On the Performance of Preconditioned Stochastic Gradient Descent

Xi-Lin Li

Abstract—This paper studies the performance of preconditioned stochastic gradient descent (PSGD), which can be regarded as an enhance stochastic Newton method with the ability to handle gradient noise and non-convexity at the same time. We have improved the implementation of PSGD, unrevealed its relationship to equilibrated stochastic gradient descent (ESGD) and feature normalization, and provided a software package (https://github.com/lixilinx/psgd_tf) implemented in Tensorflow to compare PSGD with four different preconditioners and variations of stochastic gradient descent (SGD) on a wide range of benchmark problems with commonly used neural network models, e.g., convolutional and recurrent neural networks. Comparison results clearly demonstrate the advantages of PSGD in terms of convergence speeds and generalization performances.

Index Terms—Stochastic gradient descent, preconditioner, optimization, Newton method, neural network.

I. INTRODUCTION

Stochastic gradient descent (SGD) and its variations, e.g., SGD with either classic or Nesterov momentum, RMSProp, Adam, adaptive learning rates, etc., are popular in diverse stochastic optimization problems, e.g., machine learning and on-line signal processing [1]–[6]. Off-the-shelf methods from convex optimizations, e.g., the quasi-Newton method, conjugate gradient method and truncated Newton method, i.e., the Hessian-free optimization, are attracting more attentions [7]–[10], and find many successful applications in stochastic optimizations as well. At the same time, searchings for new optimization theories and learning rules are always active, and methods like natural gradient descent, relative gradient descent, equilibrated SGD (ESGD) and feature normalization [11]–[15], provide us with great insight into the properties of the parameter spaces, cost function surfaces and deep neural network training in stochastic optimizations.

This paper studies the performance of preconditioned SGD (PSGD) [16], a second-order method which explicitly considers the non-convexity and gradient noises in stochastic optimizations. We closely study the performance of preconditioners in four forms, i.e., dense preconditioner, diagonal preconditioner, Kronecker product preconditioner and a scaling-and-normalization preconditioner. ESGD and feature normalization [14], [15] are shown to be PSGD with specific forms of preconditioners. We also consider different ways to evaluate the Hessian-vector product, an important measurement that helps PSGD to adaptively extract the curvature information of cost surfaces. We provide a software package implemented in Tensorflow[1] for the comparisons among variations of SGD and PSGD with different preconditioners on a wide range of

San Jose, CA 95132 (E-mail: lixilinx@gmail.com).
[1]https://www.tensorflow.org/

benchmark problems, which include synthetic and real world data, and involve most commonly used neural network models, e.g., recurrent and convolutional networks. We argue that Kronecker product and scaling-and-normalization preconditioners are particularly suitable for training neural networks since affine transformations are their basic building blocks.

II. BACKGROUND

A. Notations

We consider the minimization of cost function

$$f(\boldsymbol{\theta}) = E_z[\ell(\boldsymbol{\theta}, \boldsymbol{z})] \tag{1}$$

where E_z takes expectation over random variable $\boldsymbol{z}$, ℓ is a loss function, and $\boldsymbol{\theta}$ is the model parameter vector to be optimized. For example, in a classification problem, ℓ could be the cross entropy loss, $\boldsymbol{z}$ is a pair of input feature vector and class label, vector $\boldsymbol{\theta}$ consists of all the trainable parameters in the considered classification model, and E_z takes average over all samples from the training data set. By assuming second-order differentiable model and loss, we could approximate $\ell(\boldsymbol{\theta}, \boldsymbol{z})$ as a quadratic function of $\boldsymbol{\theta}$ within a trust region around $\boldsymbol{\theta}$, i.e.,

$$\ell(\boldsymbol{\theta}, \boldsymbol{z}) = \boldsymbol{b}_z^T \boldsymbol{\theta} + 0.5\boldsymbol{\theta}^T \boldsymbol{H}_z \boldsymbol{\theta} + a_z \tag{2}$$

where a_z is the sum of approximation error and constant term independent of $\boldsymbol{\theta}$, $\boldsymbol{H}_z$ is a symmetric matrix, and subscript z in $\boldsymbol{b}_z$, $\boldsymbol{H}_z$ and a_z reminds us that these three terms depend on $\boldsymbol{z}$. Now, we may rewrite (1) as

$$f(\boldsymbol{\theta}) = \boldsymbol{b}^T \boldsymbol{\theta} + 0.5\boldsymbol{\theta}^T \boldsymbol{H} \boldsymbol{\theta} + a \tag{3}$$

where $\boldsymbol{b} = E_z[\boldsymbol{b}_z]$, $\boldsymbol{H} = E_z[\boldsymbol{H}_z]$, and $a = E_z[a_z]$. We do not impose any assumption, e.g., positive definiteness, on $\boldsymbol{H}$ except for being symmetric. Thus the quadratic surface in the trust region could be non-convex. To simplify our notations, we no longer consider the higher-order approximation error in a, and simply assume that $f(\boldsymbol{\theta})$ is a quadratic function of $\boldsymbol{\theta}$ in the considered trust region.

