Variational Stochastic Gradient Descent for Deep Neural Networks

Haotian Chen [*1] **Anna Kuzina** [*2] **Babak Esmaeili** [1] **Jakub M. Tomczak** [1]

Abstract

Optimizing deep neural networks is one of the main tasks in successful deep learning. Current state-of-the-art optimizers are adaptive gradient-based optimization methods such as ADAM. Recently, there has been an increasing interest in formulating gradient-based optimizers in a probabilistic framework for better estimation of gradients and modeling uncertainties. Here, we propose to combine both approaches, resulting in the Variational Stochastic Gradient Descent (VSGD) optimizer. We model gradient updates as a probabilistic model and utilize stochastic variational inference (SVI) to derive an efficient and effective update rule. Further, we show how our VSGD method relates to other adaptive gradient-based optimizers like ADAM. Lastly, we carry out experiments on two image classification datasets and four deep neural network architectures, where we show that VSGD outperforms ADAM and SGD.

1. Introduction

The driving force for deep learning success is efficient and effective optimization (Bottou, 2012; Sun, 2020). Deep Neural Networks (DNNs) introduce multiple optimization challenges due to their complexity, size, and loss landscape with multiple local minima, plateaus, and saddle points. Since the first attempt to train DNNs with stochastic gradient descent (SGD), there have been multiple approaches to speed up the learning process and improve the final performance of DNNs.

Nowadays, adaptive gradient-based methods are the leading optimizers in deep learning. The first advancements arrived with the idea of adding momentum to the SGD update rule, resulting in SGD with momentum (SGDM) (Sutskever et al., 2013). The next major breakthrough was ADAM (Kingma & Ba, 2015), which uses the first and second momenta to adapt the learning rate and gradients. Currently, ADAM is probably the most widely used optimizer in deep learning due to its relative insensitivity to hyperparameter values, and much faster initial progress in training compared to SGD (Sun, 2020).

Recently, there has been an increasing interest in formulating SGD in a probabilistic framework. One example is (Mandt et al., 2017) in which SGD is used as an approximate Bayesian inference algorithm analyzed by relating it to the Ornstein-Uhlbeck process. A different approach utilizes a Bayesian perspective on SGD to model the uncertainty for datastreams (Liu et al., 2024).

In this paper, we propose to combine these two approaches, and, as a result, we obtain a probabilistic framework for adaptive gradient-based optimizers. First, we formulate a probabilistic model of SGD. Second, we employ stochastic variational inference SVI (Hoffman et al., 2013) to derive an efficient adaptive gradient-based update rule. We refer to our approach as Variational Stochastic Gradient Descent (VSGD). We apply our method to optimize overparameterized DNNs on image classification. Compared to ADAM and SGD, we obtain very promising results.

Our contributions can be summarized as follows:

1. We propose Variational Stochastic Gradient Descent (VSGD), a novel optimizer that adopts a probabilistic approach. In VSGD, we model the true gradient and the noisy gradient as latent and observed random variables, respectively, within a probabilistic model. Additionally, this unique perspective allows us to manage gradient noise more effectively by adopting distinct noise models for the true and noisy gradients.

2. We draw connections between VSGD and several established non-probabilistic optimizers, including NORMALIZED-SGD, ADAM, and SGDM. Our analysis reveals that many of these methods can be viewed as specific instances or simplified adaptations of VSGD when a constant noise model is assumed for both the true and observed gradients.

[*]Equal contribution [1]Department of Mathematics and Computer Science, Eindhoven University of Technology, Eindhoven, the Netherlands [2]Department of Computer Science, Vrije Universiteit Amsterdam, Amsterdam, the Netherlands. Correspondence to: Haotian Chen <h.chen5@tue.nl>, Anna Kuzina <a.kuzina@vu.nl>, Babak Esmaeili <b.esmaeili@tue.nl>, Jakub M. Tomczak <j.m.tomczak@tue.nl>.

Accepted to the Workshop on Advancing Neural Network Training at International Conference on Machine Learning (WANT@ICML 2024).

3. Lastly, we carry out an empirical evaluation of VSGD by comparing its performance against the most popular optimizers, namely ADAM and SGD in the context of large-scale image classification tasks using overparameterized deep neural networks. Our results indicate that VSGD not only achieves lower generalization errors, but also converges at a competative rate compared to traditional methods such as ADAM and SGD. We believe that these findings underscore the practical advantages of VSGD, making it a compelling choice for complex deep learning challenges.

2. Background knowledge

Training Neural Nets with Gradient Descent Let us consider a supervised setup for deep neural networks (DNNs) in which a DNN with weights $\theta \in \mathbb{R}^D$ predicts a target variable $y \in \mathcal{Y}$ for a given object $\mathbf{x} \in \mathcal{X}$, $f_\theta : \mathcal{X} \to \mathcal{Y}$. Here, we treat θ as a vector to keep the notation uncluttered. Theoretically, finding *best* values of θ can be achieved by *risk minimization*: $\mathcal{R}(\theta) = \mathbb{E}_{p(\mathbf{x},y)}\left[\ell(f_\theta(\mathbf{x}), y)\right]$, where ℓ is a loss function. In practice, we do not have access to $p(\mathbf{x}, y)$, but we are given a dataset $\mathcal{D}$. Then, training of f_θ corresponds to *empirical risk minimization* (Bottou, 2012; Sun, 2020): $\mathcal{L}(\theta; \mathcal{D}) = \frac{1}{N}\sum_{n=1}^{N} \ell(f_\theta(\mathbf{x}_n), y_n)$. We assume that the loss function is k-differentiable, $\ell \in \mathcal{C}^k$, at least $k = 1$.

Many optimization algorithms for DNNs are based on *gradient descent* (GD). Due to computational restrictions on calculating the full gradient (that is, over all training datapoints) for a DNN, the stochastic version of GD, *stochastic gradient descent* (SGD), is employed. SGD results in the following update rule:

$$\theta_t = \theta_{t-1} - \eta_t \hat{g}_t, \tag{1}$$

where η_t is a learning rate, $\hat{g}_t = \nabla_\theta \mathcal{L}(\theta; M)$ is a noisy version of the full gradient calculated over a mini-batch of M data points, $\mathcal{L}(\theta; M) = \frac{1}{|M|}\sum_{n \in M} \ell(f_\theta(\mathbf{x}_n), y_n)$, and we assume that $\mathbb{E}\left[\hat{g}_t\right] = g_t$.

Related work Optimization of DNNs is a crucial component in deep learning research and, especially, in practical applications. In fact, the success of deep learning models is greatly dependent on the effectiveness of optimizers. The loss landscapes of DNNs typically suffer from multiple local minima, plateaus, and saddle points (Du & Lee, 2018), making the optimization process challenging.

Since the introduction of backpropagation (Rumelhart et al., 1986; Schmidhuber, 2015), which is based on SGD updates, multiple techniques have been proposed to improve SGD. Some improvements correspond to a better initialization of layers that results in a *warm start* of an optimizer (e.g., (Glorot & Bengio, 2010; He et al., 2015; Sutskever et al., 2013)), or pre-defined learning schedules (e.g., (Loshchilov & Hutter, 2016)).

Another active and important line of research focuses on adaptive gradient-based optimizers. Multiple methods are proposed to calculate adaptive learning rates for DNN, such as RMSPROP (Graves, 2013; Tieleman & Hinton, 2012), ADADELTA (Zeiler, 2012), and ADAM (Kingma & Ba, 2015). ADAM is probably the most popular optimizer due to its relative insensitivity to hyperparameter values and rapid initial progress during training (Keskar & Socher, 2017; Sun, 2020). However, it was pointed out that ADAM may not converge (Reddi et al., 2018), leading to an improvement of ADAM called ADMSGRAD (Reddi et al., 2018). There are multiple attempts to understand ADAM that lead to new theoretical results and new variants of ADAM (e.g., (Barakat & Bianchi, 2019; Zou et al., 2019)). In this paper, we propose a new probabilistic framework that treats gradients as random variables and utilizes stochastic variational inference (SVI) (Hoffman et al., 2013) to derive an estimate of a gradient. We indicate that our method is closely related to ADAM and is an adaptive gradient-based optimizer.

A different line of research focuses on 2nd-order optimization methods that require the calculation of Hessian matrices. In the case of deep learning, it is infeasible to calculate Hessian matrices for all layers. Therefore, many researchers focus on approximated 2nd-order optimizers. One of the most popular methods is KFAC (Martens & Grosse, 2015). Recently, there have been new developments in this direction (e.g., (Eschenhagen et al., 2023; Lin et al., 2021; 2023)). However, extending K-FAC introduces memory overhead that could be prohibitive, and numerical instabilities might arise in low-precision settings due to the need for matrix inversions or decompositions. Here, we focus on the 1st-order optimizer. However, we see great potential for extending our framework to the 2nd-order optimization.

