UNDERSTANDING SHORT-HORIZON BIAS IN STOCHASTIC META-OPTIMIZATION

Anonymous authors
Paper under double-blind review

ABSTRACT

Careful tuning of the learning rate, or even schedules thereof, can be crucial to effective neural net training. There has been much recent interest in gradient-based meta-optimization, where one tunes hyperparameters, or even learns an optimizer, in order to minimize the expected loss when the training procedure is unrolled. But because the training procedure must be unrolled thousands of times, the meta-objective must be defined with an orders-of-magnitude shorter time horizon than is typical for neural net training. We show that such short-horizon meta-objectives cause a serious bias towards small step sizes, an effect we term short-horizon bias. We introduce a toy problem, a noisy quadratic cost function, on which we analyze short-horizon bias by deriving and comparing the optimal schedules for short and long time horizons. We then run meta-optimization experiments (both offline and online) on standard benchmark datasets, showing that meta-optimization chooses too small a learning rate by multiple orders of magnitude, even when run with a moderately long time horizon (100 steps) typical of work in the area. We believe short-horizon bias is a fundamental problem that needs to be addressed if meta-optimization is to scale to practical neural net training regimes.

1 INTRODUCTION

The learning rate is one of the most important and frustrating hyperparameters to tune in deep learning. Too small a value causes slow progress, while too large a value causes fluctuations or even divergence. From optimization theory perspective, learning rate decay is required to achieve convergence guarantee for stochastic gradient methods under certain conditions (Bottou, 1998). While a fixed learning rate often works well for simpler problems, good performance on the ImageNet (Russakovsky et al., 2015) benchmark requires a carefully tuned schedule. A variety of decay schedules have been proposed for different architectures, including (Szegedy et al., 2014), staircase (VGG, ResNet) (Simonyan & Zisserman, 2015), (He et al., 2016). Clever learning rate heuristics have resulted in large improvements in training efficiency (Goyal et al., 2017), (Smith, 2017). A related hyperparameter is momentum; typically fixed to a reasonable value such as 0.9, careful tuning can also give significant performance gains (Sutskever et al., 2013). While optimizers such as Adam (Kingma & Ba, 2015) are often described as adapting coordinate-specific learning rates, in fact they also have global learning rate and momentum hyperparameters analogously to SGD, and tuning at least the learning rate can be important to good performance.

Because of this, it is not surprising that there have been many attempts to adapt learning rates, either online during optimization (Schraudolph, 1999), (Schaul et al., 2013), by learning a policy offline corresponding to a learning rate schedule (Maclaurin et al., 2015), or even with an optimizer (Andrychowicz et al., 2016), (Li & Malik, 2017), (Chelsea Finn, 2017). All of these approaches are forms of meta-optimization, where one defines a meta-objective, the expected loss after some number of optimization steps, and tunes the hyperparameters to minimize this meta-objective. But because gradient-based meta-optimization can require thousands of updates, each of which unrolls the entire base-level optimization procedure, the meta-optimization is thousands of times more expensive than the base-level optimization. Therefore, the meta-objective must be defined with a much smaller time horizon (e.g. hundreds of updates) than we are ordinarily interested in for large-scale optimization. The hope is that the learned hyperparameters or optimizer will generalize well to much longer time horizons. However, we show that this is not achieved because of a strong tradeoff between short-term and long-term performance, which we refer to as *short-horizon bias*.

The basic reason for short-horizon bias can be seen in Figure 1, which shows a typical ImageNet training curve with a learning rate schedule. Each drop in the learning rate causes an immediate drop in the cost function. Therefore, any short-horizon meta-optimizer would want to aggressively decay the learning rate to achieve these immediate gains. But this would come at the expense of long-term optimization progress. The goals of this paper are to make this short-horizon bias more precise, provide empirical evidence for it, and to suggest a toy problem that can be used to analyze, understand, and potentially overcome this bias.

We first define a toy problem to illustrate the fundamental tradeoff between short-horizon and long-horizon performance. In particular, we analyze a noisy quadratic cost function based on (Schaul et al., 2013). We consider this a good proxy for neural net training because second-order optimization algorithms have been shown to train neural networks in orders-of-magnitude fewer iterations (Martens, 2010), suggesting that much of the difficulty of SGD training can be explained by quadratic approximations to the cost. In our noisy quadratic problem, the dynamics of SGD with momentum can be analyzed exactly, allowing us to derive the 1-step-horizon-optimal learning rate and momentum in closed form, as well as to fit the long-horizon-optimal schedule using gradient descent. We analyze the differences between the short-horizon and long-horizon schedules. Because short-horizon performance is dominated by high-curvature directions, the short-horizon optimizer neglects low-curvature directions, leading to poor long-term performance.

Next, we consider the setting of gradient-based meta-optimization. To measure short-horizon bias in a practical setting, we consider two idealized meta-optimization algorithms: an offline algorithm which fits a learning rate decay schedule by running optimization many times from scratch, and an online algorithm which adapts the learning rate and momentum during training. Since our interest is in studying the effect of the meta-objective itself, we allow the meta-optimizers sufficient time to optimize their meta-objectives well. We show that short-horizon meta-optimizers dramatically underperform a hand-tuned fixed learning rate, and sometimes cause base-level optimization progress to slow to a crawl, even with moderately long time horizons (e.g. 100 or 1000 steps) similar to those used in prior work on gradient-based meta-optimization .

In short, we expect that any meta-objective which doesn't correct for short-horizon bias will probably fail dramatically when run for a much longer time horizon than it was trained on. There are applications where short-horizon meta-optimization is directly useful, such as few-shot learning (Lake et al., 2011). In those settings, short-horizon bias is by definition not an issue. But much of the appeal of meta-optimization comes from the possibility of using it to speed up or simplify the training of large neural networks. In these settings, short-horizon bias is a fundamental obstacle that must be addressed for meta-optimization to be practically useful.

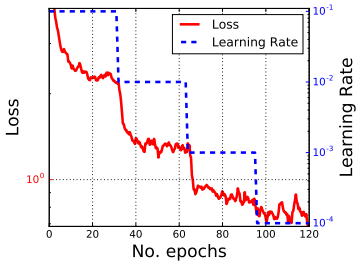


Figure 1: A typical training curve on ImageNet. A proper learning rate schedule is crucial.

2 Noisy Quadratic Problem

In this section, we develop a toy problem which demonstrates the short-horizon bias, but that we can analyze analytically. In particular, we borrow the noisy quadratic model of Schaul et al. (2013); the true function being optimized is a quadratic, but in each iteration, we observe a "noisy" version with perturbed center but exact curvature. We analyze the dynamics of SGD with momentum on this example, determine both the optimal and short-horizon-optimal learning rate schedules.

2.1 Background

Approximating the cost surface of a neural network with a quadratic function has yielded powerful insights and algorithms. Second-order optimization methods such as Newton-Raphson and natural gradient (Amari, 1998) iteratively minimize quadratic approximations to a cost function. Hessian-free (H-F) optimization (Martens, 2010) is an approximate natural gradient method which approximately minimizes the quadratic approximation using conjugate gradients; this was able to fit

deep neural networks in orders-of-magnitude fewer updates than SGD, suggesting that much of the difficulty of neural net optimization is captured by quadratic models. In the setting of Bayesian neural networks, quadratic approximations to the log-likelihood motivated the Laplace approximation (MacKay, 1992). Koh & Liang (2017) used quadratic approximations to analyze the sensitivity of a neural networks' predictions to particular training labels, thereby yielding insight into adversarial examples.

Our primary interest is to find a shared optimal learning rate and momentum across all dimensions. The reason we focus on this instead of coordinate-specific learning rates is because we are assuming doing gradient descent in a preconditioned space. Diagonal methods (Kingma & Ba, 2015), and even K-FAC (Martens & Grosse, 2015), can be viewed as gradient descent in a preconditioned space. Since current methods do not provide a perfect preconditioning, we do consider various curvature on each dimension in our model.

