A Stagewise Hyperparameter Scheduler to Improve Generalization

Jianhui Sun[1], Ying Yang[2], Guangxu Xun[1], and Aidong Zhang[1]
[1]University of Virginia, Charlottesville, VA, USA, {js9gu, gx5bt, aidong}@virginia.edu
[2]University of Michigan, Ann Arbor, MI, USA, yingyan@umich.edu

ABSTRACT

Stochastic gradient descent (SGD) augmented with various momentum variants (e.g. heavy ball momentum (SHB) and Nesterov's accelerated gradient (NAG)) has been the default optimizer for many learning tasks. Tuning the optimizer's hyperparameters is arguably the most time-consuming part of model training. Many new momentum variants, despite their empirical advantage over classical SHB/NAG, introduce even more hyperparameters to tune. Automating the tedious and error-prone tuning is essential for AutoML. This paper focuses on how to efficiently tune a large class of *multistage* momentum variants to improve generalization. We use the general formulation of quasi-hyperbolic momentum (QHM) and extend "constant and drop", the widespread learning rate α scheduler where α is set large initially and then dropped every few epochs, to other hyperparameters (e.g. batch size b, momentum parameter β, instant discount factor v). Multistage QHM is a unified framework which covers a large family of momentum variants as its special cases (e.g. vanilla SGD/SHB/NAG). Existing works mainly focus on scheduling α's decay, while multistage QHM allows additional varying hyperparameters such as b, β, and v, and demonstrates better generalization ability than only tuning α. Our tuning strategies have rigorous justifications rather than a blind trial-and-error. We theoretically prove why our tuning strategies could improve generalization. We also show the convergence of multistage QHM for general nonconvex objective functions. Our strategies simplify the tuning process and beat competitive optimizers in test accuracy empirically.

CCS CONCEPTS

• **Theory of computation** → **Sample complexity and generalization bounds**; • **Mathematics of computing** → *Nonconvex optimization.*

KEYWORDS

Deep Learning Optimization; Hyperparameter Tuning; SGD Momentum; Multistage QHM; Generalization Bound

ACM Reference Format:
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 and Data Mining (KDD '21), August 14–18, 2021, Virtual Event, Singapore.* ACM, New York, NY, USA, 11 pages. https://doi.org/10.1145/3447548.3467287

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 (1) that could generalize well to unseen test instances.

Stochastic gradient descent (SGD), the backbone of many optimizers, minimizes the objective by iteratively moving the parameters along its gradient direction. Vanilla SGD tends to converge slowly especially when it gets closer to local minima. Therefore, to accelerate the convergence, pioneered by Polyak's Heavy Ball momentum (SHB) [39] and Nesterov's Accelerated Gradient (NAG) [37], a momentum term which incorporates past gradient estimates into the current update, is often augmented to SGD.

SGD+momentum is the default optimizer for most deep learning models as it achieves an impressive training time saving and test accuracy boost compared to competing optimizers [49, 52]. In recent years, a large number of new momentum variants, many of which modify the classical SHB and NAG to accommodate more complex task-dependent objective functions, have been proposed and achieved state-of-the-art performances across domains, e.g., Synthesized Nesterov Variants (SNV) [28], PID control (PID) [2], Triple Momentum [50], Accelerated Stochastic Gradient Method (AccSGD) [23], and Quasi-Hyperbolic Momentum (QHM) [33].

Configuration of hyperparameters has a huge impact on the quality of solutions found by optimizers for deep neural networks [4]. The most reliable hyperparameter tuning strategy is still a comprehensive grid search, and is probably the most time-consuming part of training. An essential step of AutoML is to simplify and automate tuning. However, the flux of newly proposed momentum variants even cast more challenges on tuning:

(1) New momentum variants, despite their empirical advantages, often have more complex updating rules than SGD/SHB/NAG and introduce more hyperparameters. In SGD, learning rate α is often the only hyperparameter to adjust, and SHB/NAG adds an additional momentum parameter β, which is still manageable for a manual search. However, it is commonplace for any new optimizer to require at least a triplet of hyperparameters to configure, e.g., QHM adds an instant discount factor v; while PID requires a set of (k_P, k_D) to be determined apart from the

learning rate. Batch size is also a factor of model performance that has to be weighed in. Considering these hyperparameters altogether, grid search at even a moderate scale is costly.

(2) Hyperparameters are typically not fixed throughout the entire learning process. Scheduling the decay/increase of hyperparameters appropriately is nearly as important as assigning the initial values to them [52]. Existing scheduler usually decays learning rate gradually while holding other hyperparameters constant (e.g., batch size and momentum parameter). It remains unclear whether varying hyperparameters other than learning rate could help different momentum variants in general. If optimizers do benefit from varying additional hyperparameters, it introduces tremendous complexity to search manually the exact scheduling schemes for all different hyperparameters.
(3) There is a lack of unified analysis that could cover different momentum variants with one single framework, which makes it difficult to transfer existing tuning strategies from SGD/SHB/NAG. There have been a few works that introduce different learning rate tuners [5, 9, 24, 46] or momentum parameter schedulers [12, 45, 49] to help practitioners tune SGD/SHB/NAG. However, a completely new round of tuning exploration is still necessary for any newly proposed momentum scheme as its connection to SGD is largely in the dark.
(4) Existing tuning strategies are mainly from a trial-and-error process, which includes attempting different configurations and accumulating experience from the feedback of empirical experiments. This procedure is not only tedious and error-prone, but also lacks valuable insights why certain strategy works and whether it works across different models or datasets.

To tackle the aforementioned challenges, this paper provides an efficient tuning pipeline to relieve practitioners of the labor-intensive tuning process and improve the optimizer's generalization ability at the same time. We select the multistage version of a powerful momentum scheme, QHM (Quasi-Hyperbolic Momentum), which is parameterized by a triplet of hyperparameters (learning rate α, momentum parameter β, instant discount factor ν [33]) to analyze, as it includes many popular momentum variants as its special cases. Therefore, the proposed method is generally applicable to a large family of multistage momentum variants.

We extend the idea that practitioners have long used to schedule learning rate in SGD, "constant and drop" [1], to all hyperparameters involved in QHM ((α, β, ν) and batch size b), and propose multistage QHM in Section 3, where we adjust all involving hyperparameters stagewise. Existing works are unclear about how to adjust (α, β, ν, b) appropriately or whether adjustment of any of them could contribute to better generalization.