There are various attempts to treat SGD as a probabilistic framework. For instance, Mandt et al. (2017) presented an interesting perspective on how SGD can be used for approximate Bayesian posterior inference. Moreover, they showed how SGD relates to a stochastic differential equation (SDE), specifically, the Ornstein-Uhlbeck process. The idea of perceiving SGD as an SDE was further discussed in, e.g., (Yokoi & Sato, 2019) with interesting connections to stochastic gradient Langevin dynamics (Welling & Teh, 2011). Another perspective was on the phrasing SGD as a Bayesian method either for sampling (Mingard et al., 2021), modeling parameter uncertainty for streaming data (Liu et al., 2024) or generalization (Smith & Le, 2017). A different approach uses linear Gaussian models to formulate the generation process of the noisy gradients and infer the hidden true gradient with Kalman filter or Bayesian filter, e.g., (Bittner & Pronzato, 2004; Vuckovic, 2018; Yang, 2021). In this paper, we propose to model dependencies between *true* gradients and *noisy* gradients through the Bayesian perspective using a novel framework. Our framework treats both

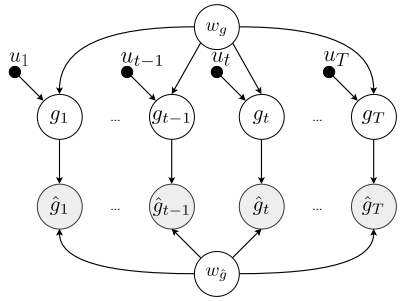

Figure 1. A probabilistic graphical model of our proposed VSGD optimizer. White circles represent latent variables, gray circles correspond to observable variables, filled small black circles denote control variables.

quantities as random variables and allows efficient inference and incorporation of prior knowledge into the optimization process. A similar line of work is the natural gradient-based methods for Bayesian Inference where the optimizer uses natural gradients instead of Euclidean gradients (Khan et al., 2018; Osawa et al., 2019). Our work can be considered orthogonal to this line of work, as they apply ADAM to a Bayesian model for learning the variational parameters. VSGD, on the other hand, assumes a Bayesian model for the gradients themselves and may not necessarily be applied to a Bayesian inference problem.

3. Methodology

3.1. Our approach: VSGD

A probabilistic model of SGD There are several known issues with applying SGD to the training of DNNs (Bottou, 2012; Sun, 2020), such as slow convergence due to gradient explosion/vanishing or too noisy gradient, poor initialization of weights, and the need for a hyperparameter search for each architecture and dataset. Here, we would like to focus on finding a better estimate of the *true* gradient. To achieve that, we propose to perceive SGD through a Bayesian perspective, in which we treat noisy gradients as observed variables and want to infer the posterior distributions over true gradients. Subsequently, we can then use the expected value of the posterior distribution over the *true* gradient g_t and plug it into the SGD update rule (Eq. 1). In the following, we present how we can achieve that.

In our considerations, we treat all quantities as random variables. Hence, the *true* gradient g_t is a latent variable, and we observe the *noisy* gradient $\hat{g}_t$. We propose to model g_t and $\hat{g}_t$ using Gaussian distributions with precision variables w_g and $w_{\hat{g}}$, representing the information of systematic and observation noises, respectively. Here, we use "observation noise" to indicate the sampling noise of the observed gradient and "systematic noise" to indicate the introduced error of trying to approximate the real, local area of the gradient surface with a simplified model. Both precision variables are treated as Gamma-distributed latent variables. The advantage of treating precision variables as latent variables instead of hyperparameters is that this way, we can incorporate prior knowledge over the systematic and observation noise as well as having uncertainty over them.

We introduce a control variate u_t to serve as a mean for g_t and is essential for our approach. We suggest defining it as a function that aggregates information from previously observed noisy gradients $u_t = h(\hat{g}_{1:t-1})$. Later (Eq. 8), we will discuss the specific formulation of the function $h(\cdot)$. In the case of $\hat{g}_t$, we propose using g_t as the mean. We assume that the gradients across each dimension are independent by modeling them independently. This approach is adopted for computational efficiency. Moving forward, unless specified otherwise, all symbols should be considered as scalars.

For the maximum number of iterations T, the joint distribution is then defined as follows (Figure 1):

$$p(w_g, w_{\hat{g}}, g_{1:T}, \hat{g}_{1:T}; u_{1:T}) \tag{2}$$

$$= p(w_g)p(w_{\hat{g}}) \prod_{t=1}^{T} p(g_t|w_g; u_t)p(\hat{g}_t|g_t, w_{\hat{g}}), \tag{3}$$

with the following distributions:

$$p(g_t|w_g; u_t) = \mathcal{N}(u_t, w_g^{-1}), \tag{4}$$

$$p(\hat{g}_t|g_t, w_{\hat{g}}) = \mathcal{N}(g_t, w_{\hat{g}}^{-1}), \tag{5}$$

$$p(w_g) = \Gamma(\gamma, \gamma), \tag{6}$$

$$p(w_{\hat{g}}) = \Gamma(\gamma, K_g\gamma), \tag{7}$$

where the Gamma distributions $\Gamma(\cdot)$ are parameterized by shape and rate. K_g is a hyperparameter designed to reflect our prior belief about the variance ratio between $\hat{g}_i$ and g_i, i.e., $K_g = \mathbb{E}\left[w_{\hat{g}}^{-1}\right] / \mathbb{E}\left[w_g^{-1}\right]$. A recommended value for K_g is greater than 1, e.g. $K_g = 30$, suggesting that the majority of observed variance stems from observation noise $w_{\hat{g}}^{-1}$ rather than systematic noise w_g^{-1}. γ is a shared hyperparameter that indicates the strength of the prior, essentially equating the prior knowledge with γ observations. A smaller value for γ, such as $1e-8$, is advised to represent our prior ignorance. The remaining key question is the choice of the control variate u_t as a way to carry information. In this paper, we aggregate the information over t by setting the prior mean for the current g_t as the posterior mean of the previous ones:

$$u_t = \mathbb{E}_{p(g_t|\hat{g}_{t-1}; u_{t-1})}[g_t], \tag{8}$$

where u_0 could be set to 0.

An alternative approach to using u_t to summarize the knowledge before t is to instead let g_t explicitly depend on $g_{1:t-1}$. However, incorporating this dependency explicitly would lead to a non-scalable, fully connected graphical model. A

common alternative solution is to adopt a Markov structure, in which g_t depends only on the most recent L observations $g_{t-L:t-1}$. This aligns the model with frameworks similar to Kalman filters or Bayesian filters (Kalman, 1960). However, these models either require known precision terms beforehand, which is often impractical or necessitate online learning of precisions through computationally intensive methods such as nested loops in variational inference or Monte Carlo methods, rendering them non-scalable for training deep neural networks. By introducing u_t and treating it as an additional observation, we can utilize SVI to avoid nested loops and still guarantee convergence to the estimated precisions (if any). As demonstrated in this paper, this strategy proves to be both scalable and efficient.

Stochastic Variational Inference Since the model defined by Eqs. 4-7 requires calculating intractable integral over latent variables, to find a good estimate of g_t, we aim to approximate the distribution $p(w_g, w_{\hat{g}}, g_{1:T}, \hat{g}_{1:T}; u_{1:T})$ with a variational distribution $q(w_g, w_{\hat{g}}, g_{1:T}; \tau, \phi_{1:T})$, parameterized by the global and local variational parameters τ and $\{\phi_t\}_{t=1}^T$ respectively. We achieve this by maximizing the evidence lower-bound (ELBO) defined by p and q

$$\arg\max_{\tau,\phi_{1:T}} \mathbb{E}_q \left[\log \frac{p(w_g, w_{\hat{g}}, g_{1:T}, \hat{g}_{1:T}; u_{1:T})}{q(w_g, w_{\hat{g}}, g_{1:T}; \tau, \phi_{1:T})} \right]. \quad (9)$$

We will demonstrate that this objective can be achieved concurrently with the original optimization objective $\theta^* = \arg\min_\theta \mathcal{L}(\theta; \mathcal{D})$, allowing each SGD iteration of θ to serve simultaneously as an SVI iteration for Eq. 9. Following SVI (Hoffman et al., 2013), we employ the mean-field approach, assuming the variational posterior is factorized as follows:

$$q(w_g, w_{\hat{g}}, g_{1:T}) = q(w_g)q(w_{\hat{g}}) \prod_{t=1}^{T} q(g_t). \quad (10)$$

We define the marginal variational distributions as:

$$q(w_g) = \Gamma(a_g, b_g), \quad (11)$$
$$q(w_{\hat{g}}) = \Gamma(a_{\hat{g}}, b_{\hat{g}}), \quad (12)$$
$$q(g_t) = \mathcal{N}(\mu_{t,g}, \sigma_{t,g}^2), \quad (13)$$

for $t = 1 : T$, where the global and local variational parameters are $\tau = (a_g, a_{\hat{g}}, b_g, b_{\hat{g}})^T$ and $\phi_t = (\mu_{t,g}, \sigma_{t,g}^2)^T$, respectively.

We now outline the complete SVI update process to meet the objective set in Eq. 9. For detailed derivations, please see App. A.1. Let $\tau_{t-1} = (a_{t-1,g}, a_{t-1,\hat{g}}, b_{t-1,g}, b_{t-1,\hat{g}})^T$ and $\phi_{t-1} = (\mu_{t-1,g}, \sigma_{t-1,g}^2)^T$ represent the global and local variational parameters updated just before the t-th SVI iteration, and we set the control variate:

$$u_t = \mu_{t-1,g}, \quad (14)$$

as the variational version of u_t as in Eq. 8, and let $u_0 = 0$. At iteration t, we start by randomly drawing a noisy gradient $\hat{g}_t$ with the sampling algorithm of your choice (e.g. using a mini-batch of samples). Then, conditioned on $\hat{g}_t$, τ_{t-1}, and u_t, we update the local parameter ϕ_t as follows:

$$\mu_{t,g} = \frac{b_{t-1,\hat{g}}}{b_{t-1,\hat{g}} + b_{t-1g}} \mu_{t-1,g} + \frac{b_{t-1,g}}{b_{t-1,\hat{g}} + b_{t-1g}} \hat{g}_t, \quad (15)$$

$$\sigma_{t,g}^2 = \left(\frac{a_{t-1,g}}{b_{t-1,g}} + \frac{a_{t-1,\hat{g}}}{b_{t-1,\hat{g}}} \right)^{-1}. \quad (16)$$

Note that $\mu_{t,g}$ represents a dynamic weighted average of the previous estimate $\mu_{t-1,g}$ and the noisy observation $\hat{g}_t$. Weights are dynamically determined based on the learned relative precision between Eq. 4 and Eq. 5. $\mu_{t,g}$ then acts as a control variate for the subsequent iteration. Recall that $\frac{a_{t,g}}{b_{t,g}} = \mathbb{E}_{q_t}[w_g]$. Therefore, the variance parameter $\sigma_{t,g}^2$ is equal to the inverse sum of the two precisions characterizing the systematic and observation noise. For an alternative kernel smoothing view on Eq. 15, please refer to App. A.2.

