# Curvature-Informed SGD via General Purpose Lie-Group Preconditioners

**Anonymous Author(s)**
Affiliation
Address
email

## Abstract

We present a novel approach to accelerate stochastic gradient descent (SGD) by utilizing curvature information obtained from Hessian-vector products or finite differences of parameters and gradients, similar to the BFGS algorithm. Our approach involves two preconditioners: a matrix-free preconditioner and a low-rank approximation preconditioner. We update both preconditioners online using a criterion that is robust to stochastic gradient noise and does not require line search or damping. To preserve the corresponding symmetry or invariance, our preconditioners are constrained to certain connected Lie groups. The Lie group's equivariance property simplifies the preconditioner fitting process, while its invariance property eliminates the need for damping, which is commonly required in second-order optimizers. As a result, the learning rate for parameter updating and the step size for preconditioner fitting are naturally normalized, and their default values work well in most scenarios. Our proposed approach offers a promising direction for improving the convergence of SGD with low computational overhead. We demonstrate that Preconditioned SGD (PSGD) outperforms SoTA on Vision, NLP, and RL tasks across multiple modern deep-learning architectures.

## 1 Introduction

Optimizing machine learning models with millions of free parameters presents a significant challenge. While conventional convex optimization algorithms such as Broyden-Fletcher-Goldfarb-Shanno (BFGS), its limited-memory version, L-BFGS, conjugate gradient (CG), and nonlinear versions like Hessian-free (HF) optimization Martens and Sutskever [2012] have succeeded in small-scale, convex mathematical optimization problems, they are rarely used for large-scale, stochastic optimization problems that arise in machine learning (ML). One of the main reasons is their reliance on the line search step. In many ML models, such as variational and reinforcement learning models, cost functions are defined as expectations and can only be evaluated through Monte Carlo (MC) sampling averages. This can result in large variances, making optimizers that rely on line search to ensure convergence problematic. Several recent extensions of these methods to deep learning, such as K-BFGS and KFAC, have foregone the line search step in favor of damping Goldfarb et al. [2020], Martens and Grosse [2015]. However, this adds complexity by introducing extra hyperparameters.

Empirical results indicate that plain SGD is a highly efficient optimizer for most ML problems. However, for problems with a large eigenvalue spread, SGD may converge slowly once the solution is located in a basin of attraction. Regret optimizers such as RMSProp and Adam Kingma and Ba [2015] converge faster but have been shown to generalize worse on many problems Wilson et al. [2017], Zhou et al. [2020]. Reducing the generalization gap between SGD and Adam remains an active topic of research Zhuang et al. [2020]. This work focuses on providing SGD with a good preconditioner to accelerate its convergence around the basin of attraction without undermining its generalization capacity. The curvature information for preconditioner fitting can be sampled from Hessian-vector products or finite differences of parameters and gradients, similar to the BFGS algorithm. However, constructing a preconditioner in a deterministic way, as in BFGS Boyd and Vandenberghe [2004], Goldfarb et al. [2020], may not be possible due to potential issues with line search and damping. Therefore, we adopt the more general and gradient-noise-robust preconditioner fitting criterion proposed in Li [2015] and fit the preconditioner online with another "gradient descent" algorithm. The key is to avoid making the preconditioner fitting problem more difficult and computationally expensive than the original parameter-learning problem.

In this paper, we propose using Lie groups as a tool for preconditioner fitting. The "gradient descent" on a Lie group is similar to common gradient descent in Euclidean space. It involves applying a series of small transforms via multiplication with $I + \mu G$, where $\mu$ is a small scalar and $G$ is the group generator (see D.1). The Lie group is a rich and convenient space to work in since moving a preconditioner around any point on the group behaves similarly to moving it around the identity element of the group, i.e., the identity matrix $I$. This is known as the Lie group equivariance property.

Recent curvature-aware optimization methods such as HessianFree, KFAC, AdaHessian, K-LBFGS, and Shampoo have shown good empirical results in Deep Learning Osawa et al. [2022]. Yet, they require damping, line search, or regret and are thus susceptible to pitfalls that do not affect PSGD.

In an empirical setting, PSGD simultaneously surpasses the corresponding SoTA optimizers across vision, natural language processing (NLP), and reinforcement learning (RL) tasks, and estabishes new SoTA for many networks and settings, *e.g.* ResNet, LSTMs Hochreiter and Schmidhuber [1997] and GPT-2 Radford et al. [2019]. We consider optimization problems involving MNIST LeCun and Cortes [2010], CIFAR-10 Krizhevsky [2009], Penn Treebank Marcus et al. [1993], the complete works of Shakespeare, the Open Web Text (OWT) dataset Gokaslan and Cohen [2019], HalfCheetah and RoboWalker Brockman et al. [2016]. PSGD outperforms SoTA methods with negligible overhead compared to SGD across a wide range of optimization problems. PSGD provides practitioners with a powerful, stable, and efficient optimization tool that can significantly enhance the performance of deep learning models in various domains.

## 2 Background

### 2.1 Notations

The objective is to minimize a loss function defined as an expectation, $f(\theta) = E_z[\ell(\theta, z)]$, where $\theta \in \mathbb{R}^n$ is the parameter vector to be optimized and $z$ is a random vector that can be sampled to evaluate the loss $\ell(\theta, z)$. We assume that the considered problem is second-order differentiable. To simplify the notation, we use $\hat{f}(\theta)$ to denote a sampled noisy evaluation of $f(\theta)$. A step of SGD with learning rate $\mu$ and an optional positive definite preconditioner $P$ is given by:

$$\theta_{i+1} = \theta_i - \mu P \, \partial \hat{f}(\theta)/\partial\theta \mid_{\theta=\theta_i} \tag{1}$$

where $i$ is the iteration index, $\mu > 0$ is the learning rate, and $P$ typically is a variable or adaptive preconditioner. Once the solution enters a basin of attraction centered at a local minimum $\theta^*$, we can approximate the iteration step in (1) as:

$$\theta_{i+1} - \theta^* \approx (I - \mu P\hat{H})(\theta_i - \theta^*) \tag{2}$$

where $\hat{H} = \frac{\partial^2 \hat{f}(\theta)}{\partial\theta^T \partial\theta} \mid_{\theta=\theta^*}$ is the sampled Hessian at the local minimum. Conceivably, the eigenvalue spread of $P\hat{H}$ largely determines the speed of convergence of the quasi-linear system in (2). Nearly quadratic convergence is possible if we can find a good approximation for $H^{-1}$. However, $\hat{H}$ is a noisy Hessian and is not necessarily positive definite, even if the exact one at $\theta^*$, i.e., $H$, is.

### 2.2 The Preconditioner Fitting Criterion

We adopt the preconditioner fitting criterion proposed in Li [2015]. Let $\delta g$ be the perturbation of gradient associated with parameter perturbation $\delta\theta$. Then, the fitting criterion is:

$$c(P) = E_{\delta\theta}[\delta g^T P \delta g + \delta\theta^T P^{-1} \delta\theta] \tag{3}$$

With auto differentiation tools, we can replace the pair $(\delta\theta, \delta g)$ with $(v, \hat{H}v)$, where $v$ is a random vector, and $\hat{H}v$ is the noisy Hessian-vector product, which can be evaluated as cheaply as gradients. Criterion (3) only has one positive definite solution, $P = (H^2 + E_v[\epsilon^2])^{-\frac{1}{2}}$, even for indefinite $H$, where $\epsilon = \hat{H} - H$ is a stochastic noise term. This preconditioner automatically dampens gradient noise. It is worth noting that criterion (3) gives the same preconditioner used in equilibrated SGD (ESGD) Dauphin et al. [2015] and AdaHessian Yao et al. [2021] when $P$ is diagonal, i.e., $E[v \odot v] \oslash E[(\hat{H}v) \odot (\hat{H}v)]$, where $\odot$ and $\oslash$ denote element-wise product and division, respectively.

### 2.3 Preconditioners on Lie Groups

It is natural to fit the preconditioner on a Lie group for several reasons. First, by rewriting equation (1) as $P^{-\frac{1}{2}}\theta_{i+1} = P^{-\frac{1}{2}}\theta_i - \mu\partial\hat{f}(\theta)/\partial(P^{-\frac{1}{2}}\theta)\mid_{\theta=\theta_i}$, it is clear that a preconditioned SGD is equivalent to SGD in a new set of coordinates defined by $\vartheta = P^{-\frac{1}{2}}\theta$. This coordinate change consists of rotations and scalings, i.e., operations on the orthogonal group $O(n)$ and the group of nonsingular diagonal matrices. We can represent this coordinate transform with matrix $Q^{-1}$ and, accordingly, $P = Q^TQ$. Thus, we pursue a variable $Q$ on the Lie group to fit this coordinate transform.

Second, PSGD can also be viewed as SGD with transformed features when the parameters to be learned are a list of affine transform matrices Li [2019]. Specifically, the most commonly used feature transformations (e.g., whitening, normalization, and scaling) can be represented as matrices on the affine groups. For example, the popular batch normalization Ioffe and Szegedy [2015], layer normalization Ba et al. [2016], and group normalization Wu and He [2018] can be represented as a sparse Lie group matrix where only the diagonal and last column can have nonzero values Li [2019] (See B.0.1). The decorrelated batch normalization Huang et al. [2018] is related to the whitening preconditioner in Li [2019]. Thus, the Lie group arises as a natural object to work with.

Lie groups have two properties that are particularly suitable for our task. Like any group, a specific Lie group preserves certain symmetries or invariances. For example, with $Q \in GL^+(n, \mathbb{R})$, the general linear group with positive determinant, $\vartheta$ and $\theta$ will always have the same orientation. This eliminates the need for damping, or similar remedies, to avoid degenerate solutions, since $P = Q^TQ$ is guaranteed to be invertible. The equivariance property of Lie groups further facilitates the preconditioner fitting. The same group generator, i.e., the one at the identity matrix, can be used to move a preconditioner on any point of the Lie group.

In fact, the preconditioner $P$ estimated by PSGD converges to the inverse of "absolute" Hessian regardless of the definiteness of Hessian. From this, one can show that the parameters converge following the established results in open literature. For mode details and proof see A.

The preconditioners proposed in Li [2019] can only be applied to a list of affine transform matrix parameters. Although many machine learning models exclusively consist of affine transforms and nonlinear activation functions, this is not always the case. Additionally, it can be impractical to reparameterize many existing modules, such as a convolutional layer, into their equivalent affine form. Hence, in this paper, we propose two types of novel general purpose preconditioners.

## 3 General Purpose Lie Group Preconditioners

### 3.1 Simple Matrix Free Preconditioners

Lie groups are a special type of abstract mathematical object that consist of transformations in vector space, specifically mappings that take vectors in $\mathbb{R}^n$ and map them to other vectors in the same space. Mathematically we have, $T : \mathbb{R}^n \mapsto \mathbb{R}^n$. The following theorem gives one systematic way to construct such sparse Lie group preconditioners.

**Theorem 3.1.** *Let $K = \{\sigma_1, \ldots, \sigma_m\}$ be a subgroup of the permutation group $S_n$. Then, linear transform $T : \mathbb{R}^n \mapsto \mathbb{R}^n, \quad T(x|a_1, \ldots, a_m) = \sum_{i=1}^m a_i \odot \sigma_i(x)$, forms a subgroup of $GL(n, \mathbb{R})$ parameterized with $\{a_1, \ldots, a_m\}$ if $T(\cdot|a_1, \ldots, a_m)$ is bijective, where both $a_i$ and $x$ are in $\mathbb{R}^n$.*

See proof of Theorem 3.1 in Appendix B.

*Example 1*: the group of invertible diagonal matrices. We must have $K = \{e\}$ if $|K| = 1$, where $e$ is the identity element of $S_n$, i.e., $e(x) = x$. Then, $T$ simply has a diagonal matrix representation, i.e., $T(x|a_1) = \mathrm{diag}(a_1)x$. Criterion (3) gives the preconditioner in ESGD Dauphin et al. [2015] and AdaHessian Yao et al. [2021] as a special case when $P$ is on this group.

