Self-Tuning Stochastic Optimization with Curvature-Aware Gradient Filtering

Ricky T. Q. Chen[*†] **Dami Choi**[*†] **Lukas Balles**[*‡] **David Duvenaud**[†] **Philipp Hennig**[‡]

Abstract

Standard first-order stochastic optimization algorithms base their updates solely on the average mini-batch gradient, and it has been shown that tracking additional quantities such as the curvature can help de-sensitize common hyperparameters. Based on this intuition, we explore the use of exact per-sample Hessian-vector products and gradients to construct optimizers that are self-tuning and hyperparameter-free. Based on a dynamics model of the gradient, we derive a process which leads to a curvature-corrected, noise-adaptive online gradient estimate. The smoothness of our updates makes it more amenable to simple step size selection schemes, which we also base off of our estimates quantities. We prove that our model-based procedure converges in the noisy quadratic setting. Though we do not see similar gains in deep learning tasks, we can match the performance of well-tuned optimizers and ultimately, this is an interesting step for constructing self-tuning optimizers.

1 Introduction

Stochastic gradient-based optimization is plagued by the presence of numerous hyperparameters. While these can often be set to rule-of-thumb constants or manually-designed schedules, it is also common belief that a more information regarding the optimization landscape can help present alternative strategies such that manual tuning has less of an impact on the end result. For instance, the use of curvature information in the form of Hessian matrices or Fisher information can be used to de-sensitize or completely remove step size parameter (Ypma, 1995; Amari, 1998; Martens, 2014), and the momentum coefficient can be set to reduce the local gradient variance (Arnold et al., 2019b).

Based on these intuitions, we investigate the use of efficient curvature and variance estimates during training to construct a *self-tuning* optimization framework. Under a Bayesian paradigm, we treat the true gradient as the unobserved state of a dynamical system and seek to automatically infer the true gradient conditioned on the history of parameter updates and stochastic gradient observations.

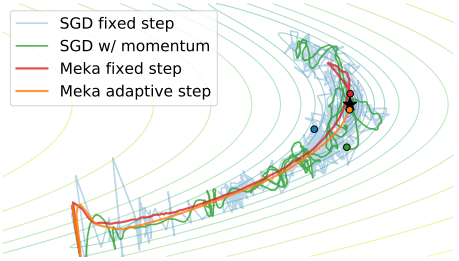


Figure 1: Stochastic gradient eventually goes into diffusion and does not converge. Our filtered gradients offer smooth convergence and complements adaptive step sizes.

Our method is enabled by evaluations of exact *per-sample* gradients and Hessian-vector products. With recent improvements in automatic differentiation tooling (*e.g.*, Bradbury et al., 2018; Agarwal and Ganichev, 2019; Dangel et al., 2020), this matches the asymptotic time cost of minibatch gradient and Hessian-vector product evaluations.

[*]Equal contribution.
[†]University of Toronto. Vector Institute. {rtqichen, choidami, duvenaud}@cs.toronto.edu
[‡]Max Planck Institute for Intelligent Systems, Tübingen, Germany. {lballes, ph}@tue.mpg.de

1st I Can't Believe It's Not Better Workshop (ICBINB@NeurIPS 2020), Vancouver, Canada.

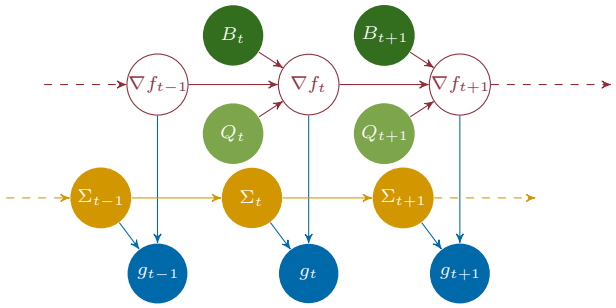


Figure 2: Graphical model of the hidden Markov dynamics model. The main idea of our algorithm is that the dynamics parameters can be cheaply estimated on each minibatch, and smoothed across time using exact Kalman filter inference. These dynamics parameters are the gradient variance Σ, the directional curvature $B\delta$ and its variance Q. We stabilize Σ with an exponential moving average, which is effectively another, more elementary form of Kalman filtering.

While our framework contains the good properties of both curvature-based updates and variance reduction—which are attested in toy and synthetic scenarios—we do not observe significant improvements empirically in optimizing deep neural networks. Notably, our approach can be viewed as an explicit form of the implicit gradient transport of Arnold et al. (2019b), yet it does not achieve the same acceleration empirically observed in practice. While we do not fully understand this behavior, we analyze the estimated quantites along the training trajectory and hypothesize that our method has a higher tendency of going down high-variance high-curvature regions whereas standard stochastic gradient descent is repelled from such regions due to gradient variance. This potentially serves as a downside of our method in the deep learning setting. Regardless, the use of efficient variance estimation and interpretation of gradient estimation as Bayesian filtering are useful constructs in the development of self-tuning stochastic optimization.

2 Bayesian Filtering for Stochastic Gradients

We consider stochastic optimization problems of the general form

$$\arg\min_{\theta \in \mathbb{R}^d} f(\theta), \quad f(\theta) = \mathbb{E}_\xi \left[\tilde{f}(\theta, \xi) \right] \tag{1}$$

where we only have access to samples ξ. Stochastic gradient descent—the prototypical algorithm for this setting—iteratively updates $\theta_{t+1} = \theta_t - \alpha_t g_t$, where

$$g_t = \frac{1}{n} \sum_{i=1}^{n} \nabla_\theta \tilde{f}(\theta_t, \xi_t^{(i)}), \quad \xi_t^{(1)}, \ldots, \xi_t^{(n)} \overset{\text{iid}}{\sim} p(\xi), \tag{2}$$

and α_t is a scalar step size. We may use notational shorthands like $f_t = f(\theta_t)$, $\nabla f_t = \nabla f(\theta_t)$.

SGD is hampered by the effects of gradient noise. It famously needs a decreasing step size schedule to converge; used with a constant step size, it goes into diffusion in a region around the optimum (see, *e.g.*, Bottou et al., 2018). Gradient noise also makes stochastic optimization algorithms difficult to tune. In particular, unreliable directions are not amenable to step size adaptation.

To stabilize update directions, we build a framework for estimating the true gradient ∇f based on Kalman filtering. This can also be viewed as a variance reduction method, but does not require the typical finite-sum structure assumption of *e.g.* Schmidt et al. (2017); Johnson and Zhang (2013).

2.1 Dynamical System Model

We treat the true gradient ∇f_t as the latent state of a dynamical system. This dynamical system is comprised of an observation model $p(g_t \mid f_t)$ and a dynamics model $p(\nabla f_t \mid \nabla f_{t-1}, \delta_{t-1})$ where $\delta_{t-1} = \theta_t - \theta_{t-1}$ is the update direction. We will later choose δ_t to be depend on our variance-reduced gradient estimates, but the gradient inference framework itself is agnostic to the choice of δ_t.

The observation model $p(g_t \mid \nabla f_t)$ describes how the gradient observations relate to the state of the dynamical system. In our case, it is relatively straight-forward, since g_t is simply an unbiased stochastic estimate of ∇f_t, but the exact distribution remains to be specified. We make the assumption that g_t follows a Gaussian distribution,

$$g_t \mid \nabla f_t \sim \mathcal{N}(\nabla f_t, \Sigma_t), \tag{3}$$

with covariance Σ_t. Since g_t is the mean of iid terms (Eq. 2), this assumption is supported by the central limit theorem when sufficiently large batch sizes are used.

The dynamics model $p(\nabla f_t \mid \nabla f_{t-1})$ describes how the gradient evolves between iterations. We base our dynamics model on a first order Taylor expansion of the gradient function **centered at $\boldsymbol{\theta_t}$**, $\nabla f(\theta_{t-1}) \approx \nabla f(\theta_t) - \nabla^2 f(\theta_t)\delta_{t-1}$. We propose to approximate the gradient dynamics by computing a stochastic estimate of the Hessian-vector product, $B_t\delta_{t-1}$, where $\mathbb{E}[B_t] = \nabla^2 f(\theta_t)$. Again, we make a Gaussian noise assumption. This implies the dynamics model

$$\nabla f_t \mid \nabla f_{t-1} \sim \mathcal{N}(\nabla f_{t-1} + B_t\delta_{t-1}, Q_t). \tag{4}$$

where Q_t is the covariance of $B_t\delta_{t-1}$, taking into account the stochasticity in B_t.