B. A Brief Review of PSGD

PSGD uses preconditioned stochastic gradient to update $\boldsymbol{\theta}$ as

$$\boldsymbol{\theta}^{[\text{new}]} = \boldsymbol{\theta}^{[\text{old}]} - \mu_\theta \boldsymbol{P} \frac{\partial \hat{f}(\boldsymbol{\theta})}{\partial \boldsymbol{\theta}} \tag{4}$$

where $0 < \mu_\theta < 1$ is a normalized step size, $\hat{f}(\boldsymbol{\theta})$ is an estimate of $f(\boldsymbol{\theta})$ obtained by replacing expectation with sample average, and $\boldsymbol{P}$ is a preconditioner adaptively updated

along with $\boldsymbol{\theta}$. Within the considered trust region, let us write the stochastic gradient, $\partial \hat{f}(\boldsymbol{\theta})/\partial \boldsymbol{\theta}$, explicitly as

$$\frac{\partial \hat{f}(\boldsymbol{\theta})}{\partial \boldsymbol{\theta}} = \hat{\boldsymbol{H}}\boldsymbol{\theta} + \hat{\boldsymbol{b}} \qquad (5)$$

where $\hat{\boldsymbol{H}}$ and $\hat{\boldsymbol{b}}$ are estimates of $\boldsymbol{H}$ and $\boldsymbol{b}$, respectively. Let $\delta\boldsymbol{\theta}$ be a random perturbation of $\boldsymbol{\theta}$, and be small enough such that $\boldsymbol{\theta} + \delta\boldsymbol{\theta}$ still resides in the same trust region. Then, (5) suggests the following resultant perturbation of stochastic gradient,

$$\delta\hat{\boldsymbol{g}} \stackrel{\text{def}}{=} \frac{\partial \hat{f}(\boldsymbol{\theta} + \delta\boldsymbol{\theta})}{\partial \boldsymbol{\theta}} - \frac{\partial \hat{f}(\boldsymbol{\theta})}{\partial \boldsymbol{\theta}} = \hat{\boldsymbol{H}}\delta\boldsymbol{\theta} = \boldsymbol{H}\delta\boldsymbol{\theta} + \boldsymbol{\varepsilon} \qquad (6)$$

where $\boldsymbol{\varepsilon}$ accounts for the error due to replacing $\hat{\boldsymbol{H}}$ with $\boldsymbol{H}$. Note that by definition, $\delta\hat{\boldsymbol{g}}$ is a random vector dependent on $\delta\boldsymbol{\theta}$. PSGD pursues the preconditioner $\boldsymbol{P}$ via minimizing criterion

$$c(\boldsymbol{P}) = E_{\delta\theta}[\delta\hat{\boldsymbol{g}}^T \boldsymbol{P}\delta\hat{\boldsymbol{g}} + \delta\boldsymbol{\theta}^T \boldsymbol{P}^{-1}\delta\boldsymbol{\theta}] \qquad (7)$$

where $E_{\delta\theta}$ takes expectation over $\delta\boldsymbol{\theta}$. Let us introduce Cholesky factorization $\boldsymbol{P} = \boldsymbol{Q}^T \boldsymbol{Q}$. With slight abuse of notation, we further rewrite $c(\boldsymbol{P})$ in (7) as a function of $\boldsymbol{Q}$. Then we can update $\boldsymbol{P}$ by learning rule

$$\boldsymbol{Q}^{[\text{new}]} = \boldsymbol{Q}^{[\text{old}]} - \mu_Q \nabla\mathcal{E}\boldsymbol{Q}^{[\text{old}]}, \qquad (8)$$

where $\mu_Q > 0$ is a step size,

$$\nabla\mathcal{E} = \left.\frac{\partial\hat{c}(\boldsymbol{Q} + \mathcal{E}\boldsymbol{Q})}{\partial\mathcal{E}}\right|_{\mathcal{E}=\boldsymbol{0}}$$

is called a stochastic relative gradient, and $\hat{c}$ is the preconditioner estimation criterion obtained by replacing the expectation in (7) with sample average. For our preconditioner estimation problem, relative gradient is equivalent to the natural gradient on the Lie group of triangular matrices with positive diagonal entries. More details on relative and natural gradients can be found in [11], [12].

Under mild conditions, criterion (7) determines a unique positive definite $\boldsymbol{P}$ [16]. The resultant preconditioner is perfect in the sense that it preconditions the stochastic gradient such that

$$\boldsymbol{P}E_{\delta\theta}[\delta\hat{\boldsymbol{g}}\delta\hat{\boldsymbol{g}}^T]\boldsymbol{P} = E_{\delta\theta}[\delta\boldsymbol{\theta}\delta\boldsymbol{\theta}^T] \qquad (9)$$

which is comparable to relationship

$$\boldsymbol{H}^{-1}\delta\boldsymbol{g}\delta\boldsymbol{g}^T\boldsymbol{H}^{-1} = \delta\boldsymbol{\theta}\delta\boldsymbol{\theta}^T \qquad (10)$$

where $\delta\boldsymbol{g}$ is the perturbation of noiseless gradient, and we assume that $\boldsymbol{H}$ is invertible such that $\delta\boldsymbol{g} = \boldsymbol{H}\delta\boldsymbol{\theta}$ can be rewritten as (10). Thus, PSGD can be viewed as an enhanced Newton method that can handle gradient noise and non-convexity at the same time.

