Scheduling Hyperparameters to Improve Generalization: From Centralized SGD to Asynchronous SGD

JIANHUI SUN, University of Virginia
YING YANG, University of Michigan
GUANGXU XUN, Baidu Research
AIDONG ZHANG, University of Virginia

This article[1] studies how to schedule hyperparameters to improve generalization of both *centralized* single-machine stochastic gradient descent (SGD) and *distributed* asynchronous SGD (ASGD). SGD augmented with momentum variants (e.g., heavy ball momentum (SHB) and Nesterov's accelerated gradient (NAG)) has been the default optimizer for many tasks, in both centralized and distributed environments. However, many advanced momentum variants, despite empirical advantage over classical SHB/NAG, introduce extra hyperparameters to tune. The error-prone tuning is the main barrier for AutoML.

***Centralized* SGD:** We first focus on *centralized* single-machine SGD and show how to efficiently schedule the hyperparameters of a large class of momentum variants to improve generalization. We propose a unified framework called multistage quasi-hyperbolic momentum (Multistage QHM), which covers a large family of momentum variants as its special cases (e.g., vanilla SGD/SHB/NAG). Existing works mainly focus on only scheduling learning rate α's decay, while multistage QHM allows additional varying hyperparameters (e.g., momentum factor), and demonstrates better generalization than only tuning α. We show the convergence of multistage QHM for general non-convex objectives.

***Distributed* SGD:** We then extend our theory to distributed asynchronous SGD (ASGD), in which a parameter server distributes data batches to several worker machines and updates parameters via aggregating batch gradients from workers. We quantify the asynchrony between different workers (i.e., gradient staleness), model the dynamics of asynchronous iterations via a stochastic differential equation (SDE), and then derive a PAC-Bayesian generalization bound for ASGD. As a byproduct, we show how a moderately large learning rate helps ASGD to generalize better.

Our tuning strategies have rigorous justifications rather than a blind trial-and-error as we theoretically prove why our tuning strategies could decrease our derived generalization errors in both cases. Our strategies simplify the tuning process and beat competitive optimizers in test accuracy empirically. Our codes are publicly available https://github.com/jsycsjh/centralized-asynchronous-tuning.

[1] A preliminary version titled "A Stagewise Hyperparameter Scheduler to Improve Generalization" appeared in Proceedings of the 27th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining (KDD 2021).

This work is supported in part by the US National Science Foundation under grants IIS-2106913, 2008208, 1955151, 1934600, 1938167. Any opinions, findings, and conclusions or recommendations expressed in this material are those of the author(s) and do not necessarily reflect the views of the National Science Foundation.
Authors' addresses: J. Sun and A. Zhang, University of Virginia, Charlottesville, VA 22904; emails: {js9gu, aidong}@virginia.edu; Y. Yang, University of Michigan, Ann Arbor, MI 48104; email: yingyan@umich.edu; G. Xun, Baidu Research, Sunnyvale, CA 94089; email: guangxuxun@baidu.com.

1556-4681/2023/03-ART29
https://doi.org/10.1145/3544782

CCS Concepts: • **Theory of computation** → **Sample complexity and generalization bounds**; • **Mathematics of computing** → *Non-convex optimization;*

Additional Key Words and Phrases: Deep learning optimization, hyperparameter tuning, SGD momentum, multistage QHM, asynchronous SGD, generalization bound

ACM Reference format:
Jianhui Sun, Ying Yang, Guangxu Xun, and Aidong Zhang. 2023. Scheduling Hyperparameters to Improve Generalization: From Centralized SGD to Asynchronous SGD. *ACM Trans. Knowl. Discov. Data.* 17, 2, Article 29 (March 2023), 37 pages.
https://doi.org/10.1145/3544782

1 INTRODUCTION

Most machine learning and data mining tasks could be formulated as the following optimization problem:

$$\min_{\theta} \mathcal{R}(\theta) = \min_{\theta} \mathbb{E}_{(x_i, y_i)\sim\mathcal{D}} l_\theta(x_i, y_i), \tag{1}$$

where θ, $\mathcal{D}$, and $l_\theta(x_i, y_i)$ are the trainable parameter, data distribution, and loss function, respectively. Generalization ability, the most important measure of learning models, depends heavily on whether the optimizer is able to reliably find a solution of Equation (1) that could generalize well to unseen test instances.

This article studies the hyperparameter tuning of **stochastic gradient descent** (**SGD**) and its variants, which minimizes the objective by iteratively moving the parameters along its negative gradient direction, in both *centralized* setting, where both gradient computation and parameter updating are completed with one single processor, and *distributed* setting, where gradient computation is distributed among different workers who communicate with a parameter server to update parameters. As first-order gradient is easy to compute and is evaluated only with a mini-batch of data instances, the per-iteration computational efficiency makes SGD the top candidate in both *centralized* and *distributed* settings.

However, centralized and distributed SGD each poses different challenges to tuning its hyperparameters, whose configuration is known to have a huge impact on the quality of solutions found by optimizers for deep neural networks [8]. In a centralized setting, a large number of new SGD variants which introduce more complicated updating rules than vanilla SGD and extra hyperparameters to tune, have no general tuning rules that are guaranteed to apply to a majority of cases. There is a very limited number of SGD variants in distributed machine learning, as most variants can not maintain their superiority over vanilla SGD as in centralized computing. However, in distributed setting, asynchrony, which indicates the inconsistent view of the optimized parameters among worker machines when conducting local computations, makes the dynamics of SGD much more challenging than its centralized counterpart to analyze [5] and how hyperparameters affect SGD in even its vanilla form is still a bit of mystery. The most reliable hyperparameter tuning strategy is thus still a comprehensive grid search, and is probably the most time-consuming part of training. This article aims at tackling the above problems in both scenarios.

1.1 Motivation

***Centralized* SGD.** Vanilla SGD tends to converge slowly especially when it gets closer to local minima. Therefore, a momentum term, which incorporates past gradient estimates into the current update, is often augmented to SGD to accelerate the convergence around stationary points, pioneered by Polyak's **Heavy Ball momentum** (**SHB**) [63] and **Nesterov's Accelerated Gradient** (**NAG**) [60].

SGD+momentum is the default optimizer in centralized computing as it converges faster than vanilla SGD and generalizes better than adaptive gradient methods (e.g., Adam [39]) [77, 80]. A large number of new momentum variants have been proposed in recent years, many of which modify the classical SHB and NAG to accommodate more complex task-dependent objective functions, and have achieved state-of-the-art performances across domains, e.g., **Synthesized Nesterov Variants** (**SNV**) [44], PID control [4], Triple Momentum [78], **Accelerated Stochastic Gradient Method** (**AccSGD**) [38], and **Quasi-Hyperbolic Momentum** (**QHM**) [53]. However, the influx of newly proposed momentum variants casts the following challenges on tuning:

—New momentum variants, despite their empirical superiority, often have more complex updating rules than SGD/SHB/NAG and introduce more hyperparameters.
—Hyperparameters are typically not fixed throughout the entire learning process. Scheduling hyperparameters appropriately is nearly as important as assigning the initial values to them [80].
—A lack of unified analysis that could cover different momentum variants with one single framework makes it difficult to learn from existing tuning strategies from SGD/SHB/NAG.
—Existing tuning strategies are mainly from a trial-and-error process with no theoretical insight of the inner mechanism, which is thus tedious and error-prone.

***Distributed* SGD.** The main enabler of recent advances in deep learning is models and data of extreme size [15, 16, 25, 33]. Though *centralized* SGD and its variants, in which all gradient computation and parameter updating are completed with one single processor, are still widely used in most research settings, parallel and distributed computing is now the status quo in most industrial deep learning deployments by massive companies like Google or Microsoft [12, 13], as no single machine is able to deal with potentially billions of parameters and data instances. A popular approach to compute on multiple machines simultaneously is "data-parallelism", in which data are divided into separate batches which are distributed to different worker machines during training [19]. A parameter server keeps a record of the parameters to be optimized, divides computation labors to worker machines, and aggregates computed updates from different workers.

This article focuses on **Asynchronous Stochastic Gradient Descent** (**ASGD**). In contrast to **Synchronous SGD** (**SSGD**) where updates only happen when all workers complete the current round of computations, in ASGD, the parameter server updates the parameter every time some worker completes its current assigned batch gradient computation, and propagates the new parameter only to that worker without waiting other workers (more details in Section 2.2). SSGD enforces all processors to perform local gradient calculations with the same optimization parameter, which in turn makes faster machines experience significant amounts of idle waiting time. ASGD ensures a better utilization of computational resources, while complicating the analysis of its dynamics with the phenomenon of gradient staleness:

*Each worker calculates its batch gradient with an outdated parameter, with no knowledge that the parameter server may have already updated the parameter several times via communicating with other faster workers. Therefore, the parameter server nearly always receives some "delayed" gradient.

This asynchrony obfuscates our understanding of how basic hyperparameters (e.g., learning rate and batch size) impact the generalization ability of ASGD. Considering every round of tuning is extremely costly as ASGD usually works with data and models of massive size, this article aims at connecting the generalization performance with hyperparameters to reduce the tuning time.

1.2 Contribution

To tackle the aforementioned challenges, this article derives a PAC-Bayesian generalization bound for both *centralized* and *distributed* SGD. In a practical manner, this bound is able to provide an efficient tuning pipeline to relieve practitioners of the labor-intensive tuning process and improve the optimizer's generalization ability at the same time.

For centralized computing, we select the multistage version of a powerful momentum scheme, QHM [53] to analyze, as it includes many popular momentum variants as its special cases. We propose multistage QHM in Section 4, where we adjust all involving hyperparameters stagewise, by extending the idea that practitioners have long used to schedule learning rate in SGD, "constant and drop",[2] to all hyperparameters involved in QHM (learning rate α, momentum parameter β, instant discount factor ν, and batch size b). Existing works are unclear about how to schedule (α, β, ν, b) appropriately or whether adjustment of any of them could contribute to better generalization.

For distributed computing, we derive a PAC-Bayesian generalization bound by modeling the continuous-time dynamics of ASGD with **stochastic differential equations** (**SDEs**) and studying its stationary distribution, which is the first generalization bound on ASGD to our knowledge (see Sections 3 and 7 for details). We theoretically and empirically show how a moderately large learning rate could help ASGD generalize better.

Our contribution could be summarized as follows:

(1) We propose practical tuning guidelines in both *centralized* and *distributed* computing environments. Our proposed methods effectively restrict the search space of hyperparameters, and help automate the tuning process.
(2) We provide theoretical evidence on whether and why our proposed methods improve generalizability. Our derived generalization bounds are of independent interest. To our best knowledge, this is the first hyperparameter tuning pipeline that is applicable to a large class of optimizers with a sound theoretical guarantee.

1.3 Article Organization

In Section 2, we introduce the background and notation. In Section 3, we elaborate the logical framework and proof schema of deriving the generalization error, which is essential to this article, before we move on to present our main results. In Section 4, we present our proposed approach, multistage QHM, to tune a large class of momentum variants in *centralized* computing environment. In Section 5, we prove Theorem 3 to show how varying hyperparameters affect the generalization bound of multistage QHM, and then based on the theoretical findings, propose the exact paradigm for our stagewise scheduler. In Section 6, we provide the convergence guarantee for multistage QHM optimizing non-convex objectives. In Section 7, we extend to distributed parallel computing regime, generalize our theory to Asynchronous SGD, and show how a moderately large learning rate helps ASGD generalize. In Section 8, extensive experiments are presented and show the empirical advantage of our proposed tuning strategies. We discuss relevant works in Section 9. In Section A, we provide detailed proofs that are omitted in the main text.

2 BACKGROUND

In this section, we introduce QHM, ASGD, and generalization error, which are pertinent to this work.

[2] "Constant and drop" essentially divides the entire training process into several stages, where initial learning rate is set to be large for faster convergence and held constant for a number of epochs, and is dropped with a fixed rate (or exponentially) at the end of every stage. "Constant and drop" step size is so popular that a number of optimizers use it by default in off-shelf softwares (e.g., PyTorch [62] or Tensorflow [1]).

2.1 Centralized Quasi-hyperbolic Momentum

Let $\{z_i = (x_i, y_i)\}_{i=1}^N$ represent the training set. The expected risk function is defined as $\mathcal{R}(\theta) \triangleq \mathbb{E}_{z\sim\mathcal{D}} l_\theta(z)$, where $l_\theta(z)$ is the loss function associated with model parameter θ and data instance z. Empirical risk $\mathcal{R}_\zeta(\theta)$ is an unbiased estimator of the expected risk function, and is defined as $\mathcal{R}_\zeta(\theta) \triangleq \frac{1}{b}\sum_{j\in\zeta}\mathcal{R}_j(\theta)$, where $\mathcal{R}_j(\theta) \triangleq l_\theta(z_j)$ is the contribution to risk from jth data point. ζ represents a mini-batch of random samples and $b \triangleq |\zeta|$ represents the batch size. Similarly, we define $\nabla_\theta\mathcal{R}$, $\nabla_\theta\mathcal{R}_j$, and $\nabla_\theta\mathcal{R}_\zeta$ as their gradients, respectively. We denote the empirical gradient as $\hat{g}(\theta) \triangleq \nabla_\theta\mathcal{R}_\zeta$ and exact gradient as $g(\theta) \triangleq \nabla_\theta\mathcal{R}$ for the simplicity of notation. We assume $\mathbb{E}[\|\hat{g}(\theta) - g(\theta)\|^2] \le \sigma^2$ (bounded noise is a standard assumption [20, 57]).

We start from the following updating rule of SGD:

$$\theta_{k+1} = \theta_k - \alpha_k d_k, \tag{2}$$

where α_k and d_k are the learning rate and search direction at kth step, respectively. (mini-batch) SGD[3] uses $\hat{g}_k \triangleq \hat{g}(\theta_k)$ as d_k.

We focus on QHM methods [53], which could be formulated as

$$\begin{aligned} d_{k+1} &= (1-\beta_k)\hat{g}_k + \beta_k d_k, \\ \theta_{k+1} &= \theta_k - \alpha_k[(1-\nu_k)\hat{g}_k + \nu_k d_k]. \end{aligned} \tag{3}$$

Note that the formulation of the QHM method is very general, and could recover many momentum methods with different specifications of $(\alpha_k, \beta_k, \nu_k)$. For example, QHM recovers plain SGD when $\nu_k = 0$.

If $\nu_k = 1$, QHM recovers SHB:

$$\begin{aligned} d_{k+1} &= (1-\beta_k)\hat{g}_k + \beta_k d_k, \\ \theta_{k+1} &= \theta_k - \alpha_k d_k, \end{aligned} \tag{4}$$

where variable d is commonly referred to as the "momentum buffer". The exponential discount factor β_k controls how slowly the momentum buffer is updated.

If $\nu_k = \beta_k$, QHM recovers NAG:

$$\begin{aligned} d_{k+1} &= (1-\beta_k)\hat{g}_k + \beta_k d_k, \\ \theta_{k+1} &= \theta_k - \alpha_k[(1-\beta_k)\hat{g}_k + \beta_k d_k]. \end{aligned} \tag{5}$$

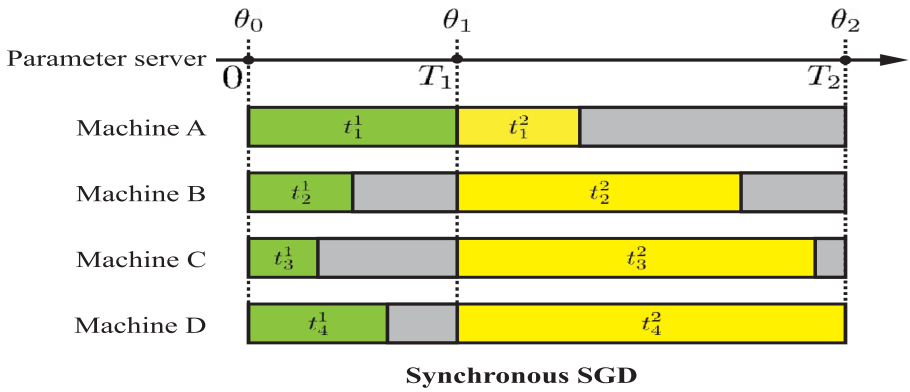
From the connection between QHM and SHB, NAG, QHM could be interpreted as a ν_k-weighted average of the momentum update step and the plain SGD update step. ν_k is referred to as an instant discount factor.

Reference [53] showed that QHM could recover many other popular momentum schemes, e.g., PID control, SNV, ASGD, and Triple Momentum, with different α_k, β_k, ν_k specifications. Therefore, our analysis based on QHM could cover a family of momentum methods as special cases.

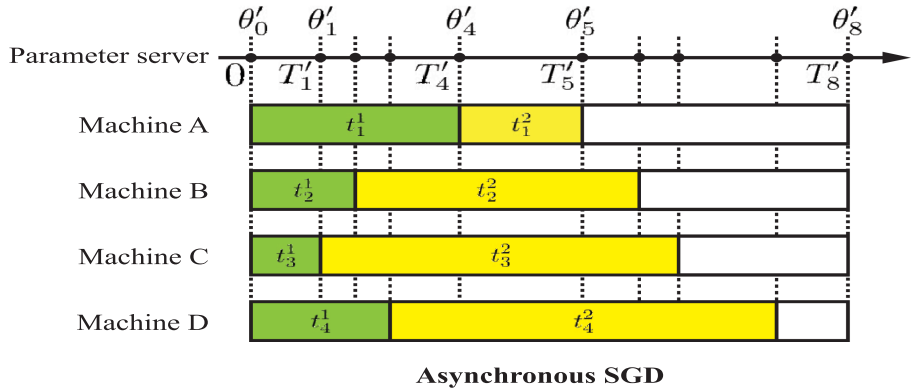
2.2 Distributed Asynchronous SGD

A natural extension of SGD from a single machine to multiple machines, in the data-parallel setting, is SSGD, whose mechanism is illustrated in Figure 1(a). Specifically, there are one parameter server and four worker machines in this toy example. Each worker machine receives two workloads during the timeframe under consideration. Let t_i^j denote the time that ith ($i = 1, 2, 3, 4$) machine takes on computing jth ($j = 1, 2$) workload. Different worker machines take different lengths of time when completing the first or second workload due to heterogeneous computational capacity and job complexity. At time 0, the parameter server propagates initial parameter θ_0 to each machine.

[3] As mini-batch GD contains "one-instance" SGD as a special case, we use "SGD" to refer to mini-batch GD throughout this article unless specified otherwise.