*Example 2*: The group of "X-shape matrices." Let $K = \{e, \sigma_f\}$, where $\sigma_f$ denotes the flipping permutation. Then, we can show that

$$\begin{aligned} T(\cdot|a,b)T(\cdot|u,v) &= T(\cdot|a \odot u + b \odot \sigma_f(v), a \odot v + b \odot \sigma_f(u)) \\ T^{-1}(\cdot|a,b) &= T(\cdot|\sigma_f(a) \oslash c, -b \oslash c) \end{aligned}$$

where $c = a \odot \sigma_f(a) - b \odot \sigma_f(b)$. Clearly, such transforms form a Lie group if they are invertible, i.e., no element of $c$ is zero. The matrix representation of this $T$ only has nonzero diagonal and anti-diagonal elements, thus the name X-shape matrix.

*Example 3*: The butterfly matrix. For an even $n$, subgroup $K = \{e, s_{\frac{n}{2}}\}$ induces a Lie group whose representations are invertible butterfly matrices, where $s_{\frac{n}{2}}$ denotes circular shifting by $\frac{n}{2}$ positions. This group of matrices are the building blocks of the Kaleidoscope matrices Dao et al. [2020].

*Example 4*: The plain dense invertible matrix. The group $GL(n, \mathbb{R})$ can be recovered by letting $K = \{e, s_1, \ldots, s_{n-1}\}$, where $s_i$ denotes circular shifting by $i$ positions.

The group $GL(n, \mathbb{R})$ is too expensive for large scale problems. The group of diagonal matrices, also called the "Jacobi preconditioner" in its matrix form, is sparse but empirically shown to be less effective without the help of momentum for certain machine learning problems Dauphin et al. [2015]. We are mostly interested in the cases with $2 \le |K| \le 4$. These Lie groups are sparse enough, yet simple enough to derive their inverse manually, and at the same time can significantly accelerate the convergence of SGD by shortcutting gradients separated far away in positions.

### 3.2 Low Rank Approximation Preconditioner

Low-rank approximation (LRA) is a standard technique for processing large-scale matrices. Commonly adopted forms of positive definite low-rank approximation, such as $P = \rho I + UU^T$, cannot always be factorized as $P = Q^TQ$ for certain Lie groups, where $\rho > 0$ is a small positive number. Additionally, this form of approximation is not effective for reducing eigenvalue spread. In many real-world problems, the Hessian has a few very large and very small eigenvalues, i.e., tails on both ends of the spectrum Sagun et al. [2016, 2017]. However, all the eigenvalues of $P$ in this form are lower bounded by $\rho$, meaning that it can only fit one tail of the spectrum when $\mathrm{rank}(U) \ll n$.

For this reason, we propose a new low-rank approximation with form $Q = \rho(I + UV^T)$, where $\rho$ is not necessarily small nor positive, and $U$ and $V$ have $r$ columns with $r \ll n$. To justify this form of approximation, we need to establish two facts. First, preconditioner $P = Q^TQ$ with this form can fit both tails of the spectra of Hessian, providing an accurate characterization the curvature of a function, improving optimization algorithms, and assessing their robustness. Second, we can update this preconditioner on Lie groups.

**Theorem 3.2.** *Preconditioner $P = Q^TQ$ with $Q = \rho(I + UV^T)$ can have positive eigenvalues arbitrarily larger than $\rho^2$ and arbitrarily smaller than $\rho^2$ with proper $U$ and $V$.*

**Theorem 3.3.** *If $\rho \neq 0$ and $(I+V^TU)^{-1}$ or $(I+U^TV)^{-1}$ exists, $A_V(\rho, U) = \rho(I+UV^T)$ defines a subgroup of $GL(n, \mathbb{R})$ parameterized with $\rho$ and $U$. Similarly, $A_U(\rho, V) = \rho(I + UV^T)$ defines another subgroup of $GL(n, \mathbb{R})$ parameterized with $\rho$ and $V$.*

See proofs of Theorem 3.2 &3.3 in Appendix C.1 & C.1.

## 4 Practical Considerations

Above-proposed preconditioners can be fit online by minimizing criterion (3) using gradient descent on Lie groups. Unlike traditional gradient descent, moving an object on a Lie group is achieved by multiplying it with $I + \mu G$, where $G$ is the group generator and $\mu$ is small enough such that $|\mu G| < 1$. This series of small movements trace a curve on the Lie group manifold, and $G$ is always in the tangent space of the group as the Lie algebra is closed. See D for mathematical considerations.

Note, that optimizer damping is neither necessary nor generally feasible on any Lie group, although it is widely used in other second-order optimizers to avoid degenerate solutions. On one hand, by

Table 1: Test accuracy of LeNet5 on MNIST over 10 runs.

| SGD | Adam | KFAC | PSGD$_{\text{A}}$ | PSGD$_{\text{XMat}}$ | PSGD$_{\text{LRA}}$ |
|---|---|---|---|---|---|
| 99.04 | 99.12 | 99.16 | 99.26 | 99.22 | 99.22 |

Table 2: Stage & cosine learning rate and residual-free ResNet18-RF on CIFAR10.

| | ResNet18 | | ResNet18-RF | |
|---|---|---|---|---|
| *lr* | *cos* | *stage* | *cos* | *stage* |
| SGD | 95.51 | 95.07 | 94.97 | 94.35 |
| PSGD | 95.54 | 95.43 | 95.36 | 95.17 |

fitting $Q$ on a connected Lie group, $P = Q^T Q$ cannot be singular. On the other hand, damping could be incompatible with certain forms of Lie groups, as we may not always be able to find another $Q'$ on the same group such that $Q'^T Q' = Q^T Q + \lambda I$, where $\lambda > 0$. This eliminates the need for setting up a proper damping schedule. However, gradient clipping can be helpful in promoting stability. The quadratic approximation leading to the quasi-linear system (2) is only valid within a certain region around $\theta$. Thus, $\|\delta\theta\| = \mu\|P\partial\hat{f}(\theta)/\partial\theta\|$ should be small enough such that $\theta + \delta\theta$ still locates in this trust region. We can adjust $\mu$ or clip $\|P\partial\hat{f}(\theta)/\partial\theta\|$ to ensure that $\|\delta\theta\|$ is small enough.

Lastly, the learning rate for parameter updating and step size for preconditioner fitting are naturally normalized to be in the range $(0, 1)$. We have found a step size of $0.01$ to be an effective initial rate, and our method is robust to a wide range of learning rates and weight decays. (see Appendix E.2)

## 5 Empirical Results

In this work, we evaluate the performance of the PSGD algorithm on a diverse set of tasks, including vision, natural language processing (NLP), and reinforcement learning (RL). In the domain of computer vision, we evaluate the algorithm's performance on the MNIST LeCun and Cortes [2010] dataset and using a LeNet5 (see Table 1) and a convex large-scale logistic regression (see Appendix E.3). Aditionally we consider the CIFAR10 Krizhevsky [2009] dataset (see Table 2) with ResNet18. Finally, we consider several variants of the CIFAR10 dataset, including those with noisy labels, blur-deficit, adversarial attacks, and class imbalances (see Table 3).

For NLP tasks, we study the use of LSTMs Hochreiter and Schmidhuber [1997] and GPT-2 style transformers Radford et al. [2019], on various text corpora, including the Penn Treebank Marcus et al. [1993], the complete works of Shakespeare, and the Open Web Text (OWT) dataset Gokaslan and Cohen [2019] (see Table 4 & 5). In the RL setting, we consider a Proximal Policy Optimization (PPO) Schulman et al. [2017] applied to the HalfCheetah and RoboWalker environmets using OpenAI's gym environments Brockman et al. [2016] (see Fig. 1).

To provide insight into the fundamental differences between SGD and PSGD, we perform uncertainty and forgettability analysis Toneva et al. [2018]. Finally, we examine the pathological delayed XOR problem, first introduced in Hochreiter and Schmidhuber [1997], in order to further our understanding of the strengths and limitations of different optimization methods.

### 5.1 CIFAR-10 and Friends on ResNet18

We examine the effectiveness of the PSGD optimizer in classifying CIFAR10-derived datasets using a ResNet18 and find it outperforms the SoTA optimizer SGD. Once the optimizers are tuned on ResNet18 with the CIFAR10 dataset, we do not change them for the rest of the experiments.

**CIFAR10 ResNet18** We evaluate the performance of PSGD and SGD on the CIFAR10 image classification task using the ResNet18 model, following the implementation of the AdaBelief algorithm Zhuang et al. [2020]. We adopt the cosine learning rate scheduler and observe that SGD achieves the SOTA test accuracy of $95.5\%$. Our results (see Table 2) show that PSGD slightly outperforms SGD when the shortcut connections are removed or a less-tuned learning rate scheduler is used. Both optimizers are tuned with hyperparameters such as learning rate and weight decay and use a momentum of 0.9. We update the preconditioner of PSGD every ten iterations, resulting in a negligible overhead compared to SGD.

**Removing Skip Connections** To increase curvature in the relatively flat modern ResNet architecture, we remove the residual skip connections that gave ResNets their namesake. A recent study Zhang et al. [2022] demonstrated the use of Tailored Activation Transformation (TAT) in conjunction with K-FAC Martens and Grosse [2015](another second-order optimizer) to close the gap between residual networks with and without residual skip connections. However, in our experiment, we do not utilize TAT and instead compare the performance of SOTA optimizers on both residual-free and

Table 3: Accuracy of ResNet18 on diverse CIFAR10 derived tasks using PSGD vs SGD.

| CIFAR10 | Standard | No Shortcut | Class Imb | Noisy Final | Noisy Best | Adversarial | Blur Deficit |
|---|---|---|---|---|---|---|---|
| PSGD + M | $95.49_{0.08}$ | $95.27_{0.09}$ | $87.16_{0.85}$ | $78.63_{0.11}$ | $78.63_{0.11}$ | $85.17_{0.04}$ | $89.51_{0.12}$ |
| SGD +M | $95.47_{0.14}$ | $94.66_{0.16}$ | $86.32_{0.84}$ | $26.9_{33.75}$ | $73.975_{5.65}$ | $83.82_{0.18}$ | $83.51_{0.15}$ |

Table 4: Test perplexity (lower is better) on Penn Treebank for one-, two- and three-layered LSTMs. All results except PSGD in the table are reported by AdaBelief and Adan.

| LSTM | PSGD | Adan | AdaBelief | SGD | AdaBound | Adam | AdamW | Padam | RAdam | Yogi |
|---|---|---|---|---|---|---|---|---|---|---|
| 1 layer | **83.5** | 83.6 | 84.2 | 85.0 | 84.3 | 85.9 | 84.7 | 84.2 | 86.5 | 86.5 |
| 2 layers | **64.9** | 65.2 | 66.3 | 67.4 | 67.5 | 67.3 | 72.8 | 67.2 | 72.3 | 71.3 |
| 3 layers | **59.7** | 59.8 | 61.2 | 63.7 | 63.6 | 64.3 | 69.9 | 63.2 | 70.0 | 67.5 |

standard ResNet18 models. The results, summarized in Table 2, indicate that PSGD outperforms SGD by $0.61\%$ and $0.20\%$ on residual-free and standard ResNet18, respectively. Our findings are consistent with the results from Zhang et al. [2022], where a difference of $0.6\%$ was observed between the optimization of residual-free using TAT and standard ResNet18.

**Modified CIFAR10** Next, we evaluate the performance of SGD and PSGD on three variations of the CIFAR10 image classification task using ResNet18: class imbalance, noisy training labels, and adversarial generalization. These variations provide a rigorous test for the generalization ability of optimizers. We maintain the hyperparameters from our previous experiments on the ResNet18 CIFAR10 dataset for a fair comparison between the optimizers.

**Class Imbalanced CIFAR10** We evaluate the optimization performance on a class-imbalanced version of the CIFAR10 dataset, where $50\%$ of the classes are randomly reduced by an order of magnitude. We compare SGD and PSGD on optimizing ResNet18 and report the results in Table 3. Our results show that PSGD outperforms the SOTA by $1.03\%$ on this dataset.