Note that in the presence of gradient noise, the optimal $\boldsymbol{P}$ and $\boldsymbol{P}^{-1}$ given by (9) are not unbiased estimates of $\boldsymbol{H}^{-1}$ and $\boldsymbol{H}$, respectively. Actually, even if $\boldsymbol{H}$ is positive definite and available, $\boldsymbol{H}^{-1}$ may not always be a good preconditioner since it could significantly amplify the gradient noise along the directions of the eigenvectors of $\boldsymbol{H}$ associated with small eigenvalues, and leads to divergence. More specifically, [16] shows that

$$\boldsymbol{H}^{-1}E_{\delta\theta}[\delta\hat{\boldsymbol{g}}\delta\hat{\boldsymbol{g}}^T]\boldsymbol{H}^{-1} \geq E_{\delta\theta}[\delta\boldsymbol{\theta}\delta\boldsymbol{\theta}^T] \qquad (11)$$

where $\boldsymbol{A} \geq \boldsymbol{B}$ means that $\boldsymbol{A} - \boldsymbol{B}$ is nonnegative definite.

III. Implementations of PSGD

A. Hessian-Vector Product Calculation

1) Approximate Solution: The original PSGD method relies on (6) to calculate the Hessian-vector product, $\hat{\boldsymbol{H}}\delta\boldsymbol{\theta}$. This numerical differentiation method is simple, and only involves gradient calculations. However, it requires $\delta\boldsymbol{\theta}$ to be small enough. In practice, numerical accuracy might be an issue when handling small numbers with floating-point arithmetic. This concern becomes more grave with the emerging of half-precision math in neural network training.

2) Exact Solution: An alternative way to calculate this Hessian-vector product is via

$$\frac{\partial}{\partial\boldsymbol{\theta}}\left\{\left[\frac{\partial\hat{f}(\boldsymbol{\theta})}{\partial\boldsymbol{\theta}}\right]^T\delta\boldsymbol{\theta}\right\} = \frac{\partial^2\hat{f}(\boldsymbol{\theta})}{\partial\boldsymbol{\theta}\partial\boldsymbol{\theta}^T}\delta\boldsymbol{\theta} = \hat{\boldsymbol{H}}\delta\boldsymbol{\theta} \qquad (12)$$

Here, we no longer require $\delta\boldsymbol{\theta}$ to be small enough. The above trick is known for a long time [17]. However, hand-coded second-order derivative is error-prone even for moderately complicated models. Nowadays, this choice becomes attractive due to the wide availability of automatic differentiation softwares, e.g., Tensorflow, Pytorch[2], Theano[3], etc.. Note that higher-order derivatives may not be fully supported in certain softwares, e.g., the latest Tensorflow, version 1.6, does not support second-order derivative for its while loop.

B. Different Forms of Preconditioner

1) Dense Preconditioner: We call $\boldsymbol{P}$ a dense preconditioner if it does not have any sparse structure except for being symmetric. A dense preconditioner is practical only for small-scale problems with up to thousands of trainable parameters, since it requires $\mathcal{O}(L^2)$ parameters to represent it, where L is the length of $\boldsymbol{\theta}$. We are mainly interested in sparse, or limited-memory, preconditioners, whose representations only require $\mathcal{O}(L)$ or less parameters.

2) Diagonal Preconditioner: Diagonal preconditioner probably is one of the simplest. From (7), we are ready to find the optimal solution as

$$\boldsymbol{P} = \text{diag}\left(\sqrt{E_{\delta\theta}[\delta\boldsymbol{\theta} \odot \delta\boldsymbol{\theta}] \oslash E_{\delta\theta}[\delta\hat{\boldsymbol{g}} \odot \delta\hat{\boldsymbol{g}}]}\right) \qquad (13)$$

where $\odot$ and $\oslash$ denote element-wise multiplication and division, respectively. For $\delta\boldsymbol{\theta}$ drawn from standard multivariate normal distribution, $E_{\delta\theta}[\delta\boldsymbol{\theta} \odot \delta\boldsymbol{\theta}]$ reduces to a vector with unit entries, and (13) gives the equilibration preconditioner in the equilibrated SGD (ESGD) proposed in [13]. Thus, ESGD is PSGD with a diagonal preconditioner. The Jacobi preconditioner is not optimal by criterion (7), and indeed is observed to show inferior performances in [13].

