Black Box Lie Group Preconditioners for SGD

Xilin Li XILIN@META.COM
Facebook SUN 103, 900 Discovery Wy, Sunnyvale, CA 94089

Abstract

A matrix free and a low rank approximation preconditioner are proposed to accelerate the convergence of stochastic gradient descent (SGD) by exploiting curvature information sampled from Hessian-vector products or finite differences of parameters and gradients similar to the BFGS algorithm. Both preconditioners are fitted with an online updating manner minimizing a criterion that is free of line search and robust to stochastic gradient noise, and further constrained to be on certain connected Lie groups to preserve their corresponding symmetry or invariance, e.g., orientation of coordinates by the connected general linear group with positive determinants. The Lie group's equivariance property facilitates preconditioner fitting, and its invariance property saves any need of damping, which is common in second-order optimizers, but difficult to tune. The learning rate for parameter updating and step size for preconditioner fitting are naturally normalized, and their default values work well in most situations.

1. Introduction

Second-order optimization for machine learning models with millions of free parameters to learn is challenging. Off-the-shelf convex optimization algorithms [1], to name a few, quasi-Newton ones like the Broyden–Fletcher–Goldfarb–Shanno (BFGS) and its limited-memory version, LM-BFGS, conjugate gradient (CG) and its nonlinear versions such as Hessian-free (HF) optimization [10], are successful for small-scale convex mathematical optimization problems, but not commonly used for large-scale stochastic optimization problems like those that arise from machine learning (ML). One of the most prominent hindrances is their dependence on the line search step. The cost functions in many ML models, e.g., variational and reinforcement learning models, are simply defined as expectations, and the only way to evaluate them is via Monte Carlo (MC) sampling averages, which could have large variances. An optimizer relying on line search to ensure convergence could be problematic for them. This issue is less grave in ML problems like classification and regression. Hopefully, line search becomes benign with increased sample sizes. But, large sample sizes bring other new issues, e.g., increased generalization gap and low computational efficiency. Empirical results suggest that the plain SGD is a highly efficient optimizer for most ML problems. Still, conceivably, SGD will converge slowly for problems with large eigenvalue spread once the solution is located in a basin of attraction. Regret optimizers like RMSProp and Adam [6] converge faster but are empirically shown to generalize less well on many problems. Reducing the generalization gap between SGD and Adam is still an active topic [12], although not the focus here. We believe that all SGD needs is 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 the Hessian-vector products or finite differences of parameters and gradients similar to the BFGS algorithm. As discussed above, we may not be able to construct a preconditioner in a deterministic way as in BFGS since line search could be problematic. Thus, we adopt a more

general and gradient noise robust preconditioner fitting criterion proposed in [7], and fit the preconditioner online with another 'gradient descent' algorithm. The point is that we should not turn the preconditioner fitting problem into a more difficult and computationally expensive one than the original parameter learning problem. Here, the Lie group is the perfect tool for preconditioner fitting. Note that the 'gradient descent' on a Lie group is similar to but different from the common gradient descent in Euclidean space. It is achieved by applying a series of small transforms via multiplication with $I + \mu G$, where μ is a small scalar, and G is the group generator. A Lie group is actually a quite friendly object to work with. Moving a preconditioner around any point on a Lie group just behaves as moving it around the identity element of the group, i.e., the identity matrix I. This is known as the equivariance property of a Lie group.

2. Background

2.1. The Notations

We are to minimize a loss function defined via expectation as $f(\theta) = E_z[\ell(\theta, z)]$, 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 always assume that the considered problem is second-order differentiable. To simplify the handwritings, we just use $\hat{f}(\theta)$ to denote one sampled noisy evaluation of $f(\theta)$. Then, one step of SGD with learning rate μ and an optional positive definite preconditioner P writes as

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

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 θ^*, 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 figure out a good approximator for H^{-1}. But, $\hat{H}$ is a noisy Hessian, and not necessarily positive definite, even if the exact one at θ^*, i.e., H, is.