**CIFAR10 with Noisy Labels** We randomly select $60\%$ of the CIFAR10 training labels, resulting in each class having approximately $46\%$ correct labels and the $54\%$ consisting of $6\%$ drawn from the other nine classes. We used SGD and PSGD to train a ResNet18 on this task for 100 epochs with 5 different seeds. Both optimizers achieved a training accuracy of around $44\%$. However, their test accuracies showed a significant difference. Among the 5 runs of SGD, only one *lucky initilization* achieved a test accuracy of $77\%$, while the other initializations had a test accuracy of $10\%$ with an average predicted probability of 0.999. This is not a standard case of over-fitting nor a case of catastrophic forgetting, since the training accuracy does not change. Instead, our experiments show there exists a tipping point in the test accuracy at around epoch 35, where within a single epoch the test accuracy drops from $71\%$ to $10\%$ while the training accuracy shows no indication of this change. Furthermore, the test accuracy does not increase for the rest of the 65 epochs. PSGD achieved an average accuracy of $78.63\%$ after 200 epochs, outperforming SGD by $51.73\%$ and exhibiting no optimization instabilities (See Table 3).

**Adversarial Attacked CIFAR10** Finally, we trained a ResNet18 on 10k unmodified CIFAR10 images and evaluated it on a test set consisting of 40k samples perturbed using Neural Tangent Generalization Attacks Yuan and Wu [2021]. As shown in Table 3, PSGD outperformed SGD by $1.35\%$.

### 5.2 Language Modeling

We conduct a performance comparison between second-order optimization methods and first-order methods for training recurrent neural networks (RNNs) and transformer models. RNNs have a recurrent structure that exhibits strong curvature properties, making them particularly suitable for second-order optimization methods. Such methods can efficiently leverage this structure to converge quickly to good solutions. Consequently, RNNs usually perform better when trained with second-order optimization methods Martens [2016], particularly when the objective function has extreme local curvature properties.

In contrast, transformers employ sparse self-attention mechanisms that allow for efficient computation of gradients. Thus, they are typically trained using first-order methods. Our aim is to provide a comprehensive comparison of these optimization methods for both RNN and transformer models.

**LSTM based models** We benchmark PSGD by predicting the Penn TreeBank Dataset using LSTMs using Zhuang et al. [2020]'s framework. Our results (see Table 4 ) indicate that PSGD consistently outperforms (lower is better) other first-order optimization algorithms, including the SoTA AdamW, on 1, 2, and 3 layer LSTMs.

**nanoGPT** In a recent study, Karpathy [2022] proposed a framework for reproducing GPT-2 results using the OpenWebText dataset. Here, we expand upon this by investigating the training of GPT-2

Table 5: Comparing different test loss of optimizers over GPT-2 style transformers on the Shakespeare-char dataset. PSGD outperforms other optimizers over various model sizes. Trained on a single 3080 GPU

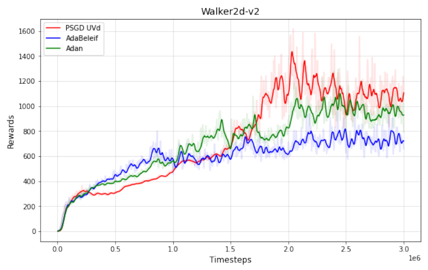
| nanoGPT | PSGD | SGD | AdamW | AdanW | AdaBelief | AdanBelief |
|---|---|---|---|---|---|---|
| 0.82M | **4.52** | 4.68 | 4.68 | 5.52 | 5.94 | 6.66 |
| 1.61M | **4.47** | 4.75 | 5.03 | 5.054 | 5.06 | 6.47 |
| 6.37M | **4.53** | 5.31 | 19.53 | 4.92 | 21.04 | 5.34 |
| 50M | **250.7** | | 591.8 | | | |

models of different sizes on both the OpenWebText and Shakespeare character datasets. Our primary objective is to benchmark the performance of PSGD against other SoTA optimization algorithms.

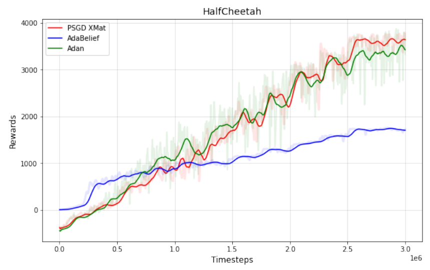
As shown in Table 5, our results indicate that PSGD consistently outperforms other optimization algorithms across various model sizes. Notably, a moderate gap in perplexity is observed for the smaller GPT-2 model trained for 5k iterations on the Shakespeare-char dataset, with a significantly larger gap observed on the OpenWebText dataset, which was trained for 600k iterations using AdamW and 100k iterations using PSGD. We found that decreasing the *precond lr* to 0.001 greatly improved the performance of transformer models. Lowering the *precond lr* smoothens the curvature in the sparse embedding layer of GPT-2 over time and enables the optimizer to consider a larger window of curvature information. This "curvature memory" improved performance and prevented divergence resulting from the otherwise sparse embedding space.

## 5.3 Reinforcement Learning

Here we consider two standard Proximal Policy Optimization (PPO) problems in Reinforcement Learning (RL): Walker2d and HalfCheetah. We compare the performance of the two SOTA optimizers AdaBelief and Adan to PSGD. We optimize the actor and critic independently. We find that PSGD can find a higher reward for both Walker2d & HalfCheetah shown in Figure 1.



(a) **Walker2d**



(b) **HalfCheetah**

Figure 1: PSGD outperforms SOTA optimizers on PPO RL on Walker2d & HalfCheetah.

## 5.4 Towards Understanding Second Order Optimizers

With the performance distinctions between PSGD and SGD now apparent, we aim to conduct a series of simple experiments to provide further insight into the reasons behind the stark contrast in the nature of the two optimizers.

**Uncertainty Analysis** We train a standard ResNet18 on CIFAR10 with PSGD and SGD, and after 200 epochs we check the entropy over the softmax-logits and miss-classification margin Toneva et al. [2018] of both NNs. We see in Table 6 that PSGD has higher entropy and a lower margin of classification compared to first-order optimizers. This very low minimum entropy of first-order optimizers may be interpreted as a form of overfitting. Intuitively, some data points (low entropy/unforgettable) have information that is easily memorized by NNs, giving up network capacity learning points that do not improve generalization. Conversely, we observe that PSGD never becomes overly certain about any data point, with the minimum entropy being six orders of magnitude larger than that of SGD. Similar trends can be observed in the mean and maximum entropy values. In terms of margin, PSGD on average has a $95\%$ certainty level for classifications, while SGD has a staggering $99.9\%$. From these statistics, we believe first-order optimizers can push NNs into a dangerous regime of overconfidence which given clean labels can reduce their generalization capacity with the potential of catastrophic forgetting. In the scenario where a network is given noisy labels (or imaginably poisoned points), training with first-order methods may lead to memorization of the training set with no generalization capacity as seen in Table 3 column Noisy Final.

PSGD's higher entropy and lower margins are indications of a larger exploration of the parameter space, more diverse solutions, and a lack of memorization. Suggesting that PSGD is able to better balance the trade-off between overfitting and underfitting, without memorizing points.

Table 6: Uncertainty Statistics: PSGD's higher uncertainty leads to better generalization and less over-fitting.

| NTK | Entropy | | | Margin | | |
|---|---|---|---|---|---|---|
| | Min | Mean | Max | Min | Mean | Max |
| PSGD | 0.139 | 0.260 | 1.545 | 0.144 | 0.956 | 0.994 |
| SGD | 7x$10^{-7}$ | 0.01 | 0.8975 | 0.3925 | 0.999 | 1 |

Table 7: Forgetting statistics for CIFAR10 on ResNet18. PSGD finds better forgettability statistics outperforming SGD.

| Forgetting | 50k | 25k | 15k | 5k |
|---|---|---|---|---|
| PSGD | 96.65 | 95.83 | 94.95 | 56.46 |
| SGD | 96.21 | 95.56 | 93.7 | 42.48 |

**Forgettability Statistics and Learning** We revisit Toneva et al. [2018]'s forgettability experiments, which found that one can prune 30% of the CIFAR-10 train samples without loss of generalization. The study found that a point's utility for generalization increases as it is learned and then subsequently forgotten during training, regardless of the optimizer or architecture used. Essentially, forgettability ordering shows which data points a NN uses to define high-dimensional boundaries akin to support vectors. We investigate whether the performance difference between PSGD and SGD's generalization performance can be attributed to this forgettability ordering. Furthermore, Toneva et al. [2018] revealed a strong correlation between forgettable points and high margin, which we explore by examining the expected forgettability ordering of the two optimizers and whether pruning points based on their orderings affects performance.

We train the ResNet18 three times pruning CIFAR10 by 25k, 35k, and 45k points based on each optimizer's forgettability score. Table 7 shows that points PSGD focuses on based on forgettability ordering outperform that of SGD. When we limit the dataset to only the 5k most forgettable datapoints, we see PSGD is able to outperform SGD by nearly 14%.

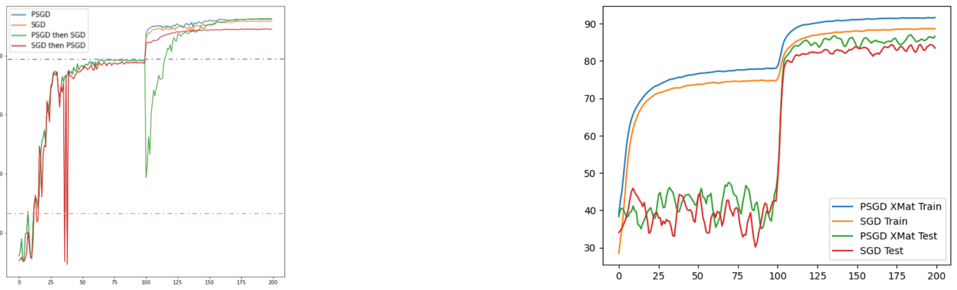
SGD and PSGD exhibit fundamentally different importance orderings, which is evident from the significant generalization gap observed when training on pruned datasets using these orderings at various degrees. We find that PSGD and SGD have a statistically significant correlation between their forgettability orderings, but the strength of the correlation is weak (Spearman coefficient of 0.02 and a p-value of $p = 1\text{x}10^{-12}$). This indicates that the nature of training observed through forgettability is different for PSGD compared to first-order optimizers.

Furthermore, we see a strong correlation coefficient of 0.75 and 0.60 between a low forgetability score and low entropy score with a p-value of $p = 0$ for PSGD and SGD respectively (see Fig 7). Hence, PSGD does a better job of shaping the NN to focus on the highly forgettable points that are important for generalization while not becoming overconfident on the easy unforgettable points giving up too much capacity, unnecessarily pushing the parameters away from initialization reducing generalization ability Cao and Gu [2019] (see 11). Note while Toneva et al. [2018] shows that forgettable points are more important for generalization for supervised learning, very recently Pooladzandi et al. [2023] showed low entropy points are the more important points for learning in the unsupervised, semi-supervised, and generative model setting. See Table 11 showing generalization gap training low and high entropy points and how it affects the distance from initialization.

**Stubborn Solutions of First Order Optimizers** From the previous examples, we learned that first-order optimizers tend to heavily overfit certain data points, reducing entropy. In the case of noisy labels, this often results in pure memorization where the network achieves the training accuracy of PSGD on the train set, but only 10% accuracy on the test dataset with an average confidence of 99.9%. To better understand the stubbornness of SGD solutions, we consider the *lucky initialization,* which resulted in a test accuracy of 77% under noisy label conditions. Here, we examine a transfer learning problem, where a ResNet18 was trained on noisy data for 100 epochs and then on clean labels for another 100 epochs. This simulates a real-world distributional shift where noisy labels may be refined throughout training, leading to a distributional shift. We compare neural networks trained with each optimizer, as well as those that changed optimizers after 100 epochs of training.