Given local parameters ϕ_t, update equations for the global parameters τ_t have the form:

$$a_{t,g} = a_{t,\hat{g}} = \gamma + 0.5, \quad (17)$$
$$b'_{t,g} = \gamma + 0.5 \left(\sigma_{t,g}^2 + (\mu_{t,g} - \mu_{t-1,g})^2 \right), \quad (18)$$
$$b'_{t,\hat{g}} = K_g \gamma + 0.5 \left(\sigma_{t,g}^2 + (\mu_{t,g} - \hat{g}_t)^2 \right). \quad (19)$$

In Eq. 17, $a_{t,g}$ and $a_{t,\hat{g}}$ remain constant throughout the iterations and depend purely on the prior strength γ. For the sake of consistency, they are included in the update equations. Thus, the posterior belief of the precision parameters w_g and $w_{\hat{g}}$ are determined purely by the rate parameters $(b'_{t,g}, b'_{t,\hat{g}})$. Intuitively, they characterize the amount of systematic noise and observation noise accordingly. In both cases, the prior value and the estimated variance of the true gradient additively contribute to the estimation. The main difference between the two comes in the third term, which depends on how much the gradient mean changed in time for the systematic noise and the difference between the observed gradient and the mean estimate for the observation noise.

Finally, the SVI updates for the global parameters are:

$$b_{t,g} = (1 - \rho_{t,1}(t))b_{t-1,g} + \rho_{t,1}(t)b'_{t,g}, \quad (20)$$
$$b_{t,\hat{g}} = (1 - \rho_{t,2}(t))b_{t-1,\hat{g}} + \rho_{t,2}(t)b'_{t,\hat{g}}. \quad (21)$$

where $\rho_{t,1}(t)$ and $\rho_{t,2}(t)$ denote the learning rates of SVI across iterations t. A typical selection for ρ could be as follows:

$$\rho_{t,1}(t) = t^{-\kappa_1}, \quad (22)$$
$$\rho_{t,2}(t) = t^{-\kappa_2}. \quad (23)$$

Algorithm 1 VSGD

Input: SVI learning rate parameter $\{\kappa_1, \kappa_2\}$, learning rate η, prior strength γ, prior variance ratio K_g.
Initialize:
$\theta_0, a_{0,g} = \gamma; a_{0,\hat{g}} = \gamma; b_{0,g} = \gamma; b_{0,\hat{g}} = K_g\gamma; \mu_{0,g} = 0$
for $t = 1$ **to** T **do**
  Compute $\hat{g}_t$ for $\mathcal{L}(\theta; \cdot)$
  $\rho_{t,1} = t^{-\kappa_1}$
  $\rho_{t,2} = t^{-\kappa_2}$
  Update $\sigma^2_{t,g}, \mu_{t,g}$ ▷ Eq. 15, 16
  Update $a_{t,g}, a_{t,\hat{g}}$ ▷ Eq. 17
  Update $b_{t,g}, b_{t,\hat{g}}$ ▷ Eq. 20,21
  Update θ_t ▷ Eq. 24
end for

In this formulation, κ_1 and κ_2 are hyperparameters that influence the behavior of the SVI learning rates. We typically limit κ_1 and κ_2 in the range $(0.5, 1]$ to let $\rho_{t,1}$ and $\rho_{t,2}$ meet the Robbins–Monro conditions (Robbins & Monro, 1951), and we select $\kappa_2 > \kappa_1$ to allow for a slower or larger decaying learning rate for w_g. For example, setting $\kappa_1 = 0.8$ and $\kappa_2 = 0.9$ allows w_g to adapt more rapidly to new developments compared to $w_{\hat{g}}$.

Note that when using a mini-batch of samples, $\hat{g}_t$ is calculated as the average of the sampled gradients in the batch. While in VSGD it is possible to treat the samples in the mini-batch separately, see App. A.3 for details.

Having concluded the t-th SVI iteration, we then incorporate the local parameter ϕ_t to finalize the t-th SGD iteration:

$$\theta_t = \theta_{t-1} - \frac{\eta}{\sqrt{\mu^2_{t,g} + \sigma^2_{t,g}}}\mu_{t,g}, \tag{24}$$

where $\sqrt{\mu^2_{t,g} + \sigma^2_{t,g}} = \sqrt{\mathbb{E}\left[g^2_t\right]}$ represents an estimation of the local Lipschitz constant. A summary of the steps of the VSGD algorithm can be seen in Algorithm 1.

3.2. VSGD, Constant VSGD and their connections to other gradient-based deep learning optimizers

Our probabilistic framework offers a way of estimating gradients. A natural question to raise here is whether this framework is related to other 1st order optimizers used in deep learning. Similarly to well-known methods like Adam, VSGD maintains a cache of the first and second momenta of the gradient, calculated as weighted moving averages. However, unlike other methods, where these variables remain constant, VSGD adjusts its weights across iterations.

VSGD and Normalized-SGD First, we can show how our VSGD method relates to a version of SGD called Normalized-SGD (Murray et al., 2019) that updates the gradients as follows:

$$\theta_t = \theta_{t-1} - \frac{\eta}{\sqrt{\hat{g}^2_t}}\hat{g}_t. \tag{25}$$

If we take $\gamma \to \infty$, $K_g \to 0$, then VSGD is approximately equivalent to Normalized-SGD (Murray et al., 2019). We can show it by taking $b_{0,g} = \gamma$, $b_{0,\hat{g}} = K_g\gamma$. Then, for $\gamma \to \infty$ and $K_g \to 0$, Eq. 15 yields:

$$\mu_{1,g} = 0 \cdot \frac{K_g\gamma}{K_g\gamma + \gamma} + \hat{g}_1\frac{\gamma}{K_g\gamma + \gamma} \approx \hat{g}_1. \tag{26}$$

In a subsequent iteration, as γ dominates all other values, we have: $a_{t,g} \approx \gamma$, $b'_{t,g} \approx \gamma$, $b'_{t,\hat{g}} \approx K_g\gamma$. As a result, $b_{t,g} \approx \gamma$, $b_{t,\hat{g}} \approx K_g\gamma$. By induction, the variational parameter for g_t at iteration t is the following:

$$\sigma^2_{t,g} = \frac{K_g\gamma^2}{\gamma(K_g\gamma + \gamma)} = \frac{K_g}{K_g + 1} \approx 0, \tag{27}$$

$$\mu_{t,g} = \mu_{t-1,g} \cdot \frac{K_g\gamma}{K_g\gamma + \gamma} + \hat{g}_t\frac{\gamma}{K_g\gamma + \gamma} \approx \hat{g}_t. \tag{28}$$

As a result, the update rule for θ is approximately the update rule of Normalize-SGD, namely:

$$\theta_t \approx \theta_{t-1} - \frac{\eta}{\sqrt{\hat{g}^2_t}}\hat{g}_t. \tag{29}$$

Constant VSGD Before we look at further connections between our VSGD and other optimizers, we first introduce a simplified version of VSGD with a constant variance ratio shared between g_t and $\hat{g}_t$. This results in the following model:

$$p(\omega, g_{1:T}, \hat{g}_{1:T}; u_{1:T}) = p(\omega)\prod_{t=1}^{T} p(g_t|\omega; u_t)p(\hat{g}_t|g_t, \omega),$$

with the following distributions:

$$p(g_t|\omega; u_t) = \mathcal{N}(u_t, K_g^{-1}\omega^{-1}), \tag{30}$$

$$p(\hat{g}_t|g_t, \omega) = \mathcal{N}(g_t, \omega^{-1}), \tag{31}$$

$$p(\omega) = \Gamma(\gamma, \gamma). \tag{32}$$

This approach is tantamount to imposing a strong prior on Eq. 3, dictating that the observation noise is the systematic noise scaled by K_g. This strong prior affects only the ratio between systematic and observation noises. Regarding the prior knowledge of the observation noise ω itself, we can maintain an acknowledgment of our ignorance by setting γ to a small value such as 10^{-8}.

Given its characteristic of maintaining a constant variance ratio, we refer to the algorithm as Constant VSGD. Since the derivation of Constant VSGD aligns with that of other variants of VSGD, we omit the detailed derivation here and directly present the update equations.