In Section 4, we first prove Theorem 1 to show how varying hyperparameters affect the generalization bound, and then based on the theoretical findings, propose the exact paradigm for our stagewise scheduler. In Section 5, we provide the convergence guarantee for multistage QHM for nonconvex objectives. In Section 6, extensive experiments are presented and show the empirical advantage of our proposed multistage QHM. We discuss relevant works in Section 7.

Our contribution could be summarized as follows:

(1) We propose multistage QHM, which is a unified and flexible framework that allows adjusting every hyperparameter across stages. We provide the exact approach to schedule the decay/increase to achieve better generalization. Our method effectively restricts the search space of hyperparameters, and helps automate the tuning process.
(2) We provide theoretical evidence whether and why our proposed multistage QHM improves generalizability. Our theoretical findings could easily transfer to many newly proposed momentum schemes. 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.

2 BACKGROUND

In this section, we introduce quasi-hyperbolic momentum, and generalization error, that are pertinent to this work.

2.1 Quasi-Hyperbolic Momentum

Let $\{z_i = (x_i, y_i)\}_{i=1}^N$ represent the training set. 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 j-th 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 [11, 35]).

We start from the following updating rule of Stochastic Gradient Descent (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 k-th step, respectively. (mini-batch) SGD [2] uses $\hat{g}_k \triangleq \hat{g}(\theta_k)$ as d_k.

We focus on Quasi-Hyperbolic Momentum methods (QHM) [33], 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 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:

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

[1] "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 [38] or Tensorflow [1]).

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

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 instant discount factor.

[33] showed that QHM could recover many other popular momentum schemes, e.g., PID control (PID), Synthesized Nesterov Variants (SNV), Accelerated Stochastic Gradient Method (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 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 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.

$$\begin{aligned} \mathcal{R}(Q) &\triangleq \mathbb{E}_{\theta\sim Q}\mathbb{E}_{(x,y)\sim\mathcal{D}} l(f_\theta(x), y), \\ \hat{\mathcal{R}}(Q) &\triangleq \mathbb{E}_{\theta\sim Q}\frac{1}{N}\sum_{j=1}^{N} l(f_\theta(x_j), y_j), \end{aligned} \tag{6}$$

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:

$$\mathcal{E} \triangleq |\mathcal{R}(Q) - \hat{\mathcal{R}}(Q)|. \tag{7}$$

3 A STAGEWISE HYPERPARAMETER SCHEDULER

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 [25, 45, 49]. 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 number of iterations/epochs), the learning process is divided into M stages, and at i-th stage SGD(α_i) is applied for T_i iterations/epochs.