A key insight is that the parameters $B_t\delta_{t-1}, Q_t, \Sigma_t$ of the model can all be "observed" directly using automatic differentiation of the loss on each minibatch of samples. We use the Hessian at θ_t so that the Hessian-vector product can be simultaneously computed with g_t with just one extra call to automatic differentiation (or "backward pass") in each iteration (note this does not require constructing the full matrix B_t). The variances Q_t and Σ_t can also be empirically estimated with some memory overhead by using auto-vectorized automatic differentiation routines. We discuss implementation details later in Section 4.

2.2 Filtering Framework for Gradient Inference

As Equations (3) and (4) define a linear-Gaussian dynamical system, exact inference on the true gradient conditioned on the history of gradient observations $p(\nabla f_t | g_{1:t}, \delta_{1:t-1})$ takes the form of the well-known Kalman filtering equations (Kalman, 1960) (review in Särkkä, 2013): We define parameters m_t^-, m_t, P_t^- and P_t such that

$$\begin{aligned} \nabla f_t \mid g_{1:t-1}, \delta_{1:t-1} &\sim \mathcal{N}(m_t^-,\ P_t^-) \\ \nabla f_t \mid g_{1:t}, \delta_{1:t-1} &\sim \mathcal{N}(m_t,\ P_t). \end{aligned} \tag{5}$$

Starting from a prior belief $\nabla f_0 \sim \mathcal{N}(m_0, P_0)$, these parameters are updated iteratively:

$$m_t^- = m_{t-1} + B_t\delta_{t-1}, \qquad P_t^- = P_{t-1} + Q_{t-1} \tag{6}$$

$$K_t = P_t^-(P_t^- + \Sigma_t)^{-1} \tag{7}$$

$$m_t = (I - K_t)m_t^- + K_t g_t, \qquad P_t = (I - K_t)P_t^-(I - K_t)^T + K_t\Sigma_t K_t^T \tag{8}$$

Equation (6) is referred to as the *prediction* step as it computes mean and covariance of the predictive distribution $p(\nabla f_t | g_{1:t-1})$. In our setting, it predicts the gradient ∇f_t based on our estimate of the previous gradient (m_{t-1}) and the Hessian-vector product approximating the change in gradient from the step $\theta_t = \theta_{t-1} + \delta_{t-1}$. Equation (8) is the *correction* step. Here, the local stochastic gradient evaluation g_t is used to correct the prediction. Importantly, the *Kalman gain* (7) determines the blend between the prediction and the observations according to the uncertainty in each.

The resulting algorithm gives an online estimation of the true gradients as the parameters θ_t are updated. We refer to this framework as MEKA, loosely based on *model-based Kalman-adjusted gradient estimation*. During optimization, we may use the posterior mean m_t as a variance-reduced gradient estimator and take steps in the direction of $\delta_t = -\alpha_t m_t$.

We note two key insights enabling MEKA: First, all parameters of the filter are not set *ad hoc*, but are directly evaluated or estimated using automatic differentiation. Secondly, the dynamics model makes explicit use of the Hessian to predict gradients. This is a first-order update. In contrast to second-order methods, like quasi-Newton methods, MEKA does not try to estimate the Hessian from gradients, but instead leverages a (noisy) projection with the actual Hessian to improve gradient estimates. This is both cheaper and more robust than second-order methods, because it does not involve solving a linear system.

2.3 ADAM-style Update Directions

While MEKA produces variance-reduced gradient estimates, it does not help with ill-conditioned optimization problems, a case where full batch gradient descent can perform poorly. To alleviate this, we may instead take update directions motivated by the ADAGRAD (Duchi et al., 2011) line of optimizers. We follow ADAM (Kingma and Ba, 2014) which proposes dividing the first moment of the gradient element-wise by the square root of the second moment, to arrive at

$$\delta_t = -\alpha_t \frac{m_t}{\sqrt{m_t + \text{diag}(P_t)} + \varepsilon}. \tag{9}$$

where ε is taken for numerical stability and simply set to 10^{-8}. Whereas ADAM makes use of two exponential moving averages to estimate the first and second moments of g_t, we have estimates automatically inferred through the filtering framework. We refer to this variant as ADAMEKA.

3 Uncertainty-informed Step Size Selection

We can adopt a similar Bayesian filtering framework for probabilistic step size adaptation. Our step size adaptation will be a simple enhancement to the quadratic rule, but takes into account uncertainty in the stochastic regime and is much more robust to stochastic observations. The standard quadratic rule if the objective f can be computed exactly is $\alpha_{\text{quadratic}} := \frac{-\delta_t^T \nabla f_t}{\delta_t^T \nabla^2 f_{t-1} \delta_t}$, which is based on minimizing a local quadratic approximation $f(\theta_t + \alpha_t \delta_t) - f_t \approx \alpha \delta_t^T \nabla f_t + \frac{\alpha^2}{2} \delta_t^T \nabla^2 f_{t-1} \delta_t$.

However, since we only have access to stochastic estimates of ∇f and $\nabla^2 f$, naïvely taking this step size with high variance samples results in unpredictable behavior and can cause divergence during optimization. To compensate for the stochasticity and inaccuracy of a quadratic approximation, adaptive step size approaches often include a "damping" term (*e.g.* Martens (2010))—where a constant is added to the denominator—and an additional scaling factor on α_t, both of which aim to avoid large steps but introduces more hyperparameters.

As an alternative, we propose a scheme that uses the variance of the estimates to adapt the step size, only taking steps into regions where we are confident about minimizing the objective function. Once again leveraging the availability of Q_t and Σ_t, our approach allows automatic trade-off between minimizing a local quadratic approximation and the uncertainty over large step sizes, foregoing manual tuning methods such as damping.

We adopt a similar linear-Gaussian dynamics model for tracking the true objective f_t, with the same assumptions as in Section 2. Due to its similarity with Section 2, we delegate the derivations to Appendix B. We again define the posterior distribution,

$$f_t \mid y_{1:t}, \delta_{1:t-1} \sim \mathcal{N}(u_t, s_t). \tag{10}$$

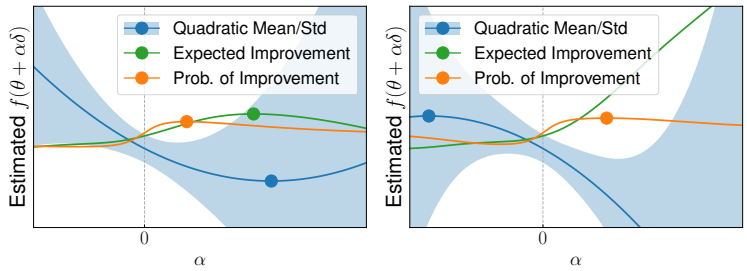
where u_t and s_t are inferred using the Kalman update equations. Finally, setting $f_{t+1} = f(\theta_t + \alpha_t \delta_t)$ for some direction δ_t, we have a predictive model of the change in function value as

$$f_{t+1} - f_t \mid y_{1:t}, g_{1:t}, \delta_{1:t} \sim \mathcal{N}\left(\alpha_t \delta_t^T m_t + \frac{\alpha_t^2}{2} \delta_t^T B_t \delta_t, 2s_t + \alpha_t^2 \delta_t^T P_t \delta_t + \frac{\alpha_t^4}{4} \delta_t^T Q_t \delta_t\right) \tag{11}$$

Contrasting this with the simple quadratic approximation, the main difference is now we take into account the uncertainty in f_t, ∇f_t, and $\nabla^2 f_t$. Each term makes different contributions to the variance as α_t increases, corresponding to different trade-offs between staying near where we are more certain about the function value and exploring regions we believe have a lower function value. Explicitly specifying this trade-off gives an *acquisition function*. These decision rules are typically used in the context of Bayesian optimization (Shahriari et al., 2016), but we adopt their use for step size selection.

3.1 Acquisition Functions for Step Size Selection

Computing the optimal step size in the context of a long but finite sequence of optimization steps is intractable in general, but many reasonable heuristics have been developed. These heuristics usually balance immediate progress against information gathering likely to be useful for later steps.



(a) Positive curvature (b) Negative curvature

Figure 3: Illustration of different acquisition functions for selecting a step size α, based on the mean and variance of our local quadratic estimate of the loss surface.