As seen in Fig 2a, PSGD finds a flexible solution when given noisy labels. This is seen since when we correct the labels both PSGD and SGD can reach an accuracy of 92.42%. In contrast, the solution found by SGD seems to be stubborn since when we correct the noisy labels, PSGD then SGD reaches an accuracy of 91.7% and SGD then PSGD has an accuracy of 88%. This shows the importance of curvature information in the early periods of training.

Intuitively, first-order methods cause NNs to dedicate parameters to memorizing some data points, unnecessarily reducing entropy. When memorization occurs given incorrect labels, it may be difficult to reshape the NN when the labels are corrected, leading to the loss in generalization accuracy seen in Fig 2a. In contrast, as PSGD finds a less certain or suborn solution in the first stage of training, either optimizer can then reshape the NN to their liking in the second stage.



(a) **Noisy Label CIFAR-10** (b) **Nero-Plasticity**

Figure 2: a) Solutions found by SGD are harder to optimize compared to PSGD. Note for SGD we used a lucky initialization (see 5.1). The blue and yellow dotted line are the average accuracy of PSGD and SGD after 100 epochs respectivly. b) Removing the blur-deficit at epoch 100, PSGD is be more neuro-plastic compared to SGD, achieving better train and test performance.

**PSGD Preserves Neuro-Plasticity** Recently, Achille et al. [2017] studied the phenomenon of neuro-plasticity in NNs. The presents an different lense to our stubborn solution hypothesis. They found that NNs exhibit a critical learning period, during which if a learning deficit is not removed, the NN becomes unable to learn effectively. To simulate this deficit, CIFAR10 images are blurred for the first half of training, after which the deficit is removed. Following Achille et al. [2017], we used an exponential learning rate decay. To investigate the impact of optimization on neuro-plasticity we compare SGD and PSGD. We find that PSGD retains neuro-plasticity outperforming SGD by 6 and 3% on test and train sets seen in Fig 2b and Table 3.

We believe the noisy-label and neuro-plasticity results have strong applicability to transfer learning and coreset selection Pooladzandi et al. [2022], particularly in scenarios where the training distribution changes during training. Recently, Osawa et al. [2022] demonstrated that the affine Kronecker variant of PSGD outperforms other first and second-order optimizers when fine-tuning a ResNet18 and ViT Dosovitskiy et al. [2020] on CIFAR10 that were pre-trained on ImageNet Deng et al. [2009].

**Understanding long term dependence via Delayed XOR Problem** We consider the delayed XOR problem proposed in the LSTM paper Hochreiter and Schmidhuber [1997]. The task is to predict the XOR relation of $a$ and $b$ scattered randomly and far away in a long sequence. This problem is challenging for many architectures and optimizers because it cannot be "partially solved" since memorizing either $a$ or $b$ alone does not help to predict XOR$(a, b)$. We consider solving this problem with standard RNNs and LSTMs optimized using SGD, AdaBelief, Adan, AdaHessian, Hessian Free, Shampoo, KFAC, and PSGD at different sequence lengths. The success rates for each optimizer are shown in Table 8. In our experiments, we find that PSGD LRA is the only method that can solve the XOR problem past the length of 32 with an RNN. Furthermore, LSTMs show no benefit to RNNs without using curvature information for the XOR problem. Finally, PSGD optimizes an RNN to solve the XOR problem better than any of the other methods using an LSTM. These result hint, that while increasing model parameters and architecture complexity may boost performance, the choice of optimizer may be more important for certain problems. See E.5 for rank analysis.

Table 8: Delayed XOR problem at length 32,55,64, and 112 averaged over 10 runs. An RNN optimized with PSGD outperforms other optimizers with an RNN or LSTM. This hints that the choice of optimizer may be more important than network architecture.

| **XOR** | **PSGD LRA** | | **AdaHessian** | | **KFAC** | | **HessianFree** | | **Shampoo** | | **SGD** | | **AdaBelief** | | **Adan** | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| Length | RNN | LSTM | RNN | LSTM | RNN | LSTM | RNN | LSTM | RNN | LSTM | RNN | LSTM | RNN | LSTM | RNN | LSTM |
| **32** | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 0 | 1 | 1 | 1 | 1 | 1 | 1 | 1 |
| **55** | 1 | 1 | 0 | 1 | 0 | 0 | 0 | 0.6 | 0 | 0.8 | 0 | 0 | 0 | 0 | 0 | 0 |
| **64** | 1 | 1 | 0 | 0.8 | 0 | 0 | 0 | 0 | 0 | 0.6 | 0 | 0 | 0 | 0 | 0 | 0 |
| **112** | 1 | 1 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |

## 6 Conclusion

In conclusion, this work has presented a comprehensive study of general purpose Lie group preconditioners for the optimization of deep learning problems. We have provided theoretical guarantees for the convergence of Lie group preconditioned optimization methods, and empirical results demonstrating PSGD outperforming SoTA optimizers in generalization, robustness, and stability across various tasks in Vision, NLP, and RL. Furthermore, our analysis of forgettability and entropy shows that PSGD effectively focuses on forgettable points, leading to improved generalization. These findings provide valuable insights for further developing efficient and effective optimization techniques for deep learning models.

## References

Alessandro Achille, Matteo Rovere, and Stefano Soatto. Critical learning periods in deep neural networks. *CoRR*, abs/1711.08856, 2017. URL http://arxiv.org/abs/1711.08856.

Jimmy Lei Ba, Jamie Ryan Kiros, and Geoffrey E. Hinton. Layer normalization, 2016. URL https://arxiv.org/abs/1607.06450.

S. Boyd and L. Vandenberghe. *Convex Optimization*. Cambridge University Press, 2004.

Greg Brockman, Vicki Cheung, Ludwig Pettersson, Jonas Schneider, John Schulman, Jie Tang, and Wojciech Zaremba. Openai gym. *arXiv preprint arXiv:1606.01540*, 2016.

Yuan Cao and Quanquan Gu. Generalization error bounds of gradient descent for learning over-parameterized deep relu networks, 2019. URL https://arxiv.org/abs/1902.01384.

Tri Dao, Nimit S. Sohoni, Albert Gu, Matthew Eichhorn, Amit Blonder, Megan Leszczynski, Atri Rudra, and Christopher Ré. Kaleidoscope: an efficient, learnable representation for all structured linear maps. In *International Conference on Learning Representations (ICLR) 2020*. OpenReview.net, 2020.

Y. N. Dauphin, H. Vries, and Y. Bengio. Equilibrated adaptive learning rates for non-convex optimization. In *NIPS*, pages 1504–1512. MIT Press, 2015.

Jia Deng, Wei Dong, Richard Socher, Li-Jia Li, Kai Li, and Li Fei-Fei. Imagenet: A large-scale hierarchical image database. In *2009 IEEE Conference on Computer Vision and Pattern Recognition*, pages 248–255, 2009. doi: 10.1109/CVPR.2009.5206848.

Alexey Dosovitskiy, Lucas Beyer, Alexander Kolesnikov, Dirk Weissenborn, Xiaohua Zhai, Thomas Unterthiner, Mostafa Dehghani, Matthias Minderer, Georg Heigold, Sylvain Gelly, Jakob Uszkoreit, and Neil Houlsby. An image is worth 16x16 words: Transformers for image recognition at scale. *CoRR*, abs/2010.11929, 2020. URL https://arxiv.org/abs/2010.11929.

Aaron Gokaslan and Vanya Cohen. Openwebtext corpus, 2019.

Donald Goldfarb, Yi Ren, and Achraf Bahamou. Practical quasi-newton methods for training deep neural networks. *CoRR*, abs/2006.08877, 2020. URL https://arxiv.org/abs/2006.08877.

S. Hochreiter and J. Schmidhuber. Long short-term memory. *Neural Computation*, 9(8):1735–1780, 1997.

Lei Huang, Dawei Yang, Bo Lang, and Jia Deng. Decorrelated batch normalization. *CoRR*, abs/1804.08450, 2018. URL http://arxiv.org/abs/1804.08450.

Sergey Ioffe and Christian Szegedy. Batch normalization: Accelerating deep network training by reducing internal covariate shift. *CoRR*, abs/1502.03167, 2015. URL http://arxiv.org/abs/1502.03167.

Andrej Karpathy. Nanogpt: Small gpt implementations. https://github.com/karpathy/nanoGPT, 2022. Accessed on 22 February 2023.

D. P. Kingma and J. L. Ba. Adam: a method for stochastic optimization. In *ICLR*. Ithaca, NY: arXiv.org, 2015.

Alex Krizhevsky. Learning multiple layers of features from tiny images. pages 32–33, 2009. URL https://www.cs.toronto.edu/~kriz/learning-features-2009-TR.pdf.

Yann LeCun and Corinna Cortes. MNIST handwritten digit database. 2010. URL http://yann.lecun.com/exdb/mnist/.

X. L. Li. Preconditioner on matrix lie group for SGD. In *7th International Conference on Learning Representations, ICLR 2019, New Orleans, LA, USA, May 6-9, 2019*. OpenReview.net, 2019.

Xi-Lin Li. Preconditioned stochastic gradient descent, 2015. URL https://arxiv.org/abs/1512.04202.

Mitchell P. Marcus, Beatrice Santorini, and Mary Ann Marcinkiewicz. Building a large annotated corpus of English: The Penn Treebank. *Computational Linguistics*, 19(2):313–330, 1993. URL `https://aclanthology.org/J93-2004`.

J. Martens and R. B. Grosse. Optimizing neural networks with Kronecker-factored approximate curvature. In *ICML*, pages 2408–2417, 2015.

J. Martens and I. Sutskever. Training deep and recurrent neural networks with hessian-free optimization. In G. Montavon, G. B. Orr, and K. R. Muller, editors, *Neural Networks: Tricks of the Trade*. Springer, Berlin Heidelberg, 2012.

James Martens. *Second-order Optimization for Neural Networks*. PhD thesis, University of Toronto, Canada, 2016. URL `http://hdl.handle.net/1807/71732`.

Kazuki Osawa, Satoki Ishikawa, Rio Yokota, Shigang Li, and Torsten Hoefler. Asdl: A unified interface for gradient preconditioning in pytorch. In *Order Up! The Benefits of Higher-Order Optimization in Machine Learning, NeurIPS 2022 Workshop*, 2022. URL `https://order-up-ml.github.io/papers/19.pdf`.

Adam Paszke, Sam Gross, Soumith Chintala, Gregory Chanan, Edward Yang, Zachary DeVito, Zeming Lin, Alban Desmaison, Luca Antiga, and Adam Lerer. Automatic differentiation in pytorch. In *NIPS-W*, 2017.

Barak A Pearlmutter. Fast exact multiplication by the hessian. *Neural computation*, 6(1):147–160, 1994.

Omead Pooladzandi, David Davini, and Baharan Mirzasoleiman. Adaptive second order coresets for data-efficient machine learning. In Kamalika Chaudhuri, Stefanie Jegelka, Le Song, Csaba Szepesvari, Gang Niu, and Sivan Sabato, editors, *Proceedings of the 39th International Conference on Machine Learning*, volume 162 of *Proceedings of Machine Learning Research*, pages 17848–17869. PMLR, 17–23 Jul 2022. URL `https://proceedings.mlr.press/v162/pooladzandi22a.html`.

Omead Pooladzandi, Pasha Khosravi, Erik Nijkamp, and Baharan Mirzasoleiman. Generating high fidelity synthetic data via coreset selection and entropic regularization, 2023. URL `https://arxiv.org/abs/2302.00138`.

Alec Radford, Jeff Wu, Rewon Child, David Luan, Dario Amodei, and Ilya Sutskever. Language models are unsupervised multitask learners. 2019.

Levent Sagun, Léon Bottou, and Yann LeCun. Singularity of the hessian in deep learning. *CoRR*, abs/1611.07476, 2016. URL `http://arxiv.org/abs/1611.07476`.

Levent Sagun, Utku Evci, V. Ugur Güney, Yann N. Dauphin, and Léon Bottou. Empirical analysis of the hessian of over-parametrized neural networks. *CoRR*, abs/1706.04454, 2017. URL `http://arxiv.org/abs/1706.04454`.