The logic behind "constant and drop" is: large but constant step size allows for faster convergence than diminishing step size in 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 [35]. 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 $\theta_0$, 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
          $\{b_i\}_{i=1}^M$;
1  for $i \in \{0, 1, ..., M-1\}$ do
2      update $\theta_{i,0} \leftarrow \theta_i$;
3      for $k \in \{0, ..., T_i - 1\}$ do
4          Sample a mini-batch $\zeta_k = \{(x_j, y_j)\}_{j=1}^{b_i}$ from training
             data uniformly;
5          Compute gradient of objective function based on $\zeta_k$,
             i.e., $\hat{g}_k = \frac{1}{b_i}\sum_{j\in\zeta_k}\nabla\mathcal{R}_j(\theta_{i,k})$;
6          Compute $d_k = (1-\beta_i)\hat{g}_k + \beta_i d_{k-1}$, with $d_0 = 0$;
7          Update $\theta_{i,k+1} \leftarrow \theta_{i,k} - \alpha_i[(1-\nu_i)\hat{g}_k + \nu_i d_k]$;
8      end
9      update $\theta_{i+1} \leftarrow \theta_{i,T_i}$;
10 end
11 return $\theta_M$
```

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 [30, 52]. Algorithm 1 is much more flexible. Based on our results, fixing β_i and b_i is suboptimal. In Section 4 we will provide theoretical justifications. In Section 6, we show our (β_i, b_i) scheduler achieves non-trivial advantages over existing schedulers.

4 HOW TO TUNE MULTISTAGE QHM?

It is well known that optimization hyperparameters have a substantial impact on the quality of 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 subsection 4.1, we will theoretically connect the generalization error of QHM with these hyperparameters. Subsection 4.2 explains how to tune to decrease the generalization error in practice.

4.1 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 1. *Assume the risk function is locally quadratic, and gradient noise is Gaussian* [3]. *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

$$\begin{aligned} C_1 &= \frac{1}{2\lambda_0}\|\theta_0\|_2^2 + \frac{1}{2}d\log\lambda_0 - \frac{1}{2}d \\ &-\frac{1}{2}\log\det(\Sigma A^{-1}) + \log\frac{1}{\epsilon} + \log N + 2, \\ C_2(\alpha,\beta,\nu,b) &= -\frac{1}{2}d\log\frac{\alpha}{2b} \\ &+\frac{\alpha tr(\Sigma A^{-1})}{4\lambda_0 b} + \frac{\alpha^2 tr(\Sigma)}{4\lambda_0 b}\frac{(4\nu^2-2\nu-1)\beta^2-2\nu\beta+1}{2(1-\beta^2)}. \end{aligned} \tag{8}$$

Proof. We defer the proof to Appendix due to space limit. Note det and tr indicate the determinant and trace of a matrix, respectively. □

In Theorem 1, 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 paper. Further notice only the last term in C_2 includes (β,ν). Let us denote the last term as H for ease of notation:

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

4.2 Hyperparameters Scheduling Scheme

We discuss how exactly Theorem 1 guides us to tune multistage QHM.

4.2.1 *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 4.1. *The impact of $\frac{b}{\alpha}$ could be summarized as follows:*

(1) If the model size d (i.e., number of parameters) is larger [4] *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. [5]

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 [40] and adaptive gradient methods (e.g., Adam or RMSProp) [24]. 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 4.1 shows to be suboptimal.

[12, 45] 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 1 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 4.1 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 4.1. Undershrinkage of batch size ($\frac{b_i}{\alpha_i}$ increases with i) negatively impacts generalization of multistage QHM in theory, and empirical experiments in Section 6 support our finding.

Corollary 4.1 also gives us insights 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 time budget.

4.2.2 *Increase momentum parameter, while keeping instant discount factor ν_i constant.*

In the original paper of QHM [33], 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 4.2. $H(\beta \to 1, \nu = 0.5) \ll H(\beta = 0.999, \nu = 0.7)$, *and consequently,* $(\beta \to 1, \nu = 0.5)$ *decreases the generalization error.*

Proof. The proof is obvious by simply substituting (β,ν) into H. [6] □

[3]The assumption is standard when approximating a stochastic algorithm with a continuous-time stochastic process (see [11, 14, 35]) and is justified when the iterates are confined to a restricted region around the minimizer.

[4]Modern large-scale deep learning models are often extremely overparameterized, with d ranging from tens to hundreds of millions (e.g., VGG [41] and ResNet [15]), and this condition is easily fulfilled.

[5][14] shows a similar bound as Theorem 1, 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 6 that such difference is empirically non-trivial.

[6]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 which is close to 1 is commonplace in practice (e.g., PyTorch SGD+momentum sets $\beta = 0.9$ by default). For SHB and NAG, [45, 49] 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 [11]). 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, generalization bound in Theorem 1 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.

4.2.3 *Increase length of training time* T_i.

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

4.2.4 *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:

$$\alpha_i = r^{i-1}\alpha_1, \quad b_i = r_b^{i-1}b_1, \quad T_i = (\frac{1}{r})^{i-1}T_1, \quad \nu_i = \frac{1}{2},$$
$$\beta_i = 1 - \frac{1}{1 + (\frac{\beta_1}{1-\beta_1})(\frac{1}{r})^{i-1}}, \quad \beta_i^{2T_i} \leq \frac{1}{2}. \tag{9}$$

In practice, a popular choice of r is $\frac{1}{2}$. r_b could be set as $\frac{1}{2}$ for convenience (Section 6.1 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 (9), we only have to manually search α_1, and Corollary 4.1 further indicates, we only need to search α_1 monotonically. For example, if we have a grid $\alpha = (0.01, 0.02, 0.03, \cdots, 1.0)$ as usual, we only need to start from 0.01, increase α, and stop immediately when the next (or next few) grid value 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.

