KOALA: A Kalman Optimization Algorithm with Loss Adaptivity

Aram Davtyan, Sepehr Sameni, Llukman Cerkezi, Givi Meishvilli, Adam Bielski, Paolo Favaro

Computer Vision Group, University of Bern
{firstname.lastname}@inf.unibe.ch

Abstract

Optimization is often cast as a deterministic problem, where the solution is found through some iterative procedure such as gradient descent. However, when training neural networks the loss function changes over (iteration) time due to the randomized selection of a subset of the samples. This randomization turns the optimization problem into a stochastic one. We propose to consider the loss as a noisy observation with respect to some reference optimum. This interpretation of the loss allows us to adopt Kalman filtering as an optimizer, as its recursive formulation is designed to estimate unknown parameters from noisy measurements. Moreover, we show that the Kalman Filter dynamical model for the evolution of the unknown parameters can be used to capture the gradient dynamics of advanced methods such as Momentum and Adam. We call this stochastic optimization method KOALA, which is short for Kalman Optimization Algorithm with Loss Adaptivity. KOALA is an easy to implement, scalable, and efficient method to train neural networks. We provide convergence analysis and show experimentally that it yields parameter estimates that are on par with or better than existing state of the art optimization algorithms across several neural network architectures and machine learning tasks, such as computer vision and language modeling.

Introduction

Optimization of functions involving large datasets and high dimensional models finds today large applicability in several data-driven fields in science and the industry. Given the growing role of deep learning, in this paper we look at optimization problems arising in the training of neural networks. The training of these models can be cast as the minimization or maximization of a certain objective function with respect to the model parameters. Because of the complexity and computational requirements of the objective function, the data and the models, the common practice is to resort to iterative training procedures, such as gradient descent. Among the iterative methods that emerged as the most effective and computationally efficient is stochastic gradient descent (SGD) (Robbins and Monro 1951). SGD owes its performance gains to the adoption of an approximate version of the objective function at each iteration step, which, in turn, yields an approximate or noisy gradient.

While SGD seems to benefit greatly (*e.g.*, in terms of rate of convergence) from such an approximation, it has also been shown that too much noise hurts the performance (Wang et al. 2013; Bottou, Curtis, and Nocedal 2018). This suggests that, to further improve over SGD, one could attempt to model the noise of the objective function. We consider the iteration-time varying loss function used in SGD as a stochastic process obtained by adding the empirical risk to zero mean Gaussian noise. A powerful approach designed to handle estimation with such processes is Kalman filtering (Kalman 1960). In fact, Kalman filtering has been used to train neural networks before (Haykin 2004; Patel 2016; Ismail et al. 2018). However, it can be applied in very different ways. Indeed, in our approach, which we call *KOALA*, we introduce a number of novel ideas that result in a practical and effective training algorithm. Firstly, we introduce drastic approximations of the estimated covariance of Kalman's dynamical state so that the corresponding matrix depends on only up to a 2×2 matrix of parameters. Secondly, we approximate intermediate Kalman filtering calculations so that more accuracy can be achieved. Thirdly, because of the way we model the objective function, we can also define a schedule for the optimization that behaves similarly to learning rate schedules used in SGD and other iterative methods (Kingma and Ba 2015).

Our contributions can be summarized as follows: 1) We design KOALA so that it can handle high-dimensional data and models, and large datasets; 2) We present analysis and conditions to ensure convergence; 3) We allow both the automated tuning of the algorithm and also the use of a learning rate schedule similar to those in existing methods; 4) We incorporate the automatic adaptation to the noise in the loss, which might vary depending on the settings of the training (*e.g.*, the minibatch size), and to the variation in the estimated weights over iteration time; 5) We show how to incorporate iteration-time dynamics of the model parameters, which are analogous to momentum (Sutskever et al. 2013); 6) We introduce KOALA as a framework so that it can be further extended (we show two variations of KOALA); 7) We show experimentally that KOALA is on par with state of the art optimizers and can yield better minima at **test time** in a number of problems from image classification to generative adversarial networks (GAN) and natural language processing (NLP).

Prior Work

First-Order Methods. First-order methods exploit only the gradient of the objective function. The main advantage of these methods lies in their speed and simplicity. (Robbins and Monro 1951) introduce the very first stochastic optimization method (SGD) in early 1951. Since then, the SGD method has been thoroughly analyzed and extended (Shang et al. 2018; Hu et al. 2020; Sung et al. 2020). However, a limitation of SGD is that the learning rate must be manually defined and it does not take any measures to improve the gradient direction.

Second-Order Methods. To address the manual tuning of the learning rates in first-order methods and to improve the convergence rate, second-order methods rely on the Hessian matrix. However, this matrix grows quadratically with the number of model parameters. Thus, most work reduces the computational complexity by approximating the Hessian (Goldfarb, Ren, and Bahamou 2020; Botev, Ritter, and Barber 2017). A number of methods looks at combining the second-order information in different ways. For example, (Roux and Fitzgibbon 2010) combine Newton's method and natural gradient. (Sohl-Dickstein, Poole, and Ganguli 2014) combine SGD with the second-order curvature information leveraged by quasi-Newton methods. (Yao et al. 2021) dynamically incorporate the curvature of the loss via adaptive estimates of the Hessian. (Henriques et al. 2019) propose a method that does not require to store the Hessian at all. In contrast to these methods, KOALA does not compute second-order derivatives, but focuses on modeling noise in the objective function.

Adaptive. An alternative to using second-order derivatives is to automatically adjust the step-size during the optimization. The adaptive selection of the update step-size has been based on several principles, including: the local sharpness of the loss function (Yue, Nouiehed, and Kontar 2020), incorporating a line search approach (Vaswani et al. 2019; Mutschler and Zell 2020; Mahsereci and Hennig 2015), the gradient change speed (Dubey et al. 2020), a "belief" in the current gradient direction (Zhuang et al. 2020), the linearization of the loss (Rolinek and Martius 2018), the per-component unweighted mean of all historical gradients (Daley and Amato 2020), handling noise by preconditioning based on a covariance matrix (Ida, Fujiwara, and Iwamura 2017), learning the update-step size (Wu, Ward, and Bottou 2020), looking ahead at the sequence of fast weights generated by another optimizer (Zhang et al. 2019a). A new family of sub-gradient methods called AdaGrad is presented in (Duchi, Hazan, and Singer 2011). AdaGrad dynamically incorporates knowledge of the geometry of the data observed in earlier iterations. (Tieleman and Hinton 2012) introduce RmsProp, further extended in (Mukkamala and Hein 2017) with logarithmic regret bounds for strongly convex functions. (Zeiler 2012) propose a per-dimension learning rate method for gradient descent called AdaDelta. (Kingma and Ba 2015) introduce Adam, based on adaptive estimates of lower-order moments. A wide range of variations and extensions of the original Adam optimizer has also been proposed (Liu, Wu, and Mozafari 2020; Reddi, Kale, and Kumar 2018; Heo et al. 2021; Loshchilov and Hutter 2019;

Chen et al. 2019; Liu et al. 2020; Luo, Xiong, and Liu 2019; Wang et al. 2020). Recent work proposes to decouple the weight decay (Granziol et al. 2021; Ginsburg et al. 2020). (Chen et al. 2020) introduces a partially adaptive momentum estimation method. Some recent work also focuses on the role of gradient clipping et al. (Zhang et al. 2020a,b). In most prior work, adaptivity comes from the introduction of extra hyper-parameters. In our case, this property is a direct byproduct of the Kalman filtering framework.