Quadratic approximations have directly yielded insight into optimization hyperparameters. In a deterministic setting, under certain conditions, second-order optimization algorithms can be run with a learning rate of 1; for this reason, H-F was able to eliminate the need to tune learning rate or momentum hyperparameters. Martens & Grosse (2015) observed that for a deterministic quadratic cost function greedily choosing the learning rate and momentum to minimize the error on the next step is equivalent to conjugate gradients. Therefore, by a general property of conjugate gradient (CG), the greedily chosen learning rates and momenta are optimal, in the sense that the greedy sequence achieves the minimum possible loss value of *any* sequence of learning rates and momenta. This property fails to hold in the stochastic setting, however, and as we show in this section, the greedy choice of learning rate and momentum can do considerably worse than optimal.

2.2 Analysis

2.2.1 Notations

We will primarily focus on the SGD with momentum algorithm in this paper. The update is written as follows,

$$v^{(t+1)} = \mu^{(t)} v^{(t)} - \alpha^{(t)} \nabla_{\theta^{(t)}} \mathcal{L} \tag{1}$$

$$\theta^{(t+1)} = \theta^{(t)} + v^{(t+1)} \tag{2}$$

where $\mathcal{L}$ is the loss function, $\alpha^{(t)}$ is the learning rate, and $v^{(t)}$ called velocity, with decay constant $\mu^{(t)}$, called momentum, at time t. For simplicity, we also denote $A(\cdot) = \mathbb{E}[\cdot]^2 + \mathbb{V}[\cdot]$, where $\mathbb{E}$ and $\mathbb{V}$ denote expectation and variance respectively.

2.2.2 Problem formulation

The problem is to find the optimal learning rate and momentum for optimizing a noisy quadratic problem. For every quadratic, one can choose a coordinate system such that the Hessian is diagonal. We assume the gradient noise variance to be co-diagonalizable with Hessians. The loss is written as,

$$\mathcal{L}(\boldsymbol{\theta}) = \mathbb{E}_{\mathbf{c}\sim\mathcal{P}}[\sum_i \frac{1}{2} h_i(\theta_i - c_i)^2] = \frac{1}{2}\sum_i h_i((\theta_i - \theta_i^*)^2 + \sigma_i^2)$$

where the expectation is taken over a data distribution $\mathcal{P}$, and θ_i, c_i are the ith coordinate of the parameter $\boldsymbol{\theta}$ and the sample $\mathbf{c}$, and $\mathbb{E}_{\mathbf{c}\sim\mathcal{P}}[c_i] = \theta_i^*$, $\mathbb{V}[c_i] = \sigma_i^2$. The stochastic gradient is written as $\nabla_{\theta_i}\mathcal{L} = h(\theta_i - c_i)$. WLOG, we assume $\theta^* = 0$, so the stochastic gradient becomes,

$$\nabla_{\theta_i}\mathcal{L} = h_i\theta_i - h_i\sigma_i\xi, \quad \xi \sim \mathcal{N}(0,1)$$

The expected loss at timestep t is,

$$\mathcal{L}(\boldsymbol{\theta}^{(t)}) = \sum_i \frac{1}{2} h_i((\theta_i^{(t)})^2 + (\sigma_i^{(t)})^2) = \sum_i \frac{1}{2} h_i(\mathbb{E}[\theta_i^{(t)}]^2 + \mathbb{V}[\theta_i^{(t)}] + (\sigma_i^{(t)})^2) \tag{3}$$

2.2.3 Coordinate-Sharing hyperparameters

We first consider the setting where the curvature already accounts for preconditioners, and hence the learning rate and momentum are shared across all dimensions. We derive the *greedy* optimal learning rate and momentum by from the dynamics of the SGD with momentum.

Several observations allow us to compactly model the dynamics following SGD with momentum on the noisy quadratic model. First, the expected loss can be expressed in terms of the variables $\mathbb{E}[\theta_i]$ and $\mathbb{V}[\theta_i]$. Second, because the Hessians and noise covariances are diagonal, each coordinate evolves independently of the others. Third, the second-order statistics of the iterate $\theta_i^{(t)}$ and the velocity $v_i^{(t)}$ can be expressed in terms of the second-order statistics of $\theta_i^{(t-1)}$ and $v_i^{(t-1)}$. Combining these observations, we model the dynamics of SGD with momentum as a *deterministic* recurrence relation with sufficient statistics $\mathbb{V}[\theta^{(t)}]$, $\mathbb{V}[v^{(t)}]$, and $\Sigma_{\theta,v}^{(t)} = \mathrm{Cov}(\theta^{(t)}, v^{(t)})$. Note that we drop the dimension subscripts because the dimensions evolve independently. The dynamics are as follows:

Theorem 1 (Mean and variance dynamics)**.** *The expectations of the parameter θ and the velocity v are updated as,*

$$\mathbb{E}[v^{(t+1)}] = \mu^{(t)}\mathbb{E}[v^{(t)}] - (\alpha^{(t)}h)\mathbb{E}[\theta^{(t)}]$$
$$\mathbb{E}[\theta^{(t+1)}] = \mathbb{E}[\theta^{(t)}] + \mathbb{E}[v^{(t+1)}]$$

The variances of the parameter θ and the velocity v are updated as,

$$\mathbb{V}[v^{(t+1)}] = (\mu^{(t)})^2\mathbb{V}[v^{(t)}] + (\alpha^{(t)}h)^2\mathbb{V}[\theta^{(t)}] - 2\mu^{(t)}\alpha^{(t)}h\Sigma_{\theta,v}^{(t)} + (\alpha^{(t)}h\sigma)^2$$
$$\mathbb{V}[\theta^{(t+1)}] = (1 - 2\alpha^{(t)}h)\mathbb{V}[\theta^{(t)}] + \mathbb{V}[v^{(t+1)}] + 2\mu^{(t)}\Sigma_{\theta,v}^{(t)}$$
$$\Sigma_{\theta,v}^{(t+1)} = \mu^{(t)}\Sigma_{\theta,v}^{(t)} - \alpha^{(t)}h\mathbb{V}[\theta^{(t)}] + \mathbb{V}[v^{(t+1)}]$$

Theorem 1 allows us to simulate a forward dynamics of how loss changes over time given any schedule of $\{\alpha^{(t)}, v^{(t)}\}_t$, and express the loss at timestep T as $\mathcal{L}(\boldsymbol{\theta}^{(T)})(\{\alpha^{(t)}, v^{(t)}\}_{t=1}^{T-1})$. It turns the stochastic problem into a deterministic one, and so we are able to solve for the optimal learning rate and momentum. Further, we can get an analytic solution if we only do 1-step lookahead, i.e., we can solve for the optimal learning rate $\alpha^{(t)*}$ and momentum $\mu^{(t)*}$ that optimizes the loss at timestep $t+1$, $\mathcal{L}(\boldsymbol{\theta}^{(t)})$. We call this as the *greedy* optimal learning rate and momentum.

Theorem 2 (Greedy optimal learning rate and momentum)**.** *The global optimal learning rate and momentum is given by,*

$$\alpha^{(t)*} = \frac{\sum_i \left(h_i^2 A(\theta_i^{(t)})\left(\sum_j h_j A(v_j^{(t)})\right) - \left(\sum_j h_j \mathbb{E}[\theta_j^{(t)} v_j^{(t)}]\right) h_i^2 \mathbb{E}[\theta_i^{(t)} v_i^{(t)}]\right)}{\sum_i \left(\left(h_i^3(A(\theta_i^{(t)}) + (\sigma_i)^2)\right)\left(\sum_j h_j A(v_j^{(t)})\right) - \left(\sum_j (h_j)^2 \mathbb{E}[\theta_j^{(t)} v_j^{(t)}]\right) h_i^2 \mathbb{E}[\theta_i^{(t)} v_i^{(t)}]\right)}$$
$$\mu^{(t)*} = -\frac{\sum_i h_i(1 - \alpha^{(t)*}h_i)\mathbb{E}[\theta_i^{(t)} v_i^{(t)}]}{\sum_i h_i A(v_i^{(t)})}$$

Note that the greedy optimal learning rate derived in Schaul et al. (2013) is a special case of Theorem 2, in which the momentum is assumed to be 0. Analytically solving for the optimal learning rate and momentum for greater than 1-step look ahead is infeasible. However, since Theorem 1 describes how the loss changes over time, we can numerically solve the problem of T-steps lookahead with Autodiff softwares and do gradient descent on the loss at timestep T with respect to the learning rate and momentum.