John Schulman, Filip Wolski, Prafulla Dhariwal, Alec Radford, and Oleg Klimov. Proximal policy optimization algorithms, 2017. URL `https://arxiv.org/abs/1707.06347`.

Mariya Toneva, Alessandro Sordoni, Remi Tachet des Combes, Adam Trischler, Yoshua Bengio, and Geoffrey J Gordon. An empirical study of example forgetting during deep neural network learning. *arXiv preprint arXiv:1812.05159*, 2018.

Andreas Veit, Michael Wilber, and Serge Belongie. Residual networks behave like ensembles of relatively shallow networks, 2016. URL `https://arxiv.org/abs/1605.06431`.

Ashia C. Wilson, Rebecca Roelofs, Mitchell Stern, Nathan Srebro, and Benjamin Recht. The marginal value of adaptive gradient methods in machine learning, 2017. URL `https://arxiv.org/abs/1705.08292`.

Yuxin Wu and Kaiming He. Group normalization. *CoRR*, abs/1803.08494, 2018. URL `http://arxiv.org/abs/1803.08494`.

Zhewei Yao, Amir Gholami, Sheng Shen, Mustafa Mustafa, Kurt Keutzer, and Michael W. Mahoney. Adahessian: an adaptive second order optimizer for machine learning. In *AAAI*, 2021.

Chia-Hung Yuan and Shan-Hung Wu. Neural tangent generalization attacks. In Marina Meila and Tong Zhang, editors, *Proceedings of the 38th International Conference on Machine Learning*, volume 139 of *Proceedings of Machine Learning Research*, pages 12230–12240. PMLR, 18–24 Jul 2021. URL `https://proceedings.mlr.press/v139/yuan21b.html`.

Guodong Zhang, Aleksandar Botev, and James Martens. Deep learning without shortcuts: Shaping the kernel with tailored rectifiers. *arXiv preprint arXiv:2203.08120*, 2022.

Pan Zhou, Jiashi Feng, Chao Ma, Caiming Xiong, Steven Hoi, and Weinan E. Towards theoretically understanding why sgd generalizes better than adam in deep learning, 2020. URL `https://arxiv.org/abs/2010.05627`.

Juntang Zhuang, Tommy Tang, Yifan Ding, Sekhar Tatikonda, Nicha Dvornek, Xenophon Papademetris, and James S. Duncan. Adabelief optimizer: adapting stepsizes by the belief in observed gradients. In *NeurIPS 2020*, 2020.

## Checklist

The checklist follows the references. Please read the checklist guidelines carefully for information on how to answer these questions. For each question, change the default **[TODO]** to [Yes] , [No] , or [N/A] . You are strongly encouraged to include a **justification to your answer**, either by referencing the appropriate section of your paper or providing a brief inline description. For example:

- Did you include the license to the code and datasets? [Yes] See Section **??**.
- Did you include the license to the code and datasets? [No] The code and the data are proprietary.
- Did you include the license to the code and datasets? [N/A]

Please do not modify the questions and only use the provided macros for your answers. Note that the Checklist section does not count towards the page limit. In your paper, please delete this instructions block and only keep the Checklist section heading above along with the questions/answers below.

1. For all authors...
   (a) Do the main claims made in the abstract and introduction accurately reflect the paper's contributions and scope? [Yes]
   (b) Did you describe the limitations of your work? [Yes] See section **??**
   (c) Did you discuss any potential negative societal impacts of your work? [Yes] See section **??**
   (d) Have you read the ethics review guidelines and ensured that your paper conforms to them? [Yes]
2. If you are including theoretical results...
   (a) Did you state the full set of assumptions of all theoretical results? [Yes] See section A
   (b) Did you include complete proofs of all theoretical results? [Yes] See section A
3. If you ran experiments...
   (a) Did you include the code, data, and instructions needed to reproduce the main experimental results (either in the supplemental material or as a URL)? [Yes] supplemental material
   (b) Did you specify all the training details (e.g., data splits, hyperparameters, how they were chosen)? [Yes] See section E.2 and E.2
   (c) Did you report error bars (e.g., with respect to the random seed after running experiments multiple times)? [Yes]
   (d) Did you include the total amount of compute and the type of resources used (e.g., type of GPUs, internal cluster, or cloud provider)? [Yes] See section E.7
4. If you are using existing assets (e.g., code, data, models) or curating/releasing new assets...
   (a) If your work uses existing assets, did you cite the creators? [Yes] See section E.7
   (b) Did you mention the license of the assets? [Yes]
   (c) Did you include any new assets either in the supplemental material or as a URL? [Yes] supplemental material
   (d) Did you discuss whether and how consent was obtained from people whose data you're using/curating? [N/A]
   (e) Did you discuss whether the data you are using/curating contains personally identifiable information or offensive content? [N/A] All datasets used are open source
5. If you used crowdsourcing or conducted research with human subjects...
   (a) Did you include the full text of instructions given to participants and screenshots, if applicable? [N/A]
   (b) Did you describe any potential participant risks, with links to Institutional Review Board (IRB) approvals, if applicable? [N/A]
   (c) Did you include the estimated hourly wage paid to participants and the total amount spent on participant compensation? [N/A]

## A On the Convergence of PSGD

In this section, we show that the preconditioner $P$ estimated by PSGD converges to the inverse of Hessian under mild conditions. We forgo convergence of network parameters since it is a well-studied topic in the literature.

To begin, let us formulate the problem clearly. Let $(v, h)$ be a pair of vector and the associated noisy Hessian-vector product. We assume that $v$ is drawn from distribution $\mathcal{N}(0, I)$. The noisy Hessian-vector product $h$ is modeled by

$$h = H_0 v + \varepsilon$$

where $H_0$ is the true Hessian, and $\varepsilon$ is a noise term independent of $v$ due to use of sample averaged loss instead of the true loss. Note, the full Hessian expansion is circumvented, and instead only requires us to calculate the much cheaper Hessian vector product $H_0 v$. Then, we can write the criterion for the preconditioner estimation in PSGD as

$$\begin{aligned} c(P) =& E[h^T P h + v^T P^{-1} v] \\ =& \mathrm{tr}(P E[hh^T] + P^{-1} E[vv^T]) \\ =& \mathrm{tr}(P H^2 + P^{-1}) \end{aligned} \tag{4}$$

where

$$H^2 = H_0^2 + E[\varepsilon\varepsilon^T]$$

Clearly, $H^2$ is positive semi-definite regardless of the definiteness of $H_0$.

Note, we do not assume any definiteness or sparsity properties for $H_0$. Hence, in general, we assume that $Q$ is fitted on a connected branch of the general linear group with learning rule

$$Q^{\text{new}} = Q^{\text{old}} + dQ = Q^{\text{old}} + Q\mathcal{E} \tag{5}$$

where $Q$ relates to $P$ as

$$P = Q^T Q,$$

and both $dQ$ and $\mathcal{E}$ are sufficiently small matrices related by $dQ = Q\mathcal{E}$. One technical difficulty in proving the convergence of $Q$ with the learning rule (5) is that it cannot exclude any rotation ambiguity of $Q$ as suggested by

$$(UQ)^T(UQ) = Q^T(U^T U)Q = Q^T Q$$

where $U$ can be any matrix on the orthogonal group. To ameliorate this technical issue, we constrain $dQ$ to take on certain structure. In our proof, we assume that both $Q^{\text{old}}$ and $dQ$ are symmetric such that $Q^{\text{new}}$ is always symmetric as well. To achieve such a constraint, we should update $Q^{\text{old}}$ on the Lie group as

$$\begin{aligned} Q' =& Q^{\text{old}} + Q\mathcal{E} \\ Q^{\text{new}} =& [(Q')^T Q']^{0.5} \end{aligned} \tag{6}$$

In this way, $dQ = Q^{\text{new}} - Q^{\text{old}}$ is guaranteed to be symmetric as long as the starting guess $Q^{\text{old}}$ is symmetric as well. Still, in practice we have used the learning rule (5), and (6) only serves for the purpose of proof.

**Theorem A.1.** *Assume that $H$ is invertible, and $dQ = -\mu \frac{\partial c}{\partial Q}$ or $\mathcal{E} = -\mu Q^T \frac{\partial c}{\partial Q}$. Then, $Q$ converges to $H^{-0.5}$ or $-H^{-0.5}$ with the learning rule* (6) *and a small enough positive step size $\mu$.*

*Proof.* Given a small perturbation $dQ$ of $Q$, the change of $P$ is given by

$$\begin{aligned} dP =& (Q + dQ)^T(Q + dQ) - Q^T Q \\ =& Q^T dQ + dQ^T Q + dQ^T dQ \end{aligned}$$

The change of $P^{-1}$ is a little more complicated. We omit any terms smaller than $\mathcal{O}[(dQ)^2]$, and can expand $dP^{-1}$ as below

$$
\begin{aligned}
dP^{-1} =&(P+dP)^{-1} - P^{-1} \\
=& -P^{-1}dPP^{-1} + P^{-1}dPP^{-1}dPP^{-1} \\
=& \left(-Q^{-1}dQP^{-1} - P^{-1}dQ^TQ^{-T} - P^{-1}dQ^TdQP^{-1}\right) \\
&+ (\ Q^{-1}dQQ^{-1}dQP^{-1} + Q^{-1}dQP^{-1}dQ^TQ^{-T} \\
&+ P^{-1}dQ^TdQP^{-1} + P^{-1}dQ^TQ^{-T}dQ^TQ^{-T}\ ) \\
=& -Q^{-1}dQP^{-1} - P^{-1}dQ^TQ^{-T} + Q^{-1}dQQ^{-1}dQP^{-1} \\
&+ Q^{-1}dQP^{-1}dQ^TQ^{-T} + P^{-1}dQ^TQ^{-T}dQ^TQ^{-T}
\end{aligned}
$$

Now, the change of the PSGD criterion is given by

$$
\begin{aligned}
dc =&\text{tr}(dPH^2 + dP^{-1}) \\
=&\text{tr}(H^2Q^TdQ + H^2dQ^TQ + H^2dQ^TdQ) \\
&+ \text{tr}(-Q^{-1}dQP^{-1} - P^{-1}dQ^TQ^{-T} + Q^{-1}dQQ^{-1}dQP^{-1} \\
&+ Q^{-1}dQP^{-1}dQ^TQ^{-T} + P^{-1}dQ^TQ^{-T}dQ^TQ^{-T}) \\
=&2\text{tr}(H^2Q^TdQ - P^{-1}Q^{-1}dQ) \\
&+ \text{tr}(dQ^TdQH^2 + 2dQQ^{-1}dQP^{-1}Q^{-1} + dQP^{-1}dQ^TQ^{-T}Q^{-1})
\end{aligned} \tag{7}
$$

From (7), the first order derivatives of $c$ with respect to $Q$ is given by

$$\frac{\partial c}{\partial Q} = QH^2 - Q^{-T}P^{-1}$$

A stationary point is obtained by letting $\frac{\partial c}{\partial Q} = 0$, which leads to $H^2 = P^{-2}$. Hence, such a $P$ always exists as long as $H^2$ is invertible. Via relationship $dQ = Q\mathcal{E}$, the gradient on the Lie group is

$$\nabla_{\mathcal{E}} = Q^T \frac{\partial c}{\partial Q}$$

Thus, fitting $Q$ on the Lie group yields the same stationary point since $Q$ is invertible. Note that the stationary points only tell us that $Q^TQ = H^{-1}$ without excluding the rotation ambiguity.

Without the constraint $dQ = dQ^T$, the second order derivative of $c$ with respect to $Q$ can be shown to be

$$\frac{\partial^2 c}{\partial(\text{vec}(Q))^2} = 2I \otimes H^2 + 2J^T[Q^{-1} \otimes (QP)^{-T}] + 2[Q^{-T} \otimes (QP)^{-1}]J + 2(QQ^T)^{-1} \otimes P^{-1}$$

where $J$ is the permutation matrix satisfying

$$\text{vec}(dQ^T) = J\,\text{vec}(dQ)$$

Via relationship $dQ = Q\mathcal{E}$, the second order derivative on the Lie group is

$$\nabla^2_{\mathcal{E}} = (Q^T \otimes I)\frac{\partial^2 c}{\partial(\text{vec}(Q))^2}(Q \otimes I) \tag{8}$$

We see that neither $\frac{\partial^2 c}{\partial(\text{vec}(Q))^2}$ nor $\nabla^2_{\mathcal{E}}$ is guaranteed to be positive definite. Hence, the learning rule (5) does not convergence to a point as expected due to the rotation ambiguity.