One natural and hyperparameter-free heuristic is maximizing the *probability of improvement* (PI) (Kushner, 1964),

$$\alpha_{\text{PI}} := \arg\max_{\alpha} \mathbb{P}\left(f_{t+1} - f_t \leq 0 \mid y_{1:t}, g_{1:t}\right) \tag{12}$$

which is simply the cumulative distribution function of (11) evaluated at zero.

Figure 3 visualizes the different step sizes chosen by maximizing different acquisition functions. The heuristic of choosing the minimum of the quadratic approximation can be a poor decision when the uncertainty rises quickly. The optimum for PI interpolates between zero and the quadratic minimum in such a way that avoids regions of high uncertainty. Expected improvement (Jones et al., 1998) is another popular acquisition function; however, in tests we found it to not be as robust as PI and often results in step sizes that require additional scaling.

Maximizing probability of improvement is equivalent to the following optimization problem

$$\alpha_{\text{PI}} = \arg\min_{\alpha} \frac{-\alpha\delta_t^T m_t + \frac{\alpha^2}{2}\delta_t^T B_t \delta_t}{\sqrt{2s_t + \alpha^2 \delta_t^T P_t \delta_t + \frac{\alpha^4}{4}\delta_t^T Q_t \delta_t}}. \tag{13}$$

We numerically solve for α_{PI} using Newton's method, which itself is only a small overhead since we only optimize in one variable with fixed constants: no further evaluations of f are required. We also note that there is exactly one optimum for $\alpha \in \mathbb{R}^+$. For optimization problems where negative curvature is a significant concern, we include a third-order correction term that ensures finite and positive step sizes (details in Appendix B.2).

4 A Practical Implementation

While the above derivations have principled motivations and are free of hyperparameters, a practical implementation of MEKA is not entirely straightforward. Below we discuss some technical aspects, simplifications and design choices that increase stability in practice, as well as recent software advances that simplify the computation of quantities of interest.

Computing Per-Example Quantities for Estimating Variance Recent extensions for automatic differentiation in the machine learning software stack (Bradbury et al., 2018; Agarwal and Ganichev, 2019; Dangel et al., 2020) implement an automatic vectorization `map` function. Vectorizing over minibatch elements allows efficient computation of gradients and Hessian-vector products of neural network parameters with respect to each data sample independently. These advances allow efficient computation of the empirical variances of gradients and Hessian-vector products, and enable our filtering-based approach to gradient estimation.

Stabilizing Filter Estimates Instead of working with the full covariance matrices Σ_t and Q_t, we approximate them as scalar objects $\sigma_t I$ and $q_t I$, with $\sigma_t, q_t \in \mathbb{R}_+$ by averaging over all dimensions. We have experimented with diagonal matrices, but found that the scalar form increases stability, generally performing better on our benchmarks. Furthermore, we use an exponential moving average for smoothing the estimated gradient variance σ_t as well as the adaptive step sizes α_t. The

coefficients of these exponential moving average are kept at 0.999 in our experiments and seem to be quite insensitive, with values in $\{0.9, 0.99, 0.999\}$ all performing near identically (see Appendix F).

5 Related Work

Designing algorithms that can self-tune its own parameters is a central theme in optimization (Eiben and Smit, 2011; Yang et al., 2013); we focus on the stochastic setting, building on and merging ideas from several research directions. The Bayesian filtering framework itself has previously been applied to stochastic optimization. To the best of our knowledge, the idea goes back to Bittner and Pronzato (2004) who used a filtering approach to devise an automatic stopping criterion for stochastic gradient methods. Patel (2016) proposed filtering-based optimization methods for large-scale linear regression problems. Vuckovic (2018) and Mahsereci (2018) used Kalman filters on general stochastic optimization problems with the goal of reducing the variance of gradient estimates. In contrast to our work, none of these existing approaches leverage evaluations of Hessian-vector products to give curvature-informed dynamics for the gradient.

In terms of online variance reduction, Gower et al. (2017) have discussed the use of Hessian-vector products to correct the gradient estimate; however, they propose methods that approximate the Hessian whereas we compute exact Hessian-vector products by automatic differentiation. Arnold et al. (2019a) recently proposed an implicit gradient transport formula analogous to our dynamics model, but they require a rather strong assumption that the Hessian is the same for all samples and parameter values. In contrast, we focus on explicitly transporting via the full Hessian. This allows us to stay within the filtering framework and automatically infer the gain parameter, whereas the implicit formulation of Arnold et al. (2019a) requires the use of a manually-tuned averaging schedule.

Step size selection under noisy observations is a difficult problem and has been tackled from multiple viewpoints. Methods include meta-learning approaches (Almeida et al., 1999; Schraudolph, 1999; Plagianakos et al., 2001; Yu et al., 2006; Baydin et al., 2017) or by assuming the interpolation regime (Vaswani et al., 2019; Berrada et al., 2019). Rolinek and Martius (2018) proposed extending a linear approximation to adapt step sizes but introduces multiple hyperparameters to adjust for the presence of noise, whereas we extend a quadratic approximation and automatically infer parameters based on noise estimates. Taking into account observation noise, Mahsereci and Hennig (2017) proposed a probabilistic line search that is done by fitting a Gaussian process to the optimization landscape. However, inference in Gaussian processes is more costly than our filtering approach.

6 Convergence in the Noisy Quadratic Setting

As a motivating example, consider a simple toy problem, where

$$f(\theta, \xi) = \frac{1}{2}(\theta - \xi)^T H(\theta - \xi), \tag{14}$$

i.e., a mixture of quadratic functions with identical Hessian but varying location determined by the "data" ξ. The full gradient is $\nabla f(\theta) = H(\theta - \mathbb{E}[\xi])$ and per-example gradients evaluate to $\nabla f(\theta, \xi) = H(\theta - \xi) = \nabla f(\theta) - H(\xi - \mathbb{E}[\xi])$. Hence, we have additive gradient noise with covariance $\Sigma = H\mathbf{Cov}[\xi]H^T$ independent of θ. Moreover, since the Hessian $\nabla^2 f(\theta, \xi) = H$ is independent of ξ, we have that $B_t\delta_{t-1} \equiv \nabla f_t - \nabla f_{t-1}$. The covariance Q_t is zero and the filter equations simplify to

$$\begin{aligned} K_t &= P_{t-1}(P_{t-1} + \Sigma)^{-1}, \\ m_t &= (I - K_t)(m_{t-1} + B_t\delta_{t-1}) + K_t g_t, \\ P_t &= (I - K_t)P_{t-1}, \end{aligned} \tag{15}$$

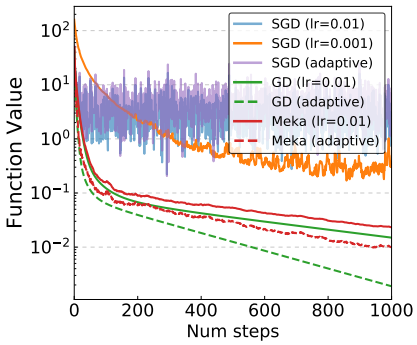


Figure 4: Filtered gradients converge with a fixed step size in the noisy quadratic regime, whereas SGD results in diffusion for the same step size.

initialized with $m_0 = g_0$, $P_0 = \Sigma$. The filter covariance P_t contracts in every step and in fact, shrinks at a rate of $O(1/t)$, meaning that the filter will narrow in on the exact gradient. We show that this enables $O(1/t)$ convergence with a *constant* step size.

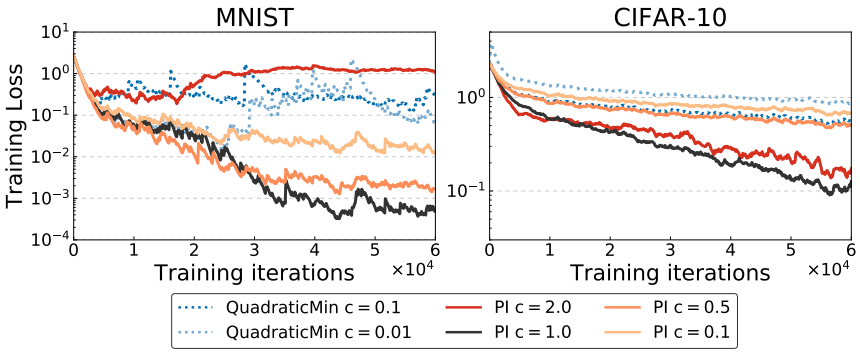


Figure 6: Adaptive step sizes based on probability of improvement work best without any additional scaling factor c for modifying the update rule: $\theta_{t+1} = \theta_t + c\alpha_t\delta_t$.