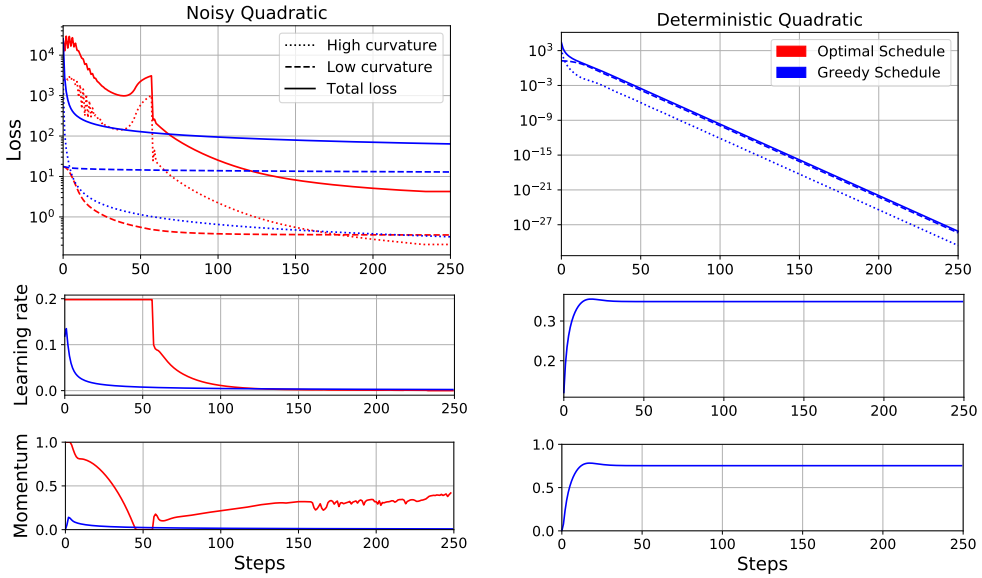
2.2.4 Coordinate-specific hyparameters

Now we consider solving the problem (Eq.(3)) using coordinate-specific hyperparameters. This reduces to a one dimensional problem as each coordinate evolves independently of others. In the vanilla SGD case, the greedy optimal learning rate is indeed optimal, in the sense that it achieves the minimum possible loss value of *any sequence* of learning rates.

Theorem 3 (Optimal learning rate)**.** *For all $T \in \mathbb{N}$, the sequence of learning rate $\{\alpha^{(t)*}\}_{t=1}^{T-1}$ that minimizes the loss at time step T, $\mathcal{L}(\theta^T)$ is given by,*

$$\alpha^{(t)*} = \frac{A(\theta^{(t)})}{h(A(\theta^{(t)}) + \sigma^2)} \tag{4}$$

Moreover, this agrees with the greedy optimal learning rate.



(a) (b)

Figure 2: Comparisons of the optimal learning rate and momentum trained by gradient descent and greedy learning rate and momentum in both noisy (a) and deterministic (b) quadratic settings. In the deterministic case, our optimized schedule matched the greedy one, just as the theory predicts.

Note that when all the coordinates have equal curvature and noise, the optimal learning rate is shared across all dimensions, so that applying the greedy optimal learning rate is optimal. Second-order methods try to precondition the problem so that the curvature is spherical. The noise variance becomes spherical when the Fisher is a good approximation to the Hessian, so that the noise variance matches the curvature. Therefore, one-step lookahead will work well with a sufficiently good natural gradient method.

2.3 Experiments

We now compare the results we obtain if we do one-step lookahead vs. T-steps lookahead, where $T = 250$. We chose a 1000 dimensional quadratic with the curvature distribution adopted from Li (2005), that which CG achieves the worst convergent rate. Since the Fisher matrix is an approximation to the Hessian matrix, we assume that $h_i = \mathbb{V}[\nabla_{\theta_i}\mathcal{L}]$, and hence $\sigma_i = \frac{1}{\sqrt{h_i}}$. We applied the greedy learning rate and momentum to this problem. We then compared it with the learning rate and momentum obtained by minimizing the T-step losses using gradient descent. The results and the corresponding learning rate and momentum are shown in Figure 2 (a). The optimal total loss decreases to a much lower value (4.25) than the loss obtained by greedy schedule (63.86).

We observe that the optimal learning rate and momentum are much higher than the greedy learning rate and momentum in the beginning of the training. We then look at the sum of the losses on the 50 highest curvature directions and 50 lowest curvature directions separately. We find that by following the optimal schedule, the loss on the high curvature directions do not decrease initially However, by keeping at a high learning rate and momentum, the loss on the low curvature directions decreases significantly. On the other hand, by following a greedy schedule, the learning rate and momentum become small very early, which makes the loss on the high curvature directions decrease instantly, along with the total loss. However, in the long term, since the learning rate is too small to make the loss on the low curvature direction decrease, it converged to a much higher loss in the end. This gives a valuable insight into the reason behind the short-horizon bias we see in meta optimization. As we will also see later in the real datasets, short horizon objective will often encourage the learning rate and momentum to be conservative, so as to achieve largest gain in short term, but performing poorly in the long term.

On the other hand, the short term horizon bias is also the result of stochasticity in the objective. As observed by Martens & Grosse (2015), the greedy learning rate is the optimal learning rate schedule in the deterministic case. We also validated it by setting the noise σ_i to zero and run greedy schedule

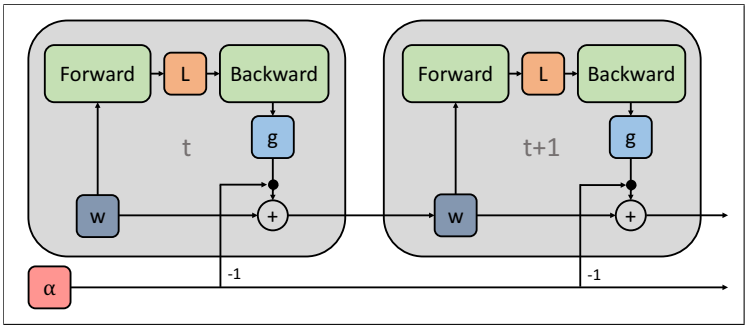


Figure 3: Regular SGD in the form of a computation graph

compared to the schedule found by gradient descent. Both curves aligned, shown in Figure 2 (b). The noise in the problem adds uncertainty to the objective, resulting in failures of greedy schedule.

3 Gradient-based meta-optimization

We now turn our attention to gradient-based hyperparameter optimization. A variety of approaches have been proposed which tune hyperparameters by doing gradient descent on a meta-objective (Schraudolph, 1999; Maclaurin et al., 2015; Andrychowicz et al., 2016). We empirically analyze an idealized meta-optimization algorithm which is simplified in two ways. First, we drop the algorithmic tricks used in prior work, and instead allow the meta-optimizer more computation than would be economical in practice. Second, we limit the representational power of our meta-model: whereas Andrychowicz et al. (2016) aimed to learn a full optimization algorithm, we focus on the much simpler problem of adapting learning rate and momentum hyperparameters, or schedules thereof. The aim of these two simplifications is that we would like to do a good enough job of optimizing the meta-objective that any failures arise from deficiencies in the meta-objective itself (i.e. short-horizon bias) rather than incomplete meta-optimization.

However, we believe our findings are applicable to the more general setting without these two simplifications. We don't see any reason why cheaper meta-optimization methods would address the shortcomings of the meta-objective. The weights of a full meta-optimizer would implicitly encode learning rate schedules. For instance, Duchi et al. (2011) implicitly uses a polynomial decay schedule because the denominator of the update uses the sum of the gradient variances rather than the mean. Therefore, any sufficiently powerful meta-optimization method needs to take into account short-horizon bias.

3.1 Background

3.1.1 Stochastic Meta Descent

The basic idea of stochastic meta descent (SMD) (Schraudolph, 1999) is to perform gradient descent on learning rate, or any other differentiable hyper-parameters. Any gradient based optimization can be unrolled as a computation graph (see Figure 3), and automatic differentiation is readily available in most deep learning libraries.