3) Kronecker Product Preconditioner: Kronecker product preconditioners have been previously exploited in [16], [18], [19]. We find that they are particularly suitable for preconditioning gradients of tensor parameters. For example, for a tensor Θ with shape (I, J, K), we flatten Θ into a

---

[2]http://pytorch.org/. Second-order derivative is supported in version 0.2.0.
[3]https://github.com/Theano/

TABLE I
NUMBER OF PARAMETERS IN $\boldsymbol{P}$ FOR PRECONDITIONING GRADIENTS OF A
MATRIX WITH SHAPE (M, N)

Dense	$0.5(M^2N^2 + MN)$
Kronecker product	$0.5(M^2 + N^2 + M + N)$
Diagonal	MN
SCAN	$M + 2N - 1$

column vector with length IJK, and use Kronecker product preconditioner

$$\boldsymbol{P} = \boldsymbol{P}_3 \otimes \boldsymbol{P}_2 \otimes \boldsymbol{P}_1 \tag{14}$$

to have preconditioned gradient $\boldsymbol{P}\partial\hat{f}/\partial\mathrm{vec}(\Theta)$, where $\otimes$ denotes Kronecker product, and $\boldsymbol{P}_1$, $\boldsymbol{P}_2$, and $\boldsymbol{P}_3$ are three positive definite matrices with shapes (I, I), (J, J), and (K, K), respectively. Such a preconditioner only requires $\mathcal{O}(I^2 + J^2 + K^2)$ parameters for its representation, while a dense one requires $\mathcal{O}(I^2J^2K^2)$ parameters. By introducing Cholesky factorization $\boldsymbol{P}_i = \boldsymbol{Q}_i^T\boldsymbol{Q}_i$ and rewriting $\boldsymbol{P}$ as $\boldsymbol{P} = \boldsymbol{Q}^T\boldsymbol{Q}$, we can learn $\boldsymbol{P}$ with learning rules similar to that in (8), where $\boldsymbol{Q} = \boldsymbol{Q}_3 \otimes \boldsymbol{Q}_2 \otimes \boldsymbol{Q}_1$, and all such $\boldsymbol{Q}$'s still form a Lie group.

4) SCaling-And-Normalization (SCAN) Preconditioner: SCAN preconditioner $\boldsymbol{P} = (\boldsymbol{Q}_2^T\boldsymbol{Q}_2) \otimes (\boldsymbol{Q}_1^T\boldsymbol{Q}_1)$ is a special Kronecker product preconditioner designed for neural network training, where $\boldsymbol{Q}_1$ is a diagonal matrix, and only entries of the diagonal and last column of $\boldsymbol{Q}_2$ can have nonzero values. As explained in Section IV.B, PSGD with a SCAN preconditioner is equivalent to SGD with normalized input features and scaled output features. It is not difficult to verify that for such sparse $\boldsymbol{Q}_1$ and $\boldsymbol{Q}_2$ with positive diagonal entries, matrices with decomposition $\boldsymbol{Q}_2 \otimes \boldsymbol{Q}_1$ form a Lie group. Hence, natural gradient descent applies to SCAN preconditioner estimation.

It is not possible to enumerate all feasible forms of preconditioners. Table I summarizes the ones we have discussed, and their degrees of freedoms. More examples can be found in [16], and our Tensorflow software package. These simple preconditioners can be used as building blocks for forming larger preconditioners via direct sum and/or Kronecker product operations. Except for a few cases, we cannot find closed-form solution for the optimal $\boldsymbol{P}$ with a desired form when given enough pairs of $(\delta\boldsymbol{\theta}, \delta\hat{\boldsymbol{g}})$. Hence, it is important to design preconditioners with proper forms such that efficient learning, e.g., natural gradient descent, is possible.

IV. APPLICATIONS TO NEURAL NETWORKS

A. Affine Transformations in Neural Networks

Element-wise nonlinearity and affine transformation,

$$\boldsymbol{y} = \Theta\boldsymbol{x} \tag{15}$$

are the two main building blocks of neural networks, where Θ is a matrix parameter, and both $\boldsymbol{x}$ and $\boldsymbol{y}$ are feature vectors optionally augmented with 1. Since most neural networks use parameterless nonlinearities, all the trainable parameters are just a list of affine transformation matrices. By assigning a Kronecker product preconditioner to each affine transformation matrix, we are using the direct sum of a list Kronecker

product preconditioners as the preconditioner for the whole model parameter vector. Our experiences suggest that this approach provides a good trade off between computational complexities and performance gains.

It is not difficult to spot out the affine transformations in most commonly used neural networks, e.g., feed-forward neural network, vanilla recurrent neural network (RNN), gated recurrent unit (GRU) [21], long short-term memory (LSTM) [22], convolutional neural network (CNN) [2], etc.. For example, in a two dimensional CNN, the input features may form a three dimensional tensor with shape (H, W, I), and the filter coefficients could form a four dimensional tensor with shape (H, W, I, O), where H is the height of image patch, W is the width of image patch, I is the number of input channels, and O is the number of output channels. To rewrite the convolution as an affine transformation, we just need to reshape the filter tensor into a matrix with size (O, HWI), and flatten the input image patch into a column vector with length HWI.