Kalman filtering. The use of Kalman filtering theory and methods for the training of neural networks is not new. For example, (Ismail et al. 2018) relates to our KOALA-V as the authors also work with scalar measurements. However, our approach differs in several ways as we introduce a way to incorporate Momentum, learning rate scheduling, noise adaptivity and provide a convergence analysis. More recently, (Shashua and Mannor 2019) incorporated Kalman filtering for Value Approximation in Reinforcement Learning. (Ollivier 2019) recovered the exact extended Kalman filter equations from first principles in statistical learning: the Extended Kalman filter is equal to Amari's online natural gradient, applied in the space of trajectories of the system. (de Vilmarest and Wintenberger 2020) applied the Extended Kalman filter to linear and logistic regressions. (Takenga et al. 2004) compared GD to methods based on either Kalman filtering or the decoupled Kalman filter. To summarize, all of these prior Kalman filtering approaches either focus on a specific non-general formulation or face difficulties when scaling to high-dimensional parameter spaces of large-scale neural models.

Risk Minimization through Kalman Filtering

In machine learning, we are interested in minimizing the *expected risk*

$$\min_{x \in R^n} L(x), \quad \text{where } L(x) \doteq E_{\xi \sim p(\xi)}[\ell(\xi; x)], \quad (1)$$

with respect to some loss ℓ that is a function of both the data $\xi \in \mathbb{R}^d$ with d the data dimensionality, $p(\xi)$ is the probability density function of ξ, and the model parameters $x \in \mathbb{R}^n$ (*e.g.*, the weights of a neural network), where n is the number of parameters in the model. We consider the big data case, which is of common interest today, where both $d \gg 1$ and $n \gg 1$ (*e.g.*, in the order of 10^6). For notational simplicity, we do not distinguish the supervised and unsupervised learning cases by concatenating all data into a single vector ξ (*e.g.*, in the case of image classification we stack in ξ both the input image and the output label). In practice, we have access to only a finite set of samples and thus settle for the *empirical risk* optimization

$$\min_{x \in R^n} \hat{L}(x), \text{where } \hat{L}(x) \doteq \frac{1}{m} \sum_{i=1}^{m} \ell(\xi_i; x), \quad (2)$$

and $\xi_i \sim p(\xi)$, for $i = 1, \ldots, m$, are our training dataset samples. The above risk is often optimized iteratively via a gradient descent method, because a closed form solution (*e.g.*, as with least squares) is typically not available.

Moreover, since in current datasets m, the number of training samples, can be very large, the computation of the

gradient of the empirical risk at each iteration is too demanding. To address this issue, the stochastic gradient descent (SGD) method (Robbins and Monro 1951) minimizes the following *minibatch risk* at each iteration time k

$$\hat{L}_k(x) \doteq \frac{1}{|\mathcal{C}_k|} \sum_{i \in \mathcal{C}_k} \ell(\xi_i; x), \tag{3}$$

where $\mathcal{C}_k \subset [1, \ldots, m]$ is a random subset of the dataset indices. Given a random initialization for x_0, SGD builds a sequence $\{x_k\}_{k=0,\ldots,T}$ by recursively updating the parameters x_k so that they decrease the k-th loss $\hat{L}_k$, *i.e.*, for $k = 0, \ldots, T-1$

$$x_{k+1} = x_k - \eta \nabla \hat{L}_k(x_k), \tag{4}$$

where $\nabla \hat{L}_k(x_k)$ denotes the gradient of $\hat{L}_k$ with respect to x and computed at x_k, and $\eta > 0$ is the *learning rate*, which regulates the speed of convergence.

Modeling Noise in Risk Minimization

In KOALA, we directly model the statistical properties of the minibatch risk $\hat{L}_k$ as a function of the empirical risk $\hat{L}$. To relate $\hat{L}_k$ to $\hat{L}$ we start by looking at the relation between $\hat{L}$ and the expected risk L. First, we point out that $\hat{L}$ is the sample mean of L. Then, we recall that, because of the central limit theorem, $\hat{L}$ converges to a Gaussian distribution with mean L as $m \to \infty$. The same analysis can be applied to the minibatch risk $\hat{L}_k$. $\hat{L}_k$ is a sample mean of $\hat{L}$ and as $|\mathcal{C}_k| \to m$, $\hat{L}_k$ converges to $\hat{L}$. Thus, the distribution of each $\hat{L}_k$ will tend towards a Gaussian random variable with mean $\hat{L}$. Finally, we can write $\forall x$

$$\hat{L}_k(x) \simeq \hat{L}(x) - v_k, \tag{5}$$

where the scalar noise variable $v_k \sim \mathcal{N}(0, R_k)$, is a zero-mean Gaussian with variance R_k. Later, we will show how to obtain an online estimate of R_k.

Risk Minimization as Loss Adaptivity

Consider a model with parameters $\hat{x}$. For example, $\hat{x}$ could be chosen as one of the solutions of the optimization (2), *i.e.*, such that $\hat{L}(\hat{x}) = \min_x \hat{L}(x)$. However, more in general, one can define $\hat{L}(\hat{x}) \doteq \hat{L}^{\text{target}}$, for some feasible $\hat{L}^{\text{target}}$. Let us now define the problem of finding x_k such that

$$\hat{L}_k(x_k) = \hat{L}^{\text{target}} - v_k, \tag{6}$$

for all k and where v_k depends on $\hat{x}$. The above formulation allows us to also solve the optimization in (2). Rather than explicitly finding the minimum of a function, in KOALA we look for the model parameters that *adapt* the minibatch risk to a given value on average. However, to solve (2) we need $\min_x \hat{L}(x)$, which is unknown. As an alternative, we iteratively approximate $\min_x \hat{L}(x)$ with a sequence of $\hat{L}_k^{\text{target}}$ that converges to it. For example, by applying Theorem 1 (see next sections), the approximation $\min_x \hat{L}(x) \simeq \hat{L}_k^{\text{target}} \doteq \hat{L}_k(x_k) - k^{-1}$ will ensure the convergence of KOALA as k grows.