Two types of automatic differentiation (auto diff) methods are commonly used: *forward mode* and *reversed mode*. Suppose a node on the computation graph has inputs $\boldsymbol{x}$ and outputs $\boldsymbol{y}$. In forward mode auto diff, gradients are computed as a dual to regular computations: $\boldsymbol{y}' = J\boldsymbol{x}'$, where J is the Jacobian. In contrast, reverse mode auto diff (a.k.a. backpropagation) computes the gradients in reversed order: $\nabla_x = J^\top \nabla_y$. Since J depends on the values of $\boldsymbol{x}$ for any non-linear functions, reverse mode auto diff needs to store the values of $\boldsymbol{x}$ until the gradients are computed.

Meta-optimization using reverse mode can be computationally demanding due to memory constraints. Maclaurin et al. (2015) got around this by cleverly exploiting reversibility to minimize the memory cost of activations. Since we are optimizing only two hyperparameters, forward mode auto diff can be computationally cheap. We provide the forward differentiation equations for obtain-

ing the gradient of vanilla SGD learning rate. Let $\frac{\partial \boldsymbol{\theta}_t}{\partial \alpha}$ be $\boldsymbol{\theta}'_{\alpha,t}$, and $\frac{\partial \mathcal{L}_t}{\partial \alpha}$ be $g_{\alpha,t}$, and the Hessian at step t to be H_t. By chain rule, we get,

$$g_{\alpha,t} = \boldsymbol{g}_{\theta,t} \cdot \boldsymbol{\theta}'_{\alpha,t-1}, \tag{5}$$

$$\boldsymbol{\theta}'_{\alpha,t} = \boldsymbol{\theta}'_{\alpha,t-1} + \alpha H_t \boldsymbol{\theta}'_{\alpha,t-1}. \tag{6}$$

The Hessian-vector product in Equation 6 can be computed efficiently using reverse-on-reverse (Werbos, 1988) or forward-on-reverse automatic differentiation (Pearlmutter, 1994), which is linear in the running time of a forward pass, without explicitly computing the full Hessian.

Gradient based meta-optimization is feasible after the gradients to the hyperparameters are computed. It is worth noting that, although SMD was originally proposed for optimizing SGD, in practice it can optimize any optimizer with few differentiable hyper-parameters, e.g. SGD with momentum, RMSProp (Tieleman & Hinton, 2012), Adam (Kingma & Ba, 2015), etc. Moreover, the meta updates can be done by any type of gradient based optimizers as well.

We provide details of SMD algorithm in Algorithm 1. α is a set of hyperparameters (e.g. learning rate), and α_0 are inital hyperparameter values. $\boldsymbol{\theta}$ is a set of optimization intermediate variables, such as weights and velocity. η is a set of meta optimizer hyperparameters (e.g. meta learning rate). Additionally, T is the look ahead window size, and M is the number of meta updates.

Algorithm 1: Stochastic Meta Descent

Input: $\alpha_0, \eta, \boldsymbol{\theta}, T, M$
Output: α
$\boldsymbol{\theta}_0 \leftarrow \boldsymbol{\theta}$;
$\alpha \leftarrow \alpha_0$;
for $m = 1$ *to* M **do**
  $\boldsymbol{\theta}'_\alpha \leftarrow \mathbf{0}$;
  for $t = 1$ *to* $T - 1$ **do**
    $X, \boldsymbol{y} \leftarrow$ `GetMiniBatch()`;
    $\boldsymbol{g}_\theta \leftarrow$ `ForwardBackProp(`$X, \boldsymbol{y}, \boldsymbol{\theta}$`)`;
    $\boldsymbol{\theta}_{\text{new}} \leftarrow$ `OptimizerStep(`$\theta, \boldsymbol{g}_\theta, \alpha$`)`;
    $g_\alpha \leftarrow \boldsymbol{g}_\theta \cdot \boldsymbol{\theta}'_\alpha$;
    $\boldsymbol{\theta}'_\alpha \leftarrow$ `ForwardGrad(`$\boldsymbol{\theta}_{new}, \alpha, \boldsymbol{\theta}'_\alpha$`)`;
    $\boldsymbol{\theta} \leftarrow \boldsymbol{\theta}_{\text{new}}$;
  $\alpha \leftarrow$ `MetaOptimizerStep(`α, g_α, η`)`;
  $\boldsymbol{\theta} \leftarrow \boldsymbol{\theta}_0$
return α

Simplifications from the original SMD algorithm. In the original algorithm (Schraudolph, 1999), 1) each parameter dimension has an individual adaptive learning rate, 2) intermediate gradients ($\boldsymbol{\theta}'_\alpha$) are accumualted throughout the process of training, 3) only one meta update step is applied. Many of those designs are based on speed reasons, and as a consequence the original algorithm does not compute the exact gradients for meta- optimization. Since we can afford extra computation to study the results of meta-optimization, we adapted the original version such that 1) only global hyper-parameters are tuned and exact gradients are computed, 2) M meta update steps are applied.

3.2 Offline meta-optimization

To understand the sensitivity of the best hyper-parameter towards the number of look ahead steps, we first design an offline experiment on a multi-layered perceptron (MLP) on MNIST (LeCun et al., 1998). Since the dynamics of training are different at the very start compared to later stages, we pretrain the network first with manually tuned learning rate and momentum.

Learnable decay schedule. We include a parameterized learning rate decay schedule, on top of the SGD with momentum updates. A standard decay schedule is the inverse time decay (Welling & Teh, 2011): $\alpha_t = \frac{\alpha_0}{(1+\frac{t}{K})^\beta}$, where t is the number of training steps, β is the learning rate decay exponent, and K is the time constant. In the SMD procedure, we optimize β together with α and μ.

Experimental details. The network layer size is [784-100-100-10], with ReLU activations. Weights are initialized with zero- mean Gaussian distribution with standard deviation 0.1. The network is pretrained with 1000 SGD with momentum steps, with $\alpha = 1, \mu = 0.9$. We train the all hyper-parameters on log space using Adam optimizer, with 2k meta steps. After one meta-step, we restore the weights back to its pretrained initialization (see Algorithm 1). We re-parameterized the momentum with $(1 - \mu)$, the learning rate with the effective learning rate $\alpha_{\text{eff}} = \frac{\alpha}{1-\mu}$.

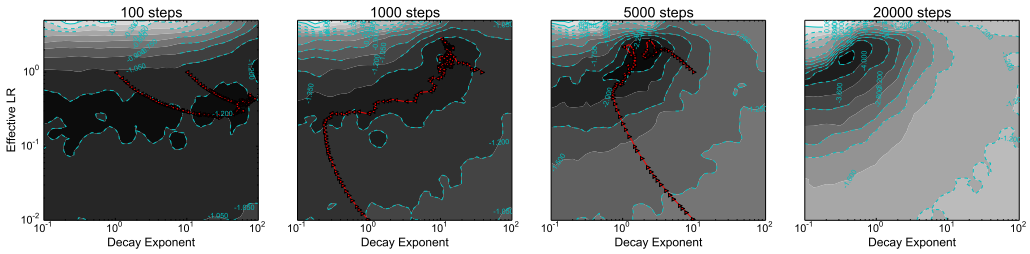


Figure 4: SMD trajectories optimizing initial effective learning rate and decay exponent by looking ahead {100, 1k, 5k, 20k} steps. 2.5k random samples with Gaussian interpolation per plots are used to illustrate the loss surface. Loss values displayed in logarithmic scale. The training losses obtrained using the selected hyperparameters are $\{4.8 \times 10^{-2}, 1.3 \times 10^{-1}, 1.3 \times 10^{-3}, 4.0 \times 10^{-5}\}$.