5 CONVERGENCE OF MULTISTAGE QHM

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

THEOREM 2. *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 (9). 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 i-th 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 $i \leq M$. Under mild regulatory conditions* [7]*, we would have:*

$$\begin{aligned}
\bar{\mathcal{G}} &\leq \varepsilon_d + \varepsilon_s, \\
\varepsilon_d &= \frac{2(\mathcal{R}(\theta_1) - \mathcal{R}^*)}{MW_2}, \\
\varepsilon_s &= \frac{1}{M}\sum_{i=1}^M \Big(1 + 24\frac{\beta_i^2\nu_i^2}{(1-\beta_i)^2}\frac{\beta_1}{\sqrt{\beta_M + \beta_M^2}} + \frac{6 + 2\beta_i^2\nu_i^2}{1-\beta_1}\Big)\alpha_i L\sigma^2.
\end{aligned} \tag{10}$$

REMARK 5.1 (NONCONVEX OJBECTIVE). *In Theorem 2, we only require the objective function to be smooth. Many existing works further require the objective function to be strongly convex [11]. As in neural network optimization, strongly convexity may not hold for highly over-parameterized neural networks (see [29]). Therefore, our settings are more general.*

REMARK 5.2 (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 larger number of stages M or 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 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 k-th 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 $\eta_{k+1} - \eta_k = -\alpha_k\hat{g}_k$ [30]. $\{\eta_k\}_{k\in\mathbb{N}}$ is devised as follows:

$$\eta_k = \begin{cases} \theta_k & k = 1 \\ \theta_k - \frac{\alpha_k\beta_k\nu_k}{1-\beta_k}d_{k-1} & k \geq 2 \end{cases} \tag{11}$$

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

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

$\{\eta_k\}_{k\in\mathbb{N}}$ 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:

$$a_{k,i} = \begin{cases} 1 - \beta_k\nu_k & i = k \\ \nu_k(1-\beta_i)\prod_{j=i+1}^k \beta_j & i < k \end{cases} \tag{12}$$

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 2:

[7]Theorem 2 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 $\frac{1-\beta_1}{\beta_1} \leq 12\frac{1-\beta_M}{\sqrt{\beta_M+\beta_M^2}}$, 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 1. *We have the following two inequalities:* $\mathbb{V}[y_k] \leq \left(6 - 4\beta_1 - 4\beta_k v_k + 2\beta_k^2 v_k^2\right)\sigma^2$, *and* $\mathbb{E}[\|g_k - \frac{1}{1-v_k \prod_{i=1}^k \beta_i}\Sigma_{i=1}^k a_{k,i} g_i\|^2] \leq \Sigma_{j=1}^{k-1} \frac{v_k \beta_k^{k-j} L^2}{1-v_k \prod_{i=1}^k \beta_i}\left(k - j + \frac{\beta_k}{1-\beta_k}\right)\mathbb{E}[\|\theta_{j+1} - \theta_j\|^2]$.

Please see Appendix for further steps. □

6 EXPERIMENTS

In this section, we present our empirical experiments to analyze the performance of our proposed multistage QHM. Our codes are publicly available [8].

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) [16], 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.

6.1 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 opposite scenario. We present our results in Figure 1 and Table 2.

Table 2 is in accordance with our claim in Section 4.2. 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 1. Therefore, we suggest overshrinkage only has marginal advantage and we could set $r_b = r$ for training with more stability.

Table 1: Experimental Settings of Table 2. All settings use the same QHM optimizer ($\beta_1 = 0.9, v = 0.5, M = 3$), but with different batch size schedulers.

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	1024	0.5	0.5	3
Batch Undershrinkage	256	0.1	0.5	3
Batch Overshrinkage	256	0.9	0.5	3

6.2 v Scheduler

We then study the effect of v in generalization ability. v does not appear in some optimizers (e.g. SGD/NAG/SHB). [33] proposed a $v = 0.7$ 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 4.1, we suggest a $v = 0.5$ in multistage QHM paradigm. We report our result when training ResNet-20 on CIFAR-10 for 50 epochs with QHM in Figure 2. All hyperparameters are exactly the same except for initial step size and v.

We could observe different v affect generalization ability at any of the given initial step sizes. It could be seen in Figure 2a that training curves are quite mixed together and no obvious training advantage could be detected from $v = 0.7$ to $v = 0.5$. However, $v = 0.5$ is better than $v = 0.7$ in the test accuracy by a large margin. The gradual increase in accuracy from smaller initial step size to larger initial step size again supports our monotone searching method.

6.3 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 two most popular momentum variants SHB and NAG also satisfy this trend. And interestingly, Adam [24] optimizer, which is not characterized by QHM formulation, also exhibits better generalizability with 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 [11, 52]. 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, 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 better test result, which shortens the tuning time.

There are several off-shelf learning rate auto-tuners, two most popular ones are cosine annealing scheduler [32] and one cycle scheduler [42, 43]. We adopt the default implementation of these two tuners [9] 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 3. 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 3a, 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 two most widespread adaptive gradient methods. We adopt the default implementation of these two optimizers [10] and sweep a large range of α_1 for these two optimizers as initial learning rate is the most influential factor for these two optimizers [52]. We pick the best performance RMSProp and Adam could get and compare them to our multistage QHM in Figure 4. All three optimizers achieve practically 0 training loss in