Kalman Filtering for Stochastic Optimization

Eq. (6) can be interpreted as a noisy observation of some unknown model parameters x, which we want to identify. Kalman filtering is a natural solution to this task. As discussed in the Prior Work section, there is an extensive literature on the application of Kalman filtering as a stochastic gradient descent algorithm. However, these methods differ from our approach in several ways. For instance, Vuckovic (Vuckovic 2018) uses the gradients as measurements. Thus, this method requires large matrix inversions, which are not scalable to the settings we consider in this paper and that are commonly used in deep learning (see section 3.3 in (Vuckovic 2018)). KOALA works instead directly with the scalar risks $\hat{L}_k$ and introduces a number of computational approximations that make the training with large datasets and high dimensional data feasible.

Let us model the uncertainty of the identified parameters x_k as a Gaussian random variable with the desired target $\hat{x}_k$ as mean. Then, a dynamical system for a sequence x_k is

$$x_k = x_{k-1} + w_{k-1} \tag{7}$$
$$\hat{L}_k^{\text{target}} = \hat{L}_k(x_k) + v_k. \tag{8}$$

Here, $w_k \sim \mathcal{N}(0, Q_k)$ is modeled as a zero-mean Gaussian variable with covariance Q_k. The dynamical model implies that the mean of the state x_k does not change when it has adapted the mean minibatch risk to the target observation $\hat{L}_k^{\text{target}}$. The equations (7) and (8) describe a dynamical system suitable for Kalman filtering (Kalman 1960). For completeness, we briefly recall here the general equations for an Extended Kalman filter

$$x_k = f_k(x_{k-1}) + w_{k-1} \tag{9}$$
$$z_k = h_k(x_k) + v_k, \tag{10}$$

where x_k are also called the *hidden state*, $z_k \in \mathbb{R}^s$ are the observations, f_k and h_k are functions that describe the state transition and the measurement dynamics respectively. The zero-mean Gaussian noises added to each equation must also be independent of the hidden state. The Extended Kalman filter infers optimal estimates of the state variables from the previous estimates of x_k and the last observation z_k. Moreover, it also estimates the a posteriori covariance matrix P_k of the state. This is done in two steps: Predict and Update, which we recall in Table 1.

If we directly apply the equations in Table 1 to our equations (7) and (8), we would immediately find that the posterior covariance P_k is an $n \times n$ matrix, which would be too large to store and update for n values used in practice. Hence, we approximate P_k as a scaled identity matrix. Since the update equation for the posterior covariance requires the computation of $K_k H_k = P_k H_k^\top H_k S_k^{-1}$, we need to approximate $H_k^\top H_k$ also with a scaled identity matrix. We do this by using its largest eigenvalue, *i.e.*,

$$H_k^\top H_k \approx |H_k|^2 I_{n \times n} = \left| \nabla_x \hat{L}_k(\hat{x}_k) \right|^2 I_{n \times n}, \tag{11}$$

where $I_{n \times n}$ denotes the $n \times n$ identity matrix. Because we work with a scalar loss $\hat{L}_k$, the innovation covariance S_k is a

Table 1: Extended Kalman filter recursive equations for a posteriori state and covariance estimation.

Predict:	$\hat{x}_k = f_k(x_{k-1})$ $\hat{P}_k = F_k P_{k-1} F_k^\top + Q_k$
Update:	$S_k = H_k \hat{P}_k H_k^\top + R_k$ $K_k = \hat{P}_k H_k^\top S_k^{-1}$ $x_k = \hat{x}_k + K_k(z_k - h_k(\hat{x}_k))$ $P_k = (I - K_k H_k)\hat{P}_k$
with:	$H_k \doteq \nabla h_k(\hat{x}_k)$ $F_k \doteq \nabla f_k(x_{k-1})$

Algorithm 1: KOALA-V (Vanilla)

Initialize x_0, P_0, Q and R
for k in range$(1, T)$ **do**
    Predict:
        $\hat{x}_k = x_{k-1}$; $\hat{P}_k = P_{k-1} + Q$
    Update:

$$x_k = \hat{x}_k - \frac{\hat{P}_k(\hat{L}_k(\hat{x}_k) - \hat{L}_k^{\text{target}})}{\hat{P}_k|\nabla \hat{L}_k(\hat{x}_k)|^2 + R} \nabla \hat{L}_k(\hat{x}_k) \quad (12)$$

$$P_k = \frac{R}{\hat{P}_k|\nabla \hat{L}_k(\hat{x}_k)|^2 + R} \hat{P}_k \quad (13)$$

end for
return x_K

---

scalar and thus it can be easily inverted. The general framework introduced so far is very flexible and allows several extensions. The parameter estimation method obtained from equations (7) and (8) is a special case of KOALA that we call *KOALA-V* (Vanilla), and summarize in Algorithm 1.

Notice that the update in eq. (12) is quite similar to the SGD update (4), where the learning rate η depends on $\hat{P}_k$, $\hat{L}_k(\hat{x}_k) - \hat{L}_k^{\text{target}}$, $\nabla \hat{L}_k(\hat{x}_k)$ and R. Thus, the learning rate in KOALA-V automatically adapts over time to the current loss value, its gradient and estimation of the parameters, while in SGD it must be manually tuned.

Incorporating Momentum Dynamics

A first important change we introduce is the incorporation of Momentum (Sutskever et al. 2013). Within our notation, this method could be written as

$$p_k = \kappa p_{k-1} - \eta \nabla \hat{L}(x_{k-1}) \quad (14)$$

$$x_k = x_{k-1} + p_k, \quad (15)$$

where p_k are so called *momentums* or velocities, that accumulate the gradients from the past. The parameter $\kappa \in (0, 1)$, commonly referred to as *momentum rate*, controls the trade-off between current and past gradients. Such updates claim to stabilize the training and prevent the parameters from getting stuck at local minima.

To incorporate the idea of Momentum within the KOALA framework, one can simply introduce the state velocities and

define the following dynamics

$$x_k = x_{k-1} + p_{k-1} + w_{k-1} \quad (16)$$

$$p_k = \kappa p_{k-1} + u_{k-1} \quad (17)$$

$$\hat{L}_k^{\text{target}} = \hat{L}_k(x_k) + v_k, \quad (18)$$

where $p_k \in \mathbb{R}^n$ and u_{k-1} is a zero-centered Gaussian random variable.