B. On the Role of Kronecker Product Preconditioner

By using Kronecker product preconditioner $\boldsymbol{P} = \boldsymbol{P}_2 \otimes \boldsymbol{P}_1$, the learning rule for Θ can be written as

$$\Theta \leftarrow \Theta - \mu_\theta \boldsymbol{P}_1 \frac{\partial\hat{f}}{\partial\Theta}\boldsymbol{P}_2 \tag{16}$$

where $\boldsymbol{P}_1$ and $\boldsymbol{P}_2$ are two positive definite matrices. With Cholesky factorizations $\boldsymbol{P}_1 = \boldsymbol{Q}_1^T\boldsymbol{Q}_1$ and $\boldsymbol{P}_2 = \boldsymbol{Q}_2^T\boldsymbol{Q}_2$, we can rewrite (16) as

$$\boldsymbol{Q}_1^{-T}\Theta\boldsymbol{Q}_2^{-1} \leftarrow \boldsymbol{Q}_1^{-T}\Theta\boldsymbol{Q}_2^{-1} - \mu_\theta\boldsymbol{Q}_1\frac{\partial\hat{f}}{\partial\Theta}\boldsymbol{Q}_2^T \tag{17}$$

Let us introduce matrix $\Theta' = \boldsymbol{Q}_1^{-T}\Theta\boldsymbol{Q}_2^{-1}$, and noticing that

$$\frac{\partial\hat{f}}{\partial\Theta'} = \boldsymbol{Q}_1\frac{\partial\hat{f}}{\partial\Theta}\boldsymbol{Q}_2^T \tag{18}$$

we can rewrite (17) simply as

$$\Theta' \leftarrow \Theta' - \mu_\theta\frac{\partial\hat{f}}{\partial\Theta'} \tag{19}$$

Correspondingly, the affine transformation in (15) is rewritten as $\boldsymbol{y}' = \Theta'\boldsymbol{x}'$, where $\boldsymbol{y}' = \boldsymbol{Q}_1^{-T}\boldsymbol{y}$ and $\boldsymbol{x}' = \boldsymbol{Q}_2\boldsymbol{x}$. Hence, the PSGD in (16) is equivalent to the SGD in (19) with transformed feature vectors $\boldsymbol{x}'$ and $\boldsymbol{y}'$.

We know that feature whitening and normalization could accelerate convergence, and batch normalization and self-normalizing neural networks are such examples [14], [15]. Actually, feature normalization can be viewed as PSGD with a specific SCAN preconditioner. This fact is best explained by considering the following example with two input features,

$$\begin{bmatrix} x_1' \\ x_2' \\ 1 \end{bmatrix} = \begin{bmatrix} 1/\sigma_1 & 0 & -\nu_1/\sigma_1 \\ 0 & 1/\sigma_2 & -\nu_2/\sigma_2 \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \\ 1 \end{bmatrix} \tag{20}$$

where ν_i and σ_i are the mean and standard deviation of x_i, respectively. However, we should be aware that explicit input feature normalization is only empirically shown to accelerate convergence, and has little meaning in certain scenarios, e.g., RNN learning where features may not have any stationary

distribution. Furthermore, feature normalization cannot normalize the input and output features simultaneously. PSGD provides a more general and principled approach to find the optimal preconditioner, and applies to a much broader range of applications. A SCAN preconditioner does not necessarily "normalize" the input features in the sense of mean removal and variance normalization.

V. EXPERIMENTAL RESULTS

A. Tensorflow Implementation

Tensorflow is one the most popular machine learning frameworks with automatic differentiation support. We have defined a bunch of benchmark problems with both synthetic and real world data, and implemented SGD, RMSProp, ESGD, and six versions of PSGD. One trick worthy to point out is that in our implementations, we use the preconditioner from last iteration to precondition the current gradient. In this way, preconditioning gradient and updating preconditioner can be processed in parallel. The original method in [16] updates preconditioner and model parameters sequentially. It may marginally speed up the convergence, but one iteration could take longer wall time.

To make our comparison results easy to analyze, we try to keep settings simple and straightforward, and do not consider commonly used neural network training tricks like momentum, time-varying step size, drop out, batch normalization, etc.. Moreover, tricks like drop out and batch normalization cannot be directly applied to RNN. Preconditioners of PSGD always are initialized with identity matrix, and updated with a constant normalized step size, 0.01, and mini-batch size 1. We independently sample each entry of $\delta\boldsymbol{\theta}$ from normal distribution with mean 0 and variance eps when (6) is used to approximate the Hessian-vector product, and mean 0 variance 1 when (12) is used, where $\text{eps} = 2^{-23}$ is the single precision machine epsilon. Mini-batch size is fixed to 128. We only report the results of PSGD with exact Hessian-vector product here since the versions with approximated one always give similar results. The training loss is smoothed to keep our plots legible. Any further details, e.g., step sizes, neural network initial guesses, training and testing sample sizes, training loss smoothing factor, etc., and further results not reported here can be found in our package at https://github.com/lixilinx/psgd_tf.

B. Selected Experiments

1) Experiment 1: We consider the addition problem first proposed in [22]. A vanilla RNN is trained to predict the mean of two marked real numbers randomly located in a long sequence. Further details can be found in [22] and our implementations. Fig. 1 summarizes the results. Only PSGD with dense, Kronecker product and SCAN preconditioners shows reasonable performance. Notably, SCAN preconditioner is the sparest one, still, it clearly outperforms the diagonal preconditioner.