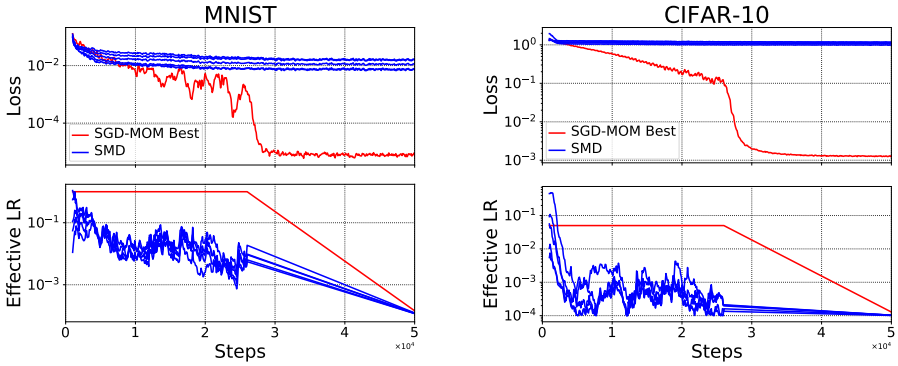


Figure 5: SMD look ahead 5 steps with different initial learning rates, compared with the best manual setting of SGD with momentum. Model is pre-trained with SGD 1000 steps. 100 meta updates using Adam optimizer tuning learning rate and momentum every 10 training steps.

Figure 4 plots SMD optimization trajectories on the loss surface, which is obtained by using random samples of hyperparameters. Multiple trajectories were run to ensure the local minima match with the ones plotted on the surface. The final hyperparameter values are not influenced much by their initializations. Importantly, looking ahead for longer steps ends up with larger initial learning rate α, and much smaller learning rate decay exponent β. The loss surface exhibit distinct landscapes, and the final chosen β differs by two orders of magnitude between 100 and 20k steps. If we use the optimal hyperparameters for 100 steps, the resulting training loss at 20k steps is three orders of magnitude larger than 20k steps.

3.3 Online Meta-Optimization

In this section, we study whether online adaptation also has a short-horizon bias. Different from the previous section, here we use Algorithm 1) to adapt the learning rate and momentum parameters during each stage of the regular training. We experimented with both MLP on MNIST and CNN on CIFAR-10 (Krizhevsky, 2009).

Experimental details. The MLP network layer size is [784-100-100-10], with ReLU activations. Weights are initialized with zero- mean Gaussian distribution with standard deviation 0.1. The CNN network is the CIFAR-10 network adapted from (Jia et al., 2014), with channel size [3, 32, 32, 64], 2×2 max pooling after every convolution layer, ReLU activations, and fully connected layers of size [1024, 100, 10]. The models were pretrained to 1000 steps using the best manually tuned learning rate and momentum parameters. 100 Adam meta steps is applied every 10 steps of regular training. We fixed the learning rate decay exponent to be zero, and applied manual decay in the middle of the training such that all runs end up with effective learning rate 1e-4.

Figure 5 shows online SMD training curves compared to manually chosen SGD with momentum. Online SMD training converge to similar learning rate values despite different initializations. SMD experiments obtain similar learning rate schedules to the greedy noisy quadatic schedules in Figure 2. The results can be explained by our noisy quadratic example: SMD tend to lower the learning rate aggressively in the beginning to obtain large decrease in high curvature directions, but later cannot move very far on small curvature direction. It does not choose to increase the learning rate afterwards since otherwise it will increase loss on high curvature directions again.

4 Conclusion

In this paper we look at the short-horizon bias in the meta-optimization objective. We try to understand the problem by analyzing a noisy quadratic model. We first derive the greedy learning rate and momentum analytically, and compared the performance on with the optimal learning rate found by gradient descent. We find that the greedy schedule starts with a very low learning rate, to achieve instant decrease in high curvature directions, but it can no longer optimize the losses on the low curvature directions. The optimal schedule, on the other hand, maintains a high learning rate in the beginning, so the losses on the low curvature directions decrease significantly. Due to this progress on the low curvature directions, the optimal schedule eventually reaches a much better solution. We also showed that if we have a sufficient second order method for preconditioning, the greedy optimal learning rate is indeed optimal. Moreover, we find the bias comes from the noise in the objective. For this reason, the greedy optimal learning rate and momentum is optimal for deterministic objective. We find the same effect of short-horizon bias in larger datasets, such as MNIST/CIFAR.

References

Shun Ichi Amari. Natural gradient works efficiently in learning. *Neural Computation*, 10(2):251–276, 1998. ISSN 0899-7667. doi: 10.1162/089976698300017746. URL http://dx.doi.org/10.1162/089976698300017746.

Marcin Andrychowicz, Misha Denil, Sergio Gomez Colmenarejo, Matthew W. Hoffman, David Pfau, Tom Schaul, and Nando de Freitas. Learning to learn by gradient descent by gradient descent. In *Advances in Neural Information Processing Systems 29: Annual Conference on Neural Information Processing Systems 2016, December 5-10, 2016, Barcelona, Spain*, pp. 3981–3989, 2016.

Léon Bottou. On-line learning in neural networks. chapter On-line Learning and Stochastic Approximations, pp. 9–42. Cambridge University Press, New York, NY, USA, 1998. ISBN 0-521-65263-4.

Sergey Levine Chelsea Finn, Pieter Abbeel. Model-agnostic meta-learning for fast adaptation of deep networks. In *ICML*, 2017.

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

Priya Goyal, Piotr Dollar, Ross Girshick, Pieter Noordhuis, Lukasz Wesolowski, Aapo Kyrola, Andrew Tulloch, Yangqing Jia, and Kaiming He. Accurate, large minibatch SGD: Training imagenet in 1 hour. Technical report, FAIR, 2017.

Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Deep residual learning for image recognition. In *2016 IEEE Conference on Computer Vision and Pattern Recognition, CVPR 2016, Las Vegas, NV, USA, June 27-30, 2016*, pp. 770–778, 2016.

Yangqing Jia, Evan Shelhamer, Jeff Donahue, Sergey Karayev, Jonathan Long, Ross Girshick, Sergio Guadarrama, and Trevor Darrell. Caffe: Convolutional architecture for fast feature embedding. In *Proceedings of the 22Nd ACM International Conference on Multimedia*, MM '14, pp. 675–678, New York, NY, USA, 2014. ACM. ISBN 978-1-4503-3063-3. doi: 10.1145/2647868.2654889. URL http://doi.acm.org/10.1145/2647868.2654889.

Diederik P. Kingma and Jimmy Ba. Adam: A method for stochastic optimization. In *3th International Conference on Learning Representations*, 2015.

P. W. Koh and P. Liang. Understanding black-box predictions via influence functions. In *International Conference on Machine Learning (ICML)*, 2017.

Alex Krizhevsky. Learning multiple layers of features from tiny images. Technical report, 2009.

Brenden M. Lake, Ruslan Salakhutdinov, Jason Gross, and Joshua B. Tenenbaum. One shot learningof simple visual concepts. In *In Proceedings of the 33th Annual Meeting of the Cognitive Science Society*, 2011.

Yann LeCun, Leon Bottou, Yoshua Bengio, and Patrick Haffner. Gradient-based learning applied to document recognition. *Proceedings of the IEEE*, 86(11):2278–2324, Nov 1998. ISSN 0018-9219. doi: 10.1109/5.726791.

Ke Li and Jitendra Malik. Learning to optimize. In *5th International Conference on Learning Representations*, 2017.

Ren-cang Li. Sharpness in rates of convergence for CG and symmetric lanczos methods. Technical report, 2005.

David J. C. MacKay. A practical bayesian framework for backpropagation networks. *Neural Comput.*, 4(3):448–472, May 1992. ISSN 0899-7667. doi: 10.1162/neco.1992.4.3.448. URL http://dx.doi.org/10.1162/neco.1992.4.3.448.

Dougal Maclaurin, David Duvenaud, and Ryan P. Adams. Gradient-based hyperparameter optimization through reversible learning. In *Proceedings of the 32nd International Conference on Machine Learning*, July 2015.

James Martens. Deep learning via Hessian-free optimization. In *ICML-10*, 2010.

James Martens and Roger Grosse. Optimizing neural networks with kronecker-factored approximate curvature. In *ICML*, 2015.

Barak A. Pearlmutter. Fast exact multiplication by the hessian. *Neural Computation*, 6(1):147–160, January 1994. ISSN 0899-7667. doi: 10.1162/neco.1994.6.1.147.