Proposition 1. *Assume a problem of the form* (14) *with* $\mu I \preceq H \preceq LI$. *If we update* $\theta_{t+1} = \theta_t - \alpha m_t$ *with* $\alpha \leq 1/L$ *and* m_t *obtained via Eq.* (15)*, then* $\mathbb{E}[f(\theta_t) - f_*] \in O(1/t)$.

Figure 4 shows experimental results for such a noisy quadratic problem of dimension $d = 20$ with a randomly-generated Hessian (with condition number > 1000) and $\xi \sim \mathcal{N}(0, I)$. Using SGD with a high learning rate simply results in diffusion, and setting the learning rate smaller results in slow convergence. Gradient descent (GD) converges nicely with the high learning rate, and using adaptive steps sizes leads to a better convergence rate. The filtered gradients from MEKA converge almost as well as gradient descent, and adaptive step sizes provide an improvement. On the other hand, SGD produces unreliable gradient directions and does not work well with adaptive step sizes. We note that the stochastic gradient has a full covariance matrix and does not match our modeling assumptions, as our model uses a diagonal covariance for efficiency. Even so, the training loss of MEKA follows that of gradient descent very closely after just a few iterations.

7 Classification Experiments

Next we test and diagnose our approach on classification benchmarks, MNIST and CIFAR-10. We use JAX's (Bradbury et al., 2018) vectorized map functionality for efficient per-example gradients and Hessian-vector products. For MNIST, we test using a multi-layer perceptron (MLP); for CIFAR-10, a convolutional neural network (CNN) and a residual network (ResNet-32) (He et al., 2016a,b). One key distinction is we replace the batch normalization layers with group normalization (Wu and He, 2018) as batch-dependent transformations conflict with our assumption that the gradient samples are independent. We note that the empirical per-iteration cost of MEKA is 1.0–1.6× that of SGD due to the computation of Hessian-vector products. Full experiment details are provided in Appendix E. A detailed comparison to tuned baseline optimizers is presented in Appendix D.1.

Online Variance Reduction We test whether the filtering procedure is correctly aligning the gradient estimate with the true gradient. For this, we use CIFAR-10 with a CNN and no data augmentation, so that the true full-batch gradient over the entire dataset can be computed. Figure 5 shows the L_2 norm difference between the gradient estimators and the full-batch gradient ∇f_t. MEKA's estimated gradients are closer to the true around by around a factor of 5 compared to the minibatch gradient sample.

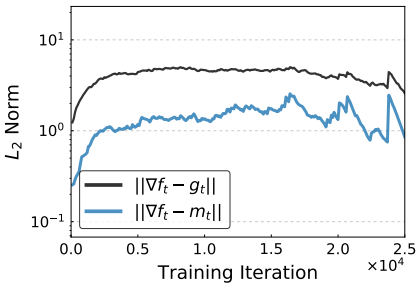


Figure 5: MEKA's estimated gradients are closer to the true full-batch gradient in L_2 norm than stochastically observed gradients by a factor of around 5.

Adaptive Step Sizes are Appropriately Scaled Without uncertainty quantification, the quadratic minimum step size scheme tends to result in step sizes too large. As such, one may include a scaling factor such that the update is modified as $\theta_{t+1} = \theta_t - c\alpha_t\delta_t$. In contrast, we find that the adaptive step sizes based on probability of improve-

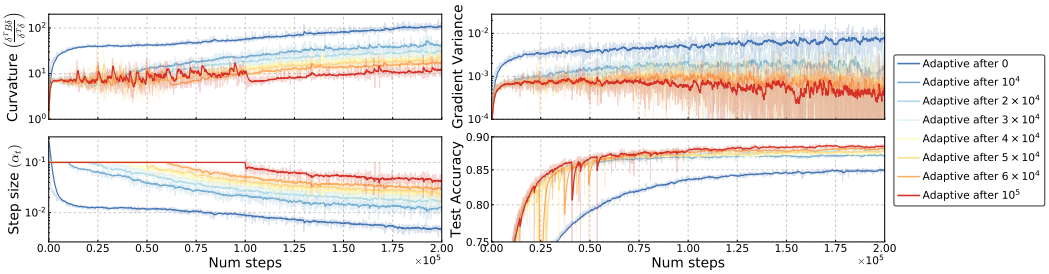


Figure 7: The performance of MEKA with adaptive step sizes on ResNet-32 can be explained by quantities captured during optimization. MEKA reaches high-curvature high-variance local minima, as soon as adaptive step sizes are used.

ment (PI) are already correctly scaled in the sense that a c different from 1.0 will generally result in worse performance. Figure 6 shows a comparison of different values for c for the quadratic and PI (12) adaptive schemes. We plot expected improvement in Appendix D, which performs poorly and requires non-unit scaling factors.

7.1 Adaptive Step Sizes Dives into High-curvature, High-variance Regions

A core aspect of our filtering approach is the ability to estimate quantities of interest during optimization. We now use these to help understand the loss landscape of ResNet-32 on CIFAR-10. We find that a cause for slow convergence of MEKA with adaptive step sizes is due to an abundance of minima that are usually too high variance for standard SGD.

Figure 7 shows estimates of the normalized curvature along the descent direction $\frac{\delta^T B_t \delta}{\delta^T \delta}$ as well as the per-sample gradient variance, averaged over parameters. To understand the loss landscape along the trajectory of optimization, we use multiple runs of MEKA with the same initialization. Each run takes a different fixed number of constant-size steps before switching to the adaptive step size scheme.

It is clear that immediately after switching to adaptive step sizes, MEKA falls into an increasingly high curvature region and remains there. The gradient variance also remains high. As our optimization procedure can handle relatively high variance and curvature, it proceeds to optimize within this sharp but potentally non-local minimum. On the other hand, it may be an advantage of fixed-step-size SGD that it skips over both high-variance and high-curvature minima.

This failing of adaptive step sizes during the initial phase of training may be related to the "short horizon bias" (Wu et al., 2018) of our one-step-ahead acquisition function. If so, compute budget can be used to approximate multi-step-ahead gains to help reduce this bias. Additionally, the ability to optimize within high-curvature high-variance regions could potentially be an advantage on problems with fewer local minima, yet this may not be the case for deep learning.

8 Conclusion

We introduced an online gradient estimation framework for stochastic gradient-based optimization, which leverages Hessian-vector products and variance estimates to perform automatic online gradient estimation and step size selection. The result is a stochastic optimization algorithm that can self-tune many important parameters such as momentum and learning rate schedules, in an online fashion without checkpointing or expensive outer-loop optimization.

While the required additional observables can be computed efficiently with recent advances in automatic differentiation tooling, they are of course not free, increasing computational cost and memory usage compared to SGD. What one gains in return is automation, so that it suffices to run the algorithm just once, without tedious tuning. Given the amount of human effort and computational resources currently invested into hyperparameter tuning, we believe our contributions are valuable steps towards fully-automated gradient-based optimization.

References

Ashish Agarwal and Igor Ganichev. Auto-vectorizing TensorFlow graphs: Jacobians, auto-batching and beyond. *arXiv preprint arXiv:1903.04243*, 2019.

Luís B Almeida, Thibault Langlois, José D Amaral, and Alexander Plakhov. Parameter adaptation in stochastic optimization. In *On-line learning in neural networks*, pages 111–134. 1999.

Shun-Ichi Amari. Natural gradient works efficiently in learning. *Neural computation*, 10(2):251–276, 1998.

Sébastien Arnold, Pierre-Antoine Manzagol, Reza Babanezhad Harikandeh, Ioannis Mitliagkas, and Nicolas Le Roux. Reducing the variance in online optimization by transporting past gradients. In *Advances in Neural Information Processing Systems 32*. 2019a.

Sébastien Arnold, Pierre-Antoine Manzagol, Reza Babanezhad Harikandeh, Ioannis Mitliagkas, and Nicolas Le Roux. Reducing the variance in online optimization by transporting past gradients. In *Advances in Neural Information Processing Systems*, pages 5391–5402, 2019b.

Atilim Gunes Baydin, Robert Cornish, David Martinez Rubio, Mark Schmidt, and Frank Wood. Online learning rate adaptation with hypergradient descent. *arXiv preprint arXiv:1703.04782*, 2017.