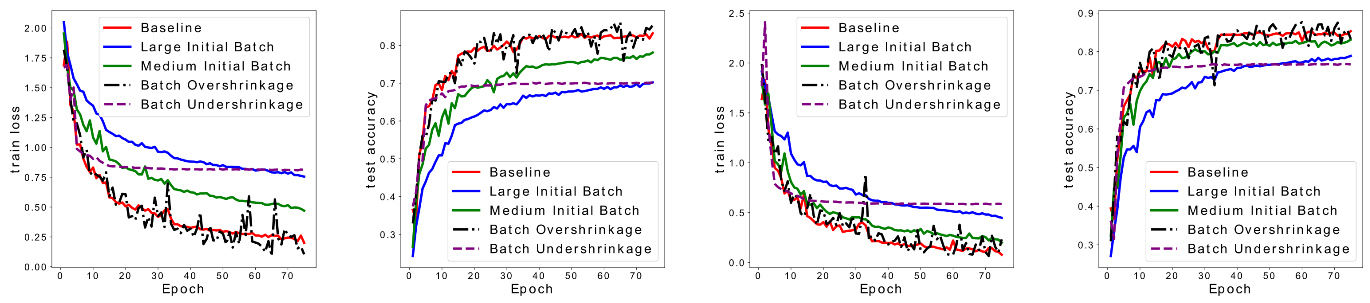
[8]https://github.com/jsycsjh/Multistage_QHM

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

[10]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.

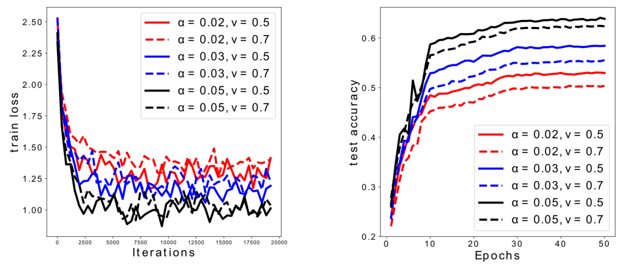
Table 2: The effect of batch size scheduler on deep models (PreAct-ResNet-110 on CIFAR-10). The pattern is consistent with our theoretical findings in Section 4: (i) batch size should decrease at least as fast as learning rate; (ii) initial batch size b_1 should be small; (iii) α_1 should be searched monotonically.

α_1	Multistage	Medium Initial Batch	Large Initial Batch	Batch Undershrinkage	Batch 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%



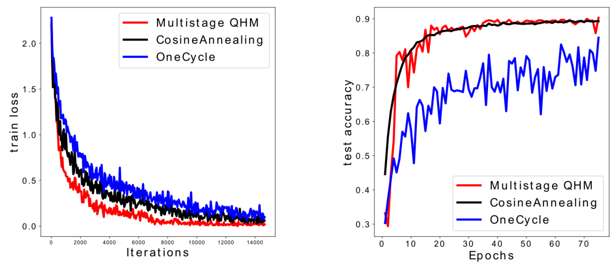
(a) Training loss (α_1 = 0.10) (b) Test accuracy (α_1 = 0.10) (c) Training loss (α_1 = 0.30) (d) Test accuracy (α_1 = 0.30)

Figure 1: Training curve and test accuracy of various batch size schedulers. Baseline is multistage QHM.



(a) Train loss (b) Test accuracy

Figure 2: Training curve and test accuracy of different v settings.

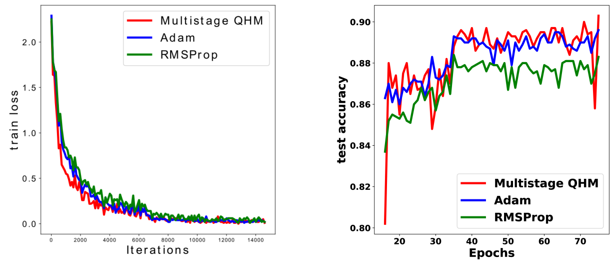


(a) Train loss (b) Test accuracy

Figure 3: Training curve and test accuracy of different learning rate schedulers (PreAct-ResNet-110 on CIFAR-10).

Table 3: The effect of initial learning rate α_1 on deep models (PreAct-ResNet-110 on CIFAR-10). The pattern indicates momentum schemes could search α_1 monotonically.

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%



(a) Train loss (b) Test accuracy

Figure 4: Training curve and test accuracy of different optimizers (PreAct-ResNet-110 on CIFAR-10).

Figure 4a. Multistage QHM is better than RMSProp in Figure 4b. Hand-tuned Adam matches Multistage QHM, with a slightly worse end-of-training test accuracy. This is in accordance with [22, 52],

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, achieves as good performance as complex counterparts in benchmark task.

7 RELATED WORK

7.1 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, human activity recognition and biomedical data mining [20, 34, 47, 48, 53–55]. 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, cyclic learning rate policy to name a few [5, 10, 18, 32, 42, 43, 46]. However, they only consider 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 [30, 49, 57]. 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 papers have also noted the impact of scheduling batch size [17, 44, 45], 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 includes not only SHB and NAG. Every hyperparameter is allowed to vary and is theoretically justified to improve generalization.

7.2 Generalization analysis

Our work aims to theoretically and empirically justify 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., [19, 21] connected the training dynamics, and the generalization to the ratio of batch size over step size.

Our generalization analysis relies on PAC-Bayesian inequalities [36], and we refer readers to a comprehensive review of PAC-Bayesian learning and references therein [13]. [14, 31] proved a PAC-Bayesian bound for vanilla SGD. Notably, [14] 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 4.1 that could not be derived only from first order bound.

7.3 Convergence analysis

The convergence of the vanilla SGD has been heavily studied and key results are highlighted in [6]. Despite the widespread use of the stochastic momentum method, there are limited definitive theoretical convergence guarantees. [8, 27, 58] studied momentum schemes but only for deterministic gradients. [56] 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 [3, 7, 26, 30, 51], largely only for the SHB or NAG, which are not directly generalizable to other momentum schemes.

8 CONCLUSIONS

To our best knowledge, this is the first general tuning guideline applicable for almost all popular momentum variants with a theoretical guarantee to boost generalization. Our multistage paradigm is natural to reason about and straightforward to implement. It effectively reduces the search space of hyperparameters, and 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.

ACKNOWLEDGMENTS

The authors would like to thank the anonymous reviewers for their valuable comments and helpful suggestions. This work is supported in part by the US National Science Foundation under grants IIS-2008208, 1938167, 1955151, and OAC-1934600. 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.

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 *12th USENIX Symposium on Operating Systems Design and Implementation (OSDI 16)*. USENIX Association, Savannah, GA, 265–283. https://www.usenix.org/conference/osdi16/technical-sessions/presentation/abadi

[2] 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 *2018 IEEE/CVF Conference on Computer Vision and Pattern Recognition*. 8522–8531.

[3] Necdet Aybat, Alireza Fallah, Mert Gürbüzbalaban, and Asuman Ozdaglar. 2020. Robust Accelerated Gradient Methods for Smooth Strongly Convex Functions. *SIAM Journal on Optimization* 30 (01 2020), 717–751. https://doi.org/10.1137/19M1244925

[4] Yoshua Bengio. 2012. Practical Recommendations for Gradient-Based Training of Deep Architectures. In *Neural Networks: Tricks of the Trade*.

[5] James Bergstra and Yoshua Bengio. 2012. Random Search for Hyper-Parameter Optimization. *J. Mach. Learn. Res.* 13, null (Feb. 2012), 281–305.

[6] L. Bottou, Frank E. Curtis, and J. Nocedal. 2018. Optimization Methods for Large-Scale Machine Learning. *ArXiv* abs/1606.04838 (2018).

[7] B. Can, Mert Gürbüzbalaban, and Lingjiong Zhu. 2019. Accelerated Linear Convergence of Stochastic Momentum Methods in Wasserstein Distances. In *ICML*.

[8] Aaron Defazio. 2019. On the Curved Geometry of Accelerated Optimization. In *Advances in Neural Information Processing Systems 32*. Curran Associates, Inc., 1766–1775. http://papers.nips.cc/paper/8453-on-the-curved-geometry-of-accelerated-optimization.pdf