Olga Russakovsky, Jia Deng, Hao Su, Jonathan Krause, Sanjeev Satheesh, Sean Ma, Zhiheng Huang, Andrej Karpathy, Aditya Khosla, Michael S. Bernstein, Alexander C. Berg, and Fei-Fei Li. Imagenet large scale visual recognition challenge. *International Journal of Computer Vision*, 115(3):211–252, 2015.

Tom Schaul, Sixin Zhang, and Yann LeCun. No more pesky learning rates. In *Proceedings of the 30th International Conference on Machine Learning, ICML 2013, Atlanta, GA, USA, 16-21 June 2013*, pp. 343–351, 2013.

Nicol N. Schraudolph. Local gain adaptation in stochastic gradient descent. In *1999 Ninth International Conference on Artificial Neural Networks ICANN 99. (Conf. Publ. No. 470)*, volume 2, pp. 569–574 vol.2, 1999. doi: 10.1049/cp:19991170.

Karen Simonyan and Andrew Zisserman. Very deep convolutional networks for large-scale image recognition. In *3th International Conference on Learning Representations*, 2015.

L. N. Smith. Cyclical learning rates for training neural networks. In *2017 IEEE Winter Conference on Applications of Computer Vision (WACV)*, pp. 464–472, March 2017. doi: 10.1109/WACV.2017.58.

Ilya Sutskever, James Martens, George Dahl, and Geoffrey Hinton. 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, pp. III–1139–III–1147. JMLR.org, 2013. URL http://dl.acm.org/citation.cfm?id=3042817.3043064.

Christian Szegedy, Wei Liu, Yangqing Jia, Pierre Sermanet, Scott E. Reed, Dragomir Anguelov, Dumitru Erhan, Vincent Vanhoucke, and Andrew Rabinovich. Going deeper with convolutions. *CoRR*, abs/1409.4842, 2014.

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

Max Welling and Yee Whye Teh. Bayesian learning via stochastic gradient langevin dynamics. In *Proceedings of the 28th International Conference on Machine Learning, ICML 2011, Bellevue, Washington, USA, June 28 - July 2, 2011*, pp. 681–688, 2011.

P. J. Werbos. Backpropagation: past and future. *IEEE 1988 International Conference on Neural Networks*, pp. 343–353 vol.1, 1988.

A PROOFS OF THEOREMS

A.1 MODEL DYNAMICS

The stochastic gradient descent with momentum is defined as follows,

$$
\begin{aligned}
v^{(t+1)} &= \mu^{(t)} v^{(t)} - \alpha^{(t)}(h\theta_t + h\sigma\xi), \quad \xi \sim \mathcal{N}(0,1) \\
\theta^{(t+1)} &= \theta^{(t)} + v^{(t+1)}
\end{aligned}
$$

A.1.1 DYNAMICS OF THE EXPECTATION

We calculate the mean of the velocity $v^{(t+1)}$,

$$
\begin{aligned}
\mathbb{E}[v^{(t+1)}] &= \mathbb{E}[\mu^{(t)} v^{(t)} - \alpha^{(t)} h\theta_t] \\
&= \mu^{(t)}\mathbb{E}[v^{(t)}] - (\alpha^{(t)} h)\mathbb{E}[\theta_t]
\end{aligned} \tag{7}
$$

We calculate the mean of the parameter $\theta^{(t+1)}$,

$$
\mathbb{E}[\theta^{(t+1)}] = \mathbb{E}[\theta^{(t)}] + \mathbb{E}[v^{(t+1)}] \tag{8}
$$

Let's assume the following initial conditions:

$$
\begin{aligned}
\mathbb{E}[v_0] &= 0 \\
\mathbb{E}[\theta_0] &= E_0
\end{aligned}
$$

Then Eq.(7) and Eq.(8) describes how $\mathbb{E}[\theta_t]$ changes over time t.

A.1.2 DYNAMICS OF THE VARIANCE

We calculate the variance of the velocity $v^{(t+1)}$,

$$
\begin{aligned}
\mathbb{V}[v^{(t+1)}] &= \mathbb{V}[\mu^{(t)} v^{(t)} - \alpha^{(t)} h\theta_t] + (\alpha^{(t)} h\sigma)^2 \\
&= \mathbb{V}[\mu^{(t)} v^{(t)} - \alpha^{(t)} h\theta_t] + (\alpha^{(t)} h\sigma)^2 \\
&= (\mu^{(t)})^2 \mathbb{V}[v^{(t)}] + (\alpha^{(t)} h)^2 \mathbb{V}[\theta_t] - 2\mu^{(t)}\alpha^{(t)} h \cdot \mathrm{Cov}(\theta_t, v^{(t)}) + (\alpha^{(t)} h\sigma)^2
\end{aligned} \tag{9}
$$

The variance of the parameter $\theta^{(t+1)}$ is given by,

$$
\mathbb{V}[\theta^{(t+1)}] = \mathbb{V}[\theta^{(t)}] + \mathbb{V}[v^{(t+1)}] + 2\mathrm{Cov}(\theta^{(t)}, v^{(t+1)}) \tag{10}
$$

We also need to derive how the covariance of θ and m changes over time:

$$
\begin{aligned}
\mathrm{Cov}(\theta^{(t)}, v^{(t+1)}) &= \mathrm{Cov}(\theta^{(t)}, \mu^{(t)} v^{(t)} - \alpha^{(t)}(h\theta_t + h\sigma\xi)) \\
&= \mu^{(t)}\mathrm{Cov}(\theta^{(t)}, v^{(t)}) - \alpha^{(t)} h\mathbb{V}[\theta_t]
\end{aligned} \tag{11}
$$

$$
\begin{aligned}
\mathrm{Cov}(\theta^{(t)}, v^{(t)}) &= \mathrm{Cov}((\theta^{(t-1)} + v^{(t)}), v^{(t)}) \\
&= \mathrm{Cov}(\theta^{(t-1)}, v_t) + \mathbb{V}[v_t] \\
&= \mu^{(t-1)}\mathrm{Cov}(\theta^{(t-1)}, v^{(t-1)}) - \alpha^{(t-1)} h\mathbb{V}[\theta^{(t-1)}] + \mathbb{V}[v_t]
\end{aligned} \tag{12}
$$

Let's assume the following initial conditions:

$$\begin{aligned}\mathbb{V}[v_0] &= 0\\ \mathbb{V}[\theta_0] &= V_0\\ \mathrm{Cov}[\theta_0, v_0] &= 0\end{aligned}$$

Combining Eq.(9-12), we obtain the following dynamics (from $t = 0, \ldots, T$):

$$\begin{aligned}\mathbb{V}[v^{(t+1)}] &= (\mu^{(t)})^2\mathbb{V}[v^{(t)}] + (\alpha^{(t)}h)^2\mathbb{V}[\theta_t] - 2\mu^{(t)}\alpha^{(t)}h \cdot \mathrm{Cov}(\theta_t, v^{(t)}) + (\alpha^{(t)}h\sigma)^2\\ \mathbb{V}[\theta^{(t+1)}] &= \mathbb{V}[\theta^{(t)}] + \mathbb{V}[v^{(t+1)}] + 2(\mu^{(t)}\mathrm{Cov}(\theta^{(t)}, v^{(t)}) - \alpha^{(t)}h\mathbb{V}[\theta_t])\\ \mathrm{Cov}(\theta^{(t+1)}, v^{(t+1)}) &= \mu^{(t)}\mathrm{Cov}(\theta^{(t)}, v^{(t)}) - \alpha^{(t)}h\mathbb{V}[\theta_t] + \mathbb{V}[v^{(t+1)}]\end{aligned}$$

