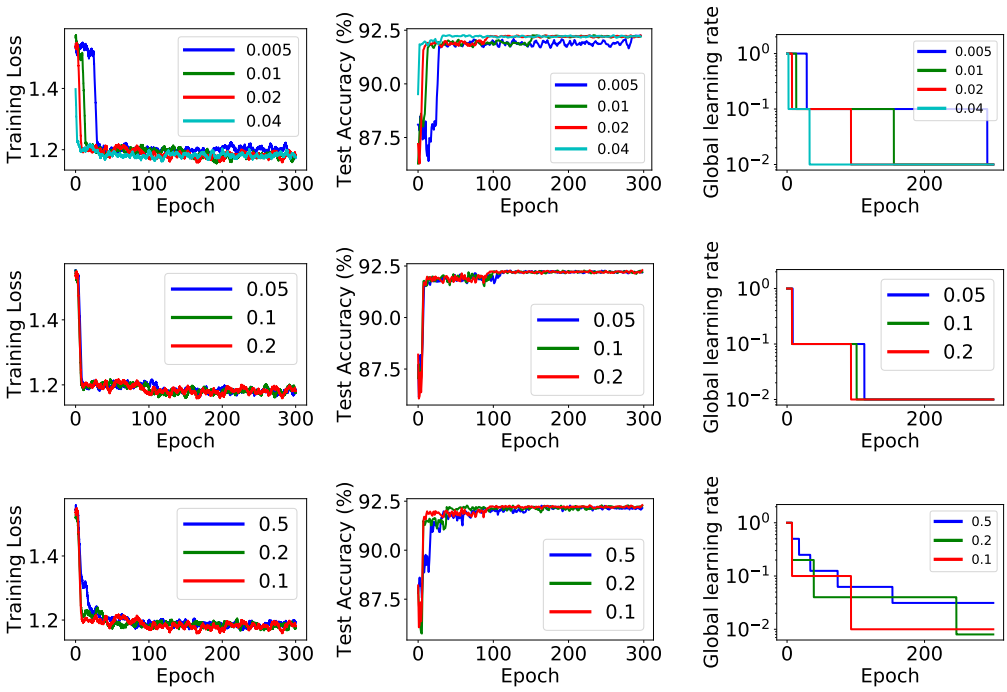


Figure 10: Training loss, test accuracy, and learning rate schedule for SASA using different values of γ, δ and ζ around the default 0.2, 0.02 and 0.1. The model is the logistic regression trained on MNIST. Top row: performance for fixed $\gamma = 0.2, \zeta = 0.1$, and $\delta \in \{0.005, 0.01, 0.02, 0.04\}$. Middle row: performance for fixed $\delta = 0.02, \zeta = 0.1$, and $\gamma \in \{0.05, 0.1, 0.2\}$. Bottom row: performance for fixed $\gamma = 0.2, \delta = 0.02$, and $\zeta \in \{0.5, 0.2, 0.1\}$. Qualitatively, increasing δ and increasing γ both cause the algorithm to drop sooner. The value of ζ does not influence the final performance, as typically the learning rate automatically decays to the same level.

training. As mentioned in Section 5, a combination of stationary detection (SASA) and overfitting detection is a promising direction toward a fully automated optimizer.

B Comparison with Yaida's test

The variance experiment in Figure 4 can be interpreted as showing that for a fixed testing frequency M, the statistical procedure (10) is more robust to changes in the noise level of the samples than the heuristic test (9). We essentially repeat this experiment in Figure 12, which shows the performance of the two testing methods (9) and (10) on a logistic regression model trained on MNIST. We used the same procedure as in all other logistic regression experiments, except with batch size one. We test the statistics every $M = 100$ iterations and plot the results of ten independent runs for each method, using a fixed M as in Figure 4. While the final training and test loss for the two methods are similar (92.7% ±0.16 for SASA, 92.7% ±0.2 for (9)), the variance in the learning rate schedules for Yaida's method is dramatically higher. On strongly convex problems, this may not cause poor performance, but as shown in Figure 4, it can cause dramatic results in more general settings. This experiment gives a further indication that when using a fixed test frequency M, explicitly accounting for the variance in $\bar{z}_N$, as in SASA, is critical for robust performance. Finally, Figure 13 shows a complementary result on CIFAR-10: even when the batch size is large (128), the statistical approach is less sensitive to using a small testing frequency M. While this effect is on a much smaller scale than the others, it indicates that Yaida's heuristic (performing the test (9) once per epoch) is more sensitive than SASA to the choice of the testing frequency.