(a) SSGD: The parameter server updates the parameter only when all workers complete their current round of computations. The parameter server broadcasts updates synchronously to all worker machines. A large amount of total idle time, but is straightforward to analyze.



(b) ASGD: The parameter server updates the parameter every time some "fast" worker completes its current assigned batch gradient computation, and only propagates the new parameter to that worker without waiting other workers. Efficient utilization of resources, but suffers from 'gradient staleness'.

Fig. 1. Comparison between Synchronous SGD and Asynchronous SGD. Green blocks represent the first round of computation jobs, and yellow blocks represent the second round of computation jobs. Grey blocks are the periods of time when machines in SSGD wait idling. t_i^j denotes the time that ith ($i = 1, 2, 3, 4$) machine takes on computing jth ($j = 1, 2$) workload. ASGD updates the parameter much more frequently during the same period time than SSGD.

The parameter server will get first update from Machine C at time t_3^1. However, parameter server will not perform any updating until t_1^1, when all machines complete their first round of computations. At $T_1 = t_1^1$, the parameter server aggregates all four batch gradients, and updates $\theta_0 \rightarrow \theta_1$. Therefore, Machines B, C, and D have to experience the idle waiting time before they could receive their second workloads.

When the number of machines goes up and the capacity across machines are heterogeneous, synchronization on every iteration in SSGD becomes a major choke point. A much more efficient alternative is ASGD or Hogwild!, which has been proposed in [12, 13, 61] to reduce the amount of

idle time of worker machines. The idea of ASGD is illustrated in Figure 1(b). Specifically, instead of waiting for the slowest machine in the first round (Machine A), the parameter server performs an update $\theta_0' \rightarrow \theta_1'$ immediately when Machine C completes its job (note $\theta_1' \neq \theta_1$). Machine C will receive θ_1' and proceed to the second job without any delay. Machine A, B, and D continue their own work and will not be notified of this progress. Apparently, in Figure 1, ASGD updates the parameter eight times in the same period of time when SSGD only updates two times. Therefore, ASGD is an effective approach to overcome the synchronization bottleneck, and is a more efficient utilization of the computation resources.

However, the asynchronization in ASGD poses challenges in theoretically analyzing the speedup and generalization performances. In SSGD, the state of parameter server is always synchronous with worker machines at any given time, i.e., the parameter with which every worker machine computes local batch gradients is always the same as in the parameter server. The analysis of SSGD is thus less challenging, as it could be regarded as dismantling and redistributing the sequence of computed batch gradients from single-machine SGD to multiple machines. In contrast, the main challenge of ASGD is that the parameter server always receives batch gradients from the workers that are computed with outdated parameters. Specifically, in Figure 1(b), when Machine B communicates with parameter server at $T_2' = t_2^1$, the state of parameter server is already $\theta = \theta_1'$. However, the batch gradient from Machine B is computed with respect to $\theta = \theta_0'$. Ideally, the parameter server should update $\theta_1' \rightarrow \theta_2'$ based on gradient calculated with θ_1', instead of the "delayed" gradient with respect to θ_0'. The same phenomenon applies to time T_3', when the parameter server is already at θ_2', two steps ahead of Machine D. Therefore, at any given time, each worker machine may have a different view of the optimization parameter. This discrepancy between parameter server and worker machines is called "gradient staleness".

Formally, recall the updating rule of single-machine SGD Equation (2) as follows:

$$\theta_{t+1} = \theta_t - \alpha_t \hat{g}(\theta_t, \zeta_t), \tag{6}$$

where α_t and ζ_t are learning rate and mini-batch at time t, respectively. $\hat{g}(\theta_t, \zeta_t)$ is the batch gradient computed with mini-batch ζ_t at parameter θ_t. Unlike Equation (6), in ASGD, the parameter each worker computes batch gradient with, is up to some random delay $\tau_t \in \mathbb{N}$,

$$\phi_t = \theta_{t-\tau_t}, \qquad \text{where} \quad P(\tau_t = l) = q_l, \tag{7}$$

where the random delays from different t's are assumed to be independent and follow a distribution whose probability mass function is denoted by q_l [59]. ϕ_t is the outdated parameter which the worker machine computes gradient with. In the toy example illustrated by Figure 1(b), for instance, at time $t = T_2'$, parameter server is at θ_1', random delay is $l = 1$ and $\phi_t = \theta_0'$.

The updating rule for ASGD is given as follows:

$$\theta_{t+1} = \theta_t - \alpha_t \hat{g}(\phi_t, \zeta_t). \tag{8}$$

As $\hat{g}(\phi_t, \zeta_t)$ is not evaluated on θ_t, Equation (8) introduces much more complex dynamics than Equation (6), and thus requiring a new analysis tool to study the generalization performances.

2.3 Generalization Error for Stochastic Algorithms

Generalization to unseen data is the essence of learning, and generalization error measures the discrepancy between the learner's performance on training and testing environments. We now formulate generalization bound in the PAC-Bayesian framework.

Traditional Frequentist learning paradigm views model parameter θ as fixed but unknown values and do not attach any probabilities to these learnable parameters. In contrast, Bayesian perspective studies a distribution of every possible setting of parameters instead of betting on one

single setting of parameters to manage model uncertainty, and has proven increasingly powerful in many applications. In the Bayesian framework, θ is assumed to follow some prior distribution P (reflects our prior knowledge of model parameters), and at each iteration of QHM, the θ distribution shifts to $\{Q_k\}_{k\geq 0}$, and converges to posterior distribution Q (reflects our knowledge of model parameters after learning with $\mathcal{D}$). We further resort to Bayesian risk function to assess the generalization bound.

$$\begin{aligned}\mathcal{R}(Q) &\triangleq \mathbb{E}_{\theta\sim Q}\mathbb{E}_{(x,y)\sim\mathcal{D}} l(f_\theta(x), y),\\ \hat{\mathcal{R}}(Q) &\triangleq \mathbb{E}_{\theta\sim Q}\frac{1}{N}\sum_{j=1}^{N} l(f_\theta(x_j), y_j),\end{aligned} \tag{9}$$

where $\hat{\mathcal{R}}(Q)$ is the risk evaluated on training set $\mathcal{T}$ and $N \triangleq |\mathcal{T}|$ is the sample size, while the expected risk $\mathcal{R}(Q)$ is the expectation of the error on unseen data. The generalization bound could therefore be defined as follows:

$$\mathcal{E} \triangleq |\mathcal{R}(Q) - \hat{\mathcal{R}}(Q)|. \tag{10}$$

3 LOGICAL FRAMEWORK: HOW TO DERIVE GENERALIZATION BOUND?

The main contribution of this article is to provide a hyperparameter tuning strategy in both *centralized* and *distributed* settings, which has rigorous justifications rather than a blind trial-and-error as we theoretically prove why our tuning strategy could decrease our derived generalization errors. How to derive a hyperparameter-dependent generalization bound is the key to our article. In this section, we elaborate on our logical framework and proof schema to derive the generalization error before we move on to present our main results in the next few sections.

3.1 Continuous-time Dynamics of Gradient-based Methods

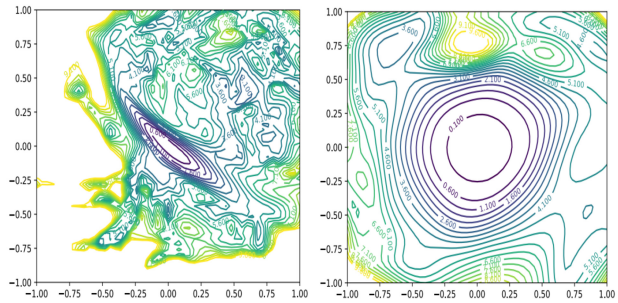
A powerful analysis tool for gradient-based methods (e.g., SGD or heavy ball momentum) is to model the continuous-time dynamics with stochastic differential equations and then study its limiting behavior [24, 30, 46, 57, 81].

$$\theta_{t+1} = \theta_t - \alpha\hat{g}(\theta_t, \zeta_t), \tag{11}$$

Let us start from characterizing the continuous-time dynamics of using a constant step size SGD Equation (11) to optimize the single-machine task (1). We first present the following standard assumptions from existing studies and discuss why these assumptions stand.

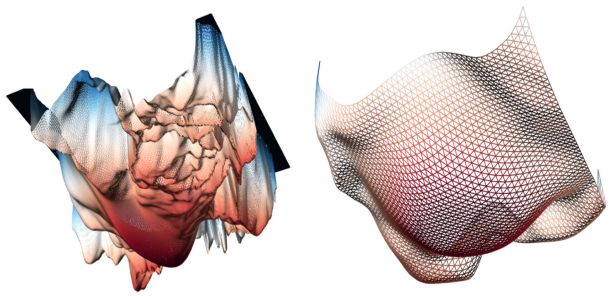
Assumption 1 ([24, 57, 81] The Second-order Taylor Approximation). *Suppose the risk function is approximately convex and 2-order differentiable, in the region close to minimum, i.e., there exists a $\delta_0 > 0$, such that $\mathcal{R}(\theta) = \frac{1}{2}(\theta-\theta^*)^T A(\theta-\theta^*)$ if $\|\theta-\theta^*\| \leq \delta_0$, where θ^* is a minimizer of $\mathcal{R}(\theta)$. Here A is the Hessian matrix $\nabla^2_\theta\mathcal{R}$ around minimizer and is positive definite. Without loss of generality, we assume a minimizer of the risk is zero, i.e., $\theta^* = 0$.*

Though here we assume a locally quadratic form of the risk function, all our results from this study apply to locally smooth and strongly convex objectives. Note that the assumption on locally quadratic structure of loss function, even for extremely non-convex objectives, could be justified empirically. [45] visualized the loss surfaces for deep structures like ResNet [25] and DenseNet [33], observing quadratic geometry around local minimum in both cases. And certain network architecture designs (e.g., skip connections) could further make neural loss geometry show no noticeable non-convexity, see e.g., Figures 2 and 3.



(a) WRN56-no-shortcut-connection (b) WRN56-shortcut-connection

Fig. 2. 2D visualization of the loss surface of Wide-ResNet-56 on CIFAR-10 both with shortcut connections (right) and without (left). Image is from [45] Figure 6.



(a) ResNet-56-shortcut-connection (b) ResNet-56-shortcut-connection

Fig. 3. 3D visualization of the loss surface of ResNet-56 on CIFAR-10 both with shortcut connections (right) and without (left). Image is from http://www.telesens.co/loss-landscape-viz/viewer.html.

Assumption 2 ([24, 57, 81] Unbiased Gradients with Bounded Noise Variance). *Suppose at each step t, gradient noise is Gaussian with mean 0 and covariance $\frac{1}{b}\Sigma(\theta_t)$, i.e.,*

$$\hat{g}(\theta_t) \approx g(\theta_t) + \frac{1}{\sqrt{b}}\Delta g(\theta_t), \quad \Delta g(\theta_t) \sim \mathcal{N}(0, \Sigma(\theta_t)).$$

We further assume that the noise covariance matrix $\Sigma(\theta_t)$ is approximately constant with respect to θ, i.e., $\Sigma(\theta_t) \approx \Sigma = CC^T$. And noises from different iterates $\{\Delta g(\theta_t)\}_{t\geq 1}$ are mutually statistically independent.

Gaussian gradient noise is natural to assume as the stochastic gradient is a sum of b independent, uniformly sampled contributions. Invoking the central limit theorem, the noise structure could be approximately Gaussian. Assumption 2 is standard when approximating a stochastic algorithm with a continuous-time stochastic process (see e.g., [57]) and is justified when the iterates are confined to a restricted region around the minimizer.

With Assumptions 1 and 2, constant-step size SGD Equation (11) could be recast as a discretization of the following continuous-time dynamics:

$$d\theta = -\alpha g(\theta)dt + \frac{\alpha}{\sqrt{b}}CdW_t, \tag{12}$$

where $dW_t = \mathcal{N}(0, Idt)$ is a Wiener process. Many SGD momentum variants (e.g., heavy ball momentum [63] and Nesterov's accelerated gradient [60]) can also be cast as a modified version of Equation (12) [20]. The stochastic process Equation (12) is known as an Ornstein–Uhlenbeck process. The following theorem shows it has an analytic stationary distribution $q(\theta)$ that is Gaussian and explicitly writes out the covariance structure of $q(\theta)$.

Theorem 1 (Stationary Structure of SGD [57]). *The stationary distribution of stochastic process Equation (12) is Gaussian, i.e.,*

$$q(\theta) \propto \exp\left\{-\frac{1}{2}\theta^T Q^{-1}\theta\right\}, \qquad \text{where} \quad QA + AQ = \frac{\alpha}{b}CC^T. \tag{13}$$

3.2 PAC-Bayesian Generalization Bound

We now introduce the classical PAC-Bayesian generalization bound.

Theorem 2 (PAC-Bayesian Generalization Bound [58, 66]). *Let $KL(Q||P)$ be the Kullback-Leibler divergence between distributions Q and P. For any positive real $\epsilon \in (0, 1)$, with probability at least $1 - \epsilon$ over a sample of size N, we have the following inequality for all distributions Q:*

$$\mathcal{R}(Q) \leq \hat{\mathcal{R}}(Q) + \sqrt{\frac{KL(Q||P) + \log\frac{1}{\epsilon} + \log N + 2}{2N - 1}}. \tag{14}$$

The prior distribution P is typically assumed to be a Gaussian prior $\mathcal{N}(\theta_0, \lambda_0 I_d)$, reflecting the common practice of Gaussian initialization [17] and L_2 regularization. The importance of Theorem 2 is that, we could easily get an upper bound of generalization bound if we could explicitly represent $\mathrm{KL}(Q||P)$, i.e., the KL divergence between posterior and prior distributions.

3.3 Roadmap for Derivation of Generalization Error

Theorem 1 gives the exact form of posterior Q, and therefore, naturally results in a generalization bound for vanilla SGD by invoking Theorem 2. More importantly, the upper bound we obtain is dependent on basic hyperparameters, i.e., α and b in vanilla SGD, through Q, which is exactly what we desire. Motivated by this result, our procedure to obtain generalization bounds for both multistage QHM (Section 5) and Asynchronous SGD (Section 7) is as follows: we model the dynamics of multistage QHM and Asynchronous SGD via a continuous-time SDE, study the limiting stationary distribution of the corresponding stochastic process, and then derive a hyperparameter-dependent generalization bound via Theorem 2.

4 CENTRALIZED STAGEWISE QHM

In this section, we present our proposed approach, stagewise QHM, to tune a large class of momentum variants in *centralized* computing environment.

One of the most effective hyperparameter scheduling rules is "constant and drop". "Constant and drop" is the de-facto learning rate scheduler in most large-scale neural networks [40, 71, 77]. In its vanilla SGD version (a.k.a. multistage SGD), with a prespecified set of learning rates $\{\alpha_i\}_i^M$ and training lengths $\{T_i\}_i^M$ (often measured by a number of iterations/epochs), the learning process is divided into M stages, and at ith stage SGD(α_i) is applied for T_i iterations/epochs.

The logic behind "constant and drop" is: a large but constant step size allows for faster convergence than diminishing step size in the early stage of training. However, constant step size SGD (or its variants) will only fluctuate in a local region around the minimum according to some stationary distribution instead of converging directly to the minimum itself [57]. Therefore, in "constant and drop" paradigm, a large learning rate is held constant for a reasonably long period to take

advantage of faster convergence until it reaches a stationary distribution around minimizer (or to a stage where it is sufficiently close to minimizer), and then the learning rate is dropped by a constant factor (or exponentially in some cases) for more refined training.

Algorithm 1 extends "constant and drop" to a large class of momentum variants. As momentum variants are more predominantly used than vanilla SGD, Algorithm 1 characterizes most real-world training. Note it recovers multistage SGD when $\nu_i = 0$, multistage SHB when $\nu_i = 1$, and multistage NAG when $\nu_i = \beta_i$.

ALGORITHM 1: Multistage QHM

Input: Objective function $\mathcal{R}(\theta)$, initialization θ_0, number of stages M, triplets of QHM specification $\{\alpha_i, \beta_i, \nu_i\}_{i=1}^M$, training lengths $\{T_i\}_{i=1}^M$, batch sizes $\{b_i\}_{i=1}^M$;

for $i \in \{0, 1, \ldots, M-1\}$ **do**
  update $\theta_{i,0} \leftarrow \theta_i$;
  for $k \in \{0, \ldots, T_i - 1\}$ **do**
    Sample a mini-batch $\zeta_k = \{(x_j, y_j)\}_{j=1}^{b_i}$ from training data uniformly (ζ_k also refers to the index set of the sampled data points if not specified otherwise);
    Compute gradient of objective function based on ζ_k, i.e., $\hat{g}_k = \frac{1}{b_i} \sum_{j \in \zeta_k} \nabla \mathcal{R}_j(\theta_{i,k})$;
    Compute $d_k = (1 - \beta_i)\hat{g}_k + \beta_i d_{k-1}$, with $d_0 = 0$;
    Update $\theta_{i,k+1} \leftarrow \theta_{i,k} - \alpha_i[(1 - \nu_i)\hat{g}_k + \nu_i d_k]$;
  end
  update $\theta_{i+1} \leftarrow \theta_{i,T_i}$;
end
return θ_M

Note existing works and most off-shelf implementations only allow α_i and T_i to vary across stages, while fixing (β_i, ν_i) and b_i as constants [50, 80]. Algorithm 1 is much more flexible. Based on our results, fixing β_i and b_i is suboptimal. In Section 5, we will provide theoretical justifications. In Section 8, we show our (β_i, b_i) scheduler achieves non-trivial advantages over existing schedulers.

5 HOW TO TUNE MULTISTAGE QHM?

It is well known that optimization hyperparameters have a substantial impact on the quality of the training process and generalizability for deep neural networks. Algorithm 1 has a large number of hyperparameters: stage-varying learning rate $\{\alpha_i\}_{i=1}^M$, momentum parameter $\{\beta_i\}_{i=1}^M$, instant discount factor $\{\nu_i\}_{i=1}^M$, and batch size $\{b_i\}_{i=1}^M$, to name a few.

Apparently, a grid search on such a large space of hyperparameters in Algorithm 1 is computationally infeasible even with substantial resources. This section will discuss how to configure these hyperparameters for better generalization. In Section 5.1, we will theoretically connect the generalization error of QHM with these hyperparameters. Section 5.2 explains how to tune to decrease the generalization error in practice.

5.1 Generalization Theorem

The connection between generalization error and hyperparameters, especially momentum-related parameters, is barely studied in existing works. We represent the generalization error as a function of hyperparameters in the following theorem.