One can rewrite these equations again as Kalman filter equations by combining the parameters x_k and the velocities p_k into one state vector $\bar{x}_k = [x_k, p_k]$ and similarly for the state noise $\zeta_{k-1} = [w_{k-1}, u_{k-1}]$. This results in the following dynamical system

$$\bar{x}_k = F\bar{x}_{k-1} + \zeta_{k-1} \quad (19)$$

$$\hat{L}_k^{\text{target}} = \hat{L}_k(\Pi\bar{x}_k) + v_k, \quad (20)$$

where $F = \begin{bmatrix} 1 & 1 \\ 0 & \kappa \end{bmatrix} \otimes I_{n \times n}$, $\Pi = \begin{bmatrix} 1 & 0 \end{bmatrix} \otimes I_{n \times n}$, and $\otimes$ denotes the Kronecker product. Similarly to the KOALA-V algorithm, we also aim to drastically reduce the dimensionality of the posterior covariance, which now is a $2n \times 2n$ matrix. We approximate P_k with the following form $\begin{bmatrix} \sigma_{x,k}^2 & \sigma_{c,k}^2 \\ \sigma_{c,k}^2 & \sigma_{p,k}^2 \end{bmatrix} \otimes I_{n \times n}$, where $\sigma_{x,k}^2, \sigma_{c,k}^2, \sigma_{p,k}^2$ are scalars. In this formulation we have that $H_k = \begin{bmatrix} \nabla \hat{l}_k^\top & \mathbf{0}^\top \end{bmatrix}$ and thus our approximation for the Kalman update of the posterior covariance will use

$$H_k^\top H_k \approx \begin{bmatrix} |\nabla \hat{L}_k|^2 I_{n \times n} & \mathbf{0} \\ \mathbf{0} & \mathbf{0} \end{bmatrix}. \quad (21)$$

The remaining equations follow directly from the application of Table 1. We call this method the *KOALA-M* (Momentum) algorithm.

Estimation of the Measurement and State Noise

In the KOALA framework we model the noise in the observations and the state transitions with zero-mean Gaussian variables with covariances R_k and Q_k respectively. So far, we assumed that these covariances were given and constant. However, they can also be estimated online, and lead to more accurate state and posterior covariance estimates. For R_k we use the following running average

$$R_k = \beta_R R_{k-1} + (1 - \beta_R)\frac{1}{|\mathcal{C}_k|} \sum_{i \in \mathcal{C}_k} \left(\ell(\xi_i; x_k) - \hat{L}_k^{\text{target}} \right)^2, \quad (22)$$

where we set $\beta_R = 0.9$. Similarly, for the covariance $Q_k \doteq \text{diag}\{q_{x,k}^2 I_{n \times n}, q_p^2 I_{n \times n}\}$, the online update for $q_{x,k}^2$ is

$$x_{\text{avg}} = \beta_x x_{\text{avg}} + (1 - \beta_x)x_{k-1} \quad (23)$$

$$q_{x,k}^2 = \frac{1}{n}|x_{k-1} - x_{\text{avg}}|^2, \quad (24)$$

where we set $\beta_x = 0.9$. This adaptivity of the noise helps both to reduce the number of hyper-parameters and to obtain better convergence.

Learning Rate Scheduling

In both the KOALA-V and the KOALA-M algorithms, the update equation for the state estimate needs $\hat{L}_k^{\text{target}}$ (see *e.g.*, eq. (12)). Thanks to Theorem 1 (see next section), we have the option to change $\hat{L}_k^{\text{target}}$ progressively with the iteration time k. For instance, we could set $\hat{L}_k^{\text{target}} = (1 - \varepsilon_k)\hat{L}_k(x_k)$, for some choice of the sequence ε_k. Using this term is equivalent to setting $\hat{L}_k^{\text{target}} = 0$ and scaling the learning rate by ε_k in eq. (12), as in many SGD implementations (Goffin 1977; Loshchilov and Hutter 2017). Notice the very different interpretation of the schedule in the case of KOALA, where we gradually decrease the target risk.

Layer-wise Approximations

Let us consider the optimization problem specifically for large neural networks. We denote with B the number of layers in a network. Next, we substitute the scalar observation eq. (20) with a B-dimensional vector of identical observations. The i-th entry in this B-dimensional observation vector depends only on the variables of the i-th block of the network, while the other variables are frozen. Thus, while in the original definition the measurement equation had H_k as an n-dimensional vector, under the proposed approximation H_k is a $B \times n$ block-diagonal matrix. Under these assumptions, the update equation (12) for both the *KOALA-V* and the *KOALA-M* algorithm will split into B layer-wise equations, where each separate equation incorporates only the gradients with respect to the parameters of a specific layer. Additionally to this, now the matrix $H_k^\top H_k S_k^{-1}$ also yields B separate blocks (one per observation), each of which gets approximated by the corresponding largest block eigenvalue. Finally, the maximum of these approximations gives us the approximation of the whole matrix

$$H_k^\top H_k S_k^{-1} \approx \max_{1 \le i \le B} \frac{|\nabla_{b_i} \hat{L}_k|^2}{S_k^{(i)}}, \tag{25}$$

where b_i is the subset of parameters corresponding to the i-th layer and $S_k^{(i)}$ is the innovation covariance corresponding to only the i-th measurement. We observe that this procedure induces better convergence in training. For more details, see the supplementary material.

Convergence Analysis

Our convergence analysis for KOALA builds on the framework introduced in (Bertsekas and Tsitsiklis 2000). The analysis is based on a general descent algorithm

$$x_{k+1} = x_k + \gamma_k(s_k + u_k), \tag{26}$$

where γ_k is the step size, s_k is a descent direction and u_k is a noise term. Here, s_k is related to the gradient of the empirical risk $\hat{L}$ and u_k satisfies some regularity conditions. In our algorithm, we also skip all update steps when the norm of the gradient of a minibatch loss is lower than a threshold. This is because such observations provide almost no information to the state. Because of this rule we can thus guarantee that $|\nabla \hat{L}_k(x)| \ge g$ for some $g > 0$. Given these settings, we

analyze the evolution of $\hat{P}_k$, showing that it stays within two positive bounds. Further, we show that the gradient of the loss goes to 0 as $k \to \infty$. This result is formalized for the KOALA-V and summarized in the following theorem with two choices for the target risks.