Figure 4, Figure 12, and Figure 13 indicate that the statistical test is more robust to changes in noise and testing frequency than Yaida's deterministic ratio test. However, Figure 13 indicates that this method can obtain similar (albeit less robust) performance on large deep learning datasets, and our

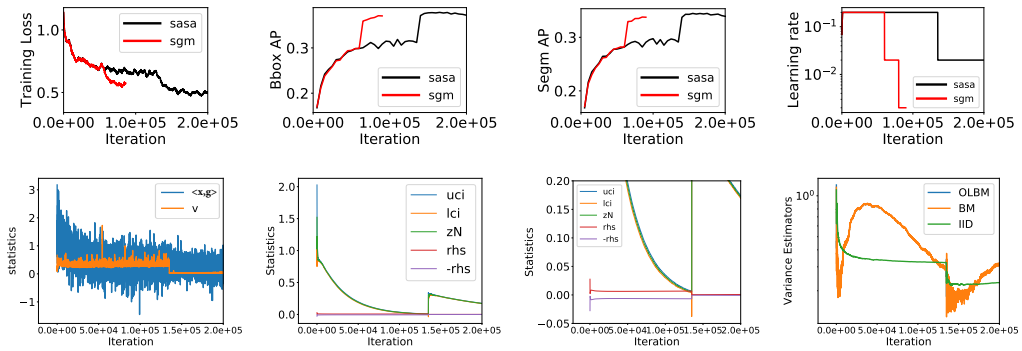
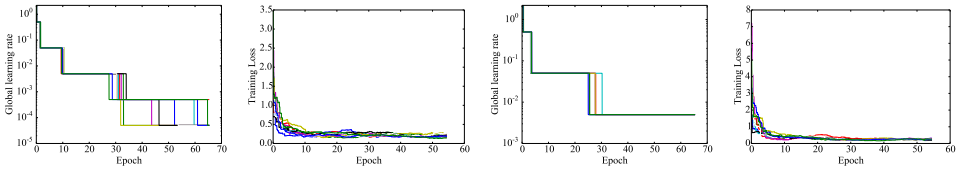


Figure 11: Top: training loss, test accuracy, and learning rate schedule for SASA and SGM for MaskRCNN training on COCO. Bottom: Evolution of the different statistics for SASA, as in Figures 5 and 7. SASA uses its default parameters $(\delta, \gamma, \zeta) = (0.02, 0.2, 0.1)$. The SGM is scheduled to decay the learning rate by 10 ($\zeta = 0.1$) twice, once at iteration 60000 and once at iteration 90000. SASA takes more iterations (double) to reach a slightly better performance without any parameter tuning.



(a) (b) (c) (d)

Figure 12: Variance in learning rate schedule and training loss for the two tests (9) (Panels (a)-(b)) and (10) (Panels (c)-(d)) for a logistic regression model on MNIST, using batch size one and test frequency $M = 100$ iterations. Ten independent runs are shown for each method. With the same value of δ, the variance in the learning rate schedule for Yaida's method (9) is much higher.

practical results can be taken more generally as large-scale evidence that methods for detecting stationarity have good practical performance when used as adaptive optimizers. Still, our formulation recovers Yaida's when $\gamma = 1$, heuristics like "test once per epoch" are not always available—such as in an online training setting—so robustness to the test frequency M is desirable, and we have demonstrated that SASA is less sensitive to noise in several regimes, such as small batch size and high test frequency. For these reasons we believe SASA will be more robust in practice, and we hope it leads to more research on using statistical tests in optimization.

C Generalized Pflug condition and comparison with Yaida's condition

In this section, we provide a generalization of Pflug's stationary condition to the case of SGM for quadratic functions. We also compare the two stationary conditions (Pflug's and Yaida's) and show that Yaida's stationary condition works much better for practical machine learning problems.

C.1 Derivation of the generalized Pflug stationary condition

As in (Pflug, 1990; Mandt et al., 2017), the derivation is based on two assumptions:

1. The quadratic objective assumption:

$$F(x) = (1/2)x^T Ax, \tag{12}$$

   where A is positive definite.