Theorem 3. *Assume the risk function is locally quadratic, and gradient noise is Gaussian (i.e., Assumptions 1 and 2).*[4] *Suppose the prior distribution of parameters is* $\mathcal{N}(\theta_0, \lambda_0 I_d)$ *with some constant* θ_0 *and* λ_0*, where* d *represents the dimension of parameters. For any positive real* $\epsilon \in (0, 1)$*, the following upper bound of generalization error holds with probability at least* $1 - \epsilon$*,*

$$\mathcal{R}(Q) - \hat{\mathcal{R}}(Q) \leq \sqrt{\frac{C_1 + C_2(\alpha, \beta, \nu, b)}{2N - 1}},$$

where

$$\begin{aligned} C_1 &= \frac{1}{2\lambda_0}\|\theta_0\|_2^2 + \frac{1}{2}d\log\lambda_0 - \frac{1}{2}d - \frac{1}{2}\log\det(\Sigma A^{-1}) + \log\frac{1}{\epsilon} + \log N + 2, \\ C_2(\alpha, \beta, \nu, b) &= -\frac{1}{2}d\log\frac{\alpha}{2b} + \frac{\alpha\, tr(\Sigma A^{-1})}{4\lambda_0 b} + \frac{\alpha^2 tr(\Sigma)}{4\lambda_0 b}\frac{(4\nu^2 - 2\nu - 1)\beta^2 - 2\nu\beta + 1}{2(1 - \beta^2)}. \end{aligned} \tag{15}$$

where det *and tr indicate the determinant and trace of a matrix, respectively.*

Proof. We defer the proof to Appendix. Here we introduce a key lemma that the proof relies on, □

Lemma 1 (Stationary Distribution of QHM Iterates). *θ optimized by QHM will converge to a stationary distribution* $\exp\{-\frac{1}{2}\theta^T\Sigma_\theta\theta\}$*. Furthermore, the trace and determinant of* Σ_θ *fulfill the following equalities:*

$$\begin{aligned} tr(\Sigma_\theta) &= \frac{\alpha}{2b}tr(\Sigma A^{-1}) + \frac{\alpha^2}{2b}\left(\frac{\nu\beta}{1 - \beta}\left(1 - \frac{2(1 + \beta - \nu\beta)}{1 + \beta} + \frac{1}{2}\right)\right) tr(\Sigma) + O(\alpha^3) \\ \det(\Sigma_\theta) &= \left(\frac{\alpha}{2b}\right)^d \det(\Sigma A^{-1}) + O(\alpha^2). \end{aligned} \tag{16}$$

We prove Lemma 1 in Appendix. This lemma essentially gives the posterior distribution Q of QHM iterates. Based on the idea we elaborate in Section 3, we only need to explicitly calculate the KL divergence between its prior distribution P and Q in order to get the generalization bound. Please refer to the Appendix for complete proof details.

In Theorem 3, generalization error is upper bounded by C_1 plus C_2, where C_2 is a function of hyperparameters of interest, i.e., (α, β, ν, b), while C_1 is constant determined only by sample size and initialization. As our concentration is to study how varying (α, β, ν, b) affects generalization error, we mainly focus on C_2 in the rest of the article. Further notice only the last term in C_2 includes (β, ν). Let us denote the last term as H for ease of notation:

$$H \triangleq \frac{\alpha^2 \mathrm{tr}(\Sigma)}{4\lambda_0 b}\frac{(4\nu^2 - 2\nu - 1)\beta^2 - 2\nu\beta + 1}{2(1 - \beta^2)}.$$

5.2 Hyperparameters Scheduling Scheme

We discuss how exactly Theorem 3 guides us to tune multistage QHM.

[4]The assumption is standard when approximating a stochastic algorithm with a continuous-time stochastic process (see [20, 24, 57]) and is justified when the iterates are confined to a restricted region around the minimizer. See Section 3 for details.

5.2.1 Decay learning rate, while keeping $\frac{b_i}{\alpha_i}$ Non-increasing. The following Corollary explains the role of $\frac{b_i}{\alpha_i}$ on each training stage:

Corollary 5.1. *The impact of $\frac{b}{\alpha}$ could be summarized as follows:*

(1) If the model size d (i.e., number of parameters) is larger[5] than $\frac{\alpha\, tr(\Sigma A^{-1})}{2\lambda_0 b} + 2H$, smaller $\frac{b}{\alpha}$ produces smaller generalization error.
(2) If $\frac{b}{\alpha}$ is fixed as constant, smaller α produces smaller generalization error.[6]

Proof. The first statement is shown by calculating the derivative of C_2 w.r.t. $s = \frac{b}{\alpha}$. The second statement is obvious by setting $b = s\alpha$ for some constant s, and substitute into C_2. □

Learning rate decay is a general consensus for most optimizers, including SGD variants [64] and adaptive gradient methods (e.g., Adam or RMSProp) [39]. However, it is less understood whether batch size should adjust alongside learning rate. Most off-shelf softwares hold it constant throughout the entire training process, for which Corollary 5.1 shows to be suboptimal.

References [21, 71] propose to scale batch size with learning rate as well (a.k.a. linear scaling rule), but in an increasing direction, for SHB and NAG. It is mainly to take advantage of the speedup of large-batch training, but with a slight sacrifice in testing accuracy. Our Theorem 3 justifies why linear scaling rule is reasonable, and also explains why linear scaling rule (in its increasing version) hurts generalization.

Specifically, the first statement in Corollary 5.1 indicates clearly if we fix batch size, decaying learning rate would hurt generalization. But if we decay batch size faster than learning rate, i.e., decrease $\frac{b_i}{\alpha_i}$, (or at least adjust batch size proportionately to learning rate, i.e., keep $\frac{b_i}{\alpha_i}$ constant), decaying learning rate would actually improve generalizability according to the second statement in Corollary 5.1. Undershrinkage of batch size ($\frac{b_i}{\alpha_i}$ increases with i) negatively impacts the generalization of multistage QHM in theory, and empirical experiments in Section 8 support our finding.

Corollary 5.1 also gives us insights into how we should set initial batch size b_1 and initial learning rate α_1. In order to maintain a smaller $\frac{b_i}{\alpha_i}$, we should select larger α_1 as long as the optimizer still converges, and smaller b_1 as long as running time does not exceed the time budget.

5.2.2 Increase Momentum Parameter, while Keeping Instant Discount Factor v_i Constant. In the original article of QHM [53], the authors proposed to set ($\beta = 0.999, \nu = 0.7$) throughout the entire learning process and achieved impressive improved training in a variety of settings. The following corollary provides a better configuration which beats ($\beta = 0.999, \nu = 0.7$) in generalization ability.

Corollary 5.2. *$H(\beta \to 1, \nu = 0.5) \ll H(\beta = 0.999, \nu = 0.7)$, and consequently, $(\beta \to 1, \nu = 0.5)$ decreases the generalization error.*

Proof. The proof is obvious by simply substituting (β, ν) into H.[7] □

[5]Modern large-scale deep learning models are often extremely overparameterized, with d ranging from tens to hundreds of millions (e.g., VGG [67] and ResNet [26]), and this condition is easily fulfilled.
[6][24] shows a similar bound as Theorem 3, but does not consider momentum parameters (β, ν); and notably, it only includes the first order term of learning rate $O(\alpha)$. The second statement of this Corollary could not be obtained with their first-order bound, and we will show in Section 8 that such difference is empirically non-trivial.
[7]Note that when $\beta \to 1$, H could be negative, but it does not indicate a negative generalization error as it is only one term of the generalization error.

Setting a β that is close to 1 is commonplace in practice (e.g., PyTorch SGD+momentum sets $\beta = 0.9$ by default). For SHB and NAG, [71, 77] also propose a scheduler for β to increase, to ensure faster convergence. However, it does not apply to QHM, as the convergence rate for QHM is much more complex, even for quadratic objective functions (see Theorem 3 in [20]). The convergence rate is not a monotone function of β. Therefore, increasing β does not necessarily guarantee faster convergence for all momentum schemes covered by QHM formulation.

However, the generalization bound in Theorem 3 is a monotonically decreasing function of β when fixing $\nu = 0.5$. Therefore, adopting an increasing $\{\beta_i\}_{i=1}^M$ is justified for better generalization performance.

5.2.3 Increase Length of Training Time T_i. As the step size is getting smaller, it is natural to allow a longer training time in later stages than in earlier stages. Therefore, we adopt the widespread tuning strategy here, i.e., keeping $T_i\alpha_i$ as a constant.

5.2.4 Tuning Strategies for Multistage QHM. Combining everything together, suppose the dropping rate of step size is $0 < r < 1$, the dropping rate of batch size is $0 < r_b \leq r < 1$ (i.e., batch size decreases faster than or proportionately to step size), for all $1 \leq i \leq M$, our tuning paradigm for Algorithm 1 is as follows:

$$\alpha_i = r^{i-1}\alpha_1, \quad b_i = r_b^{i-1}b_1, \quad T_i = \left(\frac{1}{r}\right)^{i-1} T_1, \quad \nu_i = \frac{1}{2},$$
$$\beta_i = 1 - \frac{1}{1 + \left(\frac{\beta_1}{1-\beta_1}\right)\left(\frac{1}{r}\right)^{i-1}}, \quad \beta_i^{2T_i} \leq \frac{1}{2}. \tag{17}$$

In practice, a popular choice of r is $\frac{1}{2}$. r_b could be set as $\frac{1}{2}$ for convenience (Section 8.1.1 shows smaller r_b only exhibits marginal advantage). β_1 and T_1 could just select the default values of software implementation. Our β scheduler $\beta_i = 1 - \frac{1}{1+(\frac{\beta_1}{1-\beta_1})(\frac{1}{r})^{i-1}}$ ensures the increasing trend of β_i. Condition $\beta_i^{2T_i} \leq \frac{1}{2}$ ensures T_i not to be extremely small. Note that even with β_i as large as 0.999, this condition stands if T_i is more than 500 iterations, which is much smaller than the typical number of iterations we run in practice. Therefore, this condition is easily fulfilled. b_1 should be set as small as our running time budget allows us to boost generalization, while α_1 needs to be large as long as the optimizer still converges.

With Equation (17), we only have to manually search α_1, and Corollary 5.1 further indicates, we only need to search α_1 monotonically. For example, if we have a grid $\alpha = (0.01, 0.02, 0.03, \ldots, 1.0)$ as usual, we only need to start from 0.01, increase α, and stop immediately when the next (or next few) grid values does not give us a better test accuracy. This further decreases our search space, as we do not need to try out every possible value in a large grid.

6 CONVERGENCE OF MULTISTAGE QHM

In this section, we will provide the convergence guarantee of our multistage QHM, for general non-convex objective functions.

Theorem 4. *Suppose $\mathcal{R}(\theta)$ is L-smooth and not necessarily strongly convex. We optimize $\mathcal{R}(\theta)$ using Algorithm 1 with hyperparameters specified as in Equation (17). Let $N_T = \sum_{i=1}^M T_i$ be the total number of iterations in M-stage training. Denote the expected gradient square as $\{\mathcal{G}_k \triangleq \mathbb{E}[\|g_k\|^2]\}_{k \leq N}$. We define the average expected gradient square at ith stage as $\bar{\mathcal{G}}_i \triangleq \frac{1}{T_i}\sum_{k=T_1+T_2+\cdots+T_{i-1}+1}^{T_1+T_2+\cdots+T_i} \mathcal{G}_k$ and the average expected gradient square of all M stages as $\bar{\mathcal{G}} \triangleq \frac{1}{M}\sum_{i=1}^M \bar{\mathcal{G}}_i$.*

Denote $W_1 \triangleq \frac{\alpha_i \beta_i \nu_i}{1-\beta_i}$ *and* $W_2 \triangleq T_i \alpha_i$ *for all* $i \le M$. *Under mild regulatory conditions,*[8] *we would have:*

$$\begin{aligned} \bar{\mathcal{G}} &\le \varepsilon_d + \varepsilon_s, \\ \varepsilon_d &= \frac{2(\mathcal{R}(\theta_1) - \mathcal{R}^*)}{MW_2}, \\ \varepsilon_s &= \frac{1}{M} \sum_{i=1}^{M} \left(1 + 24 \frac{\beta_i^2 \nu_i^2}{(1-\beta_i)^2} \frac{\beta_1}{\sqrt{\beta_M + \beta_M^2}} + \frac{6 + 2\beta_i^2 \nu_i^2}{1-\beta_1} \right) \alpha_i L \sigma^2. \end{aligned} \tag{18}$$

Remark 6.1 (Non-convex Objective). In Theorem 4, we only require the objective function to be smooth. Many existing works further require the objective function to be strongly convex [20]. As in neural network optimization, strongly convexity may not hold for highly over-parameterized neural networks (see [49]). Therefore, our settings are more general.

Remark 6.2 (ε_d and ε_s). ε_{d} and ε_{s} reflect two driving forces of error, ε_{d} is the deterministic approximation error, representing the iterates get closer to the minima. It will diminish with a larger number of stages M or a larger number of iterations $\{T_i\}$. However, ε_{s} is the irreducible stochastic error, describing the fluctuation around the minima due to gradient noise. And specifications of (β_i, ν_i) affect the radius of stationary distribution ε_{s}.

Proof. Due to the page limit, we defer details to Appendix and only provide a proof sketch here. Let $(\alpha_k, \beta_k, \nu_k)$ denote the hyperparameters at kth iteration. Recall the formulation of QHM (Equation (3)). Denote the update sequence $y_k \triangleq \theta_{k+1} - \theta_k$. Vanilla SGD is easier to handle as it updates $-\alpha_k \hat{g}_k$ every step. However, in QHM, $y_k \neq -\alpha_k \hat{g}_k$. We construct an auxiliary sequence $\{\eta_k\}_{k \in \mathbb{N}}$, such that $\eta_{k+1} - \eta_k = -\alpha_k \hat{g}_k$ [50]. $\{\eta_k\}_{k \in \mathbb{N}}$ is devised as follows:

$$\eta_k = \begin{cases} \theta_k & k = 1 \\ \theta_k - \frac{\alpha_k \beta_k \nu_k}{1-\beta_k} d_{k-1} & k \ge 2 \end{cases}, \tag{19}$$

where $d_0 = 0$. It is not difficult to verify $\eta_{k+1} - \eta_k = -\alpha_k \hat{g}_k$:

$$\begin{aligned} \eta_{k+1} - \eta_k &= \theta_{k+1} - W_1 d_k - (\theta_k - W_1 d_{k-1}) \\ &= y_k - \frac{\alpha_k \beta_k \nu_k}{1-\beta_k} (d_k - d_{k-1}) \\ &= -\alpha_k \Big((1-\nu_k) \hat{g}_k + \nu_k d_k \Big) - \alpha_k \beta_k \nu_k \hat{g}_{k-1} + \alpha_k \beta_k \nu_k d_{k-1} = -\alpha_k \hat{g}_k. \end{aligned}$$

$\{\eta_k\}_{k \in \mathbb{N}}$ is more similar to vanilla SGD iterates and thus easier to deal with. We then study the property of $\{\eta_k\}_{k \in \mathbb{N}}$ and its connection to $\{\theta_k\}_{k \in \mathbb{N}}$. Given the gradient sequence $\{\hat{g}_k\}_{k \in \mathbb{N}}$, set:

$$a_{k,i} = \begin{cases} 1 - \beta_k \nu_k & i = k \\ \nu_k (1-\beta_i) \prod_{j=i+1}^{k} \beta_j & i < k \end{cases}. \tag{20}$$

It is not difficult to verify $y_k = \Sigma_{i=1}^{k} a_{k,i} \hat{g}_i$ with $d_0 = 0$. Therefore, $\mathbb{E}[y_k] = \Sigma_{i=1}^{k} a_{k,i} g_i$. We then have a key lemma on the variance of QHM updating vector y_k, and the deviance between updating vector y_k and g_k, before showing Theorem 4:

[8] Theorem 4 does need some mild regulatory conditions that mainly constrains the size of α and β. It requires $W_1 = \frac{1}{48\sqrt{2}L}$, i.e., step size could not be too large; and $\frac{1-\beta_1}{\beta_1} \le 12 \frac{1-\beta_M}{\sqrt{\beta_M + \beta_M^2}}$, i.e., $\{\beta_i\}$ could not be increased too fast. These regulatory conditions could be fulfilled by typical value assignment (e.g., starting from $\beta_1 = 0.9$, and dropping step size by rate $\frac{1}{2}$).

Lemma 2. *We have the following two inequalities:* $\mathbb{V}[y_k] \leq (6 - 4\beta_1 - 4\beta_k\nu_k + 2\beta_k^2\nu_k^2)\sigma^2$, *and* $\mathbb{E}[\|g_k - \frac{1}{1-\nu_k\prod_{i=1}^k\beta_i}\Sigma_{i=1}^k a_{k,i}g_i\|^2] \leq \Sigma_{j=1}^{k-1}\frac{\nu_k\beta_k^{k-j}L^2}{1-\nu_k\prod_{i=1}^k\beta_i}(k-j+\frac{\beta_k}{1-\beta_k})\mathbb{E}[\|\theta_{j+1}-\theta_j\|^2]$.

Please see Appendix for further steps. □

7 HOW TO TUNE ASYNCHRONOUS SGD?

In this section, we extend to distributed parallel computing regime, generalize our theory to ASGD, and show how a moderately large learning rate helps ASGD generalize.

Theorem 5 (Generalization Error of ASGD). *Assume the risk function is locally quadratic, and gradient noise is Gaussian. Suppose the prior distribution of parameters is* $\mathcal{N}(\theta_0, \lambda_0 I_d)$ *with some constant* θ_0 *and* λ_0, *where* d *represents the dimension of parameters. Considering a total number of* M *working machines performing ASGD, under mild assumptions on random staleness and work process, for any positive real* $\epsilon \in (0,1)$, *the following upper bound of generalization error holds with probability at least* $1-\epsilon$,

$$\mathcal{R}(Q) - \hat{\mathcal{R}}(Q) \leq \sqrt{\frac{C_1 + C_2(\alpha, M, b)}{2N-1}},$$

where

$$\begin{aligned} C_1 &= \frac{1}{2\lambda_0}\|\theta_0\|_2^2 + \frac{1}{2}d\log\lambda_0 - \frac{1}{2}d - \frac{1}{2}\log\det(\Sigma A^{-1}) + \log\frac{1}{\epsilon} + \log N + 2, \\ C_2(\alpha, M, b) &= -\frac{1}{2}d\log\frac{\alpha}{2(M-1)b} + \frac{\alpha\, tr(\Sigma A^{-1})}{4\lambda_0(M-1)b}. \end{aligned} \tag{21}$$

Let $r = \frac{b}{\alpha}$, *if* $d \geq \frac{tr(\Sigma A^{-1})}{2(M-1)\lambda_0 r}$, *we have* $\frac{\partial C_2}{\partial r} > 0$, *i.e., a relatively small* $r = \frac{b}{\alpha}$ *(i.e., a moderately large* α*) improves the generalization of ASGD.*

Before we proceed to the proof of Theorem 5, we have the following two remarks on the assumptions and the theorem's practical implication.