Theorem 1. *Let $\hat{L}(x)$ be a continuously differentiable function and $\nabla \hat{L}(x)$ be Lipschitz-continuous. Assume that $g \le |\nabla \hat{L}_k(x)| \le G$ for all x and k, where $g, G > 0$ and $\hat{L}_k(x)$ is the minibatch loss. Let us choose the target risk*

(a) $\qquad \hat{L}_k^{target} = \hat{L}_k(x_k) - \varepsilon_k, \tag{27}$

(b) $\qquad \hat{L}_k^{target} = (1 - \varepsilon_k)\hat{L}_k(x_k), \tag{28}$

where ε_k is any sequence satisfying $\sum_{k=0}^{\infty} \varepsilon_k = \infty$ and $\sum_{k=0}^{\infty} \varepsilon_k^2 < \infty$. Then $\hat{L}(x_k)$ converges to a finite value and $\lim_{k \to \infty} \nabla \hat{L}(x_k) = 0$.

Proof. See supplementary material. $\qquad \square$

Ablations

In this section we ablate the following features and parameters of both *KOALA-V* and *KOALA-M* algorithms: the dynamics of the weights and velocities, the initialization of the posterior covariance matrix and the adaptivity of the state noise estimators. In some ablations we also separately test the *KOALA-M* algorithm with adaptive Q_k. Also, we show that our algorithm is relatively insensitive to different batch sizes and weight initializations.

We evaluate our optimization methods by computing the test performance achieved by the model obtained with the estimated parameters. Although such performance may not uniquely correlate to the performance of our method, as it might be affected also by the data, model and regularization, it is a useful indicator. In all the ablations, we choose the classification task on CIFAR-100 (Krizhevsky and Hinton 2009) with ResNet18 (He et al. 2016). We train all the models for 100 epochs and decrease the learning rate by a factor of 0.2 every 30 epochs.

For the last two ablations and in the Experiments section, we use the *KOALA-M* algorithm with $\kappa = 0.9$, adaptive R_k and Q_k, initial posterior covariance parameters $\sigma_{x,0}^2 = \sigma_{p,0}^2 = 0.1$ and $\sigma_{c,0}^2 = 0$.

Impact of the state dynamics and noise adaptivity. We compare the *KOALA-V* algorithm (*i.e.*, constant dynamics) to the *KOALA-M* (*i.e.*, with velocities). Additionally, we ablate the κ, *i.e.*, the decay rate of the velocities. The results are shown in Table 2. We observe that the use of velocities with a calibrated moment has a positive impact on the estimated parameters. Further, with adaptive noise estimations there is no need to set their initial values, which reduces the number of hyper-parameters to tune.

Posterior covariance initialization. The KOALA framework requires to initialize the matrix P_0. In the case of the *KOALA-V* algorithm, we approximate the posterior covariance with a scaled identity matrix, *i.e.*, $P_k = \sigma_k^2 I_{n \times n}$, where $\sigma_k \in \mathbb{R}$. In the case of *KOALA-M*, we approximate P_k with a 2×2 block diagonal matrix with $\sigma_{x,0}^2 I_{n \times n}$ and $\sigma_{p,0}^2 I_{n \times n}$ on the diagonal, where $\sigma_{x,0}, \sigma_{p,0} \in \mathbb{R}$. In this section we ablate

Table 2: Ablation of the state dynamics and noise adaptivity.

KOALA	κ	Top-1 Err.	Top-5 Err.
V	-	24.17	7.06
M	0.50	27.20	8.29
M	0.90	23.38	6.77
M (adapt. Q_k)	0.50	28.25	8.91
M (adapt. Q_k)	0.90	23.39	6.50

Table 3: Ablation of the posterior covariance initialization.

Parm.	Value	KOALA	Top-1 Err.	Top-5 Err.
σ_0^2	0.01	V	24.42	7.23
	0.10	V	24.17	7.06
	1.00	V	24.69	7.36
$\sigma_{x,0}^2$	0.01	M (adapt. Q_k)	23.67	6.81
	0.10	M (adapt. Q_k)	23.39	6.50
	1.00	M (adapt. Q_k)	23.82	6.53
$\sigma_{p,0}^2$	0.01	M (adapt. Q_k)	23.37	7.13
	0.10	M (adapt. Q_k)	23.39	6.50
	1.00	M (adapt. Q_k)	24.24	7.40

$\sigma_{x,0}$, $\sigma_{p,0}$ and σ_0 to show that the method quickly adapts to the observations and the initialization of P_k does not have a significant impact on the final accuracy achieved with the estimated parameters. The results are given in Table 3.

Batch size. Usually one needs to adapt the learning rate to the chosen minibatch size. In this experiment, we change the batch size in the range $[32, 64, 128, 256]$ and show that KOALA-M adapts to it naturally. Table 4 shows that the accuracy of the model does not vary significantly with a varying batch size, which is a sign of stability.

Weight initialization. Like with the batch size, we use different initialization techniques to show that the algorithm is robust to them. We apply the same initializations to SGD for comparison. We test Kaiming Uniform (He et al. 2015), Orthogonal (Saxe, McClelland, and Ganguli 2014), Xavier Normal (Glorot and Bengio 2010), Xavier Uniform (Glorot and Bengio 2010). The results are shown in Table 5.

Experiments

We evaluate KOALA-M on different tasks, including image classification (on CIFAR-10, CIFAR-100 and ImageNet (Russakovsky et al. 2015)), generative learning and language modeling. For all these tasks, we report the quality metrics on the validation sets to compare KOALA-M to the commonly used optimizers. We find that KOALA-M outperforms or is on par with the existing methods, while requiring fewer hyper-parameters to tune. We will make our code available.

CIFAR-10/100 Classification. We first evaluate KOALA-M on CIFAR-10 and CIFAR-100 using the popular ResNets (He et al. 2016) and WideResNets (Zagoruyko and Komodakis 2016) for training. We compare our results with the ones obtained with commonly used existing optimiza-

Table 4: Ablation of the batch size used for training. Classification error on CIFAR-100 with ResNet18.

Batch Size	Top-1 Error	Top-5 Error
32	24.59	7.13
64	23.11	6.93
128	23.39	6.50
256	24.34	7.59

Table 5: Ablation of different weight initializations. Classification error on CIFAR-100 with ResNet18.