A.2 Greedy learning rate and momentum

A.2.1 Univariate case

The loss at time step t is,

$$\begin{aligned}\mathcal{L}^{(t+1)} &= \frac{1}{2}h(\mathbb{E}[\theta^{(t+1)}]^2 + \mathbb{V}[\theta^{(t+1)}])\\ &= \frac{1}{2}h\Big[(\mathbb{E}[\theta^{(t)}] + \mu^{(t)}\mathbb{E}[v^{(t)}] - (\alpha^{(t)}h)\mathbb{E}[\theta_t])^2 + \mathbb{V}[\theta^{(t)}] + (\mu^{(t)})^2\mathbb{V}[v^{(t)}] + (\alpha^{(t)}h)^2\mathbb{V}[\theta_t]\\ &- 2\mu^{(t)}\alpha^{(t)}h \cdot \mathrm{Cov}(\theta_t, v^{(t)}) + (\alpha^{(t)}h\sigma)^2 + 2(\mu^{(t)}\mathrm{Cov}(\theta^{(t)}, v^{(t)}) - \alpha^{(t)}h\mathbb{V}[\theta_t])\Big]\\ &= \frac{1}{2}h\Big[((1-\alpha^{(t)}h)\mathbb{E}[\theta^{(t)}] + \mu^{(t)}\mathbb{E}[v^{(t)}])^2 + (1-\alpha^{(t)}h)^2\mathbb{V}[\theta^{(t)}] + (\mu^{(t)})^2\mathbb{V}[v^{(t)}]\\ &+ 2\mu^{(t)}(1-\alpha^{(t)}h)\mathrm{Cov}(\theta^{(t)}, v^{(t)}) + (\alpha^{(t)}h\sigma)^2\Big]\\ &= \frac{1}{2}h\Big[(1-\alpha^{(t)}h)^2\left(\mathbb{E}[\theta^{(t)}]^2 + \mathbb{V}[\theta^{(t)}]\right) + (\mu^{(t)})^2\left(\mathbb{E}[v^{(t)}]^2 + \mathbb{V}[v^{(t)}]\right)\\ &+ 2\mu^{(t)}(1-\alpha^{(t)}h)\left(\mathbb{E}[\theta^{(t)}]\mathbb{E}[v^{(t)}] + \mathrm{Cov}(\theta^{(t)}, v^{(t)})\right) + (\alpha^{(t)}h\sigma)^2\Big]\end{aligned}$$

For simplicity, we denote $A(\cdot) = \mathbb{E}[\cdot]^2 + \mathbb{V}[\cdot]$, and notice that $\mathbb{E}[\theta^{(t)}v^{(t)}] = \mathbb{E}[\theta^{(t)}]\mathbb{E}[v^{(t)}] + \mathrm{Cov}(\theta^{(t)}, v^{(t)})$, hence,

$$\mathcal{L}^{(t+1)} = \frac{1}{2}h\Big[(1-\alpha^{(t)}h)^2A(\theta^{(t)}) + (\mu^{(t)})^2A(v^{(t)}) + 2\mu^{(t)}(1-\alpha^{(t)}h)\mathbb{E}[\theta^{(t)}v^{(t)}] + (\alpha^{(t)}h\sigma)^2\Big] \quad (13)$$

In order to find the optimal learning rate and momentum for minimizing $\mathcal{L}^{(t+1)}$, we take the derivative with respect to $\alpha^{(t)}$ and $\mu^{(t)}$, and set it to 0:

$$\begin{aligned}\nabla_{\alpha^{(t)}}\mathcal{L}^{(t+1)} &= (1-\alpha^{(t)}h)A(\theta^{(t)}) \cdot (-h) - \mu^{(t)}h\mathbb{E}[\theta^{(t)}v^{(t)}] + \alpha^{(t)}(h\sigma)^2 = 0\\ \alpha^{(t)}h(A(\theta^{(t)}) + \sigma^2) &= A(\theta^{(t)}) + \mu^{(t)}\mathbb{E}[\theta^{(t)}v^{(t)}]\\ \nabla_{\mu^{(t)}}\mathcal{L}^{(t+1)} &= \mu^{(t)}A(v^{(t)}) + (1-\alpha^{(t)}h)\mathbb{E}[\theta^{(t)}v^{(t)}] = 0\\ \mu^{(t)*} &= -\frac{(1-\alpha^{(t)}h)\mathbb{E}[\theta^{(t)}v^{(t)}]}{A(v^{(t)})}\\ \alpha^{(t)}h(A(\theta^{(t)}) + \sigma^2) &= A(\theta^{(t)}) - \frac{(1-\alpha^{(t)}h)\mathbb{E}[\theta^{(t)}v^{(t)}]}{A(v^{(t)})}\mathbb{E}[\theta^{(t)}v^{(t)}]\\ \alpha^{(t)}h\left(A(v^{(t)})(A(\theta^{(t)}) + \sigma^2) - \mathbb{E}[\theta^{(t)}v^{(t)}]^2\right) &= A(\theta^{(t)})A(v^{(t)}) - \mathbb{E}[\theta^{(t)}v^{(t)}]^2\\ \alpha^{(t)*} &= \frac{A(\theta^{(t)})A(v^{(t)}) - \mathbb{E}[\theta^{(t)}v^{(t)}]^2}{h\left(A(v^{(t)})(A(\theta^{(t)}) + \sigma^2) - \mathbb{E}[\theta^{(t)}v^{(t)}]^2\right)}\end{aligned}$$

A.2.2 HIGH DIMENSION CASE

The loss is the sum of losses on each direction:

$$\mathcal{L}^{(t+1)} = \sum_i \frac{1}{2} h_i \Big[(1-\alpha^{(t)} h_i)^2 A(\theta_i^{(t)}) + (\mu^{(t)})^2 A(v_i^{(t)}) + 2\mu^{(t)}(1-\alpha^{(t)} h_i)\mathbb{E}[\theta_i^{(t)} v_i^{(t)}] + (\alpha^{(t)} h_i \sigma_i)^2 \Big]$$

Now we obtain optimal learning rate and momentum by setting the derivative to 0,

$$\nabla_{\alpha^{(t)}} \mathcal{L}^{(t+1)} = \sum_i h_i \Big[(1-\alpha^{(t)} h_i) A(\theta_i^{(t)}) \cdot (-h_i) - \mu^{(t)} h_i \mathbb{E}[\theta_i^{(t)} v_i^{(t)}] + \alpha^{(t)} (h_i \sigma_i)^2 \Big] = 0$$

$$\alpha^{(t)} \sum_i \Big((h_i)^3 (A(\theta_i^{(t)}) + (\sigma_i)^2) \Big) = \sum_i \Big((h_i)^2 A(\theta_i^{(t)}) + \mu^{(t)} (h_i)^2 \mathbb{E}[\theta_i^{(t)} v_i^{(t)}] \Big)$$

$$\nabla_{\mu^{(t)}} \mathcal{L}^{(t+1)} = \sum_i h_i \mu^{(t)} A(v_i^{(t)}) + h_i (1-\alpha^{(t)} h_i) \mathbb{E}[\theta_i^{(t)} v_i^{(t)}] = 0$$

$$\mu^{(t)*} = -\frac{\sum_i h_i (1-\alpha^{(t)} h_i) \mathbb{E}[\theta_i^{(t)} v_i^{(t)}]}{\sum_i h_i A(v_i^{(t)})}$$

$$\alpha^{(t)} \sum_i \Big((h_i)^3 (A(\theta_i^{(t)}) + (\sigma_i)^2) \Big) \cdot \Big(\sum_j h_j A(v_j^{(t)}) \Big) =$$

$$\left(\sum_i (h_i)^2 A(\theta_i^{(t)}) \Big(\sum_j h_j A(v_j^{(t)}) \Big) \right) - \left(\sum_i \Big(\sum_j h_j (1-\alpha^{(t)} h_j) \mathbb{E}[\theta_j^{(t)} v_j^{(t)}] \Big) (h_i)^2 \mathbb{E}[\theta_i^{(t)} v_i^{(t)}] \right)$$

$$\alpha^{(t)} \sum_i \left(\Big((h_i)^3 (A(\theta_i^{(t)}) + (\sigma_i)^2) \Big) \Big(\sum_j h_j A(v_j^{(t)}) \Big) - \Big(\sum_j (h_j)^2 \mathbb{E}[\theta_j^{(t)} v_j^{(t)}] \Big) (h_i)^2 \mathbb{E}[\theta_i^{(t)} v_i^{(t)}] \right) =$$

$$\sum_i \left((h_i)^2 A(\theta_i^{(t)}) \Big(\sum_j h_j A(v_j^{(t)}) \Big) - \Big(\sum_j h_j \mathbb{E}[\theta_j^{(t)} v_j^{(t)}] \Big) (h_i)^2 \mathbb{E}[\theta_i^{(t)} v_i^{(t)}] \right)$$

$$\alpha^{(t)*} = \frac{\sum_i \Big((h_i)^2 A(\theta_i^{(t)}) \Big(\sum_j h_j A(v_j^{(t)}) \Big) - \Big(\sum_j h_j \mathbb{E}[\theta_j^{(t)} v_j^{(t)}] \Big) (h_i)^2 \mathbb{E}[\theta_i^{(t)} v_i^{(t)}] \Big)}{\sum_i \Big(\Big((h_i)^3 (A(\theta_i^{(t)}) + (\sigma_i)^2) \Big) \Big(\sum_j h_j A(v_j^{(t)}) \Big) - \Big(\sum_j (h_j)^2 \mathbb{E}[\theta_j^{(t)} v_j^{(t)}] \Big) (h_i)^2 \mathbb{E}[\theta_i^{(t)} v_i^{(t)}] \Big)}$$