[9] John Duchi, Elad Hazan, and Yoram Singer. 2011. Adaptive Subgradient Methods for Online Learning and Stochastic Optimization. *Journal of Machine Learning Research* 12, 61 (2011), 2121–2159. http://jmlr.org/papers/v12/duchi11a.html

[10] R. Ge, Sham M. Kakade, R. Kidambi, and Praneeth Netrapalli. 2019. The Step Decay Schedule: A Near Optimal, Geometrically Decaying Learning Rate Procedure. In *NeurIPS*.

[11] Igor Gitman, Hunter Lang, Pengchuan Zhang, and Lin Xiao. 2019. Understanding the Role of Momentum in Stochastic Gradient Methods. In *Advances in Neural Information Processing Systems 32*. Curran Associates, Inc., 9633–9643. http://papers.nips.cc/paper/9158-understanding-the-role-of-momentum-in-stochastic-gradient-methods.pdf

[12] 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. *CoRR* abs/1706.02677 (2017). arXiv:1706.02677 http://arxiv.org/abs/1706.02677

[13] Benjamin Guedj. 2019. A Primer on PAC-Bayesian Learning. *ArXiv* abs/1901.05353 (2019).

[14] Fengxiang He, Tongliang Liu, and Dacheng Tao. 2019. Control Batch Size and Learning Rate to Generalize Well: Theoretical and Empirical Evidence. In *Advances in Neural Information Processing Systems 32*. Curran Associates, Inc., 1143–1152. http://papers.nips.cc/paper/8398-control-batch-size-and-learning-rate-to-generalize-well-theoretical-and-empirical-evidence.pdf

[15] Kaiming He, X. Zhang, Shaoqing Ren, and Jian Sun. 2016. Deep Residual Learning for Image Recognition. *2016 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)* (2016), 770–778.

[16] Kaiming He, X. Zhang, Shaoqing Ren, and Jian Sun. 2016. Identity Mappings in Deep Residual Networks. *ArXiv* abs/1603.05027 (2016).

[17] 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 (NIPS'17)*. Curran Associates Inc., Red Hook, NY, USA, 1729–1739.

[18] Gao Huang, Yixuan Li, Geoff Pleiss, Zhuang Liu, John E. Hopcroft, and Kilian Q. Weinberger. 2017. Snapshot Ensembles: Train 1, get M for free. *CoRR* abs/1704.00109 (2017). arXiv:1704.00109 http://arxiv.org/abs/1704.00109

[19] Stanisław Jastrzębski, Zac Kenton, Devansh Arpit, Nicolas Ballas, Asja Fischer, Amos Storkey, and Yoshua Bengio. 2018. Three factors influencing minima in SGD. https://openreview.net/forum?id=rJma2bZCW

[20] 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, KDD 2019, Anchorage, AK, USA, August 4-8, 2019*, Ankur Teredesai, Vipin Kumar, Ying Li, Rómer Rosales, Evimaria Terzi, and George Karypis (Eds.). ACM, 843–851. https://doi.org/10.1145/3292500.3330977

[21] 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. *CoRR* abs/1609.04836 (2016). arXiv:1609.04836 http://arxiv.org/abs/1609.04836

[22] Nitish Shirish Keskar and Richard Socher. 2017. Improving Generalization Performance by Switching from Adam to SGD. *CoRR* abs/1712.07628 (2017). arXiv:1712.07628 http://arxiv.org/abs/1712.07628

[23] Rahul Kidambi, Praneeth Netrapalli, Prateek Jain, and Sham M. Kakade. 2018. On the insufficiency of existing momentum schemes for Stochastic Optimization. *CoRR* abs/1803.05591 (2018). arXiv:1803.05591 http://arxiv.org/abs/1803.05591

[24] Diederik P. Kingma and Jimmy Ba. 2015. Adam: A Method for Stochastic Optimization. *CoRR* abs/1412.6980 (2015).

[25] Alex Krizhevsky, Ilya Sutskever, and Geoffrey E Hinton. 2012. ImageNet Classification with Deep Convolutional Neural Networks. (2012), 1097–1105. http://papers.nips.cc/paper/4824-imagenet-classification-with-deep-convolutional-neural-networks.pdf

[26] A. Kulunchakov and J. Mairal. 2019. Estimate Sequences for Variance-Reduced Stochastic Composite Optimization. In *ICML*.

[27] M. Laborde and Adam M. Oberman. 2020. A Lyapunov analysis for accelerated gradient methods: from deterministic to stochastic case. In *AISTATS*.

[28] Laurent Lessard, Benjamin Recht, and Andrew Packard. 2014. Analysis and Design of Optimization Algorithms via Integral Quadratic Constraints. *SIAM Journal on Optimization* 26 (08 2014). https://doi.org/10.1137/15M1009597

[29] 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:cs.LG/2003.00307

[30] Yanli Liu, Yuan Gao, and Wotao Yin. 2020. An Improved Analysis of Stochastic Gradient Descent with Momentum. arXiv:math.OC/2007.07989

[31] Ben London. 2017. A PAC-Bayesian Analysis of Randomized Learning with Application to Stochastic Gradient Descent. In *NIPS*.

[32] Ilya Loshchilov and Frank Hutter. 2016. SGDR: Stochastic Gradient Descent with Restarts. *CoRR* abs/1608.03983 (2016). arXiv:1608.03983 http://arxiv.org/abs/1608.03983

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

[34] Tianle Ma and Aidong Zhang. 2019. AffinityNet: Semi-Supervised Few-Shot Learning for Disease Type Prediction. In *The Thirty-Third AAAI Conference on Artificial Intelligence, AAAI 2019, The Thirty-First Innovative Applications of Artificial Intelligence Conference, IAAI 2019, The Ninth AAAI Symposium on Educational Advances in Artificial Intelligence, EAAI 2019, Honolulu, Hawaii, USA, January 27 - February 1, 2019*. AAAI Press, 1069–1076. https://doi.org/10.1609/aaai.v33i01.33011069

[35] Stephan Mandt, Matthew D. Hoffman, and David M. Blei. 2017. Stochastic Gradient Descent as Approximate Bayesian Inference. *J. Mach. Learn. Res.* 18, 1 (Jan. 2017), 4873–4907.

[36] David A. McAllester. 1998. Some PAC-Bayesian Theorems. In *Proceedings of the Eleventh Annual Conference on Computational Learning Theory (COLT' 98)*. Association for Computing Machinery, New York, NY, USA, 230–234. https://doi.org/10.1145/279943.279989

[37] Y. Nesterov. 1983. A method for solving the convex programming problem with convergence rate O(1/k^2).

[38] 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 *NeurIPS*.

[39] B.T. Polyak. 1964. Some methods of speeding up the convergence of iteration methods. *U. S. S. R. Comput. Math. and Math. Phys.* 4, 5 (1964), 1 – 17. https://doi.org/10.1016/0041-5553(64)90137-5