2.2. The Preconditioner Fitting Criterion

We adopt the preconditioner fitting criterion proposed in [7]. Following their notations, let δg be the perturbation of gradient associated with parameter perturbation $\delta\theta$. Then, this 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 simply replace 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 computationally cheap as the gradients. Criterion (3) only has one positive definite solution, $P = (H^2 + E_v[\epsilon^2])^{-1/2}$, even for indefinite H, where $\epsilon = \hat{H} - H$ is a stochastic noise term. Hence, this preconditioner automatically damps gradient noise. It is worth noting that criterion (3) gives the same preconditioner used in equilibrated SGD (ESGD) [3] and AdaHessian [11] 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 Lie group for several reasons. First, let us rewrite (1) as $P^{-1/2}\theta_{i+1} = P^{-1/2}\theta_i - \mu\, \partial \hat{f}(\theta)/\partial(P^{-1/2}\theta)\,|_{\theta=\theta_i}$. Now, it is clear that a preconditioned SGD equals SGD with a new set of coordinates defined by $\vartheta = P^{-1/2}\theta$. A coordinate change consists of rotations and scalings, i.e., operations on the orthogonal group $O(n)$ and the group of nonsingular diagonal matrices. Let us represent this coordinate transform with matrix Q^{-1}, and accordingly, $P = Q^T Q$. Now, we can pursue a variable Q on the Lie group to fit this coordinate transform.

Second, preconditioned SGD also can be viewed as SGD with transformed features when the parameters to be learned are a list of affine transform matrices [8]. Specifically, the most commonly used feature transformations, e.g., whitening, normalization, and scaling, can be represented as matrices on Lie groups. For example, the popular batch normalization operation [5] can be represented as a sparse preconditioning matrix on a Lie group where only the diagonal and last column can have nonzero values [8]. Again, the Lie group arises as a natural object to work with.

Lastly, the Lie groups have two wonderful properties suitable for our task. As with any group, a certain Lie group preserves certain symmetry or invariance. For example, with $Q \in GL^+(n, \mathbb{R})$, the general linear group with positive determinant, ϑ and θ will always have the same orientation. This saves the need for any damping or similar remedies for avoiding degenerated solutions since $P = Q^T Q$ is guaranteed to be always 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.

The preconditioners proposed in [8] can only be applied to a list of affine transform matrix parameters. Although many machine learning models indeed exclusively consist of affine transforms and nonlinear activation functions, this is not always the case. Also, practically, it is inconvenient to reparameterize many existing modules, e.g., a convolutional layer, into their equivalent affine transform forms. Hence, two types of novel black box preconditioners are proposed in this paper.

3. Black Box Lie Group Preconditioners

3.1. Simple Matrix Free Preconditioners

The term 'matrix free' suggests that we do not explicitly form the matrix representations. The Lie groups keep to be abstract forms, i.e., transformations in vector space, $T : \mathbb{R}^n \mapsto \mathbb{R}^n$. The following theorem gives one systematic way to construct such sparse Lie group preconditioners.

Proposition 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 $\mathbb{R}^n$.

Proposition 1 is proved by showing that T can be reduced into the direct sum of $\lceil n/|K| \rceil$ smaller irreducible Lie groups, where $|K|$ is the order of K. We list a few prominent examples below.

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 [3] and AdaHessian [11] as a special case when P is on this group.

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

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

where $c = a \odot \sigma_f(a) - b \odot \sigma_f(b)$. Clearly, such transforms form a Lie group if 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 [2]. For an even n, subgroup $K = \{e, s_{n/2}\}$ induces a Lie group whose representations are invertible butterfly matrices, where $s_{n/2}$ denotes circular shifting by $n/2$ positions. This group of matrices are the building blocks of the Kaleidoscope matrices [2].

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 enough, but empirically shown to be less effective without the help of momentum for certain machine learning problems. We are mostly interested in the cases with $2 \leq |K| \leq 4$. These Lie groups are sparse enough, yet simple enough to derive their inverse explicitly, and at the same time could 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 in processing large-scale matrices. Commonly adopted forms of positive definite low-rank approximation, e.g., $P = \rho I + UU^T$, cannot always be factorized as $P = Q^T Q$ such that Q is on certain Lie groups, where $\rho > 0$ is a small positive number. Furthermore, this is not an effective form of approximation for reducing eigenvalue spread. The Hessians in many real-world problems typically have a few very large and very small eigenvalues, i.e., tails on both ends of the spectra. But, all the eigenvalues of P in this form are lower bounded by ρ. Thus, it can only fit one tail of the spectra when $\text{rank}(U) \ll n$.