2) Experiment 2: Experiment 1 shows that SGD like methods have great difficulties in optimizing certain models. Hence, many thoughtfully designed neural network architectures are proposed to facilitate learning, e.g., LSTM [22], GRU [21],
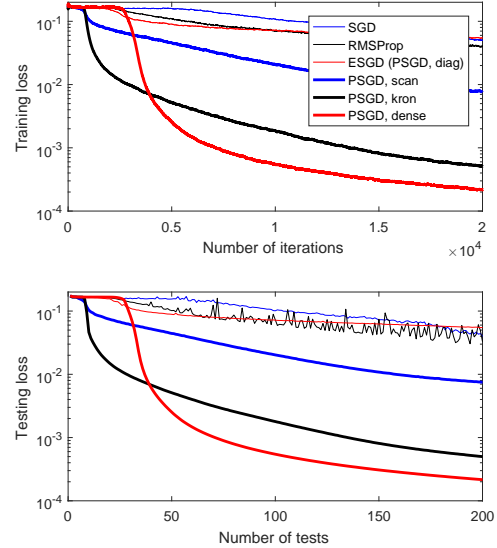

Fig. 1. Convergence curves on the addition problem with a standard RNN. Both the training and testing losses are mean square errors (MSE).

residual network [20], etc.. As revealed by its name, LSTM provides the designs for learning tasks requiring long-term memories. Still, we find that with SGD like methods, LSTM completely fails to solve the delayed-XOR problem in [22]. Fig. 2 shows the convergence curves of six tested methods. Again, only PSGD with dense, Kronecker product and SCAN preconditioners can successfully solve this problem. Bumpy convergence curves suggest that Hessians at local minima is ill-conditioned. We would like to point out that a vanilla RNN successes to solve this problem when trained with PSGD, and fails when trained with variations of SGD. More details are given in our package. Hence, selecting the right training method is at least as important as choosing a proper model.

3) Experiment 3: This experiment considers the well known MNIST[4] handwritten digits recognition task using CNN. We do not augment the training data with affine or elastic distorted images. However, we do randomly shift the original training image by ± 2 pixels both horizontally and vertically, and nest the shifted one into a larger, $(32, 32)$, image. A CNN consisting of four convolutional, two max pooling and one fully connected layers is adopted. Leaky rectified linear unit (ReLU), $\max(x, 0.3x)$, is the nonlinearity. At a first glance, this model could impose obstacles to PSGD since neither leaky ReLU nor max pooling is second-order differentiable everywhere. Nevertheless, $E_{\delta\theta}[\delta\hat{\boldsymbol{g}}\delta\boldsymbol{g}^T]$ is unlikely to be singular due to the existence of gradient noise, and thus (9) still well defines an optimal preconditioner.

Fig. 3 shows the convergence curves. PSGD with dense preconditioner is not considered due to its excessively high complexity. We find that SGD, RMSProp, ESGD and PSGD with SCAN preconditioner have comparable training losses and testing classification error rates. PSGD with Kronecker product preconditioner performs the best. Its average testing
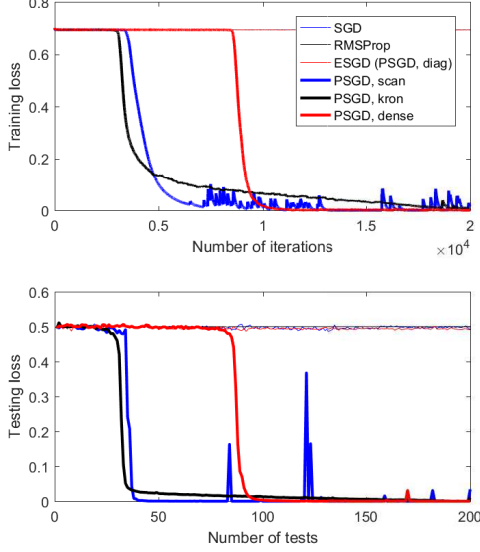
[4]http://yann.lecun.com/exdb/mnist/

Fig. 2. Convergence curves on the delayed-XOR problem with LSTM. Training and testing losses are the logistic loss and classification error rate, respectively.

classification error rate is about 0.5%, and the lowest ones are about 0.4%. Such a performance is impressive as we do not use complicated data augmentations.
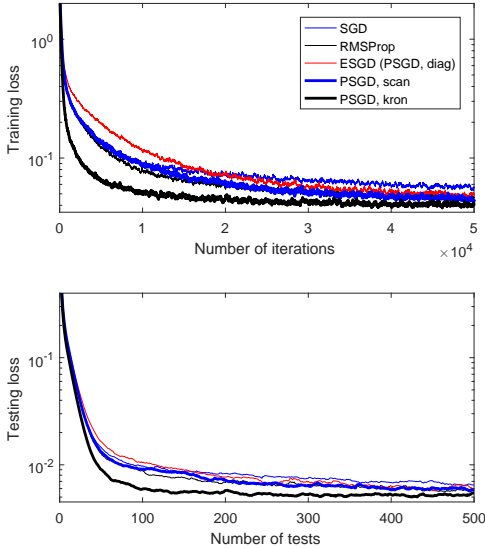

Fig. 3. Convergence curves on the MNIST handwritten digits recognition task using CNN. Training loss is the sum of cross entropy and a weighted CNN coefficients energy, and testing loss is the classification error rate. The testing loss is smoothed to make results from different methods more distinguishable.