2. The i.i.d. additive noise assumption:

$$g^k = \nabla F(x^k) + \xi^k, \tag{13}$$

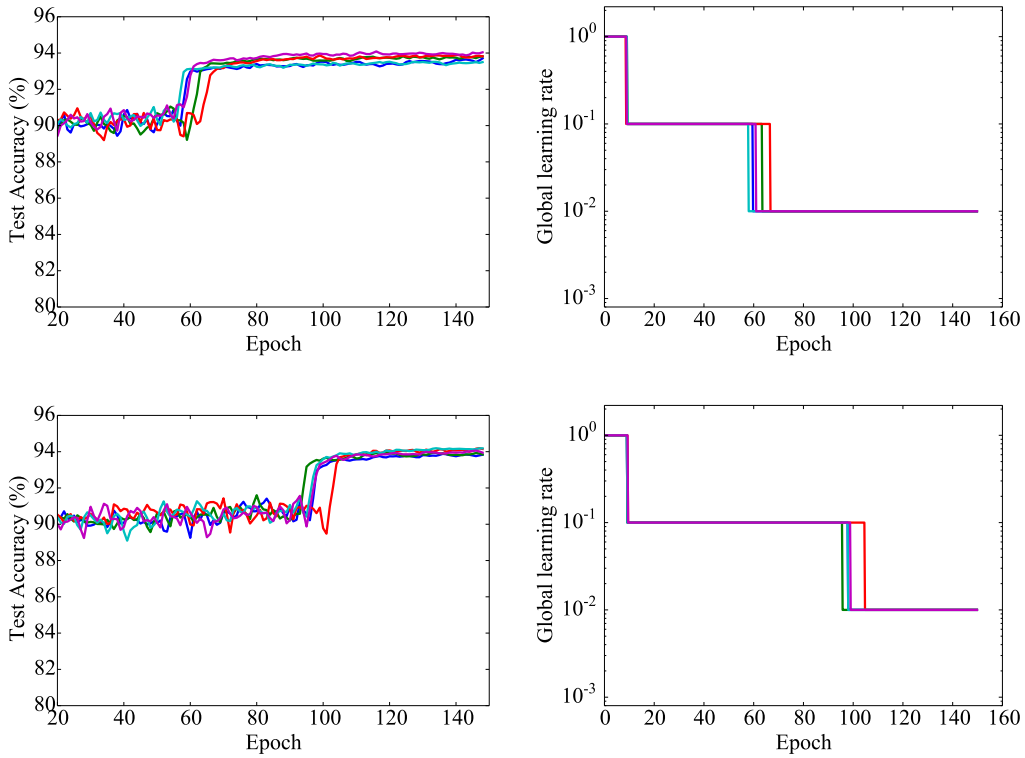


Figure 13: Test accuracy and learning rate schedule when using Yaida's ratio test (9) (top row) and our statistical test (10) (bottom row) with $M = 10$, $\delta = 0.02$, and with SASA using $\gamma = 0.2$. The standard deviation of both the learning rate drops and the best test set performance is higher for Yaida's test: 3.2 epochs vs 2.9 epochs, and 0.17% vs 0.12%. The mean performance of the statistical test is also marginally higher, 94.08% vs 93.84% test accuracy.

where ξ^k is independent of x^k, and for all $k \geq 0$ satisfies

$$\mathbf{E}\left[\xi^k\right] = 0, \qquad \mathbf{E}\left[\xi^k (\xi^k)^T\right] = \Sigma_\xi. \tag{14}$$

Mandt et al. (2017) observe that this noise assumption can hold approximately when α is small and the dynamics of SGM are approaching stationarity around a local minimum.

For the dynamics of SGM with constant α and β, i.e., (2), the sequence $\{(x^k, d^k, g^k)\}$ is assumed to converge to a stationary distribution $\pi(x, d, g)$, as we defined in Section 2. We denote x's covariance matrix under the stationary distribution as

$$\Sigma_x = \lim_{k\to\infty} \mathbf{E}\left[x^k (x^k)^T\right]. \tag{15}$$

The following theorem characterizes the dependence of Σ_x on A, α and β. It also derives an asymptotic expression of $\mathbf{E}_\pi[\langle g, d\rangle]$ in terms of A, α and β.