Remark 7.1 (Mild Assumptions on Random Staleness and Work Process). In Theorem 5, we do require some assumptions on staleness distribution and work process. These assumptions are standard in the literature on asynchronous SGD (see e.g., [3, 59]).

- —The staleness process, $(\tau_t)_t$, and the sample selection process, $(\zeta_t)_t$, are mutually independent.
- —Denote the job at step t takes the system W_t time to compute. We assume an exponential, independent work process $(W_t)_t$, i.e., $W_t \sim \exp(\lambda)$ and $(W_t)_t$ are mutually independent, where λ is the parameter for the exponential distribution.

The first assumption is valid on most neural network architectures that perform dense updates (e.g., CNN-based models), where the randomness in staleness comes from implementation and system behavior that have absolutely nothing to do with sampling process. The second assumption simplifies the queuing model that distributed computing center typically employs and such approximation proves to be satisfactory empirically [59]. These two assumptions are mild and widely used in the existing literature.

Remark 7.2 (Moderately Large Step Size Helps ASGD to Generalize). The second statement of Theorem 5 indicates, if the number of parameters d is sufficiently large (which is easily fulfilled for modern large-scale models with millions or billions of parameters), ASGD benefits from a relatively large learning rate. Such positive relationship is very important to practitioners, as it confirms that we only need to search initial α for ASGD monotonically as in multistage QHM (see

Section 5 for details). With Theorem 5, if we have a search grid for α, we only need to start from smallest possible value in this grid, increase incrementally, and stop immediately when the next (or next few) grid value does not give us a better test accuracy, which ensures we do not have to try out every possible value in a large grid. However, the condition for the positive relationship to happen, is that α has to be small enough such that the ASGD algorithm still converges.

Proof. We now prove Theorem 5 and the proof itself reveals how we use the idea in Section 3. Basically, we model ASGD's expected iterates as a momentum-like process, calculate its stationary distribution, and compute the PAC-Bayesian bound via Theorem 2.

The proof of this theorem relies on the following two lemmas on the average behavior of ASGD iterates, and stationary distribution of momentum,

Lemma 3 (Behavior of Asynchrony [59]). *Consider we have M asynchronous workers. Under the same mild assumptions on random staleness and work process as in Theorem 5, the expected update takes the following updating form:*

$$\mathbb{E}[\theta_{t+1} - \theta_t] = \left(1 - \frac{1}{M}\right)\mathbb{E}[\theta_t - \theta_{t-1}] - \frac{1}{M}\alpha\mathbb{E}[g(\theta_t)], \tag{22}$$

or equivalently, the asynchrony induces an implicit momentum.

Lemma 4 (Stationary Distribution of Momentum [57]). *Consider the following updating rule of heavy ball momentum:*

$$v_{t+1} = (1-\mu)v_t - \alpha\hat{g}(\theta_t) \qquad \theta_{t+1} = \theta_t + v_{t+1}. \tag{23}$$

The continuous-time dynamics of Equation (23) can be described by the following stochastic differential equation, under the assumption that the risk function is locally quadratic, and gradient noise is Gaussian:

$$dv = -\mu v dt - \alpha A\theta dt + \frac{1}{\sqrt{b}}\alpha C dW, \qquad d\theta = v dt. \tag{24}$$

θ optimized by Equation (23) will converge to a stationary distribution $\exp\{-\frac{1}{2}\theta^T\Sigma_\theta\theta\}$, and Σ_θ fulfills the following equation:

$$\Sigma_\theta A + A\Sigma_\theta = \frac{\alpha}{\mu b}CC^T. \tag{25}$$

We defer the proof of Lemmas 3 and 4 to Appendix. Let us elaborate how we could use these two lemmas to show Theorem 5.

The density of prior and posterior distributions is given as follow:

$$\begin{aligned} f_P &= \frac{1}{\sqrt{2\pi\det(\lambda_0 I_d)}}\exp\left\{-\frac{1}{2}(\theta-\theta_0)^T(\lambda_0 I_d)^{-1}(\theta-\theta_0)\right\}, \\ f_Q &= \frac{1}{\sqrt{2\pi\det(\Sigma_\theta)}}\exp\left\{-\frac{1}{2}\theta^T\Sigma_\theta^{-1}\theta\right\}, \qquad \text{where} \quad \Sigma_\theta A + A\Sigma_\theta = \frac{\alpha}{(M-1)b}CC^T. \end{aligned} \tag{26}$$

We calculate their $\mathrm{KL}(Q||P)$ as follows:

$$\begin{aligned} \mathrm{KL}(Q||P) &= \int\left(\frac{1}{2}\log\frac{|\lambda_0 I_d|}{|\Sigma_\theta|} - \frac{1}{2}\theta^T\Sigma_\theta^{-1}\theta + \frac{1}{2}(\theta-\theta_0)^T(\lambda_0 I_d)^{-1}(\theta-\theta_0)\right)f_Q(\theta)d\theta, \\ &= \frac{1}{2}\left\{\mathrm{tr}\left((\lambda_0 I_d)^{-1}\Sigma_\theta\right) + \theta_0^T(\lambda_0 I_d)^{-1}\theta_0 - d + \log\frac{|\lambda_0 I_d|}{|\Sigma_\theta|}\right\}, \\ &= \frac{1}{2\lambda_0}\theta_0^T\theta_0 - \frac{d}{2} + \frac{d}{2}\log\lambda_0 + \frac{1}{2\lambda_0}\mathrm{tr}(\Sigma_\theta) - \frac{1}{2}\log|\Sigma_\theta|. \end{aligned} \tag{27}$$

Starting from $\Sigma_\theta A + A\Sigma_\theta = \frac{\alpha}{(M-1)b}\Sigma$, we could get:

$$\begin{aligned}
\Sigma_\theta A + A\Sigma_\theta &= \frac{\alpha}{(M-1)b}\Sigma, \\
\Sigma_\theta + A\Sigma_\theta A^{-1} &= \frac{\alpha}{(M-1)b}\Sigma A^{-1}, \\
\mathrm{tr}(A\Sigma_\theta A^{-1} + \Sigma_\theta) &= \frac{\alpha}{(M-1)b}\mathrm{tr}(\Sigma A^{-1}),
\end{aligned} \tag{28}$$

where, $\mathrm{tr}(A\Sigma_\theta A^{-1} + \Sigma_\theta) = \mathrm{tr}(A\Sigma_\theta A^{-1}) + \mathrm{tr}(\Sigma_\theta) = \mathrm{tr}(\Sigma_\theta A^{-1} A) + \mathrm{tr}(\Sigma_\theta) = 2\mathrm{tr}(\Sigma_\theta)$,

$$\text{Thus,} \qquad \mathrm{tr}(\Sigma_\theta) = \frac{\alpha}{2(M-1)b}\mathrm{tr}(\Sigma A^{-1}).$$

Σ_θ and A are both covariance matrices, which makes it natural to assume they are both symmetric. Therefore, we have,

$$\begin{aligned}
\Sigma_\theta A + A\Sigma_\theta &= \frac{\alpha}{(M-1)b}\Sigma, \qquad 2\Sigma_\theta A = \frac{\alpha}{(M-1)b}\Sigma, \qquad \Sigma_\theta = \frac{\alpha}{2(M-1)b}\Sigma A^{-1}, \\
\det(\Sigma_\theta) &= \det\left(\frac{\alpha}{2(M-1)b}\Sigma A^{-1}\right) = \left(\frac{\alpha}{2(M-1)b}\right)^d \det(\Sigma A^{-1}), \\
\log\det(\Sigma_\theta) &= \log\left(\left(\frac{\alpha}{2(M-1)b}\right)^d \det(\Sigma A^{-1})\right), \\
&= -d\log\left(\frac{2(M-1)b}{\alpha}\right) + \log\det(\Sigma A^{-1}).
\end{aligned} \tag{29}$$

Therefore, we could get $\mathrm{KL}(Q||P)$ as follows:

$$\begin{aligned}
\mathrm{KL}(Q||P) = \frac{1}{2\lambda_0}\theta_0^T\theta_0 - \frac{d}{2} + \frac{d}{2}\log\lambda_0 + \frac{1}{4\lambda_0}\frac{\alpha}{(M-1)b}\mathrm{tr}(\Sigma A^{-1}) \\
+ \frac{d}{2}\log\left(\frac{2(M-1)b}{\alpha}\right) - \frac{1}{2}\log\det(\Sigma A^{-1}).
\end{aligned} \tag{30}$$

Plugging Equation (30) back into Theorem 2, we could consequently obtain,

$$\mathcal{R}(Q) - \hat{\mathcal{R}}(Q) \le \sqrt{\frac{C_1 + C_2(\alpha, M, b)}{2N-1}},$$

where

$$\begin{aligned}
C_1 &= \frac{1}{2\lambda_0}\|\theta_0\|_2^2 + \frac{1}{2}d\log\lambda_0 - \frac{1}{2}d - \frac{1}{2}\log\det(\Sigma A^{-1}) + \log\frac{1}{\epsilon} + \log N + 2, \\
C_2(\alpha, M, b) &= -\frac{1}{2}d\log\frac{\alpha}{2(M-1)b} + \frac{\alpha\mathrm{tr}(\Sigma A^{-1})}{4\lambda_0(M-1)b}.
\end{aligned} \tag{31}$$

Denote $r = \frac{b}{\alpha}$,

$$\begin{aligned}
C_2(r, M) &= \frac{1}{2}d\log(2(M-1)r) + \frac{\mathrm{tr}(\Sigma A^{-1})}{4\lambda_0(M-1)r}, \\
\frac{\partial C_2}{\partial r} &= \frac{d}{2r} - \frac{1}{4\lambda_0}\frac{1}{(M-1)}\frac{1}{r^2}\mathrm{tr}(\Sigma A^{-1}), \\
&= \frac{2(M-1)\lambda_0 r d - \mathrm{tr}(\Sigma A^{-1})}{4(M-1)\lambda_0 r^2}, \\
\text{if} \quad d \ge \frac{\mathrm{tr}(\Sigma A^{-1})}{2(M-1)\lambda_0 r}, &\qquad \text{we have} \quad \frac{\partial C_2}{\partial r} > 0.
\end{aligned} \tag{32}$$

□

Table 1. Experimental Settings of Table 2

Settings	b_1	r	r_b	M
Multistage	256	0.5	0.5	3
Medium Initial Batch	512	0.5	0.5	3
Large Initial Batch	1,024	0.5	0.5	3
Batch Undershrinkage	256	0.1	0.5	3
Batch Overshrinkage	256	0.9	0.5	3

All settings use the same QHM optimizer ($\beta_1 = 0.9$, $\nu = 0.5$, $M = 3$), but with different batch size schedulers.

8 EXPERIMENTS

We divide our experimental evaluations into two parts. In Section 8.1, we present the empirical evidence of the effectiveness of multistage QHM, while in Section 8.2, we show how a moderately large learning rate helps two different distributed SGD generalize. All the experiments could be reproduced with our public code repository.[9]

8.1 Evaluation of Multistage QHM

In this section, we present our empirical experiments to analyze the performance of our proposed multistage QHM.

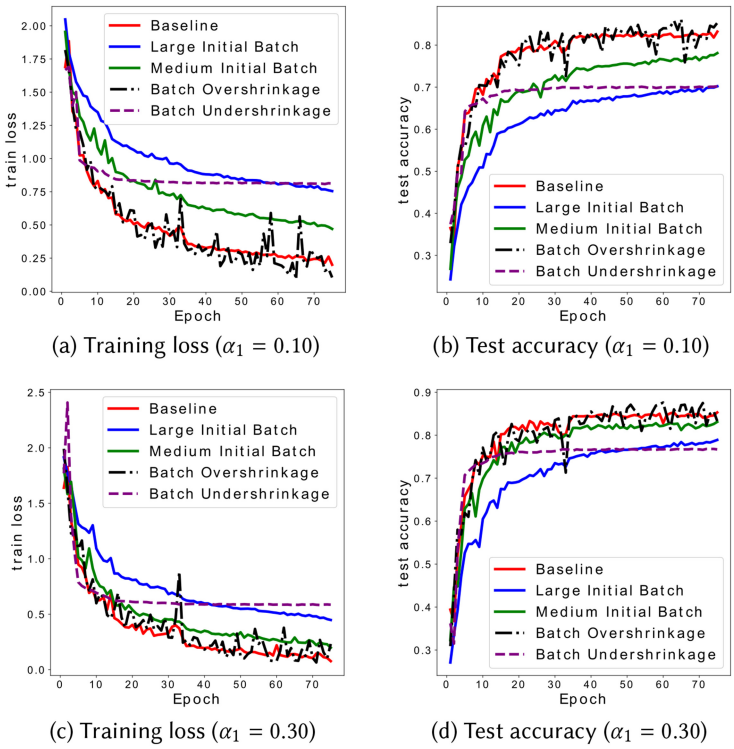
We first study how our scheduler affects the training and generalization of a CIFAR-10 image classification task. We fit the model which achieves state-of-the-art performance in various classification benchmarks, ResNet, with different optimizers and tuning schemes. We tried 110-layer ResNet with pre-activation (PreAct-ResNet-110) [27], and 20-layer ResNet with no pre-activation (ResNet-20) to ensure the robustness with varying model size. All experiments are run with NVIDIA Quadro RTX 8000 GPU.

8.1.1 Batch Size Scheduler. We start from hyperparameter $\{b_i\}_{i=1}^{M}$ (i.e., batch size), which is held constant throughout the entire learning process in most of the existing optimizers. We fit PreAct-ResNet-110 on CIFAR-10 and run for 75 epochs. We report the test accuracy after 75 epochs for each batch scheduler. Please refer to Table 1 for our experimental settings. "Overshrinkage" refers to faster batch size decay than step size decay, and "undershrinkage" refers to the opposite scenario. We present our results in Figure 4 and Table 2.

Table 2 is in accordance with our claim in Section 5.2. At any given initial learning rate, when batch size decays at least as fast as learning rate, both training loss and test accuracy are better than other schedulers by a non-trivial margin. Though overshrinkage mostly gets slightly better generalization result, we do see a less stable and more bumpy learning curve in Figure 4. Therefore, we suggest overshrinkage only has a marginal advantage and we could set $r_b = r$ for training with more stability.

8.1.2 ν Scheduler. We then study the effect of ν in generalization ability. ν does not appear in some optimizers (e.g., SGD/NAG/SHB). [53] proposed a $\nu = 0.7$ based on empirical experience and achieved state-of-the-art performance in one classification benchmark with such specification. Based on our generalization theorem in Section 5.1, we suggest a $\nu = 0.5$ in the multistage QHM paradigm. We report our result when training ResNet-20 on CIFAR-10 for 50 epochs with QHM in Figure 5. All hyperparameters are exactly the same except for initial step size and ν.

[9] https://github.com/jsycsjh/centralized-asynchronous-tuning.



(a) Training loss ($\alpha_1 = 0.10$) (b) Test accuracy ($\alpha_1 = 0.10$)

(c) Training loss ($\alpha_1 = 0.30$) (d) Test accuracy ($\alpha_1 = 0.30$)

Fig. 4. Training curve and test accuracy of various batch size schedulers. Baseline is multistage QHM.

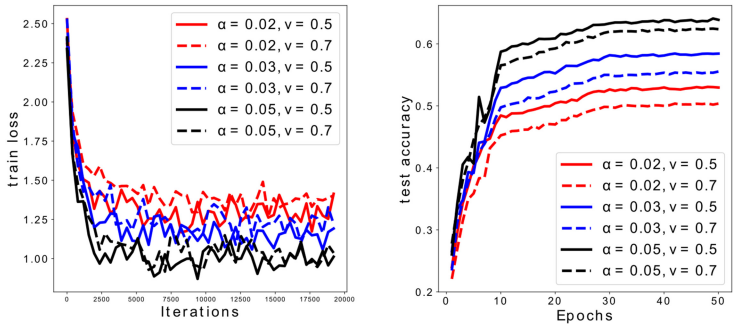
Table 2. The Effect of Batch Size Scheduler on Deep Models (PreAct-ResNet-110 on CIFAR-10)

α_1	Multistage	Medium Initial	Large Initial	Undershrinkage	Overshrinkage
0.05	73.9%	66.0%	54.4%	52.6%	82.1%
0.10	81.7%	74.6%	64.3%	62.9%	84.6%
0.15	83.2%	78.1%	70.2%	70.2%	85.3%
0.20	84.2%	81.2%	72.5%	75.0%	84.9%
0.25	85.6%	81.4%	77.6%	77.0%	87.3%
0.30	85.3%	83.1%	78.9%	76.7%	83.3%

The pattern is consistent with our theoretical findings in Section 5: (i) batch size should decrease at least as fast as learning rate; (ii) initial batch size b_1 should be small; (iii) α_1 should be searched monotonically.

We could observe different ν affect generalization ability at any of the given initial step sizes. It could be seen in Figure 5(a) that training curves are quite mixed together and no obvious training advantage could be detected from $\nu = 0.7$ to $\nu = 0.5$. However, $\nu = 0.5$ is better than $\nu = 0.7$ in the test accuracy by a large margin. The gradual increase in accuracy from a smaller initial step size to a larger initial step size again supports our monotone searching method.

8.1.3 Learning Rate Scheduler. From Table 2, it is clear that a larger initial learning rate α_1 could boost test accuracy (only if α_1 is not too large to diverge). In Table 3, we show test accuracy after running for 75 epochs with different optimizers (learning curves flatten long before 75 epochs in all cases). As QHM covers many optimizers as special cases, it is expected in Table 3 that the two most popular momentum variants SHB and NAG also satisfy this trend. And interestingly, Adam [39]



(a) Train loss (b) Test accuracy

Fig. 5. Training curve and test accuracy of different ν settings.