Algorithm 2 CONSTANT VSGD

Input: SVI learning rate parameter κ, SGD learning rate η, prior strength γ, variance ratio K_g.
Initialize:
$\theta_0, a_{0,\hat{g}} = \gamma; b_{0,\hat{g}} = \gamma; \mu_{0,g} = 0$
for $t = 1$ **to** T **do**
  Draw $\hat{g}_t$
  $\rho_t = t^{-\kappa}$
  Update $\sigma^2_{t,g}, \mu_{t,g}$ ▷ Eq. 33, 34
  Update $a_{t,\hat{g}}, b_{t,\hat{g}}$ ▷ Eq. 35,36
  Update θ_t ▷ Eq. 37
end for

$$\mu_{t,g} = \mu_{t-1,g}\frac{K_g}{K_g+1} + \hat{g}_t\frac{1}{K_g+1}, \tag{33}$$

$$\sigma^2_{t,g} = \frac{1}{K_g+1}\frac{b_{t-1,\hat{g}}}{a_{t-1,\hat{g}}}, \tag{34}$$

$$a_{t,\hat{g}} = \gamma + 1, \tag{35}$$

$$\begin{aligned} b_{t,\hat{g}} =& (1-\rho_t)b_{t-1,\hat{g}} + \rho_t\Big[\gamma \\ &+ 0.5\left(\sigma^2_{t,g} + (\mu_{t,g} - \hat{g}_t)^2\right) \\ &+ 0.5K_g\left(\sigma^2_{t,g} + (\mu_{t,g} - \mu_{t-1,g})^2\right)\Big], \end{aligned} \tag{36}$$

$$\theta_t = \theta_{t-1} - \frac{\eta}{\sqrt{\mu^2_{t,g} + \sigma^2_{t,g}}}\mu_{t,g}. \tag{37}$$

The complete CONSTANT VSGD algorithm is outlined in Algorithm 2, the updates are notably simpler compared to VSGD, as outlined in Algorithm 1, yet they still offer sufficient flexibility to estimate the uncertainties of the system. We enumerate the key distinctions between CONSTANT VSGD and VSGD as follows:

1. In VSGD, two global hidden variables, w_g and $w_{\hat{g}}$, are employed to represent systematic and observation noise, respectively. On the contrary, CONSTANT VSGD utilizes ω for observation noise and constrains the systematic noise to a fraction $\frac{1}{K_g}$ of the observation noise. Due to these reduced degrees of freedom, all the update equations in CONSTANT VSGD are simplified.

2. In the first momentum update Eq. 15 of VSGD, the weights are determined by the ratio between systematic and observation noise. However, in Eq. 33 of CONSTANT VSGD, these weights are fixed at $\{\frac{K_g}{K_g+1}, \frac{1}{K_g+1}\}$, facilitating a direct comparison with other optimizers such as ADAM.

To sum up, now we know how to simplify VSGD to CONSTANT VSGD. In the following, we will indicate how CONSTANT VSGD is connected to other optimizers, namely: ADAM, SGD with momentum, and AMSGRAD.

3.2.1. CONSTANT VSGD AND ADAM

First, we compare CONSTANT VSGD with one of the strongest and most popular optimizers for DNNs, ADAM (Kingma & Ba, 2015). At the t-th iteration, ADAM updates the first two momenta of the gradient through the following procedure:

$$m_t = \beta_1 m_{t-1} + (1-\beta_1)\hat{g}_t, \tag{38}$$

$$v_t = \beta_2 v_{t-1} + (1-\beta_2)\hat{g}^2_t, \tag{39}$$

where β_1, β_2 are hyperparameters.

Then, θ is updated as follows:

$$\theta_t = \theta_{t-1} - \frac{\eta}{\sqrt{\tilde{v}_t}}\tilde{m}_t, \tag{40}$$

where $\tilde{m}_t$ and $\tilde{v}_t$ are the bias-corrected versions of m_t and v_t, namely:

$$\tilde{m}_t = m_t/(1-\beta_1^t), \tag{41}$$

$$\tilde{v}_t = v_t/(1-\beta_2^t). \tag{42}$$

Now, the counterparts of m_t and v_t in CONSTANT VSGD, i.e., the first and second momenta:

$$\mathbb{E}\left[g_t\right] = \mu_{t,g}, \tag{43}$$

$$\mathbb{E}\left[g^2_t\right] = \mu^2_{t,g} + \sigma^2_{t,g}, \tag{44}$$

where $\mu_{t,g}$ is defined in Eq. 33. If we set $K_g = \frac{\beta_1}{1-\beta_1}$, then the first momentum updates in Eq. 33 and Eq. 38 become identical.

Next, we examine the relationship between Eq. 39 and Eq. 44. Intuitively, v_t (Eq. 39) is an approximation of the expected second momentum of the noisy gradient $\mathbb{E}\left[\hat{g}^2_t\right]$. While Eq. 44 attempts to quantify the expected second momentum of the actual gradient $\mathbb{E}\left[g^2_t\right]$.

By plugging to Eq. 33 and Eq. 34 to Eq. 44, we obtain the following:

$$\mathbb{E}\left[g^2_t\right] = \mu^2_{t-1,g}\frac{K_g^2}{(K_g+1)^2} + \hat{g}^2_t\frac{1}{(K_g+1)^2} \tag{45}$$

$$+ \frac{2K_g}{(K_g+1)^2}\mu_{t-1,g}\hat{g}_t + \frac{1}{K_g+1}\frac{b_{t-1,\hat{g}}}{a_{t-1}}. \tag{46}$$

For clarity, we divide the aforementioned equation into two components: Eq. 45 and Eq. 46. The first component aligns with Eq. 39, as both represent weighted sums of an estimated second momentum and $\hat{g}^2_t$. The distinction arises in the second component, where CONSTANT VSGD introduces two additional factors into the weighted sum: $\mu_{t-1,g}\hat{g}_t$ and $\frac{1}{K_g+1}\frac{b_{t-1,\hat{g}}}{a_{t-1}}$. Specifically, $\mu_{t-1,g}\hat{g}_t$ applies a penalty on Eq.45 when $\mu_{t-1,g}$ and $\hat{g}_t$ have opposing signs. The factor

$\frac{1}{K_g+1}\frac{b_{t-1,\hat{g}}}{a_{t-1}}$ represents the $\frac{1}{K_g+1}$ fraction of observation noise learned from the data, while a higher observation noise implies greater uncertainty in g_t, leading to a higher expected value $\mathbb{E}\left[g_t^2\right]$.

In summary, CONSTANT VSGD described in Algorithm 2 can be regarded as a variant of ADAM. The first momentum update in both algorithms is equivalent when K_g is set to $\frac{\beta_1}{1-\beta_1}$. However, the second momentum update in CONSTANT VSGD incorporates an additional data-informed component that is dynamically adjusted based on the observation noise learned from the data.

3.2.2. CONSTANT VSGD AND SGD WITH MOMENTUM

The second optimizer to which we compare CONSTANT VSGD is SGD with the momentum term (SGDM). SGDM is quite often used for DNN training due to its simplicity and improved performance compared to SGD (Liu et al., 2020b). At the t-th iteration, SGDM calculates an increment to θ_{t-1}, denoted by v_t, expressed as a weighted sum between the previous increment v_{t-1} and the latest noisy gradient $\hat{g}_t$, namely:

$$v_t = \lambda v_{t-1} + \eta \hat{g}_t, \tag{47}$$

$$\theta_t = \theta_{t-1} - v_t, \tag{48}$$

where λ is a hyperparameter. After rearranging Eq. 47, we get the following:

$$v_t = \lambda v_{t-1} + \eta \hat{g}_t \tag{49}$$

$$= (\lambda+\eta)\left[\frac{\lambda}{\lambda+\eta}v_{t-1} + \frac{\eta}{\lambda+\eta}\hat{g}_t\right]. \tag{50}$$

That is, the increment v_t in SGDM is a weighted average of v_{t-1} and $\hat{g}_t$, the counterpart of v_t in CONSTANT VSGD is $\mu_{t,g}$ updated by Eq. 33. Therefore, by choosing K_g such that $K_g = \frac{\gamma}{\eta}$, the update equations 50 and 33 become identical except for a constant scaling factor.

3.2.3. CONSTANT VSGD AND AMSGRAD

Another optimizer related to CONSTANT VSGD is AMSGRAD (Reddi et al., 2018), a variant of ADAM. AMSGRAD was introduced to address a specific issue of ADAM in estimating the second momentum. To correct the problem, a factor contributing to ADAM's suboptimal generalization performance was introduced. AMSGRAD rectifies it by incorporating the *long-term memory* of the past gradients.

Specifically, the momentum accumulation process in AMSGRAD remains identical to ADAM, as defined in Eq. 38 and Eq. 39. However, for the parameter update step, AMSGRAD differs from AMSGRAD (as in Eq. 40, whether bias-corrected or not) by incorporating the maximum of the past second momentum estimates rather than relying solely on the most recent value, that is:

$$\hat{v}_t = \max(\hat{v}_{t-1}, v_t), \tag{51}$$

$$\theta_t = \theta_{t-1} - \frac{\eta}{\sqrt{\tilde{v}_t}}\tilde{m}_t. \tag{52}$$

Here, the *long-term memory* is carried out by calculating the maximum of all historical second momentum estimations.

Similarly to AMSGRAD, CONSTANT VSGD incorporates a form of long-term memory regarding past gradient information in its second momentum update equations 45 and 46. However, this is not achieved by computing the maximum of historical values. Notably, the factor $\frac{1}{K_g+1}\frac{b_{t-1,\hat{g}}}{a_{t-1}}$ in Eq. 46 represents the expected observation noise, estimated using data up to the $(t-1)$-th iteration. In this expression, while K_g and a_{t-1} remain constant for $t \geq 2$, the variable component is $b_{t-1,\hat{g}}$, which is determined by Eq. 36, for clarity we restate it as:

$$b_{t,\hat{g}} = (1-\rho_t)b_{t-1,\hat{g}} + \rho_t s, \tag{53}$$

where:

$$s = \gamma + 0.5\left(\sigma_{t,g}^2 + (\mu_{t,g} - \hat{g}_t)^2\right) \tag{54}$$

$$+ 0.5K_g\left(\sigma_{t,g}^2 + (\mu_{t,g} - \mu_{t-1,g})^2\right). \tag{55}$$

It represents the combined squared systematic and observation noises after considering $\hat{g}_t$. Thus, Eq.53 suggests that $b_{t-1,\hat{g}}$ is a balanced summation of the historical estimate of $b_{\hat{g}}$ and the newly squared disturbances. The forgetting rate decreases over t, $\rho_t = t^{-\kappa}$, promoting longer memory retention in successive iterations. In CONSTANT VSGD, the magnitude of the memory retention can be modulated through the hyperparameter κ.