For this reason, we propose a new low-rank approximation with form $Q = \rho(I + UV^T)$, where ρ 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 show two facts. First, preconditioner $P = Q^T Q$ with this form can fit both tails of the spectra of Hessian. Second, we can update this preconditioner on Lie groups.

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

Proposition 3: If $\rho \neq 0$ and $(I + V^T U)^{-1}$ or $(I + U^T V)^{-1}$ exists, $A_V(d, U) = \rho(I + UV^T)$ defines a subgroup of $GL(n, \mathbb{R})$ parameterized with ρ and U. Similarly, $A_U(\rho, V) = \rho(I + UV^T)$ defines another subgroup of $GL(n, \mathbb{R})$ parameterized with ρ and V.

Proposition 2 can be shown to be true just by checking the simplest case with $n = 2$ and $r = 1$. Proposition 3 can be shown to be true by verifying the four requirements of being a Lie group. Notably, the Lie algebra is closed, e.g., $< U_1 V^T, U_2 V^T >= (U_1 V^T U_2 - U_2 V^T U_1) V^T$ on Lie group $A_V(U)$, where $< \cdot >$ is the Lie bracket. The condition that $(I + V^T U)^{-1}$ or $(I + U^T V)^{-1}$ exists is to ensure that $I + UV^T$ is invertible as shown by the Woodbury matrix identity. The rotation ambiguity, i.e., $I + UV^T = I + UR(VR)^T$ for any $RR^T = I$, can be removed by assuming $I + V^T U$ having the Schur decomposition form, if necessary.

Table 1: Test classification error rates (%) on MNIST with LeNet5 over ten runs. Lower is better.

SGD+M	Adam [6]	KFAC [9]	PSGD [8]	PSGD XMat	PSGD LRA
0.96 ± 0.07	0.88 ± 0.07	0.84 ± 0.07	0.74 ± 0.06	0.78 ± 0.06	0.78 ± 0.07

Table 2: Test classification accuracy (%) on CIFAR10 with ResNet18 over 16 runs. We test stage and cosine learning rate schedulers, and removal of shortcut connections. Higher is better.

	with shortcut, stage lr	with shortcut, cos lr	without shortcut, cos lr
SGD+M	95.07 ± 0.12	95.51 ± 0.11	94.97 ± 0.16
PSGD LRA	95.43 ± 0.12	95.54 ± 0.10	95.36 ± 0.09

4. Practical Considerations

All the above proposed preconditioners can be fitted online by minimizing criterion (3) with 'gradient descent' on the Lie groups. Different from the common gradient descent, moving an object on Lie group is achieved by multiplying it with $I + \mu G$, where G is the group generator, and μ is small enough such that $\|\mu G\| < 1$. Since a series of such small movements trace a curve on the Lie group manifold, we see that G indeed always is in the tangent space of the group as Lie algebra is closed. We do not detail the math derivations here. They are documented in our supplemental materials.

We would like to point out that damping is neither necessary nor generally feasible on any Lie group, although it is widely used in other second-order optimizers to avoid degenerated solutions. On one hand, by 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. We may not always be able to find another Q' on the same group such that $Q'^T Q' = Q^T Q + \lambda I$ with $\lambda > 0$. Actually, criterion (3) damps the gradient noise naturally [7]. This spares us the trouble on tuning a damping schedule.

Yet, gradient clipping could be helpful to encourage stability. The quadratic approximation leading to the quasi-linear system (2) can only be valid within a certain region around θ. 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 either adjust μ, or equivalently, clipping $\|P \partial \hat{f}(\theta)/\partial\theta\|$, to ensure that $\|\delta\theta\|$ is small enough.

Lastly, both the learning rate for parameter updating and step size for preconditioner fitting are naturally normalized to be in range $(0, 1)$. Step size 0.01 always is a good initial guess. It is not difficult to develop enough experiences for the user on setting them given an optimization problem.