Leonard Berrada, Andrew Zisserman, and M Pawan Kumar. Training neural networks for and by interpolation. *arXiv preprint arXiv:1906.05661*, 2019.

Barbara Bittner and Luc Pronzato. Kalman filtering in stochastic gradient algorithms: construction of a stopping rule. In *2004 IEEE International Conference on Acoustics, Speech, and Signal Processing*, 2004.

Léon Bottou, Frank E Curtis, and Jorge Nocedal. Optimization methods for large-scale machine learning. *Siam Review*, 60(2):223–311, 2018.

James Bradbury, Roy Frostig, Peter Hawkins, Matthew James Johnson, Chris Leary, Dougal Maclaurin, and Skye Wanderman-Milne. JAX: composable transformations of Python+NumPy programs, 2018. URL `http://github.com/google/jax`.

Dami Choi, Christopher J Shallue, Zachary Nado, Jaehoon Lee, Chris J Maddison, and George E Dahl. On empirical comparisons of optimizers for deep learning. *arXiv preprint arXiv:1910.05446*, 2019.

Felix Dangel, Frederik Kunstner, and Philipp Hennig. BackPACK: Packing more into backprop. In *International Conference on Learning Representations*, 2020.

John Duchi, Elad Hazan, and Yoram Singer. Adaptive subgradient methods for online learning and stochastic optimization. *Journal of machine learning research*, 2011.

Agoston E Eiben and Selmar K Smit. Parameter tuning for configuring and analyzing evolutionary algorithms. *Swarm and Evolutionary Computation*, 1(1):19–31, 2011.

Robert M Gower, Nicolas Le Roux, and Francis Bach. Tracking the gradients using the hessian: A new look at variance reducing stochastic methods. *arXiv preprint arXiv:1710.07462*, 2017.

Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Deep residual learning for image recognition. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, 2016a.

Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Identity mappings in deep residual networks. In *European conference on computer vision*, pages 630–645. Springer, 2016b.

Rie Johnson and Tong Zhang. Accelerating stochastic gradient descent using predictive variance reduction. In *Advances in neural information processing systems*, pages 315–323, 2013.

Donald R Jones, Matthias Schonlau, and William J Welch. Efficient global optimization of expensive black-box functions. *Journal of Global optimization*, 1998.

Rudolph Emil Kalman. A new approach to linear filtering and prediction problems. *Transactions of the ASME–Journal of Basic Engineering*, 82(Series D):35–45, 1960.

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

Harold J Kushner. A new method of locating the maximum point of an arbitrary multipeak curve in the presence of noise. 1964.

Maren Mahsereci. *Probabilistic Approaches to Stochastic Optimization*. PhD thesis, Eberhard Karls Universität Tübingen Tübingen, 2018.

Maren Mahsereci and Philipp Hennig. Probabilistic line searches for stochastic optimization. *The Journal of Machine Learning Research*, 2017.

James Martens. Deep learning via Hessian-free optimization. In *Proceedings of the 27th International Conference on International Conference on Machine Learning*, 2010.

James Martens. New insights and perspectives on the natural gradient method. *arXiv preprint arXiv:1412.1193*, 2014.

Vinod Nair and Geoffrey E Hinton. Rectified linear units improve restricted boltzmann machines. In *Proceedings of the 27th international conference on machine learning (ICML-10)*, pages 807–814, 2010.

Vivak Patel. Kalman-based stochastic gradient method with stop condition and insensitivity to conditioning. *SIAM Journal on Optimization*, 2016.

VP Plagianakos, GD Magoulas, and MN Vrahatis. Learning rate adaptation in stochastic gradient descent. In *Advances in convex analysis and global optimization*, pages 433–444. Springer, 2001.

Boris T Polyak. Some methods of speeding up the convergence of iteration methods. *USSR Computational Mathematics and Mathematical Physics*, 1964.

Michal Rolinek and Georg Martius. L4: Practical loss-based stepsize adaptation for deep learning. In *Advances in Neural Information Processing Systems*, pages 6433–6443, 2018.

Simo Särkkä. *Bayesian filtering and smoothing*, volume 3. Cambridge University Press, 2013.

Mark Schmidt, Nicolas Le Roux, and Francis Bach. Minimizing finite sums with the stochastic average gradient. *Mathematical Programming*, 162(1-2):83–112, 2017.

Frank Schneider, Lukas Balles, and Philipp Hennig. Deepobs: A deep learning optimizer benchmark suite. *arXiv preprint arXiv:1903.05499*, 2019.

Nicol N Schraudolph. Local gain adaptation in stochastic gradient descent. 1999.

B. Shahriari, K. Swersky, Z. Wang, R. P. Adams, and N. de Freitas. Taking the human out of the loop: A review of bayesian optimization. *Proceedings of the IEEE*, 2016.

Sharan Vaswani, Aaron Mishkin, Issam Laradji, Mark Schmidt, Gauthier Gidel, and Simon Lacoste-Julien. Painless stochastic gradient: Interpolation, line-search, and convergence rates. In *Advances in Neural Information Processing Systems 32*. 2019.

James Vuckovic. Kalman gradient descent: Adaptive variance reduction in stochastic optimization, 2018.

Yuhuai Wu, Mengye Ren, Renjie Liao, and Roger Grosse. Understanding short-horizon bias in stochastic meta-optimization. *arXiv preprint arXiv:1803.02021*, 2018.

Yuxin Wu and Kaiming He. Group normalization. In *Proceedings of the European Conference on Computer Vision (ECCV)*, 2018.

Xin-She Yang, Suash Deb, Martin Loomes, and Mehmet Karamanoglu. A framework for self-tuning optimization algorithm. *Neural Computing and Applications*, 23(7-8):2051–2057, 2013.

Tjalling J Ypma. Historical development of the newton–raphson method. *SIAM review*, 37(4): 531–551, 1995.

Jin Yu, Douglas Aberdeen, and Nicol N Schraudolph. Fast online policy gradient learning with smd gain vector adaptation. In *Advances in neural information processing systems*, pages 1185–1192, 2006.

Matthew D Zeiler. Adadelta: an adaptive learning rate method. *arXiv preprint arXiv:1212.5701*, 2012.

A The Full MEKA Algorithm with Adaptive Step Sizes

Algorithm 1 MEKA with adaptive step sizes based on maximizing the probability of improvement.

Hyperparameters: decay rates β_r=0.999, β_Σ=0.999, β_α=0.999
$m_0, P_0 \leftarrow \vec{0}, 10^4$ ▷ *large variance initialization ensures first Kalman gain is one*
$u_0, s_0 \leftarrow 0, 10^4$
$\delta_0 = \vec{0}$
$t \leftarrow 0$
repeat
  $t \leftarrow t + 1$
  $f_t^{(i)}, \nabla f_t^{(i)}, \nabla^2 f_t^{(i)} \delta_{t-1} \leftarrow \text{VectorizedMap}(f, \{x_i\}_{i=1}^M; \theta_{t-1})$ ▷ *compute per-example quantities*
  $y_t, r_t \leftarrow \text{MeanVarEMA}(\{f_t^{(i)}\}; \beta_r)$ ▷ *exponential moving average (EMA) on the variances*
  $g_t, \Sigma_t \leftarrow \text{MeanVarEMA}(\{\nabla f_t^{(i)}\}; \beta_\Sigma)$
  $b_t, Q_t \leftarrow \text{MeanVar}(\{\nabla^2 f_t^{(i)} \delta_{t-1}\})$
  $u_t, s_t \leftarrow \text{FilterUpdate}(u_{t-1}, s_{t-1}; m_{t-1}, P_{t-1}, y_t, r_t, b_t, Q_t)$ ▷ *filter update equations*
  $m_t, P_t \leftarrow \text{FilterUpdate}(m_{t-1}, P_{t-1}; g_t, \Sigma_t, b_t, Q_t)$ ▷ *filter update equations*
  $\alpha_t \leftarrow \arg\min_\alpha (13)$ ▷ *with an EMA (decay rate β_α) on the coefficients*
  $\delta_t \leftarrow \alpha_t m_t$
  $\theta_t \leftarrow \theta_{t-1} - \delta_t$
until convergence

B The Function Value Dynamics Model

We discuss inferring the function value f_t, taking into account uncertainty due to changes in function value and observation noise during optimization. The gradient dynamics (5) imply the following dynamics model for the function value itself:

$$\begin{aligned} f_t \mid f_{t-1} \sim \mathcal{N}(f_{t-1} + m_{t-1}^T \delta_{t-1} + \frac{1}{2}\delta_{t-1}^T B_t \delta_{t-1}, \\ \lambda_t + \delta_{t-1}^T P_{t-1} \delta_{t-1} + \frac{1}{4}\delta_{t-1}^T Q_t \delta_{t-1}) \\ y_t \mid f_t \sim \mathcal{N}(f_t, r_t) \end{aligned} \tag{16}$$

where we again use a quadratic approximation using Taylor expansion. Instead of the intractable ∇f and $\nabla^2 f$, we use the estimates from Section 2. The observations y_t and r_t are the empirical mean and variance of f_t from a minibatch. The variance terms in the dynamics model are due to the uncertainty associated with m_{t-1} and $B_t \delta_{t-1}$.