Table 1. Final Average test accuracy, over three random seeds.

	VSGD (w/ L2)	VSGD (w/o L2)	ADAM (w/o L2)	ADAMW (w/ L2)	SGD (w/ mom)
	CIFAR100				
VGG16	**70.1**	70.0	66.8	66.6	67.9
CONVMIXER	**69.8**	69.1	66.5	67.0	65.4
RESNEXT-18	**71.4**	71.2	68.2	69.7	68.5
	TINYIMAGENET-200				
VGG19	51.2	**52.0**	47.6	49.0	50.9
CONVMIXER	**53.1**	52.6	51.9	52.4	52.4
RESNEXT-18	48.7	47.2	48.8	**48.9**	47.0

4. Experiments

In this Section, we evaluate the performance of the VSGD compared to other optimizers on high-dimensional optimization problems The task we focus on in our work is training DNNs to solve classification tasks.

4.1. Setup

Data We used three benchmark datasets: CIFAR100 (Krizhevsky et al., 2009), TINYIMAGENET-200 (Deng et al., 2009b)[1], and IMAGENET-1K (Deng et al., 2009a). The CIFAR100 dataset contains 60000 small (32×32) RGB images labeled into 100 different classes, 50000 images are used for training, and 10000 are left for testing. In the case of TINYIMAGENET-200, the models are trained on 100000 images from 200 different classes and tested on 10000 images. For a large-scale experiment, we use the IMAGENET-1K dataset which contains 1,281,167 images from 1000 classes.

Architectures We evaluate VSGD on various optimization tasks and use three different neural network architectures: VGG (Simonyan & Zisserman, 2015), RESNEXT (Xie et al., 2017), and CONVMIXER (Trockman & Kolter, 2023). We keep the model hyperparameters fixed for all the optimizers in our experiments. We provide a detailed description of each architecture in App. B.1.

Hyperparameters We conducted a grid search over the following hyperparameters: Learning rate (all optimizers); Weight decay (ADAMW, VSGD); Momentum coefficient (SGD). For each set of hyperparameters, we trained the models with three different random seeds and chose the best one based on the validation dataset. The complete set of hyperparameters used in all experiments is reported in Table 3. Furthermore, we apply the learning rate scheduler, which halves the learning rate after each 10000 training iterations for CIFAR100 and every 20000 iterations for TINYIMAGENET-200. We train VGG and CONVMIXER using batch size 256 for CIFAR100 and batch size 128 for TINYIMAGENET-200. We use a smaller batch size (128 for CIFAR100 and 64 for TINYIMAGENET-200) with the RESNEXT architecture to fit training on a single GPU.

4.2. Results

In Table 1, we compare the top-1 test accuracy for ADAM, ADAMW, SGD (with momentum), VSGD, VSGD with weight decay for the different architectures and datasets. The accuracy is averaged across three random seeds. We observe that VSGD almost always converges to a better solution compared to ADAM and SGD, outperforming ADAM by an average of 2.6% for CIFAR100 and 0.9% for TINYIMAGENET-200. We present th progression of the top-1 test accuracy progresses during training in App. B.4 (Figures 2 and 3). Additionally, we provide the training curves. We observe that VSGD's convergence is often the same rate or faster compared to ADAMW.

[1]We use Tiny Imagenet(Stanford CS231N) provided on `www.image-net.org`

Table 2. Average training time on GeForce RTX 2080 Ti (seconds per training iteration) on CIFAR100 dataset.

MODEL		TRAINING TIME	
NAME	# PARAMS ($\times 10^6$)	ADAM	VSGD
VGG16	14.8	0.38	0.41
RESNEXT-18	11.1	0.84	0.87
CONVMIXER-256/8	0.6	0.36	0.38

Runtime & Stability As can be observed in Algorithm 1, VSGD requires additional operation at each gradient update step compared to the ADAM optimizer. However, we did not observe a large computational overhead compared to ADAM when training on the GPU. To illustrate this, we provide the average training time (per iteration) in Table 2.

ADAM is known to perform well on deep learning tasks without the need for extensive hyperparameter tuning. We observed that our approach also demonstrates stable performance without the need to tune the hyperparameters for different architectures and datasets separately. Therefore, we believe that VSGD is generally applicable to various DNN architectures and tasks, as is the case with ADAM.

Imagenet In order to evaluate VSGD's performance in heavily overparameterized domains, we performed an experiment on the Imagenet-1k dataset (Deng et al., 2009a) where we trained a ResNet-50 with both VSGD and ADAM. Due to limited computational resources, we did not perform any hyperparameter tuning for this experiment. For both optimizers, we used a learning rate of 0.1 with a ONECYCLELR learning rate scheduler (Smith & Topin, 2019) and a batch size of 256. As in the previous experiments, we set γ to 1e-8. We train both models for 50 epochs. The top-1 accuracy of the validation set for VSGD was 71.1%, exceeding ADAM's 68.7% by 2.4%.

5. Conclusion

In this work, we introduced VSGD, a novel optimization framework that combines gradient descent with probabilistic modeling by treating the true gradients as a hidden random variable. This approach allows for a more principled approach to model noise over gradients. Not only does this framework open a new path to optimizing DNNs, but one can establish links with other popular optimization methods by certain specific modeling choices in our method. Lastly, we evaluated VSGD on large-scale image classification tasks using a variety of deep learning architectures, where we demonstrated that VSGD consistently outperformed the baselines and achieved a competitive convergence rate.

References

Barakat, A. and Bianchi, P. Convergence analysis of a momentum algorithm with adaptive step size for non convex optimization. *arXiv preprint arXiv:1911.07596*, 2019.

Bittner, B. and Pronzato, L. Kalman filtering in stochastic gradient algorithms: construction of a stopping rule. In *2004 IEEE International Conference on Acoustics, Speech, and Signal Processing*, volume 2, pp. ii–709. IEEE, 2004.

Bottou, L. *Stochastic Gradient Descent Tricks*, pp. 421–436. Springer Berlin Heidelberg, Berlin, Heidelberg, 2012. ISBN 978-3-642-35289-8. doi: 10.1007/978-3-642-35289-8_25. URL https://doi.org/10.1007/978-3-642-35289-8_25.

Deng, J., Dong, W., Socher, R., Li, L.-J., Li, K., and Fei-Fei, L. Imagenet: A large-scale hierarchical image database. In *2009 IEEE Conference on Computer Vision and Pattern Recognition*, pp. 248–255, 2009a. doi: 10.1109/CVPR.2009.5206848.

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

Du, S. and Lee, J. On the power of over-parametrization in neural networks with quadratic activation. In *International conference on machine learning*, pp. 1329–1338. PMLR, 2018.

Eschenhagen, R., Immer, A., Turner, R. E., Schneider, F., and Hennig, P. Kronecker-Factored Approximate Curvature for Modern Neural Network Architectures. In *Thirty-seventh Conference on Neural Information Processing Systems*, 2023.

Glorot, X. and Bengio, Y. Understanding the difficulty of training deep feedforward neural networks. In *Proceedings of the thirteenth international conference on artificial intelligence and statistics*, pp. 249–256. JMLR Workshop and Conference Proceedings, 2010.

Graves, A. Generating sequences with recurrent neural networks. *arXiv preprint arXiv:1308.0850*, 2013.

He, K., Zhang, X., Ren, S., and Sun, J. Delving deep into rectifiers: Surpassing human-level performance on imagenet classification. In *Proceedings of the IEEE international conference on computer vision*, pp. 1026–1034, 2015.

Hoffman, M. D., Blei, D. M., Wang, C., and Paisley, J. Stochastic variational inference. *Journal of Machine Learning Research*, 2013.

Kalman, R. E. A new approach to linear filtering and prediction problems. 1960.

Keskar, N. S. and Socher, R. Improving generalization performance by switching from adam to sgd. *arXiv preprint arXiv:1712.07628*, 2017.

Khan, M., Nielsen, D., Tangkaratt, V., Lin, W., Gal, Y., and Srivastava, A. Fast and scalable Bayesian deep learning by weight-perturbation in Adam. In Dy, J. and Krause, A. (eds.), *Proceedings of the 35th International Conference on Machine Learning*, volume 80 of *Proceedings of Machine Learning Research*, pp. 2611–2620. PMLR, 10–15 Jul 2018. URL https://proceedings.mlr.press/v80/khan18a.html.

Kingma, D. and Ba, J. Adam: A method for stochastic optimization. In *International Conference on Learning Representations (ICLR)*, San Diega, CA, USA, 2015.

Krizhevsky, A., Hinton, G., et al. Learning multiple layers of features from tiny images. 2009.

Lin, W., Nielsen, F., Khan, M. E., and Schmidt, M. Structured second-order methods via natural gradient descent. 2021.

Lin, W., Duruisseaux, V., Leok, M., Nielsen, F., Khan, M. E., and Schmidt, M. Simplifying momentum-based positive-definite submanifold optimization with applications to deep learning. In *International Conference on Machine Learning*, pp. 21026–21050. PMLR, 2023.

Liu, H., Ong, Y.-S., Shen, X., and Cai, J. When gaussian process meets big data: A review of scalable gps. *IEEE transactions on neural networks and learning systems*, 31 (11):4405–4423, 2020a.

Liu, T., Lin, Y., and Zhou, E. Bayesian stochastic gradient descent for stochastic optimization with streaming input data. *SIAM Journal on Optimization*, 34(1):389–418, 2024.