5. Experimental Results

We have tested the proposed preconditioners on four tasks: 1) the MNIST handwriting digit recognition with LeNet5; 2) the CIFAR10 image classification with ResNet18; 3) a large scale logistic regression problem, and 4) the delayed XOR problems [4] with sequence length 64. Detailed experimental setups and results are put in the appendices. Tables 1 and 2 summarize the image classification results. Preconditioned SGD (PSGD) also outperforms LM-BFGS on the third problem. The last problem is well known to be difficult for either simple or gated recurrent neural networks (RNNs). It seems that only PSGD can reliably solve this type of pathological problem. These initial results are encouraging for second-order optimizations using preconditioners on the Lie groups.

Acknowledgments:

I would like to thank Yaroslav Bulatov for his encouragements of the work on preconditioned SGD, and Omead Pooladzandi for insightful discussions.

References

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

[2] 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.

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

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

[5] S. Ioffe and C. Szegedy. Batch normalization: accelerating deep network training by reducing internal covariate shift. In *ICML*, 2015.

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

[7] X. L. Li. Preconditioned stochastic gradient descent. *IEEE Trans. Neural Networks and Learning Systems*, 29(5):1454–1466, 2018.

[8] 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.

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

[10] 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.

[11] 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.

[12] 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.

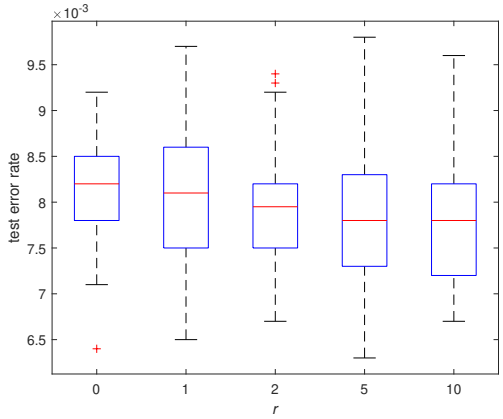

Figure 1: MNIST test classification error rates over 50 runs using preconditioners with different orders of LRA. The one with order 0 reduces to the diagonal preconditioner. Table 1 reports results of the one with $r = 5$. Lower is better.

Appendix A: More Experimental Results

A.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

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

where d is a vector. This form of Q cannot form a Lie group. Still, its two factors, $\text{diag}(d)$ and $I + UV^T$, can be fit on their 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 orders 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'.

A.2: CIFAR10 Image Classification with ResNet18

We follow the implementations of the Adabelief algorithm [12] to test preconditioned SGD (PSGD) on the CIFAR10 image classification task with the ResNet18 model. Training code of Adabelief is available at `https://github.com/juntang-zhuang/Adabelief-Optimizer`. One main difference from the implementations in [12] 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.

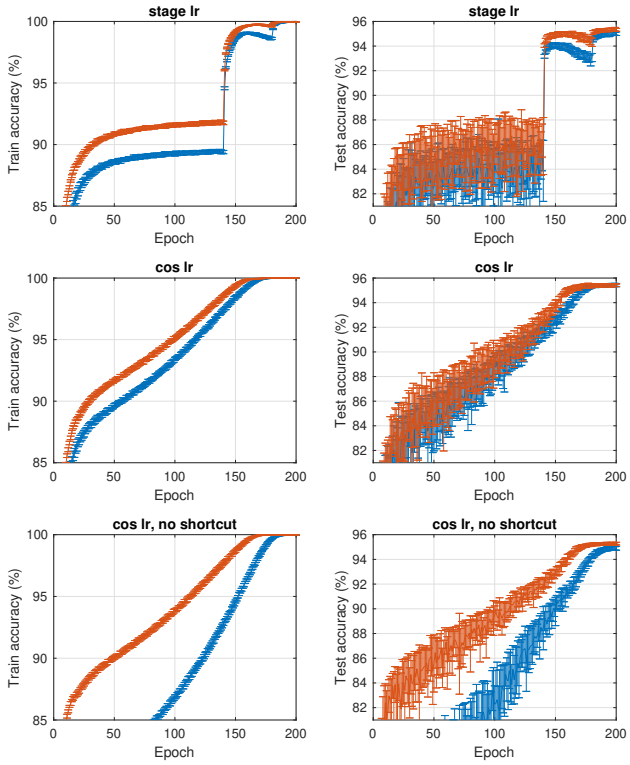

Figure 2: CIFAR10 image classification with ResNet18. The order of low rank Hessian approximation is 10. Mean and variance are estimated over 16 runs. Higher is better.