Table 3. The Effect of Initial Learning Rate α_1 on Deep Models (PreAct-ResNet-110 on CIFAR-10)

α_1	NAG	α_1	SHB	α_1	Adam
0.005	84.1%	0.10	81.3%	0.001	86.6%
0.010	86.7%	0.20	84.9%	0.002	87.3%
0.015	87.8%	0.40	87.2%	0.004	87.8%
0.020	87.8%	0.80	87.9%	0.008	88.8%
0.025	88.6%	1.60	88.7%	0.016	89.5%

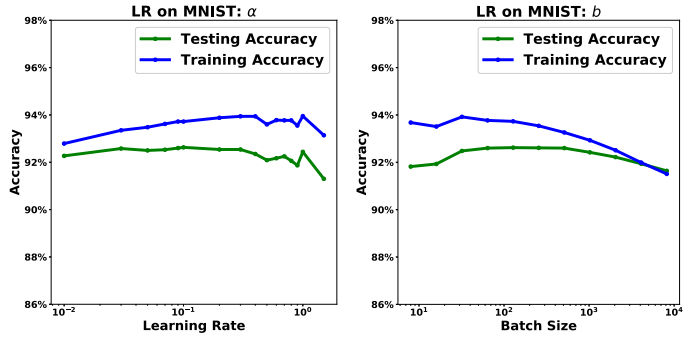
The pattern indicates momentum schemes could search α_1 monotonically.

optimizer, which is not characterized by QHM formulation, also exhibits better generalizability with an increasing initial step size. The reason why the appropriate range of step size varies for different algorithms is that each optimizer's effective step size is scaled differently [20, 80]. The test accuracy's positive relationship with initial step size helps us to simplify the procedure to search for α_1. For a given search space, the practitioner does not have to try out every possible value of α_1. Instead, he could start from the smallest to largest α_1, and stop when the next possible α_1 does not give a better test result, which shortens the tuning time.

We fit a *shallow* logistic regression on MNIST and a *deep* Preact-ResNet-110 [27] on CIFAR-10. We sweep a large range of batch size and learning rate. We could observe in Figures 6 and 7, only deep model will exhibit apparent learning rate/batch size impact, while shallow model does not. This is exactly consistent with our theoretical finding in Corollary 5.1, where we point out that only deep models will require the linear scaling rule between learning rate and batch size.

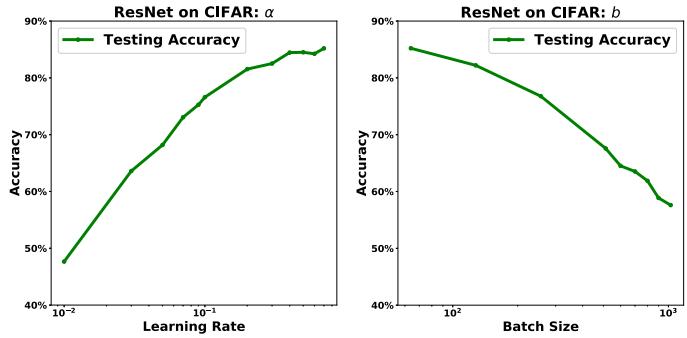
Fine-tuning is commonly used in real-world deployment, where the model is pretrained with some large source data by institutions possessing sufficient computational resources, then practitioners download the pre-trained models and only retrain a part of the model. In order to see whether our derived training guidelines work also in fine-tuning, We pretrain with ImageNet and then fine-tune the model on Aircraft [56] dataset. We test ResNet, VGG, DenseNet with SGD, SHB, NAG, and PID ($k_P = -0.1$ and $k_D = 3.0$). We sweep through a large range of α from 10^{-5} to 10^{-3} and report the test performance in Table 4. In all scenarios, larger α ensures better generalization consistently.

Deep learning models are susceptible to adversarial attacks and adversarial training is a popular approach to generating robust models. In order to see whether our proposed guidelines work with robust models, we test on CIFAR10 with SENet/ResNet trained by **Projected Gradient Descent**



(a) Impact of Learning Rate (b) Impact of Batch Size

Fig. 6. We fit a *shallow* logistic regression on MNIST. The optimizer is SHB with $\nu = 1$ and $\beta = 0.9$. We sweep a large range of batch size and learning rate. It is straightforward to see batch size and learning rate does not have much influence on the test performance.



(a) Impact of Learning Rate (b) Impact of Batch Size

Fig. 7. We fit a *deep* Preact-ResNet-110 [27] on CIFAR-10. The optimizer is SHB with $\nu = 1$ and $\beta = 0.9$. We sweep a large range of batch size and learning rate. In contrast to Figure 6. It is straightforward to see batch size and learning rate have much influence on the test performance.

Table 4. We Pretrain with ImageNet and then Fine-tune the Model on Aircraft [56] Dataset

α	ResNet+SGD	VGG+SGD	DenseNet+SGD	ResNet+SHB	ResNet+NAG	ResNet+PID
1e-05	1.50%	2.40%	1.32%	17.85%	16.02%	1.68%
2e-05	3.03%	6.51%	2.31%	32.37%	29.64%	2.49%
4e-05	6.78%	17.19%	5.55%	51.67%	52.87%	6.78%
5e-05	9.60%	23.21%	6.72%	59.23%	60.55%	8.97%
6e-05	10.68%	31.62%	9.81%	65.38%	65.53%	11.43%
8e-05	15.39%	46.44%	14.91%	71.11%	71.77%	14.46%
1e-04	20.43%	53.17%	18.96%	73.51%	73.63%	17.40%
2e-04	35.49%	67.09%	34.59%	76.54%	77.50%	32.37%
3e-04	45.42%	68.74%	48.18%	77.74%	77.56%	44.82%
4e-04	57.67%	69.49%	56.11%	78.49%	77.68%	51.88%
5e-04	63.64%	69.22%	62.83%	79.57%	79.54%	60.43%

We sweep through a large range of α from 10^{-5} to 10^{-3}. Larger α ensures better generalization consistently.

Table 5. We Test on CIFAR-10 Under the l_∞ Threat Model of Perturbation Budget $\frac{8}{255}$, without Additional Data

α	SENet	PreAct	α	SENet	PreAct	α	SENet	PreAct	α	SENet	PreAct
0.002	32.46%	34.79%	0.016	45.84%	46.02%	0.028	47.19%	46.24%	0.04	47.47%	47.07%
0.004	36.82%	38.69%	0.018	45.91%	46.21%	0.03	47.19%	46.61%	0.1	48.36%	47.21%
0.006	39.68%	41.44%	0.02	46.45%	46.30%	0.032	47.30%	46.72%	0.15	48.99%	48.05%
0.008	41.92%	43.19%	0.022	46.42%	45.92%	0.034	47.41%	46.75%	0.2	49.36%	49.04%
0.01	43.38%	44.11%	0.024	46.52%	46.47%	0.036	47.44%	46.30%	0.25	50.24%	49.34%
0.012	44.33%	44.92%	0.026	47.16%	46.53%	0.038	47.49%	46.39%	0.3	50.83%	50.01%

The vanilla PGD-AT framework [55] is used to produce adversarially robust model. The model is evaluated under 10-steps PGD attack (PGD-10) [55]. The architectures we test are SENet-18 [29], and Preact-ResNet-18 [27].

Adversarial Training (PGD-AT) [55]. The adversarial testing accuracy is reported in Table 5. Moderately larger α promotes better generalization universally in all settings.

There are several off-shelf learning rate auto-tuners, the two most popular ones are cosine annealing scheduler [52] and one cycle scheduler [68, 69]. We adopt the default implementation of these two tuners[10] and sweep a large range of α_1 for these two autotuners. We pick the best performance cosine annealing and one cycle could get and compare them to our multistage QHM in Figure 8. With careful hand-tuning, cosine annealing scheduler could match the performance of multistage QHM. However, multistage QHM has at least two advantages over cosine annealing. First, in Figure 8(a), the training curve for multistage QHM is steeper than cosine annealing, indicating it trains faster. Second, cosine annealing has a complex learning rate adjustment rule and therefore, practitioners are more difficult to understand its intrinsic process. A byproduct of its black-box nature is it lacks a theoretical convergence guarantee or justification why it improves test accuracy.

Adam and RMSProp are the two most widespread adaptive gradient methods. We adopt the default implementation of these two optimizers[11] and sweep a large range of α_1 for these two optimizers as the initial learning rate is the most influential factor for these two optimizers [80]. We pick the best performance RMSProp and Adam could get and compare them to our multistage QHM in Figure 9. All three optimizers achieve practically 0 training loss in Figure 9(a). Multistage QHM is better than RMSProp in Figure 9(b). Hand-tuned Adam matches Multistage QHM, with a slightly worse end-of-training test accuracy. This is in accordance with [37, 80], also observing momentum could achieve a very small advantage than adaptive algorithms. Therefore, our multistage QHM has a natural tuning process that is straightforward to reason about, and achieves as good performance as complex counterparts in benchmark task.

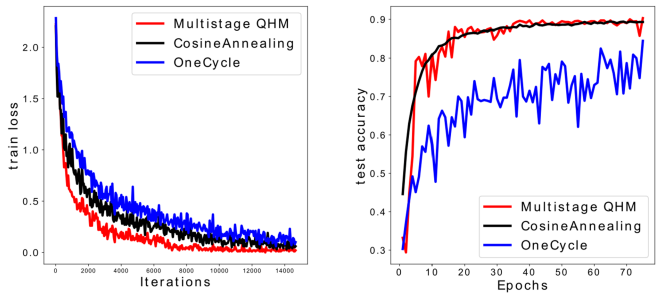
8.2 Evaluation of Asynchronous SGD

In this section, we present our empirical experiments to attest Theorem 5. We train a ResNet-50 [25] on the tiny ImageNet dataset.[12] Each node in our experiments has 4 NVIDIA TESLA K80 GPUs. Our experimental setting could be found in Table 6. Specifically, we use a similar experimental protocol as in [6, 21]. Every node uses a mini-batch size of 256. We run all experiments for 90 epochs (we observe all learning curves reach a plateau far before 90 epochs). We employ learning

[10]Please check https://pytorch.org/docs/stable/optim.html for their implementations.

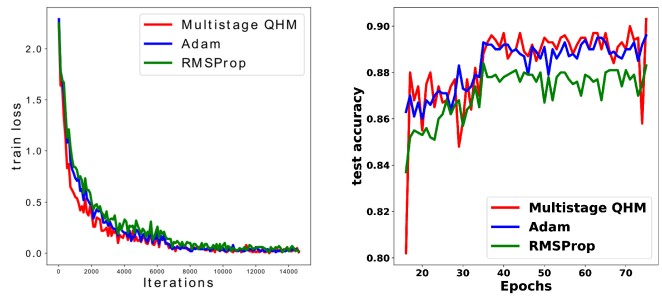
[11]Please check https://pytorch.org/docs/stable/_modules/torch/optim/adam.html#Adam and https://pytorch.org/docs/stable/_modules/torch/optim/rmsprop.html#RMSprop for their default specifications.

[12]Tiny Imagenet is a micro version of Imagenet, but is still a much larger dataset than CIFAR10. Tiny ImageNet contains 200 image classes, a training dataset of 100,000 images, a validation dataset of 10,000 images, and a test dataset of 10,000 images. The image size is 64×64.



(a) Train loss (b) Test accuracy

Fig. 8. Training curve and test accuracy of different learning rate schedulers (PreAct-ResNet-110 on CIFAR-10).



(a) Train loss (b) Test accuracy

Fig. 9. Training curve and test accuracy of different optimizers (PreAct-ResNet-110 on CIFAR-10).

Table 6. Experimental Setting for Distributed SGD

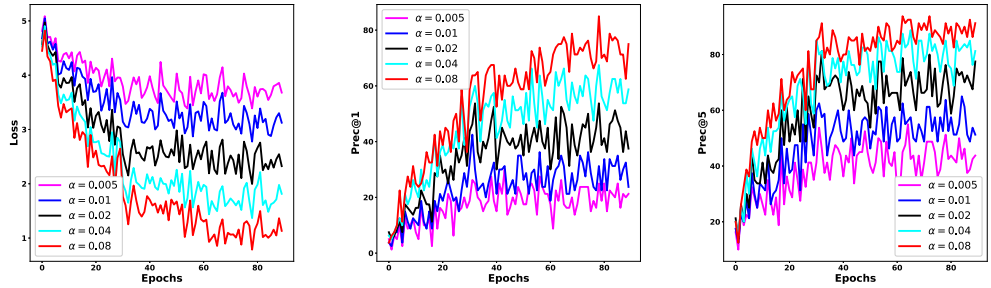
b per node	epochs	warm-up	decay factor (epoch)	network interface	GPU
256	90	TRUE	0.1 (30, 60, 80)	InfiniBand	NVIDIA K80 (4 per node)

rate warm-up (i.e., increasing the learning rate to a large value over a certain number of training iterations followed by decreasing the learning rate), which is known to improve performance in large-scale training (the learning rate warms up during the first five epochs as in [21]). The learning rate is decayed by a factor of 10 at epochs 30, 60, and 80. Our nodes communicate over 100 Gbps InfiniBand, which is widely used in high-performance computing clusters. We employ 20 CPU cores in each node.

We test two different popular implementations of distributed and asynchronous SGD: **ALLREDUCE SGD** (**AR-SGD**) [2, 21] and **Decentralized Parallel SGD** (**D-PSGD**) [47, 48], which differ in how the parameter server aggregates information and how worker machines communicate.

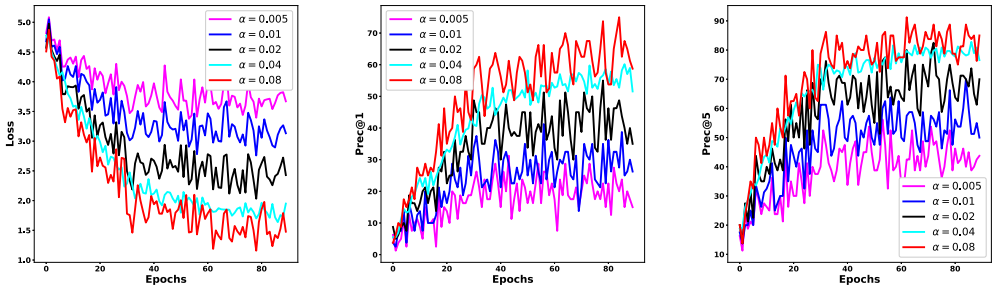
We sweep a wide range of initial α with a fine grid from $\alpha = 0.001$ to $\alpha = 0.1$. The pattern is consistent, that a larger initial learning rate guarantees a better result, in both AR-SGD and D-PSGD. We plot five learning curves with learning rates equally separated in a log scale $\alpha = 0.005, 0.01, 0.02, 0.04, 0.08$. Figures 10 and 11 present experimental results for AR-SGD and D-PSGD, respectively.

Note that the learning curve in distributed learning is much more bumpy than centralized learning curve (see Figure 4 for example). However, it does not indicate the curve does not converge yet. The bumpy curve is due to system behavior, a much more diverse dataset, and communication



(a) Validation loss (b) Validation Precision@1 (c) Validation Precision@5

Fig. 10. Validation loss and precision curves when training ResNet-50 on the tiny ImageNet dataset with different initial learning rates α by AR-SGD. Over a wide range of α, the pattern is very clear, where larger the initial α ensures better generalization.



(a) Validation loss (b) Validation Precision@1 (c) Validation Precision@5

Fig. 11. Validation loss and precision curves when training ResNet-50 on the tiny ImageNet dataset with different initial learning rates α by D-PSGD. Larger initial α ensures better generalization consistently.

overhead. Actually, we run the experiments long after 90 epochs, but the performances for these learning curves do not significant improve.

In Figures 10 and 11, we can see over a wide range of α, the improvement brought by increasing learning rate is consistent in both AR-SGD and D-PSGD. This observation proves the effectiveness of Theorem 5, and could decrease the search space by only monotonically search the initial α.

9 RELATED WORK

9.1 Tuning Approaches

The hyperparameter search space in state-of-the-art deep learning systems can be too high-dimensional for practitioners to explore manually, especially when deep learning has been increasingly important for many interesting real-world problems, e.g., document processing, time series analysis, and biomedical data mining [31, 35, 54, 73–76, 82, 83]. Costly hyperparameter tuning is the main obstacle of automated machine learning. Many approaches have been developed to help tune learning rate including random search, Bayesian optimization, geometric decay, cosine annealing with warm restarts, and cyclic learning rate policy to name a few [9, 18, 32, 52, 68, 69, 72]. However, they only consider the momentum-free scenario, or hold momentum parameter constant.

Due to the empirical advantage of SGD momentum, a lot of effort has also been devoted to adaptive momentum scheduler [50, 77, 85]. Most of the existing works are focused on either SHB or NAG and a general consensus with these two optimizers is that the momentum parameter needs to be increased. Recent articles have also noted the impact of scheduling batch size [28, 70, 71],

which mainly suggest linearly scale batch size with learning rate, but in an increasing direction. We theoretically explain why it may slightly sacrifice test accuracy.

Multistage QHM is a more general and flexible framework than existing works that include not only SHB and NAG. Every hyperparameter is allowed to vary and is theoretically justified to improve generalization.

9.2 Generalization Analysis

Our work aims at theoretically and empirically justifying multistage QHM to generalize. A number of recent works empirically report the influence of hyperparameters, largely on batch size and learning rate, and provide practical tuning guidelines, e.g., [34, 36] connected the training dynamics, and the generalization to the ratio of batch size over step size.

Our generalization analysis relies on PAC-Bayesian inequalities [58], and we refer readers to a comprehensive review of PAC-Bayesian learning and references therein [22]. [24, 51] proved a PAC-Bayesian bound for vanilla SGD. Notably, [24] showed that the ratio of batch size and step size could not be too large to ensure good generalization. Our work focuses on characterizing generalization on a class of momentum schemes, and covers their vanilla SGD analysis as a special case. Moreover, we show that our second order bound is better than their first order bound as it implicates Corollary 5.1 that could not be derived only from first order bound.

9.3 Convergence Analysis

The convergence of the vanilla SGD has been heavily studied and key results are highlighted in [10]. Despite the widespread use of the stochastic momentum method, there are limited definitive theoretical convergence guarantees. [14, 42, 87] studied momentum schemes but only for deterministic gradients. [84] studied the SHB and NAG methods and derived a convergence rate with bounded gradient assumption (which we do not require), and they obtained rates that are slower than those for SGD. Recent work establishes convergence guarantees in different settings [7, 11, 41, 50, 79], largely only for the SHB or NAG, which are not directly generalizable to other momentum schemes.