[40] Sebastian Ruder. 2016. An overview of gradient descent optimization algorithms. *ArXiv* abs/1609.04747 (2016).

[41] Karen Simonyan and Andrew Zisserman. 2015. Very Deep Convolutional Networks for Large-Scale Image Recognition. In *3rd International Conference on Learning Representations, ICLR 2015, San Diego, CA, USA, May 7-9, 2015, Conference Track Proceedings*. http://arxiv.org/abs/1409.1556

[42] L. N. Smith. 2017. Cyclical Learning Rates for Training Neural Networks. In *2017 IEEE Winter Conference on Applications of Computer Vision (WACV)*. 464–472. https://doi.org/10.1109/WACV.2017.58

[43] Leslie N. Smith and Nicholay Topin. 2017. Super-Convergence: Very Fast Training of Residual Networks Using Large Learning Rates. *CoRR* abs/1708.07120 (2017). arXiv:1708.07120 http://arxiv.org/abs/1708.07120

[44] Sam Smith and Quoc V. Le. 2018. A Bayesian Perspective on Generalization and Stochastic Gradient Descent. https://openreview.net/pdf?id=BJij4yg0Z

[45] Samuel L. Smith, Pieter-Jan Kindermans, and Quoc V. Le. 2018. Don't Decay the Learning Rate, Increase the Batch Size. In *International Conference on Learning Representations*. https://openreview.net/forum?id=B1Yy1BxCZ

[46] 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 - Volume 2 (NIPS'12)*. Curran Associates Inc., Red Hook, NY, USA, 2951–2959.

[47] Qiuling Suo, Liuyi Yao, Guangxu Xun, Jianhui Sun, and Aidong Zhang. 2019. Recurrent Imputation for Multivariate Time Series with Missing Values. In *2019 IEEE International Conference on Healthcare Informatics, ICHI 2019, Xi'an, China, June 10-13, 2019*. IEEE, 1–3. https://doi.org/10.1109/ICHI.2019.8904638

[48] 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 *IEEE International Conference on Big Data, Big Data 2020, Atlanta, GA, USA, December 10-13, 2020*. IEEE, 798–807. https://doi.org/10.1109/BigData50022.2020.9378408

[49] 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 - Volume 28 (ICML'13)*. III–1139–III–1147.

[50] 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.

[51] Sharan Vaswani, F. Bach, and M. Schmidt. 2019. Fast and Faster Convergence of SGD for Over-Parameterized Models and an Accelerated Perceptron. *ArXiv* abs/1810.07288 (2019).

[52] Ashia C Wilson, Rebecca Roelofs, Mitchell Stern, Nati Srebro, and Benjamin Recht. 2017. The Marginal Value of Adaptive Gradient Methods in Machine Learning. In *Advances in Neural Information Processing Systems 30*. Curran Associates, Inc., 4148–4158. http://papers.nips.cc/paper/7003-the-marginal-value-of-adaptive-gradient-methods-in-machine-learning.pdf

[53] Hongfei Xue, Wenjun Jiang, Chenglin Miao, Fenglong Ma, Shiyang Wang, Ye Yuan, Shuochao Yao, Aidong Zhang, and Lu Su. 2020. DeepMV: Multi-View Deep Learning for Device-Free Human Activity Recognition. *Proc. ACM Interact. Mob. Wearable Ubiquitous Technol.* 4, 1 (2020), 34:1–34:26. https://doi.org/10.1145/3380980

[54] Guangxu Xun, Kishlay Jha, Jianhui Sun, and Aidong Zhang. 2020. Correlation Networks for Extreme Multi-label Text Classification. *Proceedings of the 26th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining* (2020).

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

[56] Yan Yan, Tianbao Yang, Zhe Li, Qihang Lin, and Yi Yang. 2018. A Unified Analysis of Stochastic Momentum Methods for Deep Learning. In *IJCAI*. 2955–2961. https://doi.org/10.24963/ijcai.2018/410

[57] Jian Zhang and Ioannis Mitliagkas. 2018. YellowFin and the Art of Momentum Tuning. arXiv:stat.ML/1706.03471

[58] Z. Zhu and L. Orecchia. 2017. Linear Coupling: An Ultimate Unification of Gradient and Mirror Descent. In *ITCS*.

9 APPENDIX

In this section, we give the proof of Theorem 1, and 2. We keep the key proof steps and omit many algebraic transformations due to the page limit.

9.1 Proof of Theorem 1

In this section, we introduce two lemmas and prove Theorem 1.

Lemma 2 ([36]). *Let $KL(Q||P)$ as the KL divergence between two distributions Q and P. For any positive real $\delta \in (0,1)$, with probability at least $1-\delta$ over a sample of size N, we have the following inequality for all distributions Q:*

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

Lemma 3. *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:*

$$tr(\Sigma_\theta) = \frac{\alpha}{2b}tr(\Sigma A^{-1}) + \frac{\alpha^2}{2b}\Big(\frac{\nu\beta}{1-\beta}(1-\frac{2(1+\beta-\nu\beta)}{1+\beta}+\frac{1}{2})\Big)tr(\Sigma) + O(\alpha^3) \qquad (13)$$
$$\det(\Sigma_\theta) = (\frac{\alpha}{2b})^d\det(\Sigma A^{-1}) + O(\alpha^2)$$

proof of Lemma 3. 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 [11]:

$$\begin{aligned}&\Sigma_\theta^{(0)} = 0 \quad A\Sigma_\theta^{(1)} + \Sigma_\theta^{(1)}A = \frac{1}{b}\Sigma\\ &\Sigma_{\theta d}^{(1)} + \Sigma_{d\theta}^{(1)} = A\Sigma_\theta^{(1)} + \Sigma_\theta^{(1)}A - \frac{2(1+\beta-\nu\beta)}{1+\beta}\frac{1}{b}\Sigma\\ &A\Sigma_\theta^{(2)} + \Sigma_\theta^{(2)}A = \frac{2\nu\beta}{1-\beta}(\Sigma_{d\theta}^{(1)}A + A\Sigma_{\theta d}^{(1)}) + 2A\Sigma_\theta^{(1)}A\end{aligned} \qquad (14)$$

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:

$$\begin{aligned}&\Sigma_{\theta d}^{(1)}A^{-1} + \Sigma_{d\theta}^{(1)}A^{-1}\\ &= A\Sigma_\theta^{(1)}A^{-1} + \Sigma_\theta^{(1)} - \frac{2(1+\beta-\nu\beta)}{1+\beta}\frac{1}{b}\Sigma A^{-1}\\ &A\Sigma_\theta^{(2)}A^{-1} + \Sigma_\theta^{(2)}\\ &= \frac{2\nu\beta}{1-\beta}(\Sigma_{d\theta}^{(1)} + A\Sigma_{\theta d}^{(1)}A^{-1}) + 2A\Sigma_\theta^{(1)}\end{aligned} \qquad (15)$$

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 (14): $2\mathrm{tr}(A\Sigma_\theta^{(1)}) = \frac{1}{b}\mathrm{tr}(\Sigma)$ 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.

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