Here we have included a scalar term λ_t, which acts as a correction to the local quadratic model. This acts similar to a damping component, except we can automatically infer an optimal λ_t by maximizing the likelihood of $p(y_t \mid y_{1:t-1})$, with a closed form solution (see Appendix B.1). While damping terms are usually difficult to set empirically (Choi et al., 2019), we note that including our λ_t term is essentially free, and it automatically decays after optimization stabilizes.

B.1 Adaptively Correcting the Dynamics Model

We construct a dynamics model of the function value as follows (repeated for convenience):

$$\begin{aligned} f_t \mid f_{t-1} \sim \mathcal{N}(f_{t-1} + m_{t-1}^T \delta_{t-1} + \frac{1}{2}\delta_{t-1}^T B_t \delta_{t-1},\ \lambda_t + \delta_{t-1}^T P_{t-1} \delta_{t-1} + \frac{1}{4}\delta_{t-1}^T Q_t \delta_{t-1}) \\ y_t \mid f_t \sim \mathcal{N}(f_t, r_t) \end{aligned} \tag{17}$$

We include a scalar parameter λ_t in case the local quadratic approximation is inaccurate, ie. when y_t is significantly different from the predicted value. If this occurs, a high value of λ_t causes the Kalman gain to become large, throwing away the stale estimate and putting more weight on the new observed function value.

We pick a value for λ_t by maximizing the likelihood of $p(y_t|y_{1:t-1})$. Marginalizing over f_t, we get

$$p(y_t|y_{1:t-1}) = \mathcal{N}\left(\underbrace{u_{t-1} + m_{t-1}^T\delta_{t-1} + \frac{1}{2}\delta_{t-1}^T B_t \delta_{t-1}}_{u_t^-},\ \lambda_t + \underbrace{s_{t-1} + \delta_{t-1}^T P_{t-1}\delta_{t-1} + \frac{1}{4}\delta_{t-1}^T Q_t \delta_{t-1} + r_t}_{c_t}\right) \tag{18}$$

Taking the log and writing it out, we get

$$\log p(y_t|y_{1:t-1}) \propto -\frac{1}{2}\left[\frac{(y_t - u_t^-)^2}{\lambda_t + c_t} + \log(\lambda_t + c_t)\right] \tag{19}$$

and its derivative is

$$-\frac{1}{2}\left[\frac{-(y_t - u_t^-)^2}{(\lambda_t + c_t)^2} + \frac{1}{\lambda_t + c_t}\right] = -\frac{1}{2}\left[\frac{-(y_t - u_t^-)^2 + \lambda_t + c_t}{(\lambda_t + c_t)^2}\right] \tag{20}$$

Setting this to zero, we get

$$\lambda_t = (y_t - u_t^-)^2 - c_t \tag{21}$$

Since the role of λ_t is to ensure we are not overconfident in our predictions, and we don't want to deal with negative variance values, we set

$$\lambda_t^* = \max\{(y_t - u_t^-)^2 - c_t,\ 0\} \tag{22}$$

As can be seen from Figure 8, this λ_t term goes to zero when it is not needed, ie. when the dynamics model is correct, which occurs on MNIST after convergence. It is also a quantity that shows us just how incorrect our dynamics model is, and for the problems we tested, we find that it is significantly smaller than the posterior variance s_t. This suggests that it has minimal impact if removed, but we keep it in the algorithm for cases when a quadratic approximation is not sufficient.

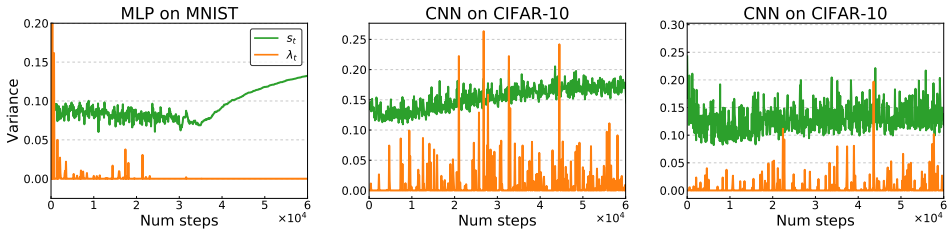
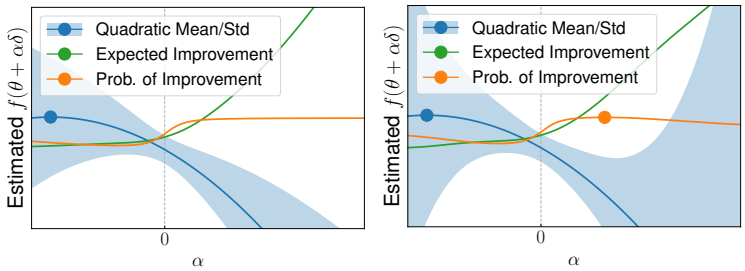


Figure 8: The quantity $2s_t + \lambda_t$ shows up as a constant variance term during step size adaptation. We find that though λ_t has an effect only during a few iterations, it is usually small enough to be ignored. This suggests that the quadratic approximation assumption is okay most of the time. Nevertheless, a self-correcting term that is essentially compute-free is a desirable component.

B.2 Dealing with negative curvature in step size adaptation

When the gradient estimating is pointing in a direction of negative curvature, there is a chance that the optimal step size is infinity (Figure 9a).



(a) Negative curvature. (b) Negative curvature with $\lambda_t||\delta||^6$.

Figure 9: Negative curvature can result in infinite step sizes. An extra correction term to the variance ensures the optimal step size is finite.

This occurs when the variance of $f_{t+1} - f_t$ rises slower than the expectation. To handle this situation, we can add a third order correction term, which appears in the variance as a term that scales with $||\delta||^6$. Using the same procedure as inferring a constant λ_t, we can instead add the term $\lambda_t ||\delta||^6$ to the variance. We then choose λ_t as

$$\lambda_t^* = \max\left\{ \frac{1}{||\delta||^6} \left((y_t - u_t^-)^2 - c_t \right),\ 0 \right\} \tag{23}$$

This extra term (if $\lambda_t > 0$) in the variance ensures that variance increases faster than the expectation. Adaptive step sizes based on the probability of improvement will then have an optimal step size that is finite in value (Figure 9b). An additional damping effect may be added by lower bounding λ_t. We did not fully test this approach as the exponential moving averaged curvature used in practice was always positive for our test problems. Incidentally, Figures 9a and 9b show that the expected improvement is not a good heuristic as it is extremely large even with this extra term. Moreover, the quadratic approximation will result in negative step sizes.

C Using the Current Hessian vs the Previous Hessian

For the dynamics model, we make the choice to use the Hessian at the updated location B_t, which is an unbiased estimate of $\nabla^2 f(\theta_t)$, instead of B_{t-1}, the Hessian at θ_{t-1}. Firstly, we made this choice for computational reasons: it is easier to compute the Hessian at θ_t since we are already computing the gradient g_t evaluated at θ_t. Secondly, we found that using the current Hessian results in better performance and more stability. Figure 10 shows this on MNIST. For CIFAR-10, we found that using the previous Hessian B_{t-1} resulted in immediate divergence and NaNs, so we do not show those plots.

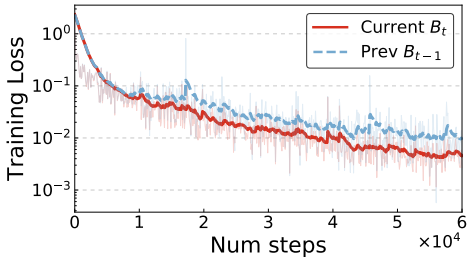


Figure 10: The choice of using current vs previous Hessian on MNIST.