On the other hand, both $Q$ and $dQ$ are symmetric with the learning rule (6). Thus, the second order derivative of $c$ with respect to $Q$ simplifies to

$$\frac{\partial^2 c}{\partial(\text{vec}(Q))^2} = 2I \otimes H^2 + 2Q^{-1} \otimes (QP)^{-T} + 2Q^{-T} \otimes (QP)^{-1} + 2(QQ^T)^{-1} \otimes P^{-1}$$

At a stationary point, we have $Q = \pm P^{0.5} = \pm H^{-0.5}$. Thus, this second order derivative reduces to

$$\frac{\partial^2 c}{\partial(\text{vec}(Q))^2} \mid_{Q=\pm H^{-0.5}} = 2I \otimes H^2 + 4H^{0.5} \otimes H^{1.5} + 2H \otimes H \tag{9}$$

By (8), the second order derivative on the Lie group is

$$\nabla^2_{\mathcal{E}} \mid_{Q=\pm H^{-0.5}} = 2H^{-1} \otimes H^2 + 4H^{-0.5} \otimes H^{1.5} + 2I \otimes H \tag{10}$$

Now, both $\frac{\partial^2 c}{\partial(\text{vec}(Q))^2}$ and $\nabla^2_{\mathcal{E}}$ are positive definite for an invertible $H$ at the stationary point. Thus, $Q$ converges to either stationary point, i.e., $H^{-0.5}$ or $-H^{-0.5}$. □

From Theorem A.1, we see that $P$ converges to $|H_0|^{-1}$ when $\varepsilon = 0$, i.e., inverse of the "absolute" Hessian. With stochastic gradient noises, $\varepsilon \neq 0$ and we always have $P \prec |H_0|^{-1}$. This is not a bug but rather a feature of PSGD that damps the gradient noises perfect such that $PE[\delta g \delta g^T]P = E[\delta\theta\delta\theta^T]$ Li [2015], a relationship generalizing the Newton method to non-convex and stochastic optimizations. Unlike the damping strategies in other second order methods, this built-in gradient noise damping mechanics of PSGD does not requires any tuning effort.

## B Construction of matrix-free preconditioner

The following statement gives a systematic way to construct a family of black-box matrix-free preconditioners.

**Theorem 3.1.** *Let $K = \{\sigma_1, \ldots, \sigma_m\}$ be a subgroup of the permutation group $S_n$. Then, linear transform $T : \mathbb{R}^n \mapsto \mathbb{R}^n$, $\quad T(x|a_1, \ldots, a_m) = \sum_{i=1}^m a_i \odot \sigma_i(x)$, forms a subgroup of $GL(n, \mathbb{R})$ parameterized with $\{a_1, \ldots, a_m\}$ if $T(\cdot|a_1, \ldots, a_m)$ is bijective, where both $a_i$ and $x$ are in $\mathbb{R}^n$.*

*Proof.* The proof follows by showing that such matrices have the following four properties required to form a Lie group.

First, we show that $I$ is the identity element. Note that $K$ has element $e$ since it is a subgroup of $S_n$. Then without the loss of generality, we can set $\sigma_1 = e$ and $a_1$ to be a vector of ones, and all the other $a_i$, $i > 1$, to be vectors of zeros. This leads to $T(x|a_1, \ldots, a_m) = x$, and thus $T(\cdot|a_1, \ldots, a_m) = I$ when represented as a matrix.

Second, we show that such transforms are closed with binary operation $T^{(1)} \circ T^{(2)}$ defined as $[T^{(1)} \circ T^{(2)}](x) = T^{(1)}[T^{(2)}(x)]$. Specifically, we have

$$\begin{aligned}
[T^{(1)} \circ T^{(2)}](x) &= \sum_{i=1}^m a_i^{(1)} \odot \sigma_i^{(1)} \left( \sum_{j=1}^m a_j^{(2)} \odot \sigma_j^{(2)}(x) \right) \\
&= \sum_{i=1}^m \sum_{j=1}^m [a_i^{(1)} \odot \sigma_i^{(1)}(a_j^{(2)})] \odot \sigma_i^{(1)}(\sigma_j^{(2)}(x))
\end{aligned}$$

Since $K$ is a subgroup of $S_n$, $\sigma_i^{(1)}(\sigma_j^2(\cdot))$ still belongs to $K$. Hence, $T^{(1)} \circ T^{(2)}$ will have the same form as $T(\cdot|a_1, \ldots, a_m)$ after merging like terms.

By representing $T(\cdot|a_1, \ldots, a_m)$ as a matrix, it is clear that the associativity property, i.e., $(T^{(1)} \circ T^{(2)}) \circ T^{(3)} = T^{(1)} \circ (T^{(2)} \circ T^{(3)})$, holds since matrix multiplication is associative. Lastly, the inverse of $T(\cdot|a_1, \ldots, a_m)$ exists since we assume that $T(\cdot|a_1, \ldots, a_m)$ is bijective, and thus its representation is an invertible matrix. □

**Corollary B.0.1.** *We want to point out that not all simple matrix-free preconditions can be constructed by Theorem 3.1. Let us take the widely used feature normalization transform, e.g., batch normalization Ioffe and Szegedy [2015], layer normalization Ba et al. [2016] and group normalization Wu and He [2018] as an example. We have*

$$T(x|\mu, \sigma) = (x - \mu)/\sigma$$

*where $\mu$ and $\sigma$ are either scalar or vector mean and standard deviation of $x$, respectively. This $T(\cdot|\mu, \sigma)$ forms a sparse affine group for $\sigma \neq 0$ Li [2019]. However, we cannot use such preconditioners as black-box ones.*

## C Construction of low-rank approximation preconditioners

The following property states that the proposed low-rank approximation preconditioners can fit both ends of the spectra of Hessian. It is not difficult to prove this statement. But, it reveals an important advantage over traditional low-rank approximation preconditions with form $P = \rho I + UU^T$, whose eigenvalues are lower bounded by $\rho$.

### C.1 Notations

Let $Q = (I + UV^T)B$, where $U$ and $V$ are two tall thin matrices, and $B$ is a matrix on certain group, e.g., the group of diagonal matrix, or simply a scalar. It is an important preconditioner as after reparameterization, we can have $Q = \mathrm{diag}(d) + UV^T$ for diagonal matrix $B$, which relates to the LM-BFGS, HF, and conjugate gradient (CG) methods. This preconditioner is efficient if low-rank modication can significantly further reduce the condition number [1] after preconditioning the Hessian with a diagonal matrix, i.e., a Jacobi preconditioner. One also can think $Q$ as a low-rank approximation of the inverse square root of a positive definite Hessian. Note that this form of preconditioner can fit both tails of the spectra of Hessian.

**Theorem 3.2.** *Preconditioner $P = Q^TQ$ with $Q = \rho(I + UV^T)$ can have positive eigenvalues arbitrarily larger than $\rho^2$ and arbitrarily smaller than $\rho^2$ with proper $U$ and $V$.*

*Proof.* Let us check the simplest case, i.e., $\rho = 1$, $U = u$ and $V = v$. Then, $P$ is shown to be

$$\begin{aligned} P =&(I + vu^T)(1 + uv^T) \\ =&I + vu^T + uv^T + (u^Tu)vv^T \end{aligned}$$

This $P$ has two eigenvalues determined by $u$ and $v$, say $\lambda_1$ and $\lambda_2$. They satisfy

$$\begin{aligned} \lambda_1\lambda_2 =&(1 + u^Tv)^2 \\ \lambda_1 + \lambda_2 =&2 + 2u^Tv + \|u\|^2\|v\|^2 \end{aligned}$$

By choosing $u$ and $v$ properly, these two eigenvalues can be arbitrarily smaller or larger than 1. For example, by letting $u^Tv = 0$ and $\|u\| = \|v\| \to \infty$, we have $\lambda_1\lambda_2 = 1$ and $\lambda_1 + \lambda_2 \to \infty$. Hence, we must have one eigenvalue arbitrarily large, and the other one arbitrarily small. In general, the order of rank can be larger than 1, and thus more degree of freedoms for fitting the spectra of Hessian. □

**Theorem 3.3.** *If $\rho \neq 0$ and $(I+V^TU)^{-1}$ or $(I+U^TV)^{-1}$ exists, $A_V(\rho, U) = \rho(I+UV^T)$ defines a subgroup of $GL(n, \mathbb{R})$ parameterized with $\rho$ and $U$. Similarly, $A_U(\rho, V) = \rho(I + UV^T)$ defines another subgroup of $GL(n, \mathbb{R})$ parameterized with $\rho$ and $V$.*

*Proof.* Without the loss of generality, we assume $\rho = 1$, and simplify rewrite $A_V(1, U)$ as $A_V(U) = I + UV^T$. We can show that $A_V(U)$ forms a Lie group by revealing the following facts

$$\begin{aligned} A_V(0) =&I \\ A_V(U_1)A_V(U_2) =&A_V(U_1 + U_2 + U_1V^TU_2) \\ A_V^{-1}(U) =&A_V[-U(I + V^TU)^{-1}] \end{aligned}$$

i.e., the existence of identity element, closed with respect to matrix multiplication, and the existence of inverse, respectively. The last equation is simply the rewriting of the Woodbury matrix identity, i.e.,

$$[I + UV^T]^{-1} = I - U(I + V^TU)^{-1}V^T$$

The condition that $(I + V^TU)^{-1}$ exists is necessary as otherwise $A_V(U)$ is singular. Lastly, the associativity property clearly holds for matrix multiplications. Hence, all such matrices form a Lie group. Similarly, we can show that $A_U(V) = I + UV^T$ defines another Lie group. □

[1]Since $PH$ is not symmetric, smaller eigenvalue spread does not always suggest smaller condition number, e.g., $[1, a; 0, 1]$ has arbitrarily large conditioner number for $|a| \to \infty$. In PSGD, $P$ does not amplify the gradient noise (ref Li [2015], page 5-6, section IV.B), and thus avoids such degraded solution.

#### C.1.1 The rotation ambiguity and Schur decomposition

Note that $UV^T = UQ(VQ)^T$ for any orthogonal matrix $Q$. Thus, we can remove this rotation ambiguity by selecting a $Q$ such that $Q^T(V^TU)Q$ is an upper triangular block matrix with block size 1 or 2, i.e., the Schur decomposition. $A_V(U)$ and $A_U(V)$ still form groups by constraining $V^TU$ to be quasi-triangular.

## D Low-rank approximation preconditioner fitting

In practice, we seldom use $Q = \rho(I + UV^T)$ as a preconditioner directly. Its limitation is clear, i.e., most of its eigenvalues are $\rho^2$ with $r \ll n$. In our method, we start from a rough preconditioner guess, say $B$, and modify it with $I + UV^T$ to have the refined preconditioner as

$$Q = (I + UV^T)B$$

If matrix $B$ is from another Lie group, we still can update $Q$ efficiently. For example, $B$ can be a diagonal matrix with nonzero diagonals. Then this composited preconditioner reduce to a diagonal one when $r = 0$.

**Computation** Note that neither of or methods require direct formation of a curvature matrix. Instead we use Hessian vector products. One can utilize auto-differentiation packages to calculate the exact Hessian vector product or approximate them with finite differences both detailed by [Pearlmutter, 1994]. Given a neural network with N parameters, the Hessian vector calculation can be done in O(N) time and space, and does not make any approximation. Additionally, we often only calculate the preconditioner with probability p = 0.1, making the PSGD as practical as SGD.