Theorem 2. *Suppose $F(x) = (1/2)x^T Ax$, where A is positive definite with maximum eigenvalue L, and g^k satisfies* (13) *and* (14)*. If we choose $\alpha \in (0, 1/L)$ and $\beta \in [0, 1)$ in* (2)*, then Σ_x defined in* (15) *exists. Moreover, we have*

$$A\Sigma_x + \Sigma_x A = \alpha\Sigma_\xi + O(\alpha^2) \tag{16}$$

and

$$\mathbf{E}_\pi[\langle g, d\rangle] = -\frac{\alpha(1-\beta)}{2(1+\beta)}\mathbf{tr}(A\Sigma_\xi) + O(\alpha^2). \tag{17}$$

Theorem 2 states that when α is small, we can approximate Σ_x by solving the linear equation $A\Sigma_x + \Sigma_x A = \Sigma_\xi$. Moreover, the variance $\mathbf{tr}(\Sigma_x)$ decreases to zero as $\alpha \to 0$. It is well known that

larger β often leads to faster transient convergence when SGM is far away from a local minimum. According to (16), it does not affect the covariance in steady state, especially for small α.

Equation (17) implies that for small α, the vectors g^k and d^k will eventually have negative correlation. Interestingly, their correlation is less negative for larger β.

Assuming ergodicity, $\mathbf{E}_\pi[\langle g, d\rangle]$ can be evaluated by the history average

$$\mathbf{E}_\pi\big[\langle g^k, d^k\rangle\big] \approx \frac{1}{N}\sum_{i=k+1}^{k+N}\langle g^i, d^i\rangle, \tag{18}$$

where N can be chosen to control the quality of estimation. If an online estimate of $\mathbf{tr}(A\Sigma_\xi)$ is also available, then we can check if the relation established in (17) holds in a statistical sense, which serves as a test of stationarity.

Since we do not assume any knowledge of A or Σ_ξ, it can be hard to estimate $\mathbf{tr}(A\Sigma_\xi)$ using simple statistics. To address this challenge, Pflug (1983) constructed a novel scheme that requires three stochastic gradients at each iteration. Specifically, at each iteration k, we first compute two stochastic gradients g_1^k and g_2^k of F at x^k, and we let $r^k = (g_1^k - g_2^k)/2$ (in a data-parallel training setting, g_1^k and g_2^k can be computed from separate processing units, and thus can be obtained without extra delay). Next, we let $\tilde{x}^k = x^k + \alpha r^k$ and compute another stochastic gradient $\tilde{g}^k$ of F at $\tilde{x}^k$. Then, it can be shown (Pflug, 1983) that

$$\mathbf{E}[\langle r^k, \tilde{g}^k\rangle] = \frac{\alpha}{2}\mathbf{tr}(A\Sigma_\xi). \tag{19}$$

We can thus obtain an online estimate of $\mathbf{tr}(A\Sigma_\xi)$ using the running average of $\langle r^k, \tilde{g}^k\rangle$ in a similar way to (18).

As suggested by Pflug (1983), a less wasteful use of the stochastic gradients is to define $g^k = (g_1^k + g_2^k)/2$ and use it in (2). This averaging reduces the covariance of g^{k+1} and d^{k+1} by a factor of $1/2$, which together with (17) implies

$$\mathbf{E}_\pi\big[\langle g, d\rangle\big] \approx -\frac{\alpha(1-\beta)}{4(1+\beta)}\mathbf{tr}(A\Sigma_\xi), \tag{20}$$

where we still use π to denote the new stationary condition. Combining (19) and (20), we conclude that for small α,

$$\mathbf{E}_\pi\big[\langle g, d\rangle\big] \approx -\frac{1-\beta}{2(1+\beta)}\mathbf{E}\big[\langle r^k, \tilde{g}^k\rangle\big] \tag{21}$$

holds if the dynamics (2) reach stationarity. Both sides of (21) can be estimated by the history average during the training, thanks to ergodicity.

Unfortunately, evaluating this estimator requires 33% more training iterations than regular SGM due to the stochastic gradients used to compute the point $\tilde{x}^k$.

C.2 Comparing stationary conditions

Figure 14 evaluates the two stationary conditions (5) and (6) by training an L2-regularized logistic regression model on MNIST and logging the estimators for both sides of each relation. The top row shows that even when the number of iterations grows large, there is still non-negligible error in Pflug's condition even though the function is strongly convex. Contrastingly, the statistics in Yaida's relation, shown in the bottom row, quickly become almost indistinguishable, as predicted by (6) and (4). Together with the difficulty of estimating its right-hand-side, this inaccuracy makes the Pflug condition unattractive for quantitative applications such as ours, which require a precise relationship to hold. However, the *qualitative* intuition given by such quadratic stationary formulae has proven useful (Mandt et al., 2017).