$$\begin{aligned}f_P &= \frac{1}{\sqrt{2\pi\det(\lambda_0 I_d)}}\exp\Big\{-\frac{1}{2}(\theta-\theta_0)^T(\lambda_0 I_d)^{-1}(\theta-\theta_0)\Big\}\\ f_Q &= \frac{1}{\sqrt{2\pi\det(\Sigma_\theta)}}\exp\Big\{-\frac{1}{2}\theta^T\Sigma_\theta^{-1}\theta\Big\}\end{aligned} \qquad (16)$$

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

Application of the determinant and trace from Lemma 3 here will complete our proof. □

9.2 Proof of Theorem 2

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

$$\begin{aligned}&\mathbb{E}_{\xi_k}[\mathcal{R}(\eta_{k+1})] \leq\\ &= \mathcal{R}(\eta_k) + \mathbb{E}_{\xi_k}[< \nabla\mathcal{R}(\eta_k), -\alpha_k\hat{g}_k >] + \frac{L\alpha_k^2}{2}\mathbb{E}_{\xi_k}[||\hat{g}_k||^2]\end{aligned} \qquad (17)$$

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

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

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

$$\begin{aligned}&\mathbb{E}[\mathcal{R}(\eta_{k+1})] \leq\\ &\mathbb{E}[\mathcal{R}(\eta_k)] + \alpha_k^3\frac{c_k}{2}L^2(\frac{\beta_k\nu_k}{1-\beta_k})^2\mathbb{E}[||d_{k-1}||^2]\\ &+(\alpha_k\frac{1}{2c_k} - \alpha_k)\mathbb{E}[||g_k||^2] + \frac{L\alpha_k^2}{2}\mathbb{E}[||\hat{g}_k||^2]\end{aligned} \qquad (18)$$

Let us make a small detour and first prove Lemma 1.

Proof of Lemma 1. We know $\mathbb{V}[y_k] = \mathbb{E}[||y_k - \Sigma_{i=1}^k a_{k,i}g_i||^2] = \mathbb{E}[||\Sigma_{i=1}^{k-1}a_{k,i}(g_i - \hat{g}_i) + (1-\nu_k\beta_k)(g_k - \hat{g}_k)||^2] \overset{(i)}{\leq} \Big(6 - 4\beta_1 - 4\beta_k\nu_k +$

$2\beta_k^2 v_k^2\Big)\sigma^2$, where (i) follows from that $\{\hat{g}_k\}_{k\in\mathbb{N}}$ are independent from each other and $\mathbb{E}_{\xi_k}[\|\hat{g}_k - g_k\|^2] \le \sigma^2$, and also Lemma 4 in [30]. We know $\Sigma_{i=1}^k a_{k,i} = 1 - \nu_k \prod_{i=1}^k \beta_i$, thus we have:

$$\mathbb{E}[\|g_k - \frac{1}{1-\nu_k\prod_{i=1}^k\beta_i}\Sigma_{i=1}^k a_{k,i}g_i\|^2] =$$
$$\frac{1}{(1-\nu_k\prod_{i=1}^k\beta_i)^2}\mathbb{E}[\|\Sigma_{i=1}^k a_{k,i}(g_k - g_i)\|^2] \overset{(i)}{\le} \frac{1}{(1-\nu_k\prod_{i=1}^k\beta^i)^2}\times$$
$$\Sigma_{i,j=1}^k a_{k,i}a_{k,j}\Big(\frac{1}{2}\mathbb{E}[\|g_k - g_j\|^2] + \frac{1}{2}\mathbb{E}[\|g_k - g_i\|^2]\Big)$$
$$= \frac{1}{1-\nu_k\prod_{i=1}^k\beta_i}\Sigma_{i=1}^k a_{k,i}\mathbb{E}[\|g_k - g_i\|^2]$$
$$\overset{(ii)}{\le} \frac{1}{1-\nu_k\prod_{i=1}^k\beta_i}\Sigma_{i=1}^k a_{k,i}\Big((k-i)\Sigma_{j=i}^{k-1}\mathbb{E}[\|g_{j+1} - g_j\|^2]\Big)$$
$$\overset{(iii)}{\le} \frac{1}{1-\nu_k\prod_{i=1}^k\beta_i}\Sigma_{i=1}^k a_{k,i}\Big((k-i)L^2\Sigma_{j=i}^{k-1}\mathbb{E}[\|\theta_{j+1} - \theta_j\|^2]\Big)$$
$$= \frac{1}{1-\nu_k\prod_{i=1}^k\beta_i}\Sigma_{j=1}^{k-1}\Big(\Sigma_{i=1}^j a_{k,i}(k-i)\Big)\mathbb{E}[\|\theta_{j+1} - \theta_j\|^2]L^2$$
$$\overset{(iv)}{\le} \Sigma_{j=1}^{k-1}\frac{\nu_k\beta_k^{k-j}L^2}{1-\nu_k\prod_{i=1}^k\beta_i}\Big(k - j + \frac{\beta_k}{1-\beta_k}\Big)\mathbb{E}[\|\theta_{j+1} - \theta_j\|^2] \tag{19}$$

where (i) follows from Cauchy-Schwarz inequality, (ii) follows from triangle inequality, (iii) follows from smoothness, (iv) follows from Proposition 5 in [30]. □

Back to Inequality (18):

$$\mathbb{E}[\mathcal{R}(\eta_{k+1})] \le \mathbb{E}[\mathcal{R}(\eta_k)] + \alpha_k^3\frac{c_k}{2}L^2(\frac{\beta_k\nu_k}{1-\beta_k})^2\mathbb{E}[\|d_{k-1}\|^2]$$
$$+(\alpha_k\frac{1}{2c_k} - \alpha_k)\mathbb{E}[\|g_k\|^2] + \frac{L\alpha_k^2}{2}\mathbb{E}[\|\hat{g}_k\|^2]$$

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 [30], where q_i are constants to be determined, omitting many algebraic transformations: we have $\mathbb{E}[L_{k+1} - L_k] \le$

$$\Big(2\alpha_k c_k L^2 W_1^2 - \alpha_k + \frac{\alpha_k}{2c_k} + \frac{1}{2}L\alpha_k^2 + 4q_1\alpha_k^2\Big)\mathbb{E}[\|g_k\|^2]$$
$$+\frac{1}{2}L\alpha_k^2\sigma^2 + \alpha_k c_k L^2 W_1^2\mathbb{E}[\|d_{k-1} - \Sigma_{i=1}^{k-1}a_{k-1,i}g_i\|^2] +$$
$$2q_1\alpha_k^2\mathbb{E}[\|y_k - \Sigma_{i=1}^k a_{k,i}g_i\|^2]$$
$$+4q_1\alpha_k^2(1-\nu_k\prod_{i=1}^k\beta_i)^2\mathbb{E}[\|g_k - \frac{1}{1-\nu_k\prod_{i=1}^k\beta_i}\Sigma_{i=1}^k a_{k,i}g_i\|^2]$$
$$+2\alpha_k c_k L^2 W_1^2\frac{1}{\beta_k^2}(1-\prod_{i=1}^k\beta_i)^2\mathbb{E}[\|g_k - \frac{1}{1-\prod_{i=1}^k\beta_i}\Sigma_{i=1}^k b_{k,i}g_i\|^2]$$
$$+\Sigma_{i=1}^{k-1}(q_{i+1} - q_i)\|\theta_{k+1-i} - \theta_{k-i}\|^2 \tag{20}$$

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 [30], we could have:

$$\mathbb{E}[L_{k+1} - L_k] \le \Big(2\alpha_k c_k L^2 W_1^2 - \alpha_k + \frac{\alpha_k}{2c_k} + \frac{1}{2}L\alpha_k^2 + 4q_1\alpha_k^2\Big)\mathbb{E}[\|g_k\|^2]$$
$$+\Big(\frac{1}{2}L\alpha_k^2 + 24\alpha_k c_k L^2 W_1^2\frac{\beta_1(1-\beta_k)}{\sqrt{\beta_n+\beta_n^2}} + 2q_1\alpha_k^2(6 - 4\beta_1 - 4\beta_k\nu_k + 2\beta_k^2\nu_k^2)\Big)\sigma^2 \tag{21}$$

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 **??**, 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 $Q_{2,k}$:

$$Q_{2,k} \le \frac{1}{2}\alpha_k^2 L + 12\alpha_k^2 L\frac{\beta_k^2\nu_k^2}{(1-\beta_k)^2}\frac{\beta_1}{\sqrt{\beta_n+\beta_n^2}} + \frac{3+\beta_k^2\nu_k^2}{1-\beta_1}\alpha_k^2 L\sigma^2 \tag{22}$$

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 $k = T_1 + T_2 + ... + T_M$:

$$\sum_{l=1}^M\frac{\alpha_l}{2}\sum_{i=T_1+T_2+...+T_{l-1}+1}^{T_1+T_2+...+T_l}\mathbb{E}[\|g_i\|^2] \le L_1$$
$$+\sum_{l=1}^M T_l\Big(\frac{1}{2}\alpha_k^2 L + 12\alpha_k^2 L\frac{\beta_k^2\nu_k^2}{(1-\beta_k)^2}\frac{\beta_1}{\sqrt{\beta_n+\beta_n^2}} + \frac{3+\beta_k^2\nu_k^2}{1-\beta_1}\alpha_k^2 L\Big)\sigma^2 \tag{23}$$

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

$$\frac{1}{M}\sum_{l=1}^M\frac{1}{T_l}\sum_{i=T_1+T_2+...+T_{l-1}+1}^{T_1+T_2+...+T_l}\mathbb{E}[\|g_i\|^2] \le$$
$$\frac{2(\mathcal{R}(\theta_1) - \mathcal{R}^*)}{MW_2} + \frac{1}{M}\sum_{l=1}^M\Big(\alpha_k L + 24\alpha_k L\frac{\beta_k^2\nu_k^2}{(1-\beta_k)^2}\frac{\beta_1}{\sqrt{\beta_n+\beta_n^2}}$$
$$+\frac{6+2\beta_k^2\nu_k^2}{1-\beta_1}\alpha_k L\Big)\sigma^2$$

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