Initialization	Optimizer	Top-1 Error	Top-5 Error
Xavier-Normal	SGD	26.71	7.59
	KOALA-M	23.34	6.78
Xavier-Uniform	SGD	26.90	7.97
	KOALA-M	23.40	6.85
Kaiming-Uniform	SGD	27.82	7.95
	KOALA-M	23.35	6.76
Orthogonal	SGD	26.83	7.59
	KOALA-M	23.27	6.63

tion algorithms, such as SGD with Momentum and Adam. For SGD we set the momentum rate to 0.9, which is the default for many popular networks, and for Adam we use the default parameters $\beta_1 = 0.9, \beta_2 = 0.999, \epsilon = 10^{-8}$. In all experiments on CIFAR-10/100, we use a batch size of 128 and basic data augmentation (random horizontal flipping and random cropping with padding by 4 pixels). For each configuration we have two runs for 100 and 200 epochs respectively. For SGD we start with a learning rate equal to 0.1, for Adam to 0.0003 and 1.0 for KOALA-M. For the 100-epochs run on CIFAR-10 (CIFAR-100) we decrease the learning rate by a factor of 0.1 (0.2) every 30 epochs. For 200-epochs on CIFAR-10 we decrease the learning rate only once at epoch 150 by the factor of 0.1. For the 200-epoch training on CIFAR-100 the learning rate is decreased by a factor of 0.2 at epochs 60, 120 and 160. For all the algorithms, we additionally use a weight decay of 0.0005. To show the benefit of using KOALA-M for training on classification tasks, we report the Top-1 and Top-5 errors on the validation set. For both the 100-epochs and 200-epochs configurations, we report the mean error among 3 runs with 3 different random seeds. Note that the 100/200-epochs configurations are not directly comparable due to the different learning rate schedules. The results are reported in Table 6. For more comparisons and training plots see the Supplementary material.

ImageNet Classification. Following (Loshchilov and Hutter 2019), we train a ResNet50 (He et al. 2016) on 32×32 downscaled images with the most common settings: 100 epochs of training with learning rate decrease of 0.1 after every 30 epochs and a weight decay of 0.0001. We use random cropping and random horizontal flipping during training and we report the validation accuracy on single center crop images. As shown in Table 6, our model achieves a comparable accuracy to SGD, but without any task-specific hyper-parameter tuning.

Table 6: Results on CIFAR-10, CIFAR-100 and ImageNet-32 datasets for 100 and 200 epochs runs.

Dataset	Architecture	Method	100-epochs		200-epochs	
			Error			
			Top-1	Top-5	Top-1	Top-5
CIFAR-10	ResNet-18	SGD	**5.60**	**0.16**	7.53	0.29
		Adam	6.58	0.28	6.46	0.28
		KOALA-M	5.69	0.21	**5.46**	**0.25**
	ResNet-50	SGD	**6.37**	**0.19**	8.10	0.27
		Adam	6.28	0.24	**5.97**	0.28
		KOALA-M	7.29	0.24	6.31	**0.13**
	W-ResNet-50-2	SGD	6.08	**0.15**	7.60	0.24
		Adam	**6.02**	0.19	5.90	0.26
		KOALA-M	6.83	0.19	**5.36**	**0.12**
CIFAR-100	ResNet-18	SGD	23.50	6.48	22.44	**5.99**
		Adam	26.30	7.85	25.61	7.74
		KOALA-M	**23.38**	**6.70**	**22.22**	6.13
	ResNet-50	SGD	25.05	6.74	22.06	5.71
		Adam	24.95	6.96	24.44	6.81
		KOALA-M	**22.34**	**5.96**	**21.03**	**5.33**
	W-ResNet-50-2	SGD	23.83	6.35	22.47	5.96
		Adam	23.73	6.64	24.04	7.06
		KOALA-M	**21.25**	**5.35**	**20.73**	**5.08**
ImageNet-32	ResNet-50	SGD	**34.07**	**13.38**	-	-
		KOALA-M	34.99	14.06	-	-

Table 7: Comparison of different optimizers on CIFAR-100 classification task with 100-epochs configuration. Mean errors across 3 runs with different random seeds are reported.

Optimizer	Top-1 Err.	Top-5 Err.
Yogi (Reddi et al. 2018)	33.99	10.90
Adamax (Kingma and Ba 2015)	32.42	10.74
AdamW (Loshchilov and Hutter 2019)	27.23	7.98
AdamP (Heo et al. 2021)	26.62	7.61
Amsgrad (Reddi, Kale, and Kumar 2018)	25.27	6.78
Fromage (Bernstein et al. 2020)	24.65	6.71
Adabelief (Zhuang et al. 2020)	23.07	6.05
KOALA-M	**22.34**	**5.96**

Table 8: Classification error on CIFAR-100 with ResNet50. Average of 3 runs with different random seeds is reported.

Optimizer	LA(SGD)	LA(Adam)	SWA(SGD)	SWA(Adam)	KOALA-M	SWA(KOALA-M)	LA(KOALA-M)
Top-1	23.36	24.90	24.24	24.49	**22.34**	22.29	**21.70**

Comparison to more recent algorithms. We compare KOALA-M with a wider range of optimizers on the CIFAR-100 classification with ResNet50 in the 100-epochs configuration. We used the same learning rate schedule as in the previous section and set the hyperparameters for the other algorithms to the ones reported by the authors. For Yogi (Reddi et al. 2018) we set the learning rate to $10^{-2}, \beta_1 = 0.9, \beta_2 = 0.999$ and $\epsilon = 10^{-3}$, as suggested in the paper. For Adamax (Kingma and Ba 2015), AdamW (Loshchilov and Hutter 2019), AdamP (Heo et al. 2021) and Amsgrad (Reddi, Kale, and Kumar 2018) we use the same hyperparameters as for Adam. For Fromage (Bernstein et al. 2020) we set the learning rate to 10^{-2}, as suggested on the project's github page[1]. For Adabelief (Zhuang et al. 2020) we follow the hyperparameters reported in the official implementation[2] . The results are shown in Table 7.

In Table 8, we show that KOALA-M is compatible with such auxiliary methods as Lookahead (LA) (Zhang et al. 2019b) and SWA (Izmailov et al. 2018). For LA we used SGD and Adam with initial learning rates equal to 0.1 and 0.0003 respectively as the inner optimizers and set the hyperparameters α and k to 0.8 and 5 respectively, as suggested by the authors. We used SWA with both SGD and Adam inner optimizers averaging every 5 epochs starting from epoch 75. Additionally, we apply LA and SWA to KOALA-M. All experiments are for CIFAR-100 classification with ResNet50. Training is done in 100-epochs config-

uration.

Memory and time complexity. KOALA-M needs at most $2\times$ the size of the network in additional memory for storing x_{avg} and state velocities p_k. Also, since we do not store the full state covariance matrices and use at most 2×2 matrices in update equations, the computational complexity of our algorithm is **linear** with respect to the network parameters. For numerical results see the Supplementary material.