### D.1 Fundamental operations on Lie Group

#### D.1.1 The concept of group generator

Unlike the additive update to move a point in a Euclidean space, we use multiplicative update to move a point on the Lie group. For example, we can move $A_V(U)$ to any its neighbor, say $A_V(U + \mathcal{E})$, as

$$A_V\left(\mathcal{E}(I + V^TU)^{-1}\right) A_V(U) = A_V(U + \mathcal{E})$$

Since $A_V(\mu U) = I + \mu UV^T = e^{\mu UV^T}$ for $\mu \to 0$, $UV^T$ is a group generator for any $U \neq 0$. Indeed, the Lie algebra is closed as shown by

$$< U_1V^T, U_2V^T >= U_1V^TU_2V^T - U_2V^TU_1V^T = (U_1V^TU_2 - U_2V^TU_1)V^T$$

where $< \cdot, \cdot >$ is the Lie bracket.

#### D.1.2 The gradients for preconditioner updating

Here, the gradient always refers to the one on the Lie group. Thus $dA$, say on group $A_V(U)$, is either $A_V(\mathcal{E})A_V(U)$ or $A_V(U)A_V(\mathcal{E})$, where $\mathcal{E} \to 0$. Since we will update both groups, we simple put it as $dA = \mathcal{E}_1A$ or $dA = A\mathcal{E}_2$. Let's drop the $E$ in the PSGD preconditioner fitting criterion and derive the stochastic gradient as below,

$$\begin{aligned}
&d(h^TPh + v^TP^{-1}v)\\
=&h^TdPh - v^TP^{-1}dPP^{-1}v\\
=&2h^TdQ^TQh - 2v^TP^{-1}dQ^TQP^{-1}v
\end{aligned}$$

Since we are to fit $A$ and $B$ on the Lie groups, thus let

$$\begin{aligned}
dQ =&dAB + AdB\\
=&\mathcal{E}_1AB + AB\mathcal{E}_2\\
=&\mathcal{E}_1Q + Q\mathcal{E}_2
\end{aligned}$$

Then, we have

$$\begin{aligned}
& d(h^T P h + v^T P^{-1} v) \\
=& 2h^T dQ^T Q h - 2v^T P^{-1} dQ^T Q P^{-1} v \\
=& 2h^T Q^T \mathcal{E}_1^T Q h + 2h^T \mathcal{E}_2^T Q^T Q h - 2v^T P^{-1} Q^T \mathcal{E}_1^T Q P^{-1} v - 2v^T P^{-1} \mathcal{E}_2^T Q^T Q P^{-1} v \\
=& 2h^T Q^T \mathcal{E}_1^T Q h + 2h^T \mathcal{E}_2^T P h - 2v^T Q^{-1} \mathcal{E}_1^T Q^{-T} v - 2v^T P^{-1} \mathcal{E}_2^T v \\
=& 2\mathrm{tr}\left\{\mathcal{E}_1^T \left[(Qh)(Qh)^T - (Q^{-T}v)(Q^{-T}v)^T\right]\right\} + 2\mathrm{tr}\left\{\mathcal{E}_2^T \left[(Ph)h^T - v(P^{-1}v)^T\right]\right\} \quad (11)
\end{aligned}$$

(12)

### D.1.2.1 Gradient with respect to $B$

For diagonal matrix, we simply have

$$0.5\nabla_B = \mathrm{diag}[(Ph)h^T - v(P^{-1}v)^T]$$

from (11), and thus update $B$ as

$$B \leftarrow B(I - \mu\nabla_B)$$

where the step size $\mu$ is small enough such that $\mu\|\nabla_B\| < 1$, and $\|\nabla_B\|$ is just the max absolute diagonal element for a diagonal matrix. Here, $\|\cdot\|$ denotes spectral norm.

For a diagonal matrix, we also can update its diagonals with element-wise step sizes as the Lie group reduces to the direct sum of a series smaller ones with dimension one. This could lead to faster convergence when the true Hessian is diagonal. Otherwise, we do not observe any significant difference between these two step size selection strategies in our numerical results.

### D.1.2.2 Gradient with respect to $U$ on group $A_V(U)$

Since

$$dA = (I + \mathcal{E}V^T)(I + UV^T) - (I + UV^T) = \mathcal{E}V^T A$$

we replace the $\mathcal{E}_1$ in (11) with $\mathcal{E}V^T$ to obtain gradient

$$0.5\nabla_U = [(Qh)(Qh)^T - (Q^{-T}v)(Q^{-T}v)^T]V$$

Then, we update $A$ as

$$A \leftarrow A - \mu\nabla_U V^T A$$

which suggests its parameter can be updated as

$$U \leftarrow U - \mu\nabla_U (I + V^T U)$$

since we are operating on the group $A_V(U)$, where the step size is small enough such that $\mu\|\nabla_U V^T\| < 1$. Note that $\nabla_U V^T$ has at most rank two. Hence, its Frobenius norm can be used to bound its spectral norm tightly, as shown by

$$\|\nabla_U V^T\|_F/\sqrt{2} \le \|\nabla_U V^T\| \le \|\nabla_U V^T\|_F$$

### D.1.2.3 Gradient with respect to $V$ on group $A_U(V)$

As

$$dA = (I + U\mathcal{E}^T)(I + UV^T) - (I + UV^T) = U\mathcal{E}^T A$$

we replace the $\mathcal{E}_1$ in (11) with $U\mathcal{E}^T$ to give gradient

$$0.5\nabla_V = [(Qh)(Qh)^T - (Q^{-T}v)(Q^{-T}v)^T]U$$

Thus, we update $A$ as

$$A \leftarrow A - \mu U\nabla_V^T A$$

which implies that its parameter $V$ should be updated as

$$V \leftarrow V - \mu(I + VU^T)\nabla_V$$

where the step size is small enough such that $\mu\|U\nabla_V^T\| < 1$. Again, $U\nabla_V^T$ has at most rank two, and its Frobenius norm gives a tight enough estimation of its spectral norm.

## E More Experimental Results

### E.1 MNIST Handwriting Digit Recognition

To have a fair comparison between the diagonal and low rank approximation (LRA) preconditioners, we slightly upgrade $Q$ in the LRA preconditioner to form

$$Q = \mathrm{diag}(d)(I + UV^T)$$

where $d$ is a vector. This form of $Q$ cannot form a Lie group. Still, its two factors, $\mathrm{diag}(d)$ and $I + UV^T$, can be fitted on their own Lie groups. Now, the diagonal preconditioner is a special case of this LRA one with order $r = 0$.

We have tested orders $r = 0, 1, 2, 5$, and $10$. The batch size is $64$. Totally ten epochs of training are performed. The learning rate for parameter updating is annealed exponentially from $0.1$ for the first epoch to $0.001$ for the tenth epoch. The step size for preconditioner fitting is annealed exponentially from $0.1$ for the first epoch to $0.01$ for the tenth epoch. Preconditioned gradient norm is clipped to $10$ if too large. No momentum is used. Figure 1 summarizes the test classification error rates for preconditioners with different order of approximations over $50$ runs. From Fig. 1, we see that the simple diagonal preconditioner performs well. Still, LRA brings marginal gains up to order $r = 5$. This cost function is fairly 'flat' since the only nonlinearities in LeNet5 are two piece-wise linear functions, i.e., the activation function ReLU and max pooling one. Only the cross entropy loss introduces the 'curvature'.

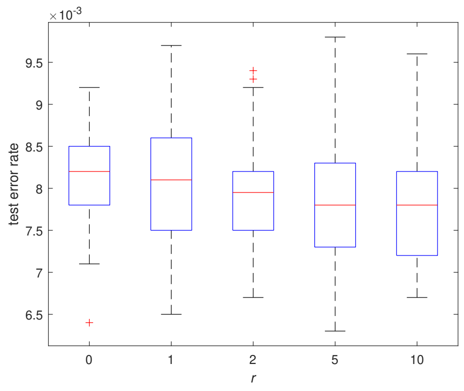


Figure 3: MNIST test classification error rates over $50$ runs using preconditioners with different orders of LRA. The one with order $0$ reduces to the diagonal preconditioner. Table 1 reports results of classification accuracy for $r = 5$. Higher is better.

### E.2 CIFAR10 Image Classification with ResNet18

We follow the implementations of Adabelief[2] algorithm Zhuang et al. [2020] to test preconditioned SGD (PSGD) on the CIFAR10 image classification task with the ResNet18 model. One main difference from the implementations in Zhuang et al. [2020] is that we reduce the learning rate by tenfold twice for all the optimizers, while the original stage learning rate scheduler only anneals the step size once. We also consider the cosine learning rate scheduler, which helps SGD to achieve the state-of-the-art (SOTA) test accuracy about $95.5\%$. Training and testing accuracy convergence curves over $16$ runs are plotted in Fig. 2. We only show the results of PSGD and SGD here as SGD is known to achieve the SOTA results for this problem.

For PSGD, we use step size $0.02$ for parameter updating and $0.01$ for preconditioner fitting. The preconditioner is only updated once per ten iterations, and thus its overhead over SGD is marginal. The same momentum factor, $0.9$, is used for both SGD and PSGD. Since the step size in PSGD is normalized, we update the momentum as $m \leftarrow 0.9m + 0.1g$, instead of $m \leftarrow 0.9m + g$ as in the SGD. No gradient clipping is used. Weight decay is realized by adding the L2 regularization term $0.5\lambda\theta^T\theta$ to the cross entropy loss. We have found that $\lambda$ between $0.01$ and $0.02$ performs the best.

[2]https://github.com/juntang-zhuang/Adabelief-Optimizer

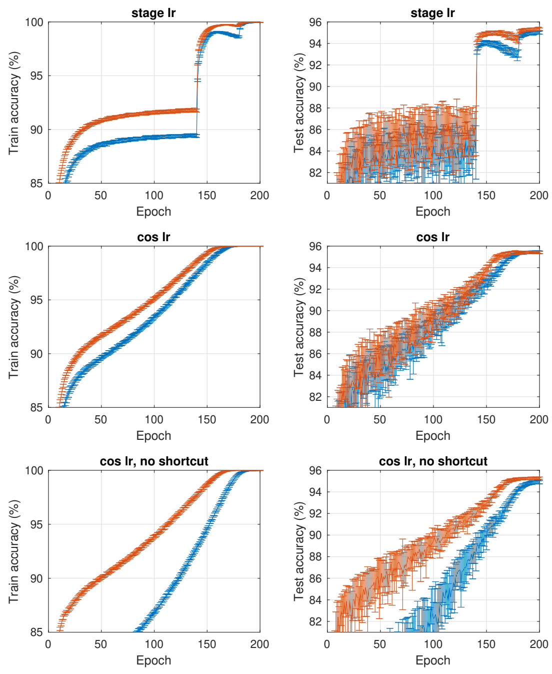


Figure 4: CIFAR10 image classification with ResNet18. The order of low-rank Hessian approximation is 10. Mean and variance are estimated over 16 runs. Higher is better.

From Fig. 2, we observe that SGD performs very well with the cosine learning rate scheduler. This is expected as these residual networks are highly evolved to be first order optimizer friendly. The extensive use of piece-wise linear functions, residual connections and batch normalizations make these models fairly 'flat' and resemble shallow models, instead of deep ones Veit et al. [2016]. Still, PSGD slightly outperforms SGD when we remove the shortcut connections or use a less tuned learning rate scheduler, e.g., the stage one here.

**PSGD is robust to hyper-parameters** To showcase the robustness of PSGD to hyper-parameters we show in table 9 the test classification accuracy of a ResNet18 trained on CIFAR10 using different learning rate and weight decay.

| PSGD XOR | Learning Rate | Weight Decay | Test Accuracy |
|---|---|---|---|
| XMat | 2e-2 | 2e-2 | 95.32 |
| LRA | 2e-2 | 2e-2 | 95.69 |
| LRA | 5e-2 | 2e-2 | 95.48 |
| LRA | 5e-2 | 2e-2 | 95.46 |
| LRA | 4e-2 | 2e-2 | 95.55 |
| LRA | 3e-2 | 2e-2 | 95.57 |
| LRA | 2e-2 | 1e-2 | 95.45 |
| LRA | 2e-2 | 3e-2 | 95.51 |

Table 9: We see that PSGD is robust to hyper-parameter selection making tuning significantly easier compared to other optimization methods.