4) Experiment 4: Here, we consider an image autoencoder consisting of three convolution layers for encoding and three deconvolution layers for decoding. Training and testing images are from the CIFAR-10 database[5]. Fig. 4 summarizes the re-

[5]https://www.cs.toronto.edu/~kriz/cifar.html

sults. SGD performs the worst. ESGD and PSGD with SCAN preconditioner give almost identical convergence curves, and outperform RMPProp. PSGD with Kronecker product preconditioner clearly outperforms all other methods.
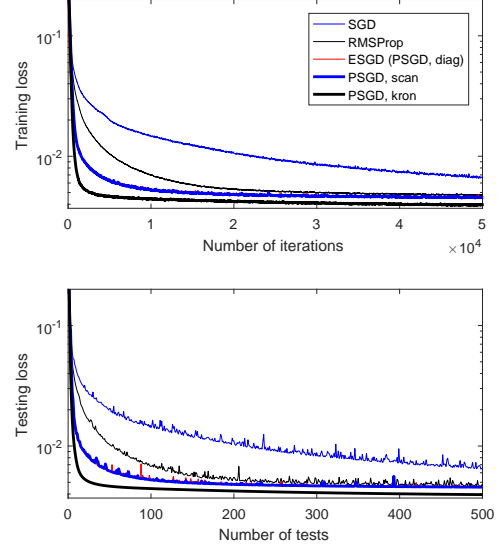

Fig. 4. Convergence curves for the image autoencoder trained on CIFAR-10 database. Both the training and testing losses are MSEs.

C. Complexities of PSGD

1) Computational Complexities: We consider computational complexity per iteration. Compared with SGD, PSGD comes with three major extra complexities:

- C1: evaluation of the Hessian-vector product;
- C2: preconditioner updating;
- C3: preconditioned gradient calculation.

C1 typically has the same complexity as SGD [17]. Depending on the neural network architectures, complexity of C1 and SGD varies a lot. For the simplest feed-forward neural network, SGD has complexity $\mathcal{O}(BMN)$, where (M, N) is the shape of the largest matrix in the model, and B is the mini-batch size. For a vanilla RNN, the complexity rises to $\mathcal{O}(DBMN)$, where D is the back propagation depth. More complicated models may have higher complexities. Complexities of C2 and C3 depend on the form of preconditioner. For a Kronecker product preconditioner, C2 has complexity $\mathcal{O}(\max(M^3, N^3))$, and C3 has complexity $\mathcal{O}(\max(M, N)MN)$. One simple way to reduce the complexities of C2 and C3 is to split those big matrices into smaller ones, and let each smaller matrix keep its own Kronecker product preconditioner. Another way is to update the preconditioner less frequently, i.e., skipping steps C1 and C2 regularly, since the curvatures are likely to evolve slower than the gradients.

2) Wall Time Comparisons: On our machines and with the above benchmark problems, the wall time of PSGD per iteration typically just doubles that of SGD. This is not astonishing since many parts of PSGD may be processed in parallel.

For example, updating preconditioner and preconditioning gradient can be executed in parallel as the preconditioner from last iteration is used to precondition the current gradient. Preconditioners for all the affine transformation matrices in the model can be updated in parallel as well once $(\delta\boldsymbol{\theta}, \delta\hat{\boldsymbol{g}})$ is prepared.

Here, we list the median wall time per iteration of each method in Experiment 5 running on an GeForce GTX 1080 Ti graphics card: 0.007 s for SGD; 0.008 s for RMSProp; 0.014 s for ESGD; 0.015 s for PSGD with SCAN preconditioner; and 0.017 s for PSGD with Kronecker product preconditioner.

D. Working with Large mini-Batch Sizes

Using large mini-batch sizes and step sizes could save training time, but also might bring new issues such as poor convergence and over fitting [23]. The gradients become more deterministic with the increase of mini-batch sizes. Without preconditioning and with badly conditioned Hessians, the model parameters will be adapted only along a few directions of the eigenvectors of Hessian associated with large eigenvalues. For many methods, this behavior leads to slow and poor convergences. PSGD seems suffer less from such concerns since the preconditioner is optimized in a way to make $P\hat{H}$ has unitary absolute eigenvalues. Hence, the model parameters are updated in a balanced manner in all directions. We have tried mini-batch size 1024 on our benchmark problems, and observe no meaningful performance loss.

E. Applications to General Optimization Problems

The boundary between stochastic and general optimization problems blurs with the increase of mini-batch sizes. Thus, a well designed stochastic optimization method should also work properly on the general mathematical optimization. Preliminary results show that PSGD does perform well on many mathematical optimization benchmark problems. A Rosenbrock function[6] minimization demo is included in our package, and PSGD finds the global minimum with about 200 iterations. Methods like SGD, RMSProp, Adam, batch normalization, etc., are apparently not prepared for these problems. Further discussion in this direction strays away from our focus.

VI. CONCLUSIONS

We have studied the performances of preconditioned stochastic gradient descent (PSGD) with approximate and exact Hessian-vector products, and with dense, diagonal, Kronecker product, and SCaling-And-Normalization (SCAN) preconditioners. The approximate Hessian-vector product by numerical differentiation is a valid alternative to the more costly exact solution, especially when automatic second-order differentiation is unavailable or the exact solution is expensive to obtain. We have shown that equilibrated SGD and feature normalization are PSGD with specific forms of preconditioners. We have compared PSGD with variations of SGD in several performance indices, e.g., training loss, testing loss, and wall time per iteration, on benchmark problems with different

levels of difficulties. These variations of SGD fail completely on tough benchmark problems with synthetic data, and show inferior performance on problems with real world data. PSGD with Kronecker product and SCAN preconditioners provides a good trade off between computational complexity and performance gain. A PSGD software package implemented in Tensorflow is available at https://github.com/lixilinx/psgd_tf.