Liu, Y., Gao, Y., and Yin, W. An improved analysis of stochastic gradient descent with momentum. *Advances in Neural Information Processing Systems*, 33:18261–18271, 2020b.

Loshchilov, I. and Hutter, F. SGDR: Stochastic gradient descent with warm restarts. *arXiv preprint arXiv:1608.03983*, 2016.

Mandt, S., Hoffman, M. D., Blei, D. M., et al. Stochastic gradient descent as approximate bayesian inference. *Journal of Machine Learning Research*, 18(134):1–35, 2017.

Martens, J. and Grosse, R. Optimizing neural networks with kronecker-factored approximate curvature. In *International conference on machine learning*, pp. 2408–2417. PMLR, 2015.

Mingard, C., Valle-Pérez, G., Skalse, J., and Louis, A. A. Is sgd a bayesian sampler? well, almost. *The Journal of Machine Learning Research*, 22(1):3579–3642, 2021.

Murray, R., Swenson, B., and Kar, S. Revisiting normalized gradient descent: Fast evasion of saddle points. *IEEE Transactions on Automatic Control*, 64(11):4818–4824, 2019.

Osawa, K., Swaroop, S., Khan, M. E. E., Jain, A., Eschenhagen, R., Turner, R. E., and Yokota, R. Practical deep learning with bayesian principles. *Advances in neural information processing systems*, 32, 2019.

Quinonero-Candela, J. and Rasmussen, C. E. A unifying view of sparse approximate gaussian process regression. *The Journal of Machine Learning Research*, 6: 1939–1959, 2005.

Reddi, S. J., Kale, S., and Kumar, S. On the convergence of adam and beyond. In *International Conference on Learning Representations*, 2018.

Robbins, H. and Monro, S. A stochastic approximation method. *The annals of mathematical statistics*, pp. 400–407, 1951.

Rumelhart, D. E., Hinton, G. E., and Williams, R. J. Learning representations by back-propagating errors. *nature*, 323(6088):533–536, 1986.

Schmidhuber, J. Deep learning in neural networks: An overview. *Neural networks*, 61:85–117, 2015.

Simonyan, K. and Zisserman, A. Very deep convolutional networks for large-scale image recognition. In *International Conference on Learning Representations*, 2015.

Smith, L. N. and Topin, N. Super-convergence: Very fast training of neural networks using large learning rates. In *Artificial intelligence and machine learning for multi-domain operations applications*, volume 11006, pp. 369–386. SPIE, 2019.

Smith, S. L. and Le, Q. V. A bayesian perspective on generalization and stochastic gradient descent. *arXiv preprint arXiv:1710.06451*, 2017.

Sun, R.-Y. Optimization for deep learning: An overview. *Journal of the Operations Research Society of China*, 8 (2):249–294, 2020.

Sutskever, I., Martens, J., Dahl, G., and Hinton, G. On the importance of initialization and momentum in deep learning. In *International conference on machine learning*, pp. 1139–1147. PMLR, 2013.

Tieleman, T. and Hinton, G. Lecture 6.5-rmsprop, coursera: Neural networks for machine learning. *University of Toronto, Technical Report*, 6, 2012.

Trockman, A. and Kolter, J. Z. Patches are all you need? *Transactions on Machine Learning Research*, 2023.

Vuckovic, J. Kalman gradient descent: Adaptive variance reduction in stochastic optimization. *arXiv preprint arXiv:1810.12273*, 2018.

Welling, M. and Teh, Y. W. Bayesian learning via stochastic gradient langevin dynamics. In *Proceedings of the 28th international conference on machine learning (ICML-11)*, pp. 681–688. Citeseer, 2011.

Xie, S., Girshick, R., Dollár, P., Tu, Z., and He, K. Aggregated residual transformations for deep neural networks. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pp. 1492–1500, 2017.

Yang, X. Kalman optimizer for consistent gradient descent. In *ICASSP 2021-2021 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*, pp. 3900–3904. IEEE, 2021.

Yokoi, S. and Sato, I. Bayesian interpretation of SGD as Ito process. *arXiv preprint arXiv:1911.09011*, 2019.

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

Zou, F., Shen, L., Jie, Z., Zhang, W., and Liu, W. A sufficient condition for convergences of adam and rmsprop. In *Proceedings of the IEEE/CVF Conference on computer vision and pattern recognition*, pp. 11127–11135, 2019.

A. Derivations

A.1. Derivation of VSGD

Here, we derive Eq. 15 to Eq. 19 following the SVI technique introduced by (Hoffman et al., 2013). First, we provide a brief overview of the SVI procedure before diving into the details.

In the t-th SVI iteration, we realize the following steps:

1. Sample $\hat{g}$ from $\hat{g}_{1:T}$, where the sample is denoted by $\hat{g}_t$.
2. Update the local parameters $\phi_{t,1:T}$ with τ_{t-1} and $\hat{g}_t$, such that
$$\phi_{t,k} = \arg\max_{\phi_k} \mathbb{E}_{q(w_g, w_{\hat{g}}; \tau_{t-1})} \left[\log \frac{p(g_k|w_g; u_t) p(\hat{g}_t|g_k, w_{\hat{g}})}{q(g_k; \phi_k)} \right] \qquad k = 1:T, \tag{56}$$
where $p(g_k|w_g; u_t)$ and $p(\hat{g}_t|g_k, w_{\hat{g}})$ are defined in Eq. 4 and Eq. 5, respectively. Note that here we use the same g_t for each $k \in 1:T$, as if the same $\hat{g}_t$ is observed T times.
3. Calculate the intermediate global parameters τ'_t with $\phi_{t,1:T}$, such that:
$$\tau'_t = \arg\max_{\tau} \mathbb{E}_{q(g_{1:T}; \phi_{t,1:T})} \left[\log \frac{p(w_g) p(w_{\hat{g}}) \prod_{k=1}^{T} p(g_k|w_g; u_t) p(\hat{g}_t|g_k, w_{\hat{g}})}{q(w_g, w_{\hat{g}}; \tau)} \right], \tag{57}$$
where $p(w_g)$ and $p(w_{\hat{g}})$ are defined in Eq. 6 and Eq. 7, respectively.
4. Update the global parameters τ_t using:
$$\tau_t = (1-\rho)\tau_{t-1} + \rho \tau'_t, \tag{58}$$
where $\rho_t = (\rho_{t,1}, \rho_{t,2})^T$ is defined in Eq. 22 and Eq. 23.
5. Pick any $\phi_{t,k} = (\mu_{t,k}, \sigma^2_{t,k})^T \in \phi_{t,1:T}$, and set $u_{t+1} = \mu_{t,k}$, where u_{t+1} serves as the control variable for the next iteration. We can choose any $\phi_{t,k}$ because by the definition (Eq. 56), $\phi_{t,k}$ and $\phi_{t,j}$ are identical for any $i, j \in 1:T$.

Since all $\phi_{t,k}$ are identical for $k = 1:T$, for notational ease, we will use $\phi_t = (\mu_{t,g}, \sigma^2_{t,g})^T$ to represent the unique local parameter updated in the t-th SVI iteration. Distributions defined in Eq. 3 are in the conjugate exponential family, therefore, we can derive the closed-form update rule for ϕ_t according to Eq. 56:

$$\log q(g_k; \mu_{t,g}, \sigma^2_{t,g}) \equiv \mathbb{E}_{q(w_g, w_{\hat{g}}; \tau_{t-1})} \left[\log p(g_k|w_g; u_t) + \log p(\hat{g}_t|g_k, w_{\hat{g}}) \right]. \tag{59}$$

Next, by matching the coefficients of the natural parameters, we get:

$$\mu_{t,g} = u_t \frac{b_{t-1,\hat{g}}}{b_{t-1,\hat{g}} + b_{t-1g}} + \hat{g}_t \frac{b_{t-1,g}}{b_{t-1,\hat{g}} + b_{t-1g}}, \tag{60}$$

$$\sigma^2_{t,g} = \frac{b_{t-1,\hat{g}} b_{t-1g}}{a_{t-1,g}(b_{t-1,\hat{g}} + b_{t-1g})}, \tag{61}$$

where Eq. 61 corresponds to Eq. 16. Putting the control variate $u_t = \mu_{t-1,g}$ in Eq. 60 results in Eq. 15. Note that the shape parameters $a_{t-1,g}$ and $a_{t-1,\hat{g}}$ are cancelled out while deriving Eq. 60, reasons being that the shapes are all equal to the same constant over the iterations, as later we will show in Eq. 66.

Similarly for $\tau'_t = (a'_{t,g}, a'_{t,\hat{g}}, b'_{t,g}, b'_{t,\hat{g}})$, according to Eq. 57:

$$\log q(w_g; a'_{t,g}, b'_{t,g}) \equiv \mathbb{E}_{q(\hat{g}_{1:T}; \phi_t)} \left[\log p(w_g) + \sum_{k=1}^{T} \log p(g_k|u_t, w_g) \right], \tag{62}$$

$$= \mathbb{E}_{q(\hat{g}_{1:T}; \phi_t)} \left[\log p(w_g) + T \log p(g_1|u_t, w_g) \right], \tag{63}$$

$$\log q(w_{\hat{g}}; a'_{t,\hat{g}}, b'_{t,\hat{g}}) \equiv \mathbb{E}_{q(\hat{g}_{1:T}; \phi_t)} \left[\log p(w_{\hat{g}}) + \sum_{k=1}^{T} \log p(\hat{g}_t|g_k, w_{\hat{g}}) \right], \tag{64}$$

$$= \mathbb{E}_{q(\hat{g}_{1:T}; \phi_t)} \left[\log p(w_{\hat{g}}) + T \log p(\hat{g}_t|g_1, w_{\hat{g}}) \right]. \tag{65}$$

Again, by matching the coefficients of the natural parameters:

$$a'_{t,g} = a'_{t,\hat{g}} = \gamma + 0.5T \tag{66}$$

$$b'_{t,g} = \gamma + 0.5T(\sigma^2_{t,g} + \mu^2_{t,g} - 2\mu_{t,g}\mu_{t-1,g} + {\mu_{t-1,g}}^2) \tag{67}$$

$$b'_{t,\hat{g}} = K_g\gamma + 0.5T(\sigma^2_{t,g} + \mu^2_{t,g} - 2\mu_{t,g}\hat{g}_t + \hat{g}^2_t) \tag{68}$$

In Eq. 66, $a'_{t,g}$ and $a'_{t,\hat{g}}$ equal the same constant over the iterations. As a result, $a_{t,g}$ and $a_{t,\hat{g}}$ are also equal constants, according to Eq .58.