We also consider the cosine learning rate scheduler, which helps SGD to achieve the state-of-the-art (SOTA) test accuracy of 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 λ between 0.01 and 0.02 performs the best.

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. 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.

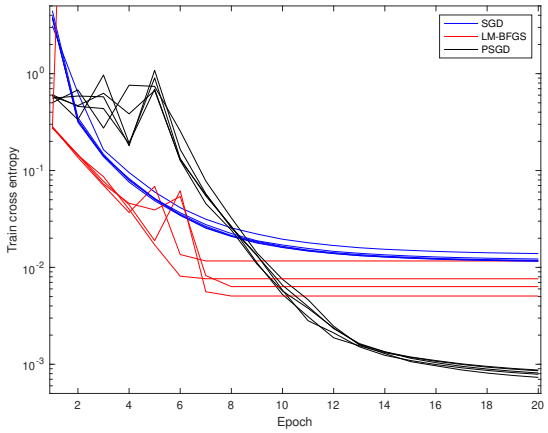

Figure 3: Typical convergence curves on the logistic regression problem. Lower is better. When comparing the convergence speed, one should be aware that one step of LM-BFGS may take up to ten iterations, while SGD and PSGD always one iteration per step.

A.3: A Large Scale Logistic Regression Problem

We use logistic regression to solve the MNIST image classification problem. Let x be the vector of 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.

We compare PSGD with the algorithm of choice for this type of problem, LM-BFGS. 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 we exponentially anneal down it. We have found that LM-BFGS diverges on roughly one third of the trials with initial learning rate 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 learning rate 0.1, we only show a few typical convergence curves of SGD, LM-BFGS and PSGD in Fig. 3. LM-BFGS converges to regression losses a few times smaller than that of SGD. PSGD could converges 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.

A.4: The Delayed XOR Problem

We consider the delayed XOR problem proposed in the LSTM paper [4]. The task is to predict the XOR relation of a and b scattered randomly and far away in a long sequence. It is a tough problem because it cannot be 'partially solved' since memorizing either a or b alone does not help to predict

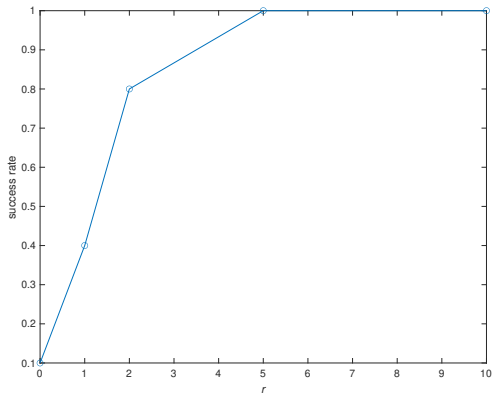

Figure 4: Success rate over ten runs in solving the XOR problem with a simple RNN and LRA preconditioners of orders $0, 1, 2, 5$ and 10. Higher is better.

XOR(a, b). This problem is equally challenging to both simple and gated recurrent neural networks (RNNs).

A simple RNN with 30 hidden states is adopted. The sequence length is 64. Tested orders of low rank approximation are $r = 0, 1, 2, 5$, and 10. Both step sizes for parameter and preconditioner updating are fixed at 0.01. The gradient clipping threshold is set to 1. No momentum is used. A trial fails if it does not converge within $100,000$ iterations. The success rates over ten runs for each r are plotted in Fig. 4. This example shows a typical problem where the diagonal preconditioner struggles, while a low-order Hessian approximation works perfectly for preconditioning.

Supplementary Materials:

Derivations of the two preconditioners are put at https://drive.google.com/file/d/1CTNx1q67_py87jn-0OI-vSLcsM1K7VsM/view?usp=sharing.