REFERENCES

[1] B. Widrow and S. D. Stearns, *Adaptive Signal Processing*. Englewood Cliffs, New Jersey: Prentice-Hall, Inc., 1985.
[2] Y. LeCun, L. Bottou, Y. Bengio, and P. Haffner, "Gradient based learning applied to document recognition," *Proc. IEEE*, vol. 86, no. 11, pp. 2278–2324, Nov. 1998.
[3] I. Sutskever, J. Martens, G. Dahl, and G. E. Hinton, "On the importance of momentum and initialization in deep learning," In *30th Int. Conf. Machine Learning*, Atlanta, 2013, pp. 1139–1147.
[4] G. Hinton, *Neural Networks for Machine Learning*. Retrieved from http://www.cs.toronto.edu/~tijmen/csc321/slides/lecture_slides_lec6.pdf.
[5] D. P. Kingma and J. Ba, "Adam: a method for stochastic optimization," in *3rd Int. Conf. Learning Representations*, San Diego, 2015.
[6] T. Schaul, S. Zhang, and Y. LeCun, "No more pesky learning rates," arXiv:1206.1106, 2013.
[7] J. Martens and I. Sutskever, "Training deep and recurrent neural networks with Hessian-free optimization," In *Neural Networks: Tricks of the Trade*, 2nd ed., vol. 7700, G. Montavon, G. B. Orr, and K.-R. Müller, Ed. Berlin Heidelberg: Springer, 2012, pp. 479–535.
[8] N. N. Schraudolph, J. Yu, and S. Günter, "A stochastic quasi-Newton method for online convex optimization," *J. Mach. Learn. Res.*, vol. 2, pp. 436–443, Jan. 2007.
[9] R. H. Byrd, S. L. Hansen, J. Nocedal, and Y. Singer, "A stochastic quasi-Newton method for large-scale optimization," *SIAM J. Optimiz.*, vol. 26, no. 2, pp. 1008–1031, Jan. 2014.
[10] B. Antoine, B. Leon, and G. Patrick, "SGD-QN: careful quasi-Newton stochastic gradient descent," *J. Mach. Learn. Res.*, vol. 10, pp. 1737–1754, Jul. 2009.
[11] J.-F. Cardoso and B. Laheld, "Equivariant adaptive source separation," *IEEE Trans. Signal Process.*, vol. 44, no. 12, pp. 3017–3030, Dec. 1996.
[12] S. Amari, "Natural gradient works efficiently in learning," *Neural Computation*, vol. 10, no. 2, pp. 251–276, Feb. 1998.
[13] Y. N. Dauphin, H. Vries, and Y. Bengio, "Equilibrated adaptive learning rates for non-convex optimization," in *Advances in Neural Information Processing Systems*, 2015, pp. 1504–1512.
[14] S. Ioffe and C. Szegedy, "Batch normalization: accelerating deep network training by reducing internal covariate shift," https://arxiv.org/abs/1502.03167, 2015.
[15] G. Klambauer, T. Unterthiner, A. Mayr, and S. Hochreiter, "Self-normalizing neural networks," in *Advances in Neural Information Processing Systems*, 2017.
[16] X.-L. Li, "Preconditioned stochastic gradient descent," *IEEE Trans. Neural Networks and Learning Systems*, 2017.
[17] B. A. Pearlmutter, "Fast exact multiplication by the Hessian," *Neural Computation*, vol. 6, pp. 147–160, 1994.
[18] J. Martens and R. B. Grosse, "Optimizing neural networks with Kronecker-factored approximate curvature," in *Proc. 32nd Int. Conf. Machine Learning*, 2015, pp. 2408–2417.
[19] D. Povey, X. Zhang, and S. Khudanpur, "Parallel training of DNNs with natural gradient and parameter averaging," in *Proc. Int. Conf. Learning Representations*, 2015.
[20] K. He, X. Zhang, S. Ren, and J. Sun, "Deep residual learning for image recognition," https://arxiv.org/abs/1512.03385, 2015.
[21] K. Cho, B. Merrienboer, C. Gulcehre, D. Bahdanau, F. Bougares, H. Schwenk, and Y. Bengio, "On the properties of neural machine translation: encoder-decoder approaches," in *Eighth Workshop on Syntax, Semantics and Structure in Statistical Translation*, 2014.
[22] S. Hochreiter and J. Schmidhuber, "Long short-term memory," *Neural Computation*, vol. 9, no.8, pp. 1735–1780, 1997.
[23] N. S. Keskar, D. Mudigere, J. Nocedal, M. Smelyanskiy, and P. Tang, "On large-batch training for deep learning: generalization gap and sharp minima," in *5th Int. Conf. Learning Representations*, Toulon, France, 2017.

[6]https://en.wikipedia.org/wiki/Rosenbrock_function