GANs and language modeling. KOALA-M also works well for training GANs (Goodfellow et al. 2014) and on NLP tasks. For numerical results, see the supplementary material.

Conclusions

We have introduced KOALA, a novel Kalman filtering-based approach to stochastic optimization. KOALA is suitable to train modern neural network models on current large scale datasets with high-dimensional data. The method can self-tune and is quite robust to wide range of training settings. Moreover, we design KOALA so that it can incorporate optimization dynamics such as those in Momentum and Adam, and learning rate schedules. The efficacy of this method is demonstrated on several experiments in image classification, image generation and language processing.

References

Bernstein, J.; Vahdat, A.; Yue, Y.; and Liu, M.-Y. 2020. On the distance between two neural networks and the stability of learning. In *NeurIPS*.

Bertsekas, D. P.; and Tsitsiklis, J. N. 2000. Gradient convergence in gradient methods with errors. *SIAM Journal on Optimization*, (3).

---

[1] https://github.com/jxbz/fromage#voulez-vous-du-fromage

[2] https://github.com/juntang-zhuang/Adabelief-Optimizer# hyper-parameters-in-pytorch

Botev, A.; Ritter, H.; and Barber, D. 2017. Practical Gauss-Newton Optimisation for Deep Learning. In *ICML*.

Bottou, L.; Curtis, F. E.; and Nocedal, J. 2018. Optimization methods for large-scale machine learning. *Siam Review*, (2).

Chen, J.; Zhou, D.; Tang, Y.; Yang, Z.; Cao, Y.; and Gu, Q. 2020. Closing the Generalization Gap of Adaptive Gradient Methods in Training Deep Neural Networks. ArXiv:1806.06763.

Chen, X.; Liu, S.; Sun, R.; and Hong, M. 2019. On the Convergence of A Class of Adam-Type Algorithms for Non-Convex Optimization. In *ICLR*.

Daley, B.; and Amato, C. 2020. Expectigrad: Fast Stochastic Optimization with Robust Convergence Properties. ArXiv:2010.01356.

de Vilmarest, J.; and Wintenberger, O. 2020. Stochastic Online Optimization using Kalman Recursion. ArXiv:2002.03636.

Dubey, S. R.; Chakraborty, S.; Roy, S. K.; Mukherjee, S.; Singh, S. K.; and Chaudhuri, B. B. 2020. diffGrad: An Optimization Method for Convolutional Neural Networks. *IEEE Transactions on Neural Networks and Learning Systems*.

Duchi, J.; Hazan, E.; and Singer, Y. 2011. Adaptive Subgradient Methods for Online Learning and Stochastic Optimization. *Journal of Machine Learning Research*.

Ginsburg, B.; Castonguay, P.; Hrinchuk, O.; Kuchaiev, O.; Lavrukhin, V.; Leary, R.; Li, J.; Nguyen, H.; Zhang, Y.; and Cohen, J. M. 2020. Stochastic Gradient Methods with Layerwise Adaptive Moments for Training of Deep Networks. ArXiv:1905.11286.

Glorot, X.; and Bengio, Y. 2010. Understanding the difficulty of training deep feedforward neural networks. In *Proceedings of the Thirteenth International Conference on Artificial Intelligence and Statistics*.

Goffin, J. 1977. On convergence rates of subgradient optimization methods. *Mathematical Programming.*, 13: 329–347.

Goldfarb, D.; Ren, Y.; and Bahamou, A. 2020. Practical Quasi-Newton Methods for Training Deep Neural Networks. In *NeurIPS*.

Goodfellow, I.; Pouget-Abadie, J.; Mirza, M.; Xu, B.; Warde-Farley, D.; Ozair, S.; Courville, A.; and Bengio, Y. 2014. Generative Adversarial Nets. In *NeurIPS*.

Granziol, D.; Wan, X.; Albanie, S.; and Roberts, S. 2021. Beyond SGD: Iterate Averaged Adaptive Gradient Method. ArXiv:2003.01247.

Haykin, S. 2004. *Kalman filtering and neural networks*. John Wiley & Sons.

He, K.; Zhang, X.; Ren, S.; and Sun, J. 2015. Delving Deep into Rectifiers: Surpassing Human-Level Performance on ImageNet Classification. In *ICCV*.

He, K.; Zhang, X.; Ren, S.; and Sun, J. 2016. Deep Residual Learning for Image Recognition. In *CVPR*.

Henriques, J. F.; Ehrhardt, S.; Albanie, S.; and Vedaldi, A. 2019. Small Steps and Giant Leaps: Minimal Newton Solvers for Deep Learning. In *ICCV*.

Heo, B.; Chun, S.; Oh, S. J.; Han, D.; Yun, S.; Kim, G.; Uh, Y.; and Ha, J.-W. 2021. AdamP: Slowing Down the Slowdown for Momentum Optimizers on Scale-invariant Weights. In *ICLR*.

Hu, Y.; Zhang, S.; Chen, X.; and He, N. 2020. Biased Stochastic First-Order Methods for Conditional Stochastic Optimization and Applications in Meta Learning. In *NeurIPS*.

Ida, Y.; Fujiwara, Y.; and Iwamura, S. 2017. Adaptive Learning Rate via Covariance Matrix Based Preconditioning for Deep Neural Networks. In *Proceedings of the Twenty-Sixth International Joint Conference on Artificial Intelligence*.

Ismail, M.; Attari, M.; Habibi, S.; and Ziada, S. 2018. Estimation theory and Neural Networks revisited: REKF and RSVSF as optimization techniques for Deep-Learning. *Neural Networks*, 108: 509–526.

Izmailov, P.; Podoprikhin, D.; Garipov, T.; Vetrov, D.; and Wilson, A. G. 2018. Averaging Weights Leads to Wider Optima and Better Generalization. *arXiv preprint arXiv:1803.05407*.

Kalman, R. E. 1960. A new approach to linear filtering and prediction problems. *Transactions of the ASME – Journal of Basic Engineering (Series D)*.

Kingma, D. P.; and Ba, J. 2015. Adam: A Method for Stochastic Optimization. In *ICLR*.

Krizhevsky, A.; and Hinton, G. 2009. Learning multiple layers of features from tiny images.

Liu, L.; Jiang, H.; He, P.; Chen, W.; Liu, X.; Gao, J.; and Han, J. 2020. On the Variance of the Adaptive Learning Rate and Beyond. In *ICLR*.

Liu, R.; Wu, T.; and Mozafari, B. 2020. Adam with Bandit Sampling for Deep Learning. In *NeurIPS*.