D Additional SASA discussion

D.1 The missing step to derive the stationary condition (6)

Assuming the existence of a stationary condition $\pi(d, x, g)$ for the SGM dynamics (2), Yaida (2018) showed

$$\mathbf{E}_\pi[\langle x, \nabla F(x)\rangle] = \frac{\alpha}{2}\frac{1+\beta}{1-\beta}\mathbf{E}_\pi[\langle d, d\rangle]. \tag{22}$$

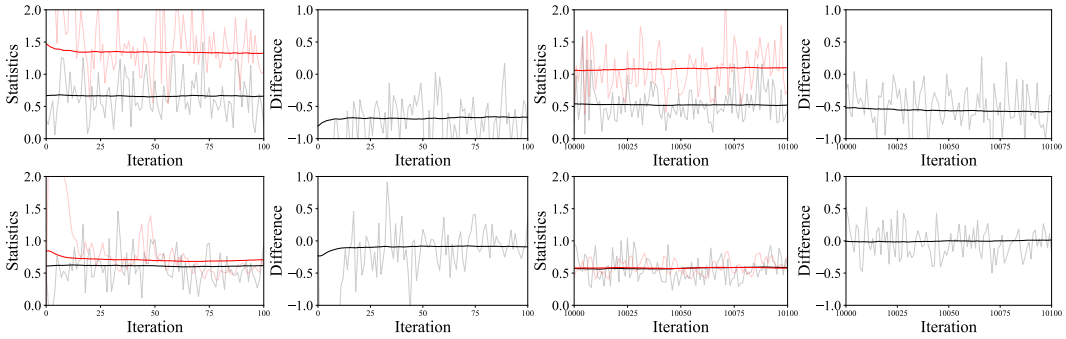


Figure 14: The two conditions (5) (top) and (6) (bottom) evaluated on a logistic regression model trained on MNIST, with $\alpha = 1.0$. Left two columns: iteration 0-100; Right two columns: iteration 10000-10100. In the statistics plots, the red and black curves (dark) are the running estimates of the left-hand and right-hand side of each condition, respectively. The light curves show the raw value of each estimator at each iteration. Even when the number of iterations is very large, the statistics suggested by (5) still do not match. On the other hand, the difference between the statistics in (6) quickly converges to zero.

Using history average to estimate the left hand side needs the full gradient of F, which is not available (or expensive to compute) during training. Instead, both Yaida (2018) and SASA use (6) in practice, i.e.,

$$\mathbf{E}_\pi[\langle x, g\rangle] = \frac{\alpha}{2}\frac{1+\beta}{1-\beta}\mathbf{E}_\pi[\langle d, d\rangle], \tag{23}$$

where the left hand side can be estimated with nearly no computational overhead. Here, we provide the missing step from (22) to (23).

By the law of total probability, we have

$$\mathbf{E}_\pi[\langle x, g\rangle] = \mathbf{E}_\pi\left[\mathbf{E}_\pi[\langle x, g\rangle | x, d]\right] = \mathbf{E}_\pi\left[\langle x, \mathbf{E}_\pi[g|x, d]\rangle\right].$$

We denote the time-independent transition kernel from (x^k, d^k, g^k) to $(x^{k+1}, d^{k+1}, g^{k+1})$ in (2) as $\mathbf{T}$. Then since π is the stationary distribution, we have the pushforward measure of π under $\mathbf{T}$ is still π, i.e., $\mathbf{T}^\sharp\pi = \pi$. Then we have

$$\mathbf{E}_\pi[g|x, d] = \mathbf{E}_{\mathbf{T}^\sharp\pi}[g|x, d] \overset{(*)}{=} \mathbf{E}_{\mathbf{T}^\sharp\pi}[g(x)|x, d] = \nabla F(x),$$

where the definition of the transition kernel (2) is used in the step (*) and the unbiasedness of the stochastic gradient, see Eqn. (3), in the last step.

D.2 Discussion on the multiple-test problem

Although SASA performs sequential hypothesis testing, it does not seem to suffer from the issue of inflated false discovery rate McDonald (2009). That is, we do not observe that the test fires earlier than it "should" in our numerical experiments. From Figure 5, we can see that the statistic $\bar{z}_N$ is either monotonically decreasing to 0 or first decreasing and then increasing to 0, leading to high positive correlation among the tests. This high correlation may prevent proportional inflated false discovery rates; see, e.g., Benjamini and Hochberg (1995); Blanchard and Roquain (2009); Lindquist and Mejia (2015).