9.4 Asynchronous SGD

For a comprehensive coverage of advances in asynchronous parallel and distributed optimization, we refer readers to [5] and references therein. Our article studies how basic hyperparameters impact the generalization performances of ASGD. [3, 59] both observed the momentum method has analogous performance as the asynchronous method, specifically, asynchrony is equivalent to adding an implicit momentum to the SGD iteration, and [59] further proposed to counteract asynchrony by reducing explicit momentum. [23] also focused on the momentum term, where a variation of shifted momentum is proposed and empirically superior compared to no momentum or with the original momentum. [65] conducted a comprehensive experimental evaluation and showed how the increasing batch size will impact the required number of training steps to reach a goal out-of-sample error. [65] demonstrated that optimal batch size depends heavily on model, training algorithm, and dataset, and popular heuristic (e.g., linearly scaling the learning rate with the batch size) typically does not universally apply to every case. [19, 86] both proposed a staleness-dependent learning rate scheme, specifically, to scale the learning rate inversely to the delay in order to reach a better generalization performance. [43] extended from [65] and studied the interaction of data parallelism and sparsity (via parameter pruning) in deep networks through extensive empirical experiments and confirmed a general scaling rule between increasing batch size and the necessary number of training steps required. Our study is the first generalization

bound of ASGD to our best knowledge, and based on the bound, we point out a moderately large learning rate is beneficial.

10 CONCLUSIONS

In this article, we propose a powerful and general pipeline to derive PAC-Bayesian generalization bounds for stochastic algorithms, which has applications in both *centralized* and *distributed* environments. In *centralized* computing, this article is the first general tuning guideline applicable for almost all popular momentum variants with a theoretical guarantee to boost generalization. Our multistage paradigm shortens the tuning time without sacrificing any learning performances. Our theoretical findings on the impact of initial step size and shrinking batch size are transferable to new momentum variants and are of independent interest for practitioners to tune their own optimizers. In *distributed* computing, our study is the first generalization theory of Asynchronous SGD, and based on our derived bound, we point out how learning rate impacts ASGD's performance and could effectively reduce the manual search space.

A APPENDIX

In this section, we give the proof of Theorems 3, 4, and other important lemmas. We keep the key proof steps and omit many algebraic transformations due to the page limit.

A.1 Proof of Lemma 1

Proof. We start from the Taylor expansion of $\Sigma_\theta = \Sigma_\theta^{(0)} + \alpha\Sigma_\theta^{(1)} + \frac{\alpha^2}{2}\Sigma_\theta^{(2)} + O(\alpha^3)$. We know from [20]:

$$\begin{aligned} \Sigma_\theta^{(0)} = 0 \quad A\Sigma_\theta^{(1)} + \Sigma_\theta^{(1)}A &= \frac{1}{b}\Sigma, \\ \Sigma_{\theta d}^{(1)} + \Sigma_{d\theta}^{(1)} = A\Sigma_\theta^{(1)} + \Sigma_\theta^{(1)}A &- \frac{2(1+\beta-\nu\beta)}{1+\beta}\frac{1}{b}\Sigma \quad , \\ A\Sigma_\theta^{(2)} + \Sigma_\theta^{(2)}A = \frac{2\nu\beta}{1-\beta}&\left(\Sigma_{d\theta}^{(1)}A + A\Sigma_{\theta d}^{(1)}\right) + 2A\Sigma_\theta^{(1)}A, \end{aligned} \tag{33}$$

where $\Sigma_{\theta d}$ is the covariance matrix between d and θ. Recall Taylor expansion: $\mathrm{tr}(\Sigma_\theta) = \alpha\mathrm{tr}(\Sigma_\theta^{(1)}) + \frac{\alpha^2}{2}\mathrm{tr}(\Sigma_\theta^{(2)}) + O(\alpha^3)$. Therefore, we derive formula for $\mathrm{tr}(\Sigma_\theta^{(1)})$ and $\mathrm{tr}(\Sigma_\theta^{(2)})$. We could get:

$$\begin{aligned} &\Sigma_{\theta d}^{(1)}A^{-1} + \Sigma_{d\theta}^{(1)}A^{-1} \\ &= A\Sigma_\theta^{(1)}A^{-1} + \Sigma_\theta^{(1)} - \frac{2(1+\beta-\nu\beta)}{1+\beta}\frac{1}{b}\Sigma A^{-1} \\ &\quad A\Sigma_\theta^{(2)}A^{-1} + \Sigma_\theta^{(2)} \\ &= \frac{2\nu\beta}{1-\beta}\left(\Sigma_{d\theta}^{(1)} + A\Sigma_{\theta d}^{(1)}A^{-1}\right) + 2A\Sigma_\theta^{(1)}. \end{aligned} \tag{34}$$

Taking trace on both sides and recall the trace is invariant under cyclic permutations: $\mathrm{tr}(\Sigma_{\theta d}^{(1)}A^{-1}) + \frac{1+\beta-\nu\beta}{1+\beta}\frac{1}{b}\mathrm{tr}(\Sigma A^{-1}) = \mathrm{tr}(\Sigma_\theta^{(1)})$ and $\mathrm{tr}(\Sigma_\theta^{(2)}) = \frac{2\nu\beta}{1-\beta}\mathrm{tr}(\Sigma_{\theta d}^{(1)}) + \mathrm{tr}(A\Sigma_\theta^{(1)})$.

We know from Equation (33): $2\mathrm{tr}(A\Sigma_\theta^{(1)}) = \frac{1}{b}\mathrm{tr}(\Sigma)$ and $2\mathrm{tr}(\Sigma_{\theta d}^{(1)}) = \mathrm{tr}(A\Sigma_\theta^{(1)} + \Sigma_\theta^{(1)}A) - \frac{2(1+\beta-\nu\beta)}{1+\beta}\frac{1}{b}\mathrm{tr}(\Sigma)$.

Therefore, we have $\mathrm{tr}(\Sigma_\theta^{(2)}) = (\frac{\nu\beta}{1-\beta}(1 - \frac{2(1+\beta-\nu\beta)}{1+\beta}) + \frac{1}{2})\frac{1}{b}\mathrm{tr}(\Sigma)$.

From $A\Sigma_\theta^{(1)}A^{-1} + \Sigma_\theta^{(1)} = \frac{1}{b}\Sigma A^{-1}$ we could get $\mathrm{tr}(\Sigma_\theta^{(1)}) = \frac{1}{2b}\mathrm{tr}(\Sigma A^{-1})$.

Together we finish the proof regarding trace. Next we prove the determinant part. From $A\Sigma_\theta^{(1)} + \Sigma_\theta^{(1)}A = \frac{1}{b}\Sigma$ we could get $A\Sigma_\theta + \Sigma_\theta A = \frac{\alpha}{b}\Sigma + O(\alpha^2)$.

Assuming Σ_θ is symmetric, $\Sigma_\theta A = \frac{\alpha}{2b}\Sigma + O(\alpha^2)$. It is not difficult to show: $\det(\Sigma_\theta) = (\frac{\alpha}{2b})^d \det(\Sigma A^{-1}) + O(\alpha^2)$ given $\Sigma_\theta = \frac{\alpha}{2b}\Sigma A^{-1} + O(\alpha^2)$, with d is the dimension. Due to page limit, we omit the detail. □

As we are mainly concerned about how α, β, ν affect the trend of generalization bound, we ignore higher order terms for now. The experiments have shown that our approximations are satisfactory.

A.2 Proof of Theorem 3

Proof. With Theorem 2 and Lemma 1, we are ready to prove Theorem 3. Recall the density of prior and posterior distributions:

$$f_P = \frac{1}{\sqrt{2\pi \det(\lambda_0 I_d)}} \exp\left\{ -\frac{1}{2}(\theta - \theta_0)^T(\lambda_0 I_d)^{-1}(\theta - \theta_0)\right\},$$
$$f_Q = \frac{1}{\sqrt{2\pi \det(\Sigma_\theta)}} \exp\left\{ -\frac{1}{2}\theta^T \Sigma_\theta^{-1}\theta\right\}. \tag{35}$$

We calculate their KL$(Q||P)$ as follows: KL$(Q||P) = \int (\frac{1}{2}\log\frac{|\lambda_0 I_d|}{|\Sigma_\theta|} - \frac{1}{2}\theta^T\Sigma_\theta^{-1}\theta + \frac{1}{2}(\theta-\theta_0)^T(\lambda_0 I_d)^{-1}$ $(\theta - \theta_0))f_Q(\theta)d\theta = \frac{1}{2}\{\mathrm{tr}((\lambda_0 I_d)^{-1}\Sigma_\theta) + \theta_0^T(\lambda_0 I_d)^{-1}\theta_0 - d + \log\frac{|\lambda_0 I_d|}{|\Sigma_\theta|}\} = \frac{1}{2\lambda_0}\theta_0^T\theta_0 - \frac{d}{2} + \frac{d}{2}\log\lambda_0 + \frac{1}{2\lambda_0}\mathrm{tr}(\Sigma_\theta) - \frac{1}{2}\log|\Sigma_\theta|$.

Application of the determinant and trace from Lemma 1 here will complete our proof. □

A.3 Proof of Theorem 4

Proof. We now study $\mathcal{R}(\eta_{k+1}) - \mathcal{R}(\eta_k)$:

$$\mathbb{E}_{\xi_k}[\mathcal{R}(\eta_{k+1})] \leq$$
$$= \mathcal{R}(\eta_k) + \mathbb{E}_{\xi_k}[<\nabla\mathcal{R}(\eta_k), -\alpha_k\hat{g}_k>] + \frac{L\alpha_k^2}{2}\mathbb{E}_{\xi_k}[\|\hat{g}_k\|^2]. \tag{36}$$

Taking full expectation $\mathbb{E} = \mathbb{E}_{\xi_1}\mathbb{E}_{\xi_2}\ldots\mathbb{E}_{\xi_k}$ on both sides:

$$\begin{aligned}
&\mathbb{E}[\mathcal{R}(\eta_{k+1})] \\
&\leq \mathbb{E}[\mathcal{R}(\eta_k)] + \mathbb{E}[<\nabla\mathcal{R}(\eta_k), -\alpha_k g_k>] + \frac{L\alpha_k^2}{2}\mathbb{E}[\|\hat{g}_k\|^2] \\
&\leq \mathbb{E}[\mathcal{R}(\eta_k)] + \frac{L\alpha_k^2}{2}\mathbb{E}[\|\hat{g}_k\|^2] - \alpha_k\mathbb{E}[\|g_k\|^2] \\
&\quad + \alpha_k\frac{c_k}{2}L^2\mathbb{E}[\|\eta_k - \theta_k\|^2] + \alpha_k\frac{1}{2c_k}\mathbb{E}[\|g_k\|^2],
\end{aligned}$$

for $c_k > 0$ as any positive constant. And we know $\eta_k - \theta_k = -\frac{\alpha_k\beta_k\nu_k}{1-\beta_k}d_{k-1}$:

$$\begin{aligned}
\mathbb{E}[\mathcal{R}(\eta_{k+1})] \leq \mathbb{E}[\mathcal{R}(\eta_k)] &+ \alpha_k^3\frac{c_k}{2}L^2\left(\frac{\beta_k\nu_k}{1-\beta_k}\right)^2\mathbb{E}[\|d_{k-1}\|^2] \\
&+ \left(\alpha_k\frac{1}{2c_k} - \alpha_k\right)\mathbb{E}[\|g_k\|^2] + \frac{L\alpha_k^2}{2}\mathbb{E}[\|\hat{g}_k\|^2].
\end{aligned} \tag{37}$$

Let us make a small detour and first prove Lemma 2.

Proof of Lemma 2. We know $\mathbb{V}[y_k] = \mathbb{E}[\|y_k - \Sigma_{i=1}^k a_{k,i} g_i\|^2] = \mathbb{E}[\|\Sigma_{i=1}^{k-1} a_{k,i}(g_i - \hat{g}_i) + (1 - \nu_k\beta_k)(g_k - \hat{g}_k)\|^2] \overset{(i)}{\leq} (6 - 4\beta_1 - 4\beta_k\nu_k + 2\beta_k^2\nu_k^2)\sigma^2$, where (i) follows from that $\{\hat{g}_k\}_{k\in\mathbb{N}}$ are independent from each other and $\mathbb{E}_{\xi_k}[\|\hat{g}_k - g_k\|^2] \leq \sigma^2$, and also Lemma 4 in [50]. We know $\Sigma_{i=1}^k a_{k,i} = 1 - \nu_k \prod_{i=1}^k \beta_i$, thus we have:

$$
\begin{aligned}
&\mathbb{E}\left[\left\| g_k - \frac{1}{1 - \nu_k \prod_{i=1}^k \beta_i} \Sigma_{i=1}^k a_{k,i} g_i \right\|^2\right] \\
&= \frac{1}{(1 - \nu_k \prod_{i=1}^k \beta_i)^2} \mathbb{E}\left[\left\| \Sigma_{i=1}^k a_{k,i}(g_k - g_i) \right\|^2\right] \\
&\overset{(i)}{\leq} \frac{1}{(1 - \nu_k \prod_{i=1}^k \beta^i)^2} \Sigma_{i,j=1}^k a_{k,i} a_{k,j} \left(\frac{1}{2}\mathbb{E}[\|g_k - g_j\|^2] + \frac{1}{2}\mathbb{E}[\|g_k - g_i\|^2]\right) \\
&= \frac{1}{1 - \nu_k \prod_{i=1}^k \beta_i} \Sigma_{i=1}^k a_{k,i} \mathbb{E}[\|g_k - g_i\|^2], \\
&\overset{(ii)}{\leq} \frac{1}{1 - \nu_k \prod_{i=1}^k \beta_i} \Sigma_{i=1}^k a_{k,i} \left((k-i)\Sigma_{j=i}^{k-1} \mathbb{E}[\|g_{j+1} - g_j\|^2]\right), \qquad (38) \\
&\overset{(iii)}{\leq} \frac{1}{1 - \nu_k \prod_{i=1}^k \beta_i} \Sigma_{i=1}^k a_{k,i} \left((k-i)L^2\Sigma_{j=i}^{k-1} \mathbb{E}[\|\theta_{j+1} - \theta_j\|^2]\right) \\
&= \frac{1}{1 - \nu_k \prod_{i=1}^k \beta_i} \Sigma_{j=1}^{k-1} \left(\Sigma_{i=1}^j a_{k,i}(k-i)\right) \mathbb{E}[\|\theta_{j+1} - \theta_j\|^2] L^2, \\
&\overset{(iv)}{\leq} \Sigma_{j=1}^{k-1} \frac{\nu_k \beta_k^{k-j} L^2}{1 - \nu_k \prod_{i=1}^k \beta_i} \left(k - j + \frac{\beta_k}{1 - \beta_k}\right) \mathbb{E}[\|\theta_{j+1} - \theta_j\|^2].
\end{aligned}
$$

where (i) follows from Cauchy–Schwarz inequality, (ii) follows from triangle inequality, (iii) follows from smoothness, (iv) follows from Proposition 5 in [50]. □

Back to Inequality Equation (37):

$$
\begin{aligned}
\mathbb{E}[\mathcal{R}(\eta_{k+1})] \leq \mathbb{E}[\mathcal{R}(\eta_k)] &+ \alpha_k^3 \frac{c_k}{2} L^2 \left(\frac{\beta_k \nu_k}{1 - \beta_k}\right)^2 \mathbb{E}[\|d_{k-1}\|^2] \\
&+ \left(\alpha_k \frac{1}{2c_k} - \alpha_k\right) \mathbb{E}[\|g_k\|^2] + \frac{L\alpha_k^2}{2} \mathbb{E}[\|\hat{g}_k\|^2].
\end{aligned}
$$

We study the following sequence: $L_k \triangleq \mathcal{R}(\eta_k) - \mathcal{R}^* + \Sigma_{i=1}^{k-1} q_i \|\theta_{k+1-i} - \theta_{k-i}\|^2$ following the idea from [50], where q_i are constants to be determined, omitting many algebraic transformations: we have

$$
\begin{aligned}
&\mathbb{E}[L_{k+1} - L_k] \\
&\leq \left(2\alpha_k c_k L^2 W_1^2 - \alpha_k + \frac{\alpha_k}{2c_k} + \frac{1}{2} L\alpha_k^2 + 4q_1\alpha_k^2\right) \mathbb{E}[\|g_k\|^2]
\end{aligned}
$$

$$
\begin{aligned}
&+ \frac{1}{2}L\alpha_k^2\sigma^2 + \alpha_k c_k L^2 W_1^2 \mathbb{E}\left[\|d_{k-1} - \Sigma_{i=1}^{k-1} a_{k-1,i} g_i\|^2\right] + 2q_1\alpha_k^2 \mathbb{E}\left[\|y_k - \Sigma_{i=1}^{k} a_{k,i} g_i\|^2\right] \\
&+ 4q_1\alpha_k^2\left(1 - \nu_k \prod_{i=1}^{k}\beta_i\right)^2 \mathbb{E}\left[\|g_k - \frac{1}{1-\nu_k\prod_{i=1}^{k}\beta_i}\Sigma_{i=1}^{k} a_{k,i} g_i\|^2\right] \\
&+ 2\alpha_k c_k L^2 W_1^2 \frac{1}{\beta_k^2}\left(1 - \prod_{i=1}^{k}\beta_i\right)^2 \mathbb{E}\left[\|g_k - \frac{1}{1-\prod_{i=1}^{k}\beta_i}\Sigma_{i=1}^{k} b_{k,i} g_i\|^2\right] \\
&+ \Sigma_{i=1}^{k-1}(q_{i+1} - q_i)\|\theta_{k+1-i} - \theta_{k-i}\|^2.
\end{aligned} \tag{39}
$$

where $b_{k,i} = (1-\beta_i)\prod_{j=i+1}^{k}\beta_j$. Set:

$$
q_1 = \frac{L^3 \frac{\alpha_1^2\nu_1^2(\beta_n+\beta_n^2)}{(1-\beta_1)(1-\beta_n)^2}}{\frac{(1-\beta_n)^2}{(\beta_n+\beta_n^2)L^2} - 4\alpha_1^2\nu_k},
$$

where n denotes the number of iterations, it is not difficult to verify the sum of last three terms is negative. Therefore, combining Lemma 5 in [50], we could have:

$$
\begin{aligned}
\mathbb{E}[L_{k+1} - L_k] \le &\left(2\alpha_k c_k L^2 W_1^2 - \alpha_k + \frac{\alpha_k}{2c_k} + \frac{1}{2}L\alpha_k^2 + 4q_1\alpha_k^2\right)\mathbb{E}[\|g_k\|^2] \\
&+ \left(\frac{1}{2}L\alpha_k^2 + 24\alpha_k c_k L^2 W_1^2 \frac{\beta_1(1-\beta_k)}{\sqrt{\beta_n+\beta_n^2}} + 2q_1\alpha_k^2(6 - 4\beta_1 - 4\beta_k\nu_k + 2\beta_k^2\nu_k^2)\right)\sigma^2.
\end{aligned} \tag{40}
$$

Note that if $W_1 = \frac{1}{48\sqrt{2}L}$, and $\frac{1-\beta_1}{\beta_1} \le 12\frac{1-\beta_n}{\sqrt{\beta_n+\beta_n^2}}$ as in Remark 6.2, we could get: $q_1 \le \frac{L}{4(1-\beta_1)}$.