Loshchilov, I.; and Hutter, F. 2017. SGDR: Stochastic Gradient Descent with Warm Restarts. In *ICLR*.

Loshchilov, I.; and Hutter, F. 2019. Decoupled Weight Decay Regularization. In *ICLR*.

Luo, L.; Xiong, Y.; and Liu, Y. 2019. Adaptive Gradient Methods with Dynamic Bound of Learning Rate. In *ICLR*.

Mahsereci, M.; and Hennig, P. 2015. Probabilistic Line Searches for Stochastic Optimization. In *NeurIPS*.

Mukkamala, M. C.; and Hein, M. 2017. Variants of RMSProp and Adagrad with Logarithmic Regret Bounds. In *ICML*.

Mutschler, M.; and Zell, A. 2020. Parabolic Approximation Line Search for DNNs. In *NeurIPS*.

Ollivier, Y. 2019. The Extended Kalman Filter is a Natural Gradient Descent in Trajectory Space. ArXiv:1901.00696.

Patel, V. 2016. Kalman-Based Stochastic Gradient Method with Stop Condition and Insensitivity to Conditioning. *SIAM Journal on Optimization*.

Reddi, S.; Zaheer, M.; Sachan, D.; Kale, S.; and Kumar, S. 2018. Adaptive methods for nonconvex optimization. In *Proceeding of 32nd Conference on Neural Information Processing Systems (NIPS 2018)*.

Reddi, S. J.; Kale, S.; and Kumar, S. 2018. On the Convergence of Adam and Beyond. In *ICLR*.

Robbins, H.; and Monro, S. 1951. A Stochastic Approximation Method. *The Annals of Mathematical Statistics*, (3).

Rolinek, M.; and Martius, G. 2018. L4: Practical loss-based stepsize adaptation for deep learning. In *NeurIPS*.

Roux, N. L.; and Fitzgibbon, A. W. 2010. A fast natural Newton method. In *ICML*.

Russakovsky, O.; Deng, J.; Su, H.; Krause, J.; Satheesh, S.; Ma, S.; Huang, Z.; Karpathy, A.; Khosla, A.; Bernstein, M.; Berg, A. C.; and Fei-Fei, L. 2015. ImageNet Large Scale Visual Recognition Challenge. *Int. J. Comput. Vision*, 115(3).

Saxe, A. M.; McClelland, J. L.; and Ganguli, S. 2014. Exact solutions to the nonlinear dynamics of learning in deep linear neural networks. In *ICLR*.

Shang, F.; Zhou, K.; Liu, H.; Cheng, J.; Tsang, I. W.; Zhang, L.; Tao, D.; and Jiao, L. 2018. VR-SGD: A Simple Stochastic Variance Reduction Method for Machine Learning. arXiv:1802.09932.

Shashua, S. D.-C.; and Mannor, S. 2019. Trust Region Value Optimization using Kalman Filtering. ArXiv:1901.07860.

Sohl-Dickstein, J.; Poole, B.; and Ganguli, S. 2014. Fast large-scale optimization by unifying stochastic gradient and quasi-Newton methods. In *ICML*.

Sung, W.; Choi, I.; Park, J.; Choi, S.; and Shin, S. 2020. S-SGD: Symmetrical Stochastic Gradient Descent with Weight Noise Injection for Reaching Flat Minima. ArXiv:2009.02479.

Sutskever, I.; Martens, J.; Dahl, G.; and Hinton, G. 2013. On the importance of initialization and momentum in deep learning. In *ICML*.

Takenga, C. M.; Anne, K. R.; Kyamakya, K.; and Chedjou, J. C. 2004. Comparison of gradient descent method, Kalman filtering and decoupled Kalman in training neural networks used for fingerprint-based positioning. In *IEEE 60th Vehicular Technology Conference*.

Tieleman, T.; and Hinton, G. 2012. Lecture 6.5-rmsprop: Divide the gradient by a running average of its recent magnitude.

Vaswani, S.; Mishkin, A.; Laradji, I.; Schmidt, M.; Gidel, G.; and Lacoste-Julien, S. 2019. Painless Stochastic Gradient: Interpolation, Line-Search, and Convergence Rates. In *NeurIPS*.

Vuckovic, J. 2018. Kalman Gradient Descent: Adaptive Variance Reduction in Stochastic Optimization. ArXiv:1810.12273.

Wang, C.; Chen, X.; Smola, A.; and P Xing, E. 2013. Variance reduction for stochastic gradient optimization. *NeurIPS*.

Wang, G.; Lu, S.; Cheng, Q.; wei Tu, W.; and Zhang, L. 2020. SAdam: A Variant of Adam for Strongly Convex Functions. In *ICLR*.

Wu, X.; Ward, R.; and Bottou, L. 2020. WNGrad: Learn the Learning Rate in Gradient Descent. ArXiv:1803.02865.

Yao, Z.; Gholami, A.; Shen, S.; Keutzer, K.; and Mahoney, M. W. 2021. ADAHESSIAN: An Adaptive Second Order Optimizer for Machine Learning. *AAAI*.

Yue, X.; Nouiehed, M.; and Kontar, R. A. 2020. SALR: Sharpness-aware Learning Rates for Improved Generalization. ArXiv:2011.05348.

Zagoruyko, S.; and Komodakis, N. 2016. Wide Residual Networks.

Zeiler, M. D. 2012. ADADELTA: An Adaptive Learning Rate Method. ArXiv:1212.5701.

Zhang, J.; Karimireddy, S. P.; Veit, A.; Kim, S.; Reddi, S.; Kumar, S.; and Sra, S. 2020a. Why are Adaptive Methods Good for Attention Models? In *NeurIPS*.

Zhang, J.; Karimireddy, S. P.; Veit, A.; Kim, S.; Reddi, S.; Kumar, S.; and Sra, S. 2020b. Why are Adaptive Methods Good for Attention Models? In *NeurIPS*.

Zhang, M.; Lucas, J.; Ba, J.; and Hinton, G. E. 2019a. Lookahead Optimizer: k steps forward, 1 step back. In *NeurIPS*.

Zhang, M. R.; Lucas, J.; Hinton, G.; and Ba, J. 2019b. Lookahead Optimizer: k steps forward, 1 step back. *Conference on Neural Information Processing Systems*.

Zhuang, J.; Tang, T.; Ding, Y.; Tatikonda, S. C.; Dvornek, N.; Papademetris, X.; and Duncan, J. 2020. AdaBelief Optimizer: Adapting Stepsizes by the Belief in Observed Gradients. In *NeurIPS*.