Eq. 66 to Eq. 68 reveals that during the posterior updates of w_g and $w_{\hat{g}}$, choosing a larger value for T increases the emphasis on the information conveyed by $\hat{g}_t$, consequently diminishing the relative influence of the priors. In VSGD, we have already made the prior strength, γ, a free parameter, then we can replace T with a constant and still pertain the ability to adjust the strength between prior and new observation through values of γ.

That means we use a constant value, such as 1, to replace T in Eq. 66, Eq. 67 and Eq. 68, which result in Eq. 18 and Eq. 19. Eq. 20 and Eq. 21 are derived by simply following Eq. 58. Eq. 17 is derived from the fact that Eq. 66 is a constant, and interpolation between two equal constants results in the same constant.

A.2. A Kernel Smoothing View on VSGD

If we look at VSGD from a kernel smoothing perspective, making use of u_t serves as a cheap approximation to conditioning on all of the historical data. Namely, VSGD seeks to smooth the local gradient function in a similar way with the following kernel smoothing task.

There are two noisy observations of the gradient function at $\theta = \theta_t$. They are (θ_t, u_t) and $(\theta_t, \hat{g}_t)$. We can define two Gaussian kernels for each of the noisy observations, governed by w_g^{-1} and $w_{\hat{g}}^{-1}$. By denoting the Gaussian kernels κ_1 and κ_2, the smoothed gradient value at θ_t should be

$$g_t = \frac{\kappa_1(\theta_t, \theta_t)}{\kappa_1(\theta_t, \theta_t) + \kappa_2(\theta_t, \theta_t)} u_t + \frac{\kappa_2(\theta_t, \theta_t)}{\kappa_1(\theta_t, \theta_t) + \kappa_2(\theta_t, \theta_t)} \hat{g}_t \tag{69}$$

After taking the expectation step of SVI (expect out w_g and $w_{\hat{g}}$), the above equation simplifies to Eq (15).

Note that u_t in VSGD is introduced as a learned summary of the noisy observations before t and we only need to smooth on u_t as a cheap approximation to smoothing on all of the historical data. Similar simplification technique is commonly seen in scalable Gaussian Process solutions, as discussed in (Quinonero-Candela & Rasmussen, 2005; Liu et al., 2020a). In these references, the learned summaries are called the "inducing points".

If we look at VSGD as an approximation to the local areas of the gradient function, then the precision term w_g can be seen as a smoothness measure of the gradient surface. In VSGD we assume a global w_g, which is equivalent to assuming that the smoothness of the whole gradient function is governed by a constant. This assumption is not necessarily ideal, but we still treat it as a global parameter for simplicity and practical reasons.

A.3. Treating Samples Separately in a Mini-bach

Assuming the size of the mini-batch is M, denote the sample gradients $\{\hat{g}_t^{(i)}\}_{i=1:M}$. According to (Hoffman et al., 2013), we would need to calculate Eq. 15 M times and average them in Eq. 18 and Eq. 19. Specifically, we would need to replace Eq. 15, Eq. 18 and Eq. 19 with:

$$\mu^{(i)}_{t,g} = \mu_{t-1,g}\frac{b_{t-1,\hat{g}}}{b_{t-1,\hat{g}} + b_{t-1g}} + \hat{g}^{(i)}_t \frac{b_{t-1,g}}{b_{t-1,\hat{g}} + b_{t-1g}} \qquad i = 1:M \tag{70}$$

$$b'_{t,g} = \frac{\sum_{i=1}^{M} \gamma + 0.5\left(\sigma^2_{t,g} + (\mu^{(i)}_{t,g} - \mu_{t-1,g})^2\right)}{M} \tag{71}$$

$$b'_{t,\hat{g}} = \frac{\sum_{i=1}^{M} K_g\gamma + 0.5\left(\sigma^2_{t,g} + (\mu^{(i)}_{t,g} - \hat{g}^{(i)}_t)^2\right)}{M} \tag{72}$$

And introduce an additional step

$$\mu_{t,g} = \frac{\sum_{i=1}^{M} \mu_{t,g}^{(i)}}{M} \tag{73}$$

before Eq. 24.

Based on the equations presented, the computational cost grows linearly with the batch size M. Given the importance of efficiency in contemporary optimization techniques, above equations were omitted from the main paper. Nonetheless, it is conceivable that the increased computational expense could be offset by enhanced convergence rates, a hypothesis reserved for future investigation.

B. Experimental Details

B.1. Architecture

We have used open-source implementations of all three models.

VGG We use VGG16 and VGG19[2] with batch normalization. We add adaptive average pooling before the final linear layer to make the architecture suitable for both 32×32 and 64×64 images.

ConvMixer We use ConvMixer[3] with 256 channels and set the depth to 8.

ResNeXt ResNeXt[4] we set the cardinality (or number of convolution groups) to 8 and the depth to 18 layers. The widen factor is set to 4, resulting in channels being `[64, 256, 512, 1024]`. We add adaptive average pooling before the final linear layer to make the architecture suitable for both 32×32 and 64×64 images.

B.2. Hyperparameters

The full set of hyperparameters used in all experiments is reported in Table 3.

Table 3. Hyperparameter values used in **all** the experiments.

	ADAM	VSGD
(β_1, β_2)	(0.9, 0.999)	—
(κ_1, κ_2)	—	(0.9, 0.81)
γ	—	1e-8
K_g	—	30
learning rate	$\{0.001, 0.005, 0.01, 0.02\}$	
weight deacy	$\{0, 0.01\}$	
momentum coefficient	$\{0.9, 0.99\}$	

B.3. Data Augmentations

We use data augmentations during training to improve the generalizability of the models. These augmentations were the same for the optimizers and only differ between datasets.

Table 4. Data augmentations.

CIFAR100	TINY IMAGENET-200
`RandomCrop(32, padding=4)`	`RandomCrop(64, padding=4)`
`RandomHorizontalFlip`	`RandomHorizontalFlip`
	`RandomAffine(degrees=45, translate=(0.1, 0.1), scale=(0.9, 1.1))`
	`ColorJitter(brightness=0.2, contrast=0.2, saturation=0.2)`

[2] `github.com/alecwangcq/KFAC-Pytorch/blob/master/models/cifar/vgg.py`
[3] `github.com/locuslab/convmixer-cifar10`
[4] `github.com/prlz77/ResNeXt.pytorch`

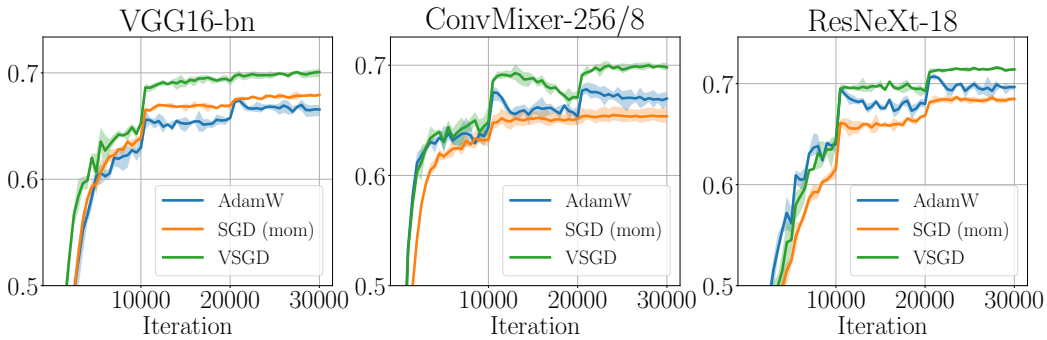

Figure 2. Top-1 test accuracy on CIFAR100 dataset.

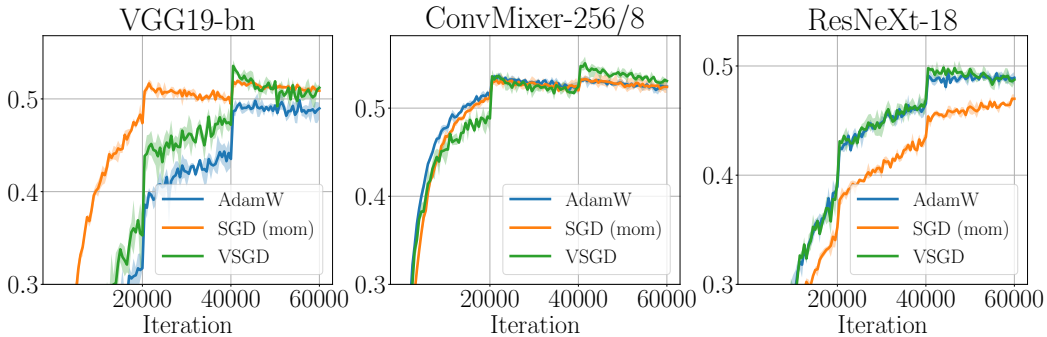

Figure 3. Top-1 test accuracy on TINYIMAGENET-200 dataset.