We now have: $\mathbb{E}[L_{k+1} - L_k] \le -Q_{1,k}\mathbb{E}[\|g_k\|^2] + Q_{2,k}$, where $Q_{1,k} \triangleq -2\alpha_k c_k L^2 W_1^2 + \alpha_k - \frac{\alpha_k}{2c_k} - \frac{1}{2}L\alpha_k^2 - 4q_1\alpha_k^2$, and $Q_{2,k} \triangleq \frac{1}{2}L\alpha_k^2 + 24\alpha_k c_k L^2 W_1^2 \frac{\beta_1(1-\beta_k)}{\sqrt{\beta_n+\beta_n^2}} + 2q_1\alpha_k^2(6 - 4\beta_1 - 4\beta_k\nu_k + 2\beta_k^2\nu_k^2)\sigma^2$.

Set $c_i = \frac{1-\beta_i}{2L\alpha_i}$, and recall $\alpha_i = \frac{W_1(1-\beta_i)}{\beta_i\nu_i} = \frac{1-\beta_i}{24\sqrt{2}L\beta_i}$, we could verify $Q_{1,k} \ge \frac{\alpha_k}{2}$.

We could bound $Q_{2,k}$:

$$
Q_{2,k} \le \frac{1}{2}\alpha_k^2 L + 12\alpha_k^2 L\frac{\beta_k^2\nu_k^2}{(1-\beta_k)^2}\frac{\beta_1}{\sqrt{\beta_n+\beta_n^2}} + \frac{3+\beta_k^2\nu_k^2}{1-\beta_1}\alpha_k^2 L\sigma^2. \tag{41}
$$

As $L_1 \ge \mathbb{E}[L_1 - L_{k+1}] \ge \Sigma_{i=1}^{k} Q_{1,i}\mathbb{E}[\|g_i\|^2] - \Sigma_{i=1}^{k} Q_{2,i}$, as $k = T_1 + T_2 + \cdots + T_M$:

$$
\begin{aligned}
&\sum_{l=1}^{M}\frac{\alpha_l}{2}\sum_{i=T_1+T_2+\ldots+T_{l-1}+1}^{T_1+T_2+\ldots+T_l}\mathbb{E}[\|g_i\|^2] \\
&\quad\le L_1 + \sum_{l=1}^{M} T_l\left(\frac{1}{2}\alpha_k^2 L + 12\alpha_k^2 L\frac{\beta_k^2\nu_k^2}{(1-\beta_k)^2}\frac{\beta_1}{\sqrt{\beta_n+\beta_n^2}} + \frac{3+\beta_k^2\nu_k^2}{1-\beta_1}\alpha_k^2 L\right)\sigma^2.
\end{aligned} \tag{42}
$$

Dividing both sides by $\frac{1}{2}MW_2 = \frac{1}{2}M\alpha_l T_l$

$$\frac{1}{M}\sum_{l=1}^{M}\frac{1}{T_l}\sum_{i=T_1+T_2+\cdots+T_{l-1}+1}^{T_1+T_2+\cdots+T_l}\mathbb{E}[\|g_i\|^2]$$

$$\leq \frac{2(\mathcal{R}(\theta_1)-\mathcal{R}^*)}{MW_2}+\frac{1}{M}\sum_{l=1}^{M}\left(\alpha_k L+24\alpha_k L\frac{\beta_k^2\nu_k^2}{(1-\beta_k)^2}\frac{\beta_1}{\sqrt{\beta_n+\beta_n^2}}+\frac{6+2\beta_k^2\nu_k^2}{1-\beta_1}\alpha_k L\right)\sigma^2.$$

As the last stage is M, substitute $\beta_n = \beta_M$ will complete our proof. □

A.4 Proof of Lemma 3

Proof. We include the proof here for the sake of completeness, which is adapted from [59]. We use Equation (8) twice, and subtracting θ_t from θ_{t+1},

$$\theta_{t+1}-\theta_t = \theta_t-\theta_{t-1}-\alpha(\hat{g}(\phi_t,\zeta_t)-\hat{g}(\phi_{t-1},\zeta_{t-1})).$$

Let $\mathcal{T}$ denote the smallest σ-algebra under which all random staleness variables are measurable. Then, by the assumption that staleness and instance selection are independent, we have

$$\mathbb{E}[\theta_{t+1}-\theta_t|\mathcal{T}] = \mathbb{E}[\theta_t-\theta_{t-1}|\mathcal{T}]-\alpha(\mathbb{E}[\hat{g}(\phi_t)|\mathcal{T}]-\mathbb{E}[\hat{g}(\phi_{t-1})|\mathcal{T}]),$$

where the expectation is taken with respect to the random selection of ζ_t's.

Finally, integrating overall randomness,

$$\begin{aligned}
\mathbb{E}[\theta_{t+1}-\theta_t] &= \mathbb{E}[\theta_t-\theta_{t-1}]-\alpha\left(\sum_{l=0}^{\infty}q_l\mathbb{E}[\hat{g}(\theta_{t-l})]-\sum_{l=0}^{\infty}q_l\mathbb{E}[\hat{g}(\theta_{t-l-1})]\right),\\
&= \mathbb{E}[\theta_t-\theta_{t-1}]-\alpha\left(q_0\mathbb{E}[\hat{g}(\theta_t)]+\sum_{l=1}^{\infty}q_l\mathbb{E}[\hat{g}(\theta_{t-l})]-\sum_{l=0}^{\infty}q_l\mathbb{E}[\hat{g}(\theta_{t-l-1})]\right),\\
&= \mathbb{E}[\theta_t-\theta_{t-1}]-\alpha q_0\mathbb{E}[\hat{g}(\theta_t)]-\alpha\left(\sum_{l=0}^{\infty}q_{l+1}\mathbb{E}[\hat{g}(\theta_{t-l-1})]-\sum_{l=0}^{\infty}q_l\mathbb{E}[\hat{g}(\theta_{t-l-1})]\right),\\
&= \mathbb{E}[\theta_t-\theta_{t-1}]-\alpha q_0\mathbb{E}[\hat{g}(\theta_t)]+\alpha\sum_{l=0}^{\infty}(q_l-q_{l+1})\mathbb{E}[\hat{g}(\theta_{t-l-1})].
\end{aligned}$$

Let W_t denote the time the tth iteration takes. Under the assumption that $(W_t)_t$ are mutually independent and exponential work time $W_t \sim \exp\lambda$, we could obtain,

$$\tau_t \sim \text{Poisson}(\lambda(M-1)W_t).$$

It is straightforward to show that τ_t is geometrically distributed on $\{0,1,\dots\}$,

$$\tau_t \sim \text{Geom}(p), \qquad \text{where} \quad p=\frac{1}{M},$$

where M is the number of workers. Note that $\mathbb{E}[\tau_t] = M - 1$. Denote $\mu = (1 - \frac{1}{M})$ and $c = 1 - \mu$. We know the following,

$$\begin{aligned}
\mathbb{E}[\theta_{t+1} - \theta_t] &= \mathbb{E}[\theta_t - \theta_{t-1}] - \alpha q_0 \mathbb{E}[\hat{g}(\theta_t)] + \alpha \sum_{l=0}^{\infty} (q_l - q_{l+1}) \mathbb{E}[\hat{g}(\theta_{t-l-1})], \\
&= \mathbb{E}[\theta_t - \theta_{t-1}] - \alpha c \mathbb{E}[\hat{g}(\theta_t)] + \alpha \sum_{l=0}^{\infty} (c\mu^l - c\mu^{l+1}) \mathbb{E}[\hat{g}(\theta_{t-l-1})], \\
&= \mathbb{E}[\theta_t - \theta_{t-1}] - \alpha c \mathbb{E}[\hat{g}(\theta_t)] - c\mathbb{E}[\theta_t - \theta_{t-1}], \\
&= \mu \mathbb{E}[\theta_t - \theta_{t-1}] - \alpha c \mathbb{E}[\hat{g}(\theta_t)], \\
&= \mu \mathbb{E}[\theta_t - \theta_{t-1}] - \alpha(1 - \mu) \mathbb{E}[\hat{g}(\theta_t)].
\end{aligned}$$

Substitute $\mu = (1 - \frac{1}{M})$, we get,

$$\mathbb{E}[\theta_{t+1} - \theta_t] = \left(1 - \frac{1}{M}\right) \mathbb{E}[\theta_t - \theta_{t-1}] - \frac{\alpha}{M} \mathbb{E}[\hat{g}(\theta_t)]. \qquad \square$$

A.5 Proof of Lemma 4

Proof. We include the proof here for the sake of completeness, which is adapted from [57]. The first moments of the coupled stochastic differential Equations (24) are

$$d\mathbb{E}[v] = -\mu \mathbb{E}[v]dt - \alpha A \mathbb{E}[\theta]dt, \qquad d\mathbb{E}[\theta] = \mathbb{E}[v]dt. \tag{43}$$

Equations for the second moments are

$$\begin{aligned}
d\mathbb{E}[\theta\theta^T] =& \mathbb{E}[d\theta\theta^T + \theta d\theta^T] = (\mathbb{E}[v\theta^T] + \mathbb{E}[\theta v^T])dt, \\
d\mathbb{E}[\theta v^T] =& \mathbb{E}[d\theta v^T + \theta dv^T] = \mathbb{E}[vv^T]dt - \mu \mathbb{E}[\theta v^T]dt - \alpha \mathbb{E}[\theta\theta^T]Adt, \\
d\mathbb{E}[v\theta^T] =& \mathbb{E}[dv\theta^T + vd\theta^T] = \mathbb{E}[vv^T]dt - \mu \mathbb{E}[v\theta^T]dt - \alpha A \mathbb{E}[\theta\theta^T]dt, \\
d\mathbb{E}[vv^T] =& \mathbb{E}[dvv^T + vdv^T] + \mathbb{E}[dvdv^T] \\
=& -2\mu \mathbb{E}[vv^T]dt - \alpha A \mathbb{E}[\theta v^T]dt - \alpha \mathbb{E}[v\theta^T]Adt + \frac{\alpha^2}{b} CC^T dt.
\end{aligned} \tag{44}$$

We set the left-hand sides of all equations to zero to find a stationary solution. The first equation implies that,

$$\mathbb{E}[v\theta^T] + \mathbb{E}[\theta v^T] = 0. \tag{45}$$

We can add the second equation and the third equation to get,

$$0 = d\mathbb{E}[v\theta^T + \theta v^T] = 2\mathbb{E}[vv^T]dt - \alpha A \mathbb{E}[\theta\theta^T]dt - \alpha \mathbb{E}[\theta\theta^T]Adt. \tag{46}$$

Combining this with the fourth equation,

$$\begin{aligned}
\mathbb{E}[vv^T] &= \frac{\alpha}{2} \mathbb{E}[\theta\theta^T]A + \frac{\alpha}{2} A \mathbb{E}[\theta\theta^T], \\
\mu \mathbb{E}[vv^T] &= \frac{\alpha^2}{2b} CC^T - \frac{1}{2} \alpha (A \mathbb{E}[\theta v^T] + \mathbb{E}[v\theta^T]A).
\end{aligned} \tag{47}$$

By some algebraic transformations, we could show $-\frac{1}{2}\alpha(A\mathbb{E}[\theta v^T] + \mathbb{E}[v\theta^T]A) = 0$, which yields,

$$\Sigma_\theta A + A\Sigma_\theta = \frac{\alpha}{\mu b} CC^T. \qquad \square$$

ACKNOWLEDGMENTS

The authors would like to thank the anonymous reviewers for their valuable comments and helpful suggestions.

REFERENCES

[1] Martín Abadi, Paul Barham, Jianmin Chen, Zhifeng Chen, Andy Davis, Jeffrey Dean, Matthieu Devin, Sanjay Ghemawat, Geoffrey Irving, Michael Isard, Manjunath Kudlur, Josh Levenberg, Rajat Monga, Sherry Moore, Derek G. Murray, Benoit Steiner, Paul Tucker, Vijay Vasudevan, Pete Warden, Martin Wicke, Yuan Yu, and Xiaoqiang Zheng. 2016. TensorFlow: A system for large-scale machine learning. In *Proceedings of the 12th USENIX Symposium on Operating Systems Design and Implementation*. USENIX Association, Savannah, GA, 265–283. Retrieved from https://www.usenix.org/conference/osdi16/technical-sessions/presentation/abadi.

[2] Takuya Akiba, Shuji Suzuki, and Keisuke Fukuda. 2017. Extremely large minibatch SGD: Training ResNet-50 on ImageNet in 15 minutes. arXiv:1711.04325. Retrieved from https://arxiv.org/abs/1711.04325.

[3] Jing An, J. Lu, and Lexing Ying. 2018. Stochastic modified equations for the asynchronous stochastic gradient descent. arXiv: 1805.08244. Retrieved from https://arxiv.org/abs/1805.08244.

[4] W. An, H. Wang, Q. Sun, J. Xu, Q. Dai, and L. Zhang. 2018. A PID controller approach for stochastic optimization of deep networks. In *Proceedings of the 2018 IEEE/CVF Conference on Computer Vision and Pattern Recognition*. 8522–8531.

[5] By Mahmoud Assran, Arda Aytekin, Hamid Reza Feyzmahdavian, Mikael Johansson, and Michael G. Rabbat. 2020. Advances in asynchronous parallel and distributed optimization. *Proceedings of the IEEE* 108, 11 (2020), 2013–2031. DOI: https://doi.org/10.1109/JPROC.2020.3026619

[6] Mahmoud Assran, Nicolas Loizou, Nicolas Ballas, and Michael G. Rabbat. 2019. Stochastic gradient push for distributed deep learning. In *Proceedings of the International Conference on Machine Learning*.

[7] Necdet Aybat, Alireza Fallah, Mert Gürbüzbalaban, and Asuman Ozdaglar. 2020. Robust accelerated gradient methods for smooth strongly convex functions. *SIAM Journal on Optimization* 30, 1 (2020), 717–751. DOI: https://doi.org/10.1137/19M1244925

[8] Yoshua Bengio. 2012. Practical recommendations for gradient-based training of deep architectures. In *Proceedings of the Neural Networks: Tricks of the Trade*.

[9] James Bergstra and Yoshua Bengio. 2012. Random search for hyper-parameter optimization. *Journal of Machine Learning Research* 13, 2 (2012), 281–305.

[10] L. Bottou, Frank E. Curtis, and J. Nocedal. 2018. Optimization methods for large-scale machine learning. arXiv: 1606.04838. Retrieved from https://arxiv.org/abs/1606.04838.

[11] B. Can, Mert Gürbüzbalaban, and Lingjiong Zhu. 2019. Accelerated linear convergence of stochastic momentum methods in wasserstein distances. In *Proceedings of the International Conference on Machine Learning*.

[12] Trishul Chilimbi, Yutaka Suzue, Johnson Apacible, and Karthik Kalyanaraman. 2014. Project adam: Building an efficient and scalable deep learning training system. In *Proceedings of the 11th USENIX Symposium on Operating Systems Design and Implementation*. USENIX Association, Broomfield, CO, 571–582. Retrieved from https://www.usenix.org/conference/osdi14/technical-sessions/presentation/chilimbi.

[13] Jeffrey Dean, Greg S. Corrado, Rajat Monga, Kai Chen, Matthieu Devin, Quoc V. Le, Mark Z. Mao, Marc'Aurelio Ranzato, Andrew Senior, Paul Tucker, Ke Yang, and Andrew Y. Ng. 2012. Large scale distributed deep networks. In *Proceedings of the 25th International Conference on Neural Information Processing Systems*. Curran Associates Inc., Red Hook, NY, 1223–1231.

[14] Aaron Defazio. 2019. On the curved geometry of accelerated optimization. In *Proceedings of the Advances in Neural Information Processing Systems*. Curran Associates, Inc., 1766–1775. Retrieved from http://papers.nips.cc/paper/8453-on-the-curved-geometry-of-accelerated-optimization.pdf.

[15] Jia Deng, Wei Dong, Richard Socher, Li-Jia Li, Kai Li, and Li Fei-Fei. 2009. ImageNet: A large-scale hierarchical image database. In *Proceedings of the 2009 IEEE Conference on Computer Vision and Pattern Recognition*. 248–255. DOI: https://doi.org/10.1109/CVPR.2009.5206848

[16] Jacob Devlin, Ming-Wei Chang, Kenton Lee, and Kristina Toutanova. 2018. BERT: Pre-training of deep bidirectional transformers for language understanding. arXiv:1810.04805. Retrieved from http://arxiv.org/abs/1810.04805.

[17] Simon S. Du and Wei Hu. 2019. Width provably matters in optimization for deep linear neural networks. arXiv:1901.08572. Retrieved from http://arxiv.org/abs/1901.08572.

[18] R. Ge, Sham M. Kakade, R. Kidambi, and Praneeth Netrapalli. 2019. The step decay schedule: A near optimal, geometrically decaying learning rate procedure. In *Proceedings of the Advances in Neural Information Processing Systems*.

[19] Niv Giladi, Mor Shpigel Nacson, Elad Hoffer, and Daniel Soudry. 2020. At stability's edge: How to adjust hyperparameters to preserve minima selection in asynchronous training of neural networks?. In *Proceedings of the International Conference on Learning Representations*. Retrieved from https://openreview.net/forum?id=Bkeb7lHtvH.

[20] Igor Gitman, Hunter Lang, Pengchuan Zhang, and Lin Xiao. 2019. Understanding the role of momentum in stochastic gradient methods. In *Proceedings of the Advances in Neural Information Processing Systems*. Curran Associates, Inc., 9633–9643. Retrieved from http://papers.nips.cc/paper/9158-understanding-the-role-of-momentum-in-stochastic-gradient-methods.pdf.