We clearly see the test accuracy of PSGD in a wide range or learning rate and weight decay converges within the expected rage of $95.49 \pm 0.08\%$.

## E.3 A Large-Scale Logistic Regression Problem

We consider a simple logistic regression model to solve the MNIST classification problem, with and without Bernoulli noise. We considered AdaHessian, AdaBelief, SGD, LBFGS, PSGD LRA/XMat, KFAC, and HessianFree optimizers. Let $x$ be the vector of the image with length $28^2$. Instead of regression on vector $x$, we do the regression on the outer product vector of $x$, which has length $28^4$. This significantly increases the test classification accuracy but leads to a large regression matrix with over six million coefficients. The KFAC preconditioner did not fit on our GPU and the HessianFree optimizer would diverge far from all other optimizers in more than 50% of our runs. Both are omitted from further discussion in this section.

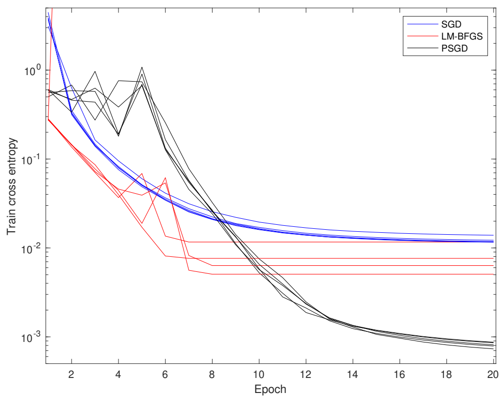
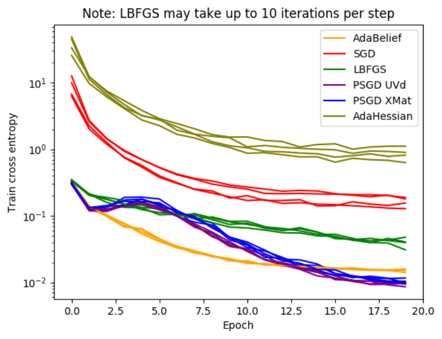


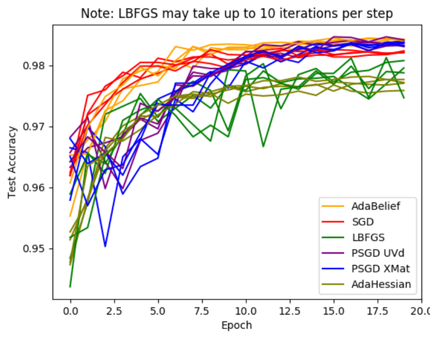
Figure 5: Standard MNIST dataset: Typical convergence curves on the logistic regression problem. Lower is better. When comparing the convergence speed, one should be aware that one step of LM-BFGS may take up to ten iterations, while SGD and PSGD always one iteration per step.

No momentum is considered since this is the case for LM-BFGS. The train batch size is $500$. It is tricky to select the initial learning rate for LM-BFGS even though we exponentially anneal it. We have found that LM-BFGS diverges on roughly one-third of the trials with an initial learning rate of $0.1$, but $0.05$ is too small and may lead to worse performance than SGD. For PSGD, we consider the LRA preconditioner with order $10$ and set the learning rates for parameters and preconditioner to $0.05$ and $0.1$, respectively. Since LM-BFGS might diverge with a learning rate of $0.1$, we only show a few typical convergence curves of SGD, LM-BFGS, and PSGD in Fig. 5. LM-BFGS converges to regression losses a few times smaller than SGD. PSGD could converge to losses about one order of magnitude lower than that of SGD and LM-BFGS. Regarding test classification error rate, we have $2.37\% \pm 0.08$, $2.09\% \pm 0.18$, $1.98\% \pm 0.08$ for SGD, LM-BFGS, and PSGD, respectively, averaged over ten runs. Again, LM-BFGS outperforms SGD, and PSGD performs the best on the test classification error rate as well.

Next, we simply randomly add Bernoulli noise to the MNIST dataset to add diversity to the dataset. Even though LM-BFGS is the standard optimizer for large-scale logistic regression we wanted to consider some other SOTA optimizers for deep learning. PSGD UVd/XMat performed the best in this scenario but we found that AdaBelief does surprisingly well given its lack of second-order information. Losses and Accuracies plotted in Fig 6



(a) **Train Loss**



(b) **Test Accuracy**

Figure 6: We consider logistic regression on the MNIST dataset with Bernoulli noise. We see PSGD finds the lowest loss on the train set, with the Belief mechanism also working well. PSGD and AdaBelief generalize well to the Accuracy of the Test Dataset.

We see that PSGD significantly outperforms the SOTA optimizers in the convex logistic regression setting under noise-free or noisy domains while being memory efficient.

### E.4 Forgettability & Uncertainty: Rank Analysis

Very recently, Toneva et al. [2018] shows that Forgettable points are crucial for generalization in the supervised setting. In the unsupervised, semi-supervised, self-supervised and generative setting Pooladzandi et al. [2023] shows that one can use the low entropy samples generated by the generative model to significantly boost classification performance of the Latent Energy Based Model which acts as generative-classifier. Here we acknowledge the previous findings and focus on the supervised setting. We train a ResNet18 on CIFAR-10 and record entropy and forgettability statistics. We plot the strong correlation between low forgettability score and low entropy found in our statistics in Fig. 7 and provide correlation coefficients in Table 10.

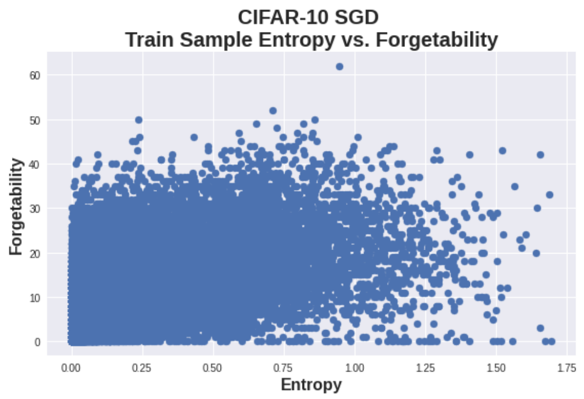


Figure 7: There is a strong correlation between the forgettability ordering and the entropy ordering. i.e. points that are unforgettable have a very low entropy.

| Entropy v Forgettability Score | Correlation Coefficient SGD | PSGD | p-value SGD | PSGD |
|---|---|---|---|---|
| Spearman | 0.72 | 0.73 | 0 | 0 |
| Pearson | 0.57 | 0.65 | 0 | 0 |
| Kedal Tau | 0.54 | 0.56 | 0 | 0 |
| Weighted Tau | 0.60 | 0.75 | 0 | 0 |

Table 10: Comparison of correlation metrics comparing Entropy Score vs Forgettability score between SGD and PSGD. We see that PSGD's average correlation between entropy and forgettability is stronger than that of SGD.

**Generalization Gap Between High and Low Entropy Subsets** We train an oracle LeNet5 on the full MNIST dataset for 20 epochs, we categorize the train set into two subsets; a high entropy subset and a low entropy subset, each consisting of 10k points. The entropy is defined over the softmax of the logits. We then train two different LeNet5's on each dataset. We find that the low entropy dataset does not generalize to the test set well achieving a test accuracy of 74%, whereas the high entropy dataset achieves a 99.3% which is on par with oracle LeNet5 (see 11). This clearly shows for supervised learning the high entropy points are most important for generalization.

Furthermore, we find that training on low entropy points results in low entropy a lower mean entropy network. This supports the hypothesis that there are certain points that the net can lower the entropy over which do not lead to generalization. In fact we see in Table 11 that training over the full dataset resulted in a higher mean low entropy compared to that of the full dataset. This supports the idea that high entropy points are important for supervised learning.

In terms of distance from initialization, we see that the network is pushed farther from initialization when using the full dataset than while using the high entropy points and the least when using the low entropy points. This supports that training with low-entropy or unforgettable points unnecessarily pushes a network's parameters far from initialization reducing generalization capacity. Furthermore, we see that when training on low entropy points, PSGD does not push the neural network far from initialization preserving generalization capacity Cao and Gu [2019].

| MNIST Statistics | Accuracy | Expected Low Entropy | Expected High Entropy | Distance from Initilization SGD | Distance from Initialization PSGD |
|---|---|---|---|---|---|
| Full Dataset | 99.3% | 5e-16 | 6e-3 | 23.1 | 26 |
| High Entropy Subset | 99.3% | 1e-10 | 2e-2 | 21.7 | 21.2 |
| Low Entropy Subset | 74.2% | 6e-18 | 4e-4 | 9.2 | 8.7 |

Table 11: Comparing generalization accuracy of a LeNet5 trained on either 10k high or 10k low entropy datapoints. We see high entropy datasets can match generalization of the full dataset whith a higher test entropy compared to the other datasets. We see that the distance from initialization is less for the low entropy points.

Finally, we see that a NN trained on low entropy points has mean low entropy 2 orders of magnitude smaller than training on the full dataset, and 8 orders of magnitude smaller than training on the high entropy subset. This shows that low entropy points cause a level of confidence in a NN which is not needed and in some cases can be hurtful to generalization 5.1.

### E.5 Effect of Rank on XOR

The full rank version of PSGD Li [2019] can solve the XOR problem with an RNN past delay of 128. Since we can arbitrarily increase the rank of Hessian approximation in our LRA version of PSGD, we consider the effect of rank on convergence on the XOR problem using an RNN. We find rank approximations of $r = 0, 1, 2, 5$, and 10 converge with probability $p = 0.1, 0.4, 0.8, 1$ and 1 respectively.

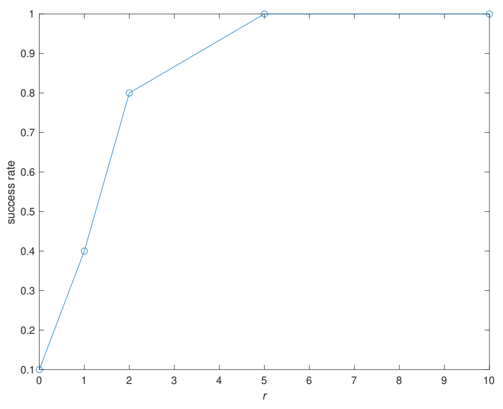


Figure 8: Success rate over ten runs in solving the XOR problem with a simple RNN and LRA preconditioners of orders 0, 1, 2, 5 and 10. Higher is better.

This example shows a typical problem where the diagonal preconditioner struggles, while a low order Hessian approximation works perfectly for preconditioning.

For all experiments both step sizes for parameter and preconditioner updating are fixed to $0.01$. The gradient clipping threshold is set to $1$. No momentum is used. We run 10 runs per optimizer for $100,000$ iterations or until convergence. The success rates over ten runs for each r are plotted in Fig. 8. This example shows a typical problem where the diagonal preconditioner struggles, while a low-order Hessian approximation works perfectly for preconditioning.

### E.6 Potential Social Impacts and Limitations

Regarding social impact, our optimization method can outperform other optimization methods in generalization accuracy as well as minimizing loss with negligible computational overhead and less tuning efforts due to normalized learning rates. This will enable better training of machine learning systems. Furthermore, since PSGD is better at generalization based on imbalanced datasets, it has the potential to reduce bias and provide better representation for under-represented classes and sub-classes.

The main potential limitation for PSGD we see, is that its certain forms. e.g., low-rank approximation one, require more memory to store the curvature information. Yet, it is still negligible compared to other popular second-order methods like KFAC that require per-sample derivatives.

### E.7 Hardware & Software

All experiments were run on a single NVIDIA 3080 10GB GPU with an 11th gen Intel i7 processor. We utilized PyTorch Paszke et al. [2017] version 1.13.

For now we have zipped the codebase to re-create our results, and will later host on github.