B.4. Training Curves

C. Hyperparameter Sensitivity Analysis

In this Section, we investigate the shape parameter γ's sensitivity to VSGD's performance. Higher γ implies a stronger weight on the prior while a lower γ implies a stronger weight on the current observation. We train VSGD on a wide range of γ values on CIFAR100 using the CONVMIXER architecture. The results can be observed in Figures 6, 7 and Table 5. Inspecting Figure 6 (Right) more closely, we observe that while there is a statistically significant difference between accuracies obtained by different γs, they all perform relatively well as long as γ is not too large.

γ	Accuracy
1e-9	67.75
5e-9	68.59
1e-8	69.03
5e-8	69.77
1e-7	69.71
1e-3	60.97

Table 5. Test accuracy on CIFAR100 trained with CONVMIXER for different γ values.

D. Constant VSGD

In this Section, we offer a comparison between VSGD and CONSTANT VSGD. From the modeling point of view, VSGD has more free parameters that allow our model to learn more complicated distributions. While CONSTANT VSGD imposes a constraint on VSGD, which is less flexible but may lead to better out-of-sample performance. To investigate this, we ran additional experiments on CIFAR100 with CONVMIXER architecture. The final accuracy for VSGD was 69.04 ± 0.4, while the accuracy for CONSTANT VSGD was 68.49 ± 0.19. The learning curves can be observed in Figure 8.

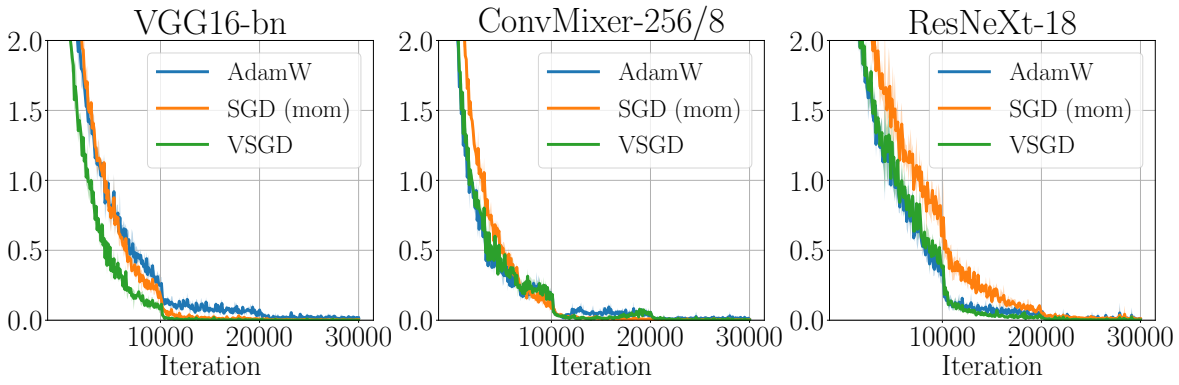

Figure 4. Training loss on CIFAR100 dataset.

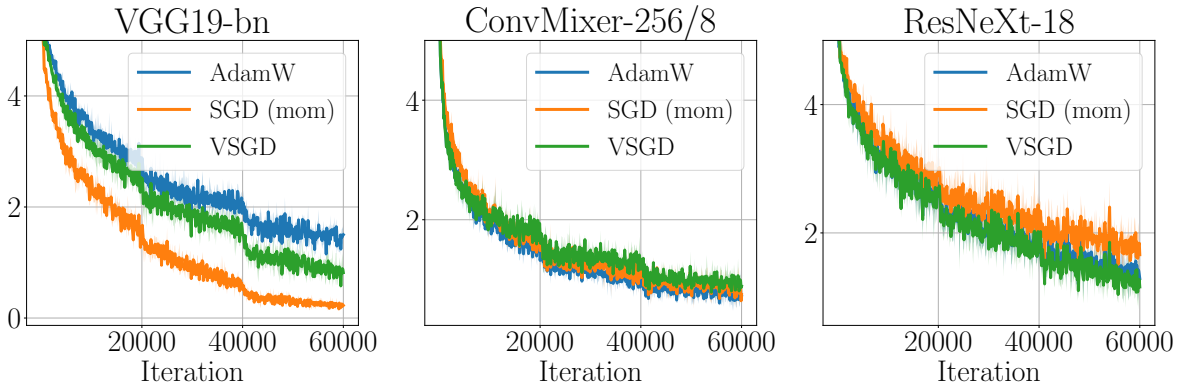

Figure 5. Training loss on TinyImagenet-200 dataset.

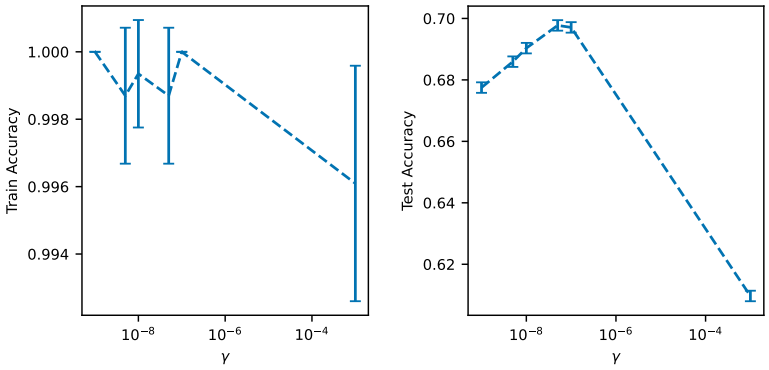

Figure 6. Analysis of the hyperparameter γ'sensitivity on the VSGD's performance in terms of accuracy.

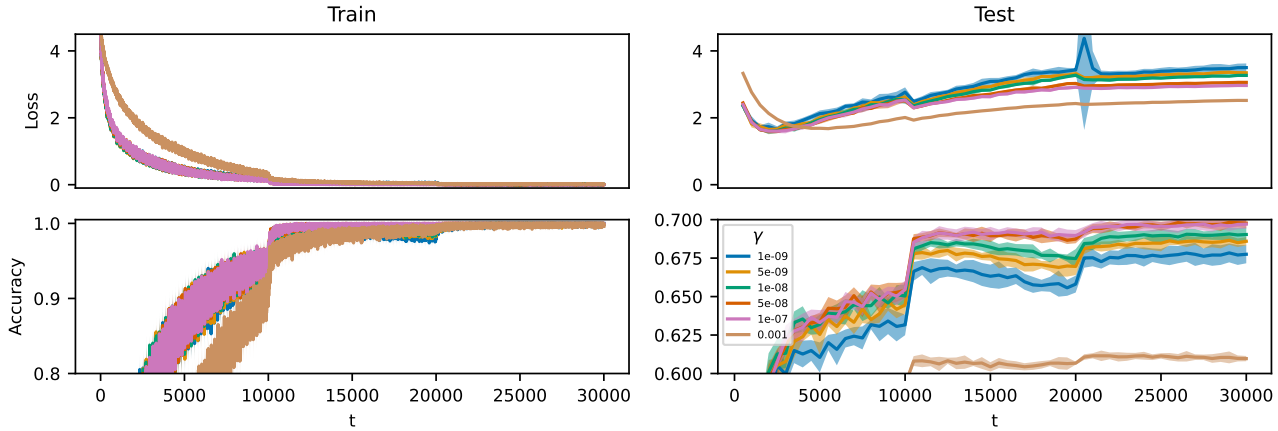

Figure 7. CIFAR100 learning curves trained with CONVMIXER architecture for different γ values.

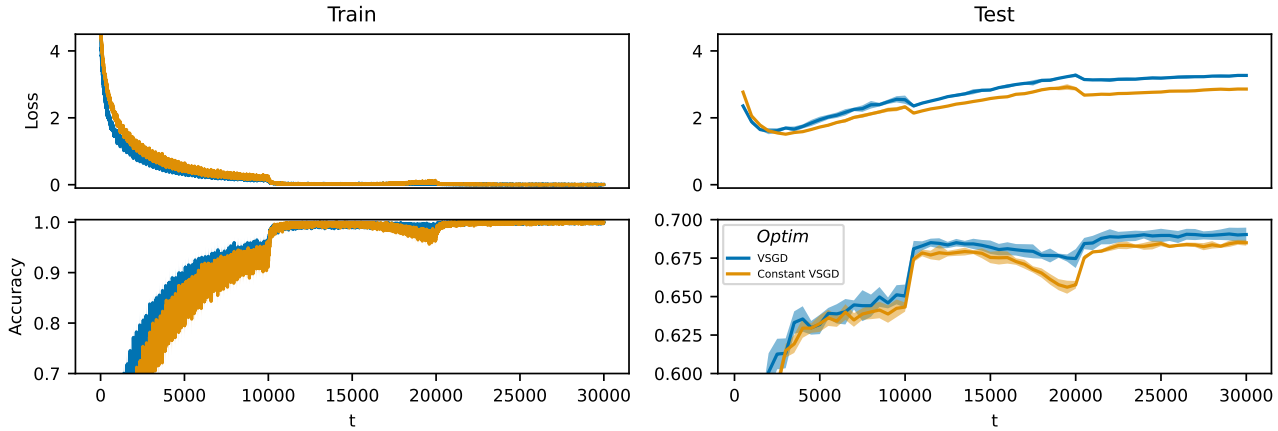

Figure 8. Comparison of CONSTANT-VSGD and VSGD learning curves trained on CIFAR100 with CONVMIXER architecture. The final accuracy for VSGD is 69.04 ± 0.4, while the accuracy for CONSTANT-VSGD is 68.49 ± 0.19.