[21] Priya Goyal, Piotr Dollár, Ross B. Girshick, Pieter Noordhuis, Lukasz Wesolowski, Aapo Kyrola, Andrew Tulloch, Yangqing Jia, and Kaiming He. 2017. Accurate, large minibatch SGD: Training ImageNet in 1 hour. arXiv:1706.02677. Retrieved from http://arxiv.org/abs/1706.02677.

[22] Benjamin Guedj. 2019. A primer on PAC-Bayesian learning. arXiv:1901.05353. Retrieved from https://arxiv.org/abs/1901.05353.

[23] Ido Hakimi, Saar Barkai, Moshe Gabel, and A. Schuster. 2019. Taming momentum in a distributed asynchronous environment. arXiv:1907.11612. Retrieved from https://arxiv.org/abs/1907.11612.

[24] Fengxiang He, Tongliang Liu, and Dacheng Tao. 2019. Control batch size and learning rate to generalize well: Theoretical and empirical evidence. In *Proceedings of the Advances in Neural Information Processing Systems*. Curran Associates, Inc., 1143–1152. Retrieved from http://papers.nips.cc/paper/8398-control-batch-size-and-learning-rate-to-generalize-well-theoretical-and-empirical-evidence.pdf.

[25] K. He, X. Zhang, S. Ren, and J. Sun. 2016. Deep residual learning for image recognition. In *Proceedings of the 2016 IEEE Conference on Computer Vision and Pattern Recognition*. 770–778.

[26] Kaiming He, X. Zhang, Shaoqing Ren, and Jian Sun. 2016. Deep residual learning for image recognition. In *Proceedings of the 2016 IEEE Conference on Computer Vision and Pattern Recognition*. 770–778.

[27] Kaiming He, X. Zhang, Shaoqing Ren, and Jian Sun. 2016. Identity mappings in deep residual networks. arXiv:1603.05027. Retrieved from https://arxiv.org/abs/1603.05027.

[28] Elad Hoffer, Itay Hubara, and Daniel Soudry. 2017. Train longer, generalize better: Closing the generalization gap in large batch training of neural networks. In *Proceedings of the 31st International Conference on Neural Information Processing Systems*. Curran Associates Inc., Red Hook, NY, 1729–1739.

[29] Jie Hu, Li Shen, and Gang Sun. 2018. Squeeze-and-excitation networks. In *Proceedings of the 2018 IEEE/CVF Conference on Computer Vision and Pattern Recognition*. 7132–7141. DOI: https://doi.org/10.1109/CVPR.2018.00745

[30] Wenqing Hu, Chris Junchi Li, Lei Li, and Jian-Guo Liu. 2018. On the Diffusion Approximation of Nonconvex Stochastic Gradient Descent. arXiv:1705.07562. Retrieved from https://arxiv.org/abs/1701.00133.

[31] Mengdi Huai, Jianhui Sun, Renqin Cai, Liuyi Yao, and Aidong Zhang. 2020. Malicious attacks against deep reinforcement learning interpretations. In *Proceedings of the 26th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining*. Association for Computing Machinery, New York, NY, 472–482. DOI: https://doi.org/10.1145/3394486.3403089

[32] Gao Huang, Yixuan Li, Geoff Pleiss, Zhuang Liu, John E. Hopcroft, and Kilian Q. Weinberger. 2017. Snapshot ensembles: Train 1, get M for free. arXiv:1704.00109. Retrieved from http://arxiv.org/abs/1704.00109.

[33] G. Huang, Z. Liu, L. Van Der Maaten, and K. Q. Weinberger. 2017. Densely connected convolutional networks. In *Proceedings of the 2017 IEEE Conference on Computer Vision and Pattern Recognition*. 2261–2269.

[34] Stanisław Jastrzębski, Zac Kenton, Devansh Arpit, Nicolas Ballas, Asja Fischer, Amos Storkey, and Yoshua Bengio. 2018. Three Factors Influencing Minima in SGD. Retrieved from https://openreview.net/forum?id=rJma2bZCW.

[35] Kishlay Jha, Guangxu Xun, Yaqing Wang, and Aidong Zhang. 2019. Hypothesis generation from text based on co-evolution of biomedical concepts. In *Proceedings of the 25th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining*. Ankur Teredesai, Vipin Kumar, Ying Li, Rómer Rosales, Evimaria Terzi, and George Karypis (Eds.). ACM, 843–851. DOI: https://doi.org/10.1145/3292500.3330977

[36] Nitish Shirish Keskar, Dheevatsa Mudigere, Jorge Nocedal, Mikhail Smelyanskiy, and Ping Tak Peter Tang. 2016. On large-batch training for deep learning: Generalization gap and sharp minima. arXiv:1609.04836. Retrieved from http://arxiv.org/abs/1609.04836.

[37] Nitish Shirish Keskar and Richard Socher. 2017. Improving generalization performance by switching from adam to SGD. arXiv:1712.07628. Retrieved from http://arxiv.org/abs/1712.07628.

[38] Rahul Kidambi, Praneeth Netrapalli, Prateek Jain, and Sham M. Kakade. 2018. On the insufficiency of existing momentum schemes for stochastic optimization. arXiv:1803.05591. Retrieved from http://arxiv.org/abs/1803.05591.

[39] Diederik P. Kingma and Jimmy Ba. 2015. Adam: A method for stochastic optimization. arXiv:1412.6980. Retrieved from https://arxiv.org/abs/1412.6980.

[40] Alex Krizhevsky, Ilya Sutskever, and Geoffrey E. Hinton. 2012. ImageNet classification with deep convolutional neural networks. *Advances in Neural Information Processing Systems* 25 (2012), 1097–1105. Retrieved from http://papers.nips.cc/paper/4824-imagenet-classification-with-deep-convolutional-neural-networks.pdf.

[41] A. Kulunchakov and J. Mairal. 2019. Estimate sequences for variance-reduced stochastic composite optimization. In *Proceedings of the International Conference on Machine Learning*.

[42] M. Laborde and Adam M. Oberman. 2020. A lyapunov analysis for accelerated gradient methods: From deterministic to stochastic case. In *Proceedings of the International Conference on Artificial Intelligence and Statistics*.

[43] Namhoon Lee, Thalaiyasingam Ajanthan, Philip Torr, and Martin Jaggi. 2021. Understanding the effects of data parallelism and sparsity on neural network training. In *Proceedings of the International Conference on Learning Representations*. Retrieved from https://openreview.net/forum?id=rsogjAnYs4z.

[44] Laurent Lessard, Benjamin Recht, and Andrew Packard. 2014. Analysis and design of optimization algorithms via integral quadratic constraints. *SIAM Journal on Optimization* 26, 1 (2014), 57–95. DOI: https://doi.org/10.1137/15M1009597

[45] Hao Li, Zheng Xu, Gavin Taylor, Christoph Studer, and Tom Goldstein. 2018. Visualizing the loss landscape of neural nets. In *Proceedings of the 32nd International Conference on Neural Information Processing Systems*. Curran Associates Inc., Red Hook, NY, 6391–6401.

[46] Qianxiao Li, Cheng Tai, and Weinan E. 2017. Stochastic modified equations and adaptive stochastic gradient algorithms. In *Proceedings of the International Conference on Machine Learning*. PMLR, International Convention Centre, Sydney, Australia, 2101–2110. Retrieved from http://proceedings.mlr.press/v70/li17f.html.

[47] Xiangru Lian, Ce Zhang, Huan Zhang, Cho-Jui Hsieh, Wei Zhang, and Ji Liu. 2017. Can decentralized algorithms outperform centralized algorithms? A case study for decentralized parallel stochastic gradient descent. In *Proceedings of the 31st International Conference on Neural Information Processing Systems*. Curran Associates Inc., Red Hook, NY, 5336–5346.

[48] Xiangru Lian, Wei Zhang, Ce Zhang, and Ji Liu. 2018. Asynchronous decentralized parallel stochastic gradient descent. In *Proceedings of the International Conference on Machine Learning*. 3049–3058. Retrieved from http://proceedings.mlr.press/v80/lian18a.html.

[49] Chaoyue Liu, Libin Zhu, and Mikhail Belkin. 2020. Toward a Theory of Optimization for Over-parameterized Systems of Non-linear Equations: The Lessons of Deep Learning. arXiv:2003.00307. Retrieved from https://arxiv.org/abs/2003.00307.

[50] Yanli Liu, Yuan Gao, and Wotao Yin. 2020. An Improved Analysis of Stochastic Gradient Descent with Momentum. arXiv:2007.07989. Retrieved from https://arxiv.org/abs/2007.07989.

[51] Ben London. 2017. A PAC-Bayesian analysis of randomized learning with application to stochastic gradient descent. In *Proceedings of the Advances in Neural Information Processing Systems*.

[52] Ilya Loshchilov and Frank Hutter. 2016. SGDR: Stochastic gradient descent with restarts. arXiv:1608.03983. Retrieved from http://arxiv.org/abs/1608.03983.

[53] Jerry Ma and Denis Yarats. 2019. Quasi-hyperbolic momentum and Adam for deep learning. In *Proceedings of the International Conference on Learning Representations*. Retrieved from https://openreview.net/forum?id=S1fUpoR5FQ.

[54] Tianle Ma and Aidong Zhang. 2019. AffinityNet: Semi-supervised few-shot learning for disease type prediction. In *Proceedings of the 33rd AAAI Conference on Artificial Intelligence, 32st Innovative Applications of Artificial Intelligence Conference, 9th AAAI Symposium on Educational Advances in Artificial Intelligence*. AAAI Press, 1069–1076. DOI: https://doi.org/10.1609/aaai.v33i01.33011069

[55] Aleksander Madry, Aleksandar Makelov, Ludwig Schmidt, Dimitris Tsipras, and Adrian Vladu. 2018. Towards deep learning models resistant to adversarial attacks. In *Proceedings of the International Conference on Learning Representations*. Retrieved from https://openreview.net/forum?id=rJzIBfZAb.

[56] Subhransu Maji, Esa Rahtu, Juho Kannala, Matthew B. Blaschko, and Andrea Vedaldi. 2013. Fine-grained visual classification of aircraft. arXiv:1306.5151. Retrieved from http://arxiv.org/abs/1306.5151.

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

[58] David A. McAllester. 1998. Some PAC-Bayesian theorems. In *Proceedings of the 11th Annual Conference on Computational Learning Theory*. Association for Computing Machinery, New York, NY, 230–234. DOI: https://doi.org/10.1145/279943.279989

[59] Ioannis Mitliagkas, Ce Zhang, Stefan Hadjis, and C. Ré. 2016. Asynchrony begets momentum, with an application to deep learning. *2016 54th Annual Allerton Conference on Communication, Control, and Computing (Allerton)*. 997–1004.

[60] Y. Nesterov. 1983. A method for solving the convex programming problem with convergence rate O($1/k^2$). *Dokl. Akad. Nauk Sssr* 269 (1983), 543–547.

[61] Feng Niu, Benjamin Recht, Christopher Re, and Stephen J. Wright. 2011. HOGWILD! A lock-free approach to parallelizing stochastic gradient descent. In *Proceedings of the 24th International Conference on Neural Information Processing Systems*. Curran Associates Inc., Red Hook, NY, 693–701.

[62] Adam Paszke, S. Gross, Francisco Massa, A. Lerer, J. Bradbury, G. Chanan, T. Killeen, Z. Lin, N. Gimelshein, L. Antiga, Alban Desmaison, Andreas Köpf, E. Yang, Zach DeVito, Martin Raison, Alykhan Tejani, Sasank Chilamkurthy, B. Steiner, Lu Fang, Junjie Bai, and Soumith Chintala. 2019. PyTorch: An imperative style, high-performance deep learning library. In *Proceedings of the Advances in Neural Information Processing Systems*.

[63] B. T. Polyak. 1964. Some methods of speeding up the convergence of iteration methods. *U. S. S. R. Computational Mathematics and Mathematical Physics* 4, 5 (1964), 1–17. DOI: https://doi.org/10.1016/0041-5553(64)90137-5

[64] Sebastian Ruder. 2016. An overview of gradient descent optimization algorithms. arXiv:1609.04747. Retrieved from https://arxiv.org/abs/1609.04747.

[65] Christopher J. Shallue, Jaehoon Lee, Joseph M. Antognini, Jascha Sohl-Dickstein, Roy Frostig, and George E. Dahl. 2018. Measuring the effects of data parallelism on neural network training. arXiv:1811.03600. Retrieved from http://arxiv.org/abs/1811.03600.

[66] John Shawe-Taylor and Robert C. Williamson. 1997. A PAC analysis of a Bayesian estimator. In *Proceedings of the 10th Annual Conference on Computational Learning Theory*. Association for Computing Machinery, New York, NY, 2–9. DOI: https://doi.org/10.1145/267460.267466

[67] Karen Simonyan and Andrew Zisserman. 2015. Very deep convolutional networks for large-scale image recognition. In *Proceedings of the 3rd International Conference on Learning Representations*. arXiv:1409.1556. Retrieved from http://arxiv.org/abs/1409.1556.

[68] L. N. Smith. 2017. Cyclical learning rates for training neural networks. In *Proceedings of the 2017 IEEE Winter Conference on Applications of Computer Vision*. 464–472. DOI: https://doi.org/10.1109/WACV.2017.58

[69] Leslie N. Smith and Nicholay Topin. 2017. Super-convergence: Very fast training of residual networks using large learning rates. arXiv:1708.07120. Retrievd from http://arxiv.org/abs/1708.07120.

[70] Sam Smith and Quoc V. Le. 2018. A Bayesian perspective on generalization and stochastic gradient descent. Retrieved from https://openreview.net/pdf?id=BJij4yg0Z.

[71] Samuel L. Smith, Pieter-Jan Kindermans, and Quoc V. Le. 2018. Don't decay the learning rate, increase the batch size. In *Proceedings of the International Conference on Learning Representations*. Retrieved from https://openreview.net/forum?id=B1Yy1BxCZ.

[72] Jasper Snoek, Hugo Larochelle, and Ryan P. Adams. 2012. Practical Bayesian optimization of machine learning algorithms. In *Proceedings of the 25th International Conference on Neural Information Processing Systems*. Curran Associates Inc., Red Hook, NY, 2951–2959.

[73] Jianhui Sun, Mengdi Huai, Kishlay Jha, and Aidong Zhang. 2022. Demystify hyperparameters for stochastic optimization with transferable representations. In *Proceedings of the 28th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining*. Association for Computing Machinery, New York, NY. DOI: https://doi.org/10.1145/3534678.3539298

[74] Jianhui Sun, Ying Yang, Guangxu Xun, and Aidong Zhang. 2021. A stagewise hyperparameter scheduler to improve generalization. In *Proceedings of the 27th ACM SIGKDD Conference on Knowledge Discovery & Data Mining*. Association for Computing Machinery, New York, NY, 1530–1540. DOI: https://doi.org/10.1145/3447548.3467287

[75] Qiuling Suo, Liuyi Yao, Guangxu Xun, Jianhui Sun, and Aidong Zhang. 2019. Recurrent imputation for multivariate time series with missing values. In *Proceedings of the 2019 IEEE International Conference on Healthcare Informatics*. IEEE, 1–3. DOI: https://doi.org/10.1109/ICHI.2019.8904638

[76] Qiuling Suo, Weida Zhong, Guangxu Xun, Jianhui Sun, Changyou Chen, and Aidong Zhang. 2020. GLIMA: Global and local time series imputation with multi-directional attention learning. In *Proceedings of the IEEE International Conference on Big Data*. IEEE, 798–807. DOI: https://doi.org/10.1109/BigData50022.2020.9378408

[77] Ilya Sutskever, James Martens, George Dahl, and Geoffrey Hinton. 2013. On the importance of initialization and momentum in deep learning. In *Proceedings of the 30th International Conference on International Conference on Machine Learning*. III–1139–III–1147.

[78] B. Van Scoy, R. A. Freeman, and K. M. Lynch. 2018. The fastest known globally convergent first-order method for minimizing strongly convex functions. *IEEE Control Systems Letters* 2, 1 (2018), 49–54.

[79] Sharan Vaswani, F. Bach, and M. Schmidt. 2019. Fast and faster convergence of SGD for over-parameterized models and an accelerated perceptron. arXiv:1810.07288. Retrieved from https://arxiv.org/abs/1810.07288.

[80] Ashia C. Wilson, Rebecca Roelofs, Mitchell Stern, Nati Srebro, and Benjamin Recht. 2017. The marginal value of adaptive gradient methods in machine learning. In *Proceedings of the Advances in Neural Information Processing Systems*. Curran Associates, Inc., 4148–4158. Retrieved from http://papers.nips.cc/paper/7003-the-marginal-value-of-adaptive-gradient-methods-in-machine-learning.pdf.

[81] Zeke Xie, Li Yuan, Zhanxing Zhu, and Masashi Sugiyama. 2021. Positive-negative momentum: Manipulating stochastic gradient noise to improve generalization. In *Proceedings of the 38th International Conference on Machine Learning*. Marina Meila and Tong Zhang (Eds.), PMLR, 11448–11458.

[82] Guangxu Xun, Kishlay Jha, Jianhui Sun, and Aidong Zhang. 2020. Correlation networks for extreme multi-label text classification. In *Proceedings of the 26th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining*.

[83] Guangxu Xun, Kishlay Jha, Ye Yuan, Yaqing Wang, and Aidong Zhang. 2019. MeSHProbeNet: A self-attentive probe net for MeSH indexing. *Bioinformatics* 35, 19 (2019), 3794–3802. DOI: https://doi.org/10.1093/bioinformatics/btz142

[84] Yan Yan, Tianbao Yang, Zhe Li, Qihang Lin, and Yi Yang. 2018. A unified analysis of stochastic momentum methods for deep learning. In *Proceedings of the IJCAI*. 2955–2961. DOI: https://doi.org/10.24963/ijcai.2018/410

[85] Jian Zhang and Ioannis Mitliagkas. 2018. YellowFin and the Art of Momentum Tuning. arXiv:1706.03471. Retrieved from https://arxiv.org/abs/1706.03471.

[86] Wei Zhang, Suyog Gupta, Xiangru Lian, and Ji Liu. 2016. Staleness-aware async-SGD for distributed deep learning *(IJCAI'16)*. AAAI Press, 2350–2356.

[87] Z. Zhu and L. Orecchia. 2017. Linear coupling: An ultimate unification of gradient and mirror descent. In *Proceedings of the ITCS*.

Received 6 September 2021; revised 23 March 2022; accepted 13 May 2022