D Comparison of Adaptive Step Size Schemes

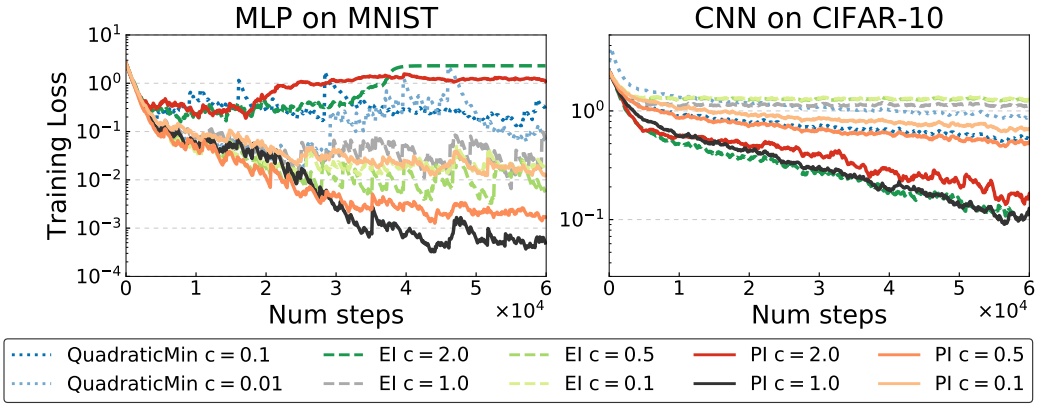


Figure 11: Comparing step sizes with a scaling factor c such that the update is $\theta_t = \theta_{t-1} + c\alpha_t\delta_t$. This comparison includes expected improvement (EI). We note that it performs poorly on MNIST and requires a scaling of 2.0 to match PI on CIFAR-10. Using a smaller scaling factor results in worse performance.

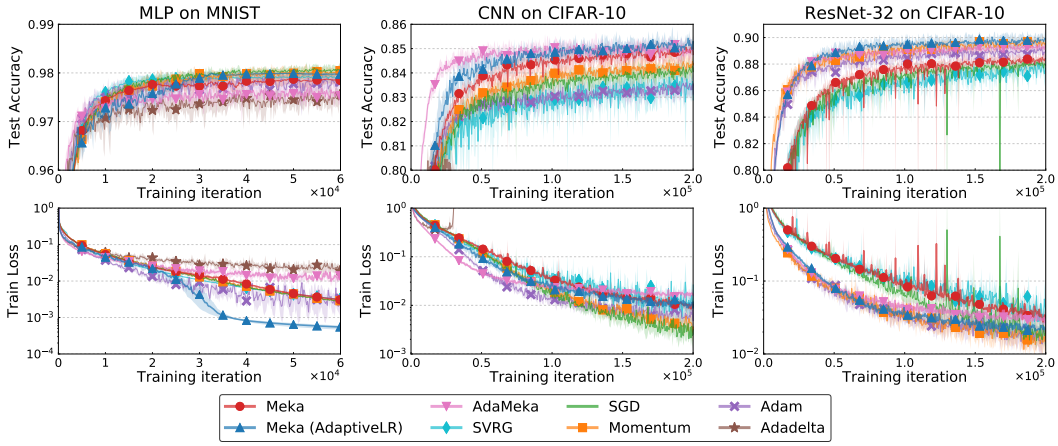


Figure 12: MEKA is competitive with optimizers that have additional tunable hyperparameters. Results are averaged over 5 random seeds; shaded regions are 5th and 95th percentiles.

D.1 Comparison with Tuned Optimizers

We measure the performance of our method against a variety of other approaches. We compare with fixed step size versions of SGD, SGD with momentum (Polyak, 1964), and ADAM (Kingma and Ba, 2014). For these, we tune the step size using grid search. We also compare against ADADELTA (Zeiler, 2012), which is a competing learning rate-free algorithm, and SVRG (Johnson and Zhang, 2013), an optimization algorithm focused around estimating the full batch gradient, using the same step size as the tuned SGD. For comparison, we implemented MEKA using the same constant learning rate as the tuned SGD, and ADAMEKA with the same learning rate as the tuned ADAM. We applied fully adaptive step sizes with MEKA update directions; on ResNet-32, we used ADAMEKA update directions to mitigate poor conditioning.

Figure 12 shows the resulting loss and accuracy curves. As this includes optimizers across a wide range of motivations, we highlight some specific comparisons. MEKA generally performs better than SGD in terms of test accuracy, showing that too much stochasticity can hurt generalization. Though both designed with gradient estimation in mind, MEKA seems to compare favorable against SVRG in terms of performance. We note that the per-iteration costs of MEKA were also cheaper as SVRG requires two gradient evaluations. With the default learning rate of 1.0, ADADELTA performs decently on MNIST but ends up diverging on CIFAR-10. In comparison, our adaptive step sizes converge well and generally outperform fixed step sized MEKA (or ADAMEKA) in both trainng loss and test accuracy.

E Experiment Details

E.1 Dataset Description

We used the official train and test split for MNIST and CIFAR-10. We did not do any data augmentation for MNIST. For CIFAR-10, we normalized the images by subtracting every pixel with the global mean and standard deviation across the training set. In addition to this, unless specified otherwise, we pre-processed the images following He et al. (2016a) by padding the images by 4 pixels on each side and applying random cropping and horizontal flips.

E.2 Architecture Description

All models use the the ReLU (Nair and Hinton, 2010) activation function.

MLP We used an MLP with 1 hidden layer of 100 hidden units.

CNN We used the same architecture as the "3c3d" architecture in Schneider et al. (2019), which consists of 3 convolutional layers with max pooling, followed by 3 fully connected layers. The first

convolutional layer has a kernel size of 5×5 with stride 1, "valid" padding, and 64 filters. The second convolutional layer has a kernel size of 3×3 with stride 1, "valid" padding, and 96 filters. The third convolutional layer has a kernel size of 3×3 with stride 1, "same" padding, and 128 filters. The max pooling layers have a window size of 3×3 with stride 2. The 2 fully connected layers have 512 and 256 units respectively.

ResNet-32 Our ResNet-32 (He et al., 2016a) model uses residual blocks based on He et al. (2016b). We replaced the batch normalization layers with group normalization (Wu and He, 2018) as batch-dependent transformations conflict with our assumption that the gradient samples are independent, and hinder our method to estimate gradient variance.

E.3 Optimizer Comparisons Description

We tuned the step size of SGD in a grid of $\{0.001, 0.01, 0.1, 1.0\}$. We tuned the step size of SGD with momentum, and ADAM in a grid of $\{0.0001, 0.001, 0.01, 0.1\}$. We chose the best step size of SGD for the variant of MEKA with a constant learning rate, and for SVRG.

The chosen step size for SGD was 0.1 for MNIST, and 0.1 for CIFAR-10. For SGD with momentum, the chosen step size was 0.01 for MNIST, 0.1 for ResNet-32 on CIFAR-10, and 0.001 for CNN on CIFAR-10. The best step size for ADAM was 0.001 for MNIST and ResNet-32 on CIFAR-10, and 0.0001 for CNN on CIFAR-10.

For SGD with momentum, the momentum coefficient γ was fixed to 0.9. For ADAM, β_1, β_2, ε were fixed to 0.9, 0.999, and 10^{-8} respectively. For ADADELTA, ρ and ε were fixed to 0.95 and 10^{-6} respectively.

F Sensitivity of MEKA's Hyperparameters

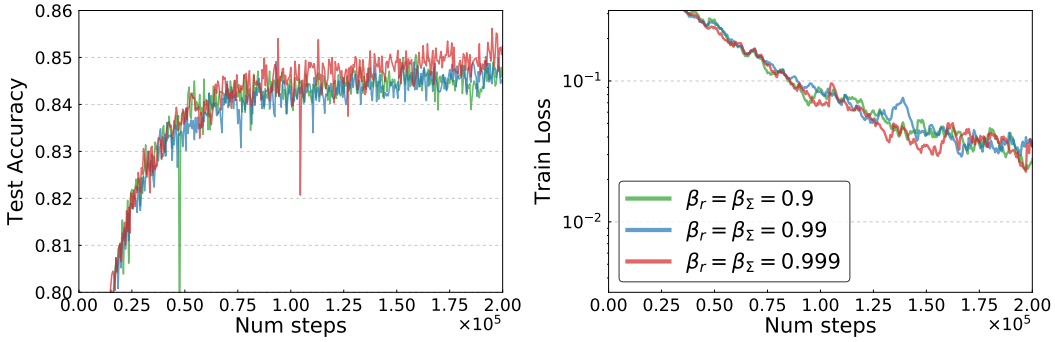


Figure 13: The performance of MEKA with constant learning rate for CNN on CIFAR-10 is not sensitive to the choice of the exponential moving average decay rates β_r and β_Σ.