A.3 COORDINATE-SPECIFIC OPTIMALITY IN SGD

We now consider a dynamic programming approach to solve the problem. We formalize the optimization problem of $\{\alpha_i\}$ as follows. We first denote $f^{(t)}$ as the minimum expected loss at times step T (i.e., under the optimal learning rate) as a function of the mean $\mathbb{E}[\theta^{(t)}]$ and variance $\mathbb{V}[\theta^{(t)}]$ of $\theta^{(t)}$ at step t.

$$f^{(t)}(\mathbb{E}[\theta^{(t)}], \mathbb{V}[\theta^{(t)}]) = \min_{\alpha^{(t)}, \alpha^{(t+1)}, \ldots, \alpha^{(T-1)}} \mathbb{E}_{\xi^{(t)}, \xi^{(t+1)}, \ldots, \xi^{(T-1)}} [\mathcal{L}(\theta^{(T)})]$$

Because we can express the $\mathbb{E}[\theta^{(t)}]$ and $\mathbb{V}[\theta^{(t)}]$ in terms of the $\mathbb{E}[\theta^{(t-1)}]$ and $\mathbb{V}[\theta^{(t-1)}]$ of $\theta^{(t-1)}$ and the learning rate $\alpha^{(t-1)}$ at the previous time step,

$$\theta^{(t)} = (1-\alpha^{(t-1)} h)\theta^{(t-1)} + \alpha^{(t-1)} h \sigma \xi^{(t-1)}$$

$$\Rightarrow (\mathbb{E}[\theta^{(t)}], (\mathbb{V}[\theta^{(t)}])^2) = ((1-\alpha^{(t-1)} h)\mathbb{E}[\theta^{(t-1)}], ((1-\alpha^{(t-1)} h)\mathbb{V}[\theta^{(t-1)}]^2 + (\alpha^{(t-1)} h \sigma)^2),$$

we can solve the minimization problem recursively by,

$$f^{(t)}(\mathbb{E}[\theta^{(t-1)}], \mathbb{V}[\theta^{(t-1)}] = \min_{\alpha^{(t-1)}} f^{(t)}(\mathbb{E}[\theta^{(t-1)}], \mathbb{V}[\theta^{(t-1)}], \alpha^{(t-1)}).$$

Now, let's first derive the form of f for $T, T-1, T-2$ for illustration. Let $A^{(t)} = (\mathbb{E}[\theta^{(t)}])^2 + (\mathbb{V}[\theta^{(t)}])^2$. We can find a recurrence relation in terms of A:

$$A^{(t)} = (1-\alpha^{(t-1)} h)^2 ((\mathbb{E}[\theta^{(t-1)}])^2 + \mathbb{V}[\theta^{(t-1)}]^2) + (\alpha^{(t-1)} h \sigma)^2 = (1-\alpha^{(t-1)} h)^2 A^{(t-1)} + (\alpha^{(t-1)} h \sigma)^2$$

Since f depends on $\mathbb{E}[\theta]$ and σ only through A, so we instead write f as a function of A:

$$f^{(t)}(A^{(T)}) = \frac{1}{2}hA^{(T)} + \sigma^2$$

Now since $A^{(T)}$ is a function of $\alpha^{(T-1)}$, if we take the derivative of $f^{(t)}$ w.r.t. $\alpha^{(T-1)}$ and setting it to zero, we get:

$$\begin{aligned}
\frac{df^{(t)}}{d\alpha^{(T-1)}} = \frac{1}{2}h\frac{dA^{(T-1)}}{d\alpha^{(T-2)}} &= 0 \\
\Rightarrow \quad \frac{dA^{(T)}}{d\alpha^{(T-1)}} &= 0 \\
\Rightarrow \quad \alpha^{(T-1)*} &= \frac{A^{(T-1)}}{h(A^{(T-1)} + \sigma^2)}
\end{aligned}$$

Thus we can write $A^{(T)}$ in terms of $A^{(T-1)}$ and the optimal $\alpha^{(T-1)*}$:

$$\begin{aligned}
A^{(T)}(A^{(T-1)}, \alpha^{(T-1)*}) &= (1 - \frac{A^{(T-1)}}{A^{(T-1)} + \sigma^2})^2 A^{(T-1)} + (\frac{A^{(T-1)}}{A^{(T-1)} + \sigma^2}\sigma)^2 \\
&= (\frac{\sigma^2}{A^{(T-1)} + \sigma^2})^2 A^{(T-1)} + (\frac{A^{(T-1)}}{A^{(T-1)} + \sigma^2}\sigma)^2 \\
&= \frac{A^{(T-1)}\sigma^2}{A^{(T-1)} + \sigma^2}.
\end{aligned}$$

Therefore,

$$\begin{aligned}
f^{(T-1)}(A^{(T-1)}) &= \frac{1}{2}hA^{(T)} + \sigma^2 \\
&= \frac{1}{2}h(\frac{A^{(T-1)}\sigma^2}{A^{(T-1)} + \sigma^2}) + \sigma^2
\end{aligned}$$

Now since $A^{(T-1)}$ is a function of $\alpha^{(T-2)}$, if we take the derivative of $f^{(T-1)}$ w.r.t. $\alpha^{(T-2)}$ and setting it to zero, we get:

$$\begin{aligned}
\frac{df^{(T-1)}}{d\alpha^{(T-2)}} = \frac{1}{2}\frac{h\sigma^4 \frac{dA^{(T-1)}}{d\alpha^{(T-2)}}}{(A^{(T-1)} + \sigma^2)^2} &= 0 \\
\Rightarrow \quad \frac{dA^{(T-1)}}{d\alpha^{(T-2)}} &= 0 \\
\Rightarrow \quad \alpha^{(T-2)*} &= \frac{A^{(T-2)}}{h(A^{(T-2)} + \sigma^2)} \\
\Rightarrow \quad f^{(T-2)}(A^{(T-2)}) &= \frac{1}{2}h(\frac{A^{(T-2)}\sigma^2}{2A^{(T-2)} + \sigma^2}) + \sigma^2
\end{aligned}$$

Theorem 4. *For all $T \in \mathbb{N}$, and $k \in \mathbb{N}$, $1 \leq k \leq T$, we have,*

$$f^{(t)}(A^{(T-k)}) = \frac{1}{2}h(\frac{A^{(T-k)}\sigma^2}{kA^{(T-k)} + \sigma^2}) + \sigma^2. \tag{14}$$

Therefore, the optimal learning $\alpha^{(t)}$ at timestep t is given as,

$$\alpha^{(t)*} = \frac{A^{(t)}}{h(A^{(t)} + \sigma^2)} \tag{15}$$

Proof. The form of f_{T-k} can be easily proven by induction on k, and use the identity that,

$$\frac{(\frac{ab}{a+b})b}{k(\frac{ab}{a+b}) + b} = \frac{ab}{(k+1)a + b}.$$

The learning rate then follows immediately by taking the derivative of f_{T-k} w.r.t. $\alpha^{(T-k-1)}$ and setting it to zero. □