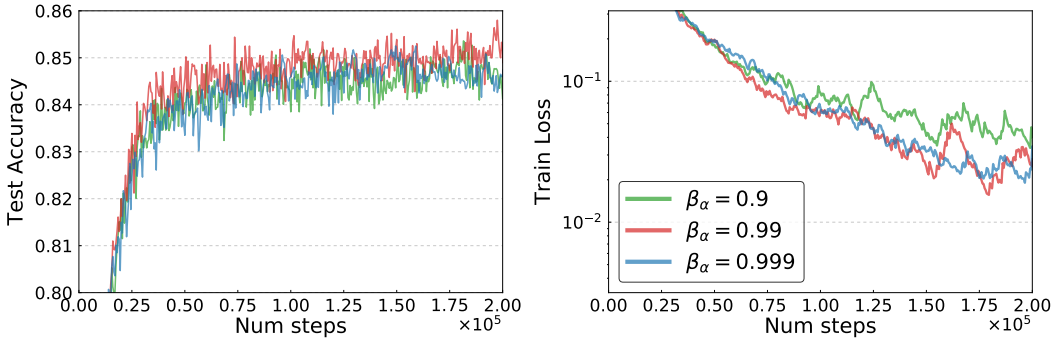


Figure 14: The performance of MEKA with the PI adaptive scheme for CNN on CIFAR-10 is not sensitive to the choice of the exponential moving average decay rate β_α.

G Additional Cost of MEKA

Table 1: The memory cost of using a vectorized map to obtain individual gradients is greater than taking the gradient of a sum over the minibatch, whereas the asymptotic compute cost is the same. B is the batch size, $|\theta|$ is the number of parameters, and D is the number of activations in the model.

	$\nabla \sum_{i=1}^{m} f_t^{(i)}$	VMap$(\nabla, f_t^{(i)})$
Memory	$\mathcal{O}\left(\lvert\theta\rvert + BD\right)$	$\mathcal{O}\left(B(\lvert\theta\rvert + D)\right)$
Compute	$\mathcal{O}\left(BD\lvert\theta\rvert\right)$	$\mathcal{O}\left(BD\lvert\theta\rvert\right)$

Table 2: The ratio of the time it takes to complete one iteration for MEKA versus SGD. Note that in addition to the vector mapped gradients, we also compute an additional Hessian-vector product. The runtimes are after just-in-time compilation of JAX has settled. Runtimes are tested on the NVIDIA TITAN Xp GPU.

Dataset	Architecture	SGD	Meka (fixed lr)	Meka (PI adaptive)
MNIST	MLP	1.00	1.10	1.00
CIFAR-10	CNN	1.00	0.83	1.34
CIFAR-10	ResNet-32	1.00	1.68	2.97

H Proofs

Proof of Proposition 1. A standard Lipschitz bound yields

$$\begin{aligned}\mathbb{E}[f_{t+1}] &\leq \mathbb{E}[f_t] - \alpha\mathbb{E}[\nabla f_t^T m_t] + \frac{L\alpha^2}{2}\mathbb{E}[\|m_t\|^2] \\ &\leq \mathbb{E}[f_t] - \frac{\alpha}{2}\left(\mathbb{E}[2\nabla f_t^T m_t - \|m_t\|^2]\right) \\ &= \mathbb{E}[f_t] - \frac{\alpha}{2}\left(\mathbb{E}[\|\nabla f_t\|^2] - \mathbb{E}[\|m_t - \nabla f_t\|^2]\right).\end{aligned} \tag{24}$$

Using strong convexity ($\|\nabla f_t\|^2 \geq 2\mu(f_t - f_*)$) and subtracting f_* from both sides results in

$$\mathbb{E}[f_{t+1} - f_*] \leq (1 - \alpha\mu)\mathbb{E}[f_t - f_*] + \frac{\alpha}{2}\mathbb{E}[\|m_t - \nabla f_t\|^2] \tag{25}$$

So in each step, we get a multiplicative decrease in the expected function value (left term) but we add a term that depends on the variance of our filtered gradient estimate m_t. So, in essence, to establish convergence, we have to show that $\mathbb{E}[\|m_t - \nabla f_t\|^2]$ decreases to zero sufficiently fast.

Since all assumptions of the Kalman filter are satisfied, we know that $\mathbb{E}[m_t] = \mathbb{E}[\nabla f_t]$ and $\mathbb{E}[(m_t - \nabla f_t)(m_t - \nabla f_t)^T] = P_t$. Hence, $\mathbb{E}[\|m_t - \nabla f_t\|^2] = \operatorname{tr}(P_t)$. We now show inductively that $P_t = \frac{1}{t+1}\Sigma$. This holds for $t = 0$ by construction. Assume it holds for arbitrary but fixed $t - 1$. Then

$$K_t = P_{t-1}(P_{t-1} + \Sigma)^{-1} = \frac{1}{t}\Sigma\left(\frac{1}{t}\Sigma + \Sigma\right)^{-1} = \frac{1}{t}\Sigma\left(\frac{t+1}{t}\Sigma\right)^{-1} = \frac{1}{t+1}I \tag{26}$$

and, thus,

$$P_t = (I - K_t)P_{t-1} = \left(I - \frac{1}{t+1}I\right)\frac{1}{t}\Sigma = \frac{1}{t+1}\Sigma \tag{27}$$

Plugging $\mathbb{E}[\|m_t - \nabla f_t\|^2] = \operatorname{tr}(P_t) = \frac{1}{t+1}\operatorname{tr}(\Sigma)$ back into Eq. (25) and introducing the shorthands $e_t = \mathbb{E}[f_t - f_*]$ and $\sigma^2 := \operatorname{tr}(\Sigma)$ reads

$$e_t \leq (1 - \alpha\mu)e_{t-1} + \frac{\alpha\sigma^2}{2}\frac{1}{t}. \tag{28}$$

Iterating backwards results in

$$e_t \leq (1-\alpha\mu)^t e_0 + \frac{\alpha\sigma^2}{2}\sum_{s=0}^{t-1}\frac{(1-\alpha\mu)^{t-1-s}}{s+1} = (1-\alpha\mu)^t e_0 + \frac{\alpha\sigma^2}{2}\sum_{s=1}^{t}\frac{(1-\alpha\mu)^{t-s}}{s}. \tag{29}$$

Lemma 1 shows that the sum term is $O(1/t)$. The first (exponential) term is trivially $O(1/t)$, which concludes the proof. □

Lemma 1. *Let* $0 < c < 1$ *and define the sequence (for* $t \geq 1$*)*

$$a_t = \sum_{s=1}^{t}\frac{c^{t-s}}{s}.$$

Then $a_t \in O(\frac{1}{t})$.

Proof. Let T be the smallest index such that $c\frac{T+1}{T} < 1$, i.e., $T = \lceil c/(1-c) \rceil$. Define

$$M = \max\left(Ta_T, \left(1 - c\frac{T+1}{T}\right)^{-1}\right) \tag{30}$$

This ensures that $a_T \leq \frac{M}{T}$ and

$$\frac{1}{M} + c\frac{t+1}{t} \leq 1 \tag{31}$$

for all $t \geq T$. We now show inductively that $a_t \leq \frac{M}{t}$ for all $t \geq T$. It holds for $t = T$ by construction of M. Assume it holds for some $t \geq T$. Then

$$\begin{aligned} a_{t+1} &= \sum_{s=1}^{t+1}\frac{c^{t+1-s}}{s} = \frac{1}{t+1} + c\underbrace{\sum_{s=1}^{t}\frac{c^{t-s}}{s}}_{=a_t \leq M/t} \\ &\leq \frac{1}{t+1} + c\frac{M}{t} = \frac{M}{t+1}\underbrace{\left(\frac{1}{M} + c\frac{t+1}{t}\right)}_{\leq 1 \text{ by Eq. (31)}} \leq \frac{M}{t+1}. \end{aligned} \tag{32}$$

□