Recurrent neural network training with preconditioned stochastic gradient descent

Xi-Lin Li, lixilinx@gmail.com

Abstract

Recurrent neural networks (RNN), especially the ones requiring extremely long term memories, are difficult to training. Hence, they provide an ideal testbed for benchmarking the performance of optimization algorithms. This paper reports test results of a recently proposed preconditioned stochastic gradient descent (PSGD) algorithm on RNN training. We find that PSGD may outperform Hessian-free optimization which achieves the state-of-the-art performance on the target problems, although it is only slightly more complicated than stochastic gradient descent (SGD) and is user friendly, virtually a tuning free algorithm.

Index Terms

Preconditioned stochastic gradient descent, recurrent neural network, optimization.

I. INTRODUCTION

In its simplest form, a standard recurrent neural network (RNN) can be defined as

\[ x(t) = \phi \left( W_1 \begin{bmatrix} u(t) \\ x(t-1) \end{bmatrix} \right), \quad y(t) = W_2 \begin{bmatrix} x(t) \\ 1 \end{bmatrix}, \]

(1)

where \( t \) is a discrete time index, \( \phi \) an element-wise sigmoid function, \( u \) the input sequence, \( x \) the hidden state sequence, \( y \) the output sequence, and \( W_1 \) and \( W_2 \) are two weight matrices with proper dimensions. RNN is a powerful tool for sequence modeling, and its gradient can be conveniently evaluated, e.g., via backpropagation through time (BPTT) \([1]\). Actually, RNN training turns out to be extremely difficult when solving problems requiring long term memories \([2]\)–\([4]\). Exploding and vanishing gradients, especially the latter, are suspected to be the causes. Hence long and short term memory (LSTM) and its variants \([4], [5]\) are invented to overcome the
vanishing gradient issue mainly by the use of forgetting gates. However, as a specially modified model, LSTM may fail to solve certain problems that it is designed for, e.g., finding XOR relationship between two binary symbols with a long lag. Furthermore, as a more complicated model, it does not necessarily outperform a simple RNN model on several natural problems as reported in [2], [3]. Another way to address vanishing gradient is to directly penalize RNN connections encouraging vanishing gradients [2]. However, as an ad hoc method, its impact on the convergence and performance of RNN training is unclear. One important discovery made in [3] is that RNN requiring long term memories can be trained using Hessian free optimization, a conjugate gradient (CG) method tailored for neural network training, notably, a backpropagation like algorithm for curvature matrix-vector product evaluation [6]. However, due to its use of line search, Hessian free optimization requires a large mini-batch size for gradient and cost function evaluations, making it computationally demanding for problems with large training sample sizes.

Recently, a preconditioned stochastic gradient descent (PSGD) algorithm is proposed in [7]. It is a simple and general procedure to upgrade a stochastic gradient descent (SGD) algorithm to a second-order algorithm by exploiting the information extracted exclusively from noisy stochastic gradients. It is virtually tuning free, and applicable equally well to both convex and non-convex problems, a striking difference from many algorithms, including the Hessian free optimization, which assume a positive definite Hessian at least for their derivations. Naturally, we are curious about its performance on RNN training, especially on those pathological synthetic problems since they are effectively impossible for standard SGD [3], [4]. Our results suggest that although the issue of exploding and vanishing gradients arises naturally in RNN, efficient training is still possible when the gradients are properly preconditioned.

II. PSGD AND RNN TRAINING

A. **PSGD**

We briefly summarize the PSGD theory in [7]. Let us consider the minimization of cost function,

\[ f(\theta) = E[\ell(\theta, z)], \]

where \( \theta \) is a parameter vector to be optimized, \( z \) is a random vector, \( \ell \) is a loss function, and \( E \) takes expectation over \( z \). At the \( k \)th iteration of PSGD, we evaluate two stochastic
gradients over the same randomly drawn samples: the original gradient $g_k$ at point $\theta = \theta_k$, and a perturbed gradient $\tilde{g}_k$ at point $\theta = \theta_k + \delta \theta_k$, where $\delta \theta_k$ is a tiny random vector. By introducing gradient perturbation as $\delta g_k = \tilde{g}_k - g_k$, a positive definite preconditioner, $P_k$, can be pursued by minimizing criterion

$$E \left[ \delta g_k^T P_k \delta g_k + \delta \theta_k^T P_k^{-1} \delta \theta_k \right],$$  

(3)

where $E$ takes expectation over random variable $\delta \theta_k$. Under mild conditions, such a $P_k$ exists and is unique \([7]\). The rationality is that minimizing criterion (3) leads to a preconditioner scaling the gradient such that $P_k \delta g_k$ approximately matches $\delta \theta_k$. In the context of RNN training, $P_k$ damps exploding gradients and amplifies vanishing gradients. In general, $P_k$ preconditions the Hessian at $\theta_k$ such that the preconditioned locally linear iteration system has approximately unitary absolute eigenvalues \([7]\). As a result, PSGD learning rule,

$$\theta_{k+1} = \theta_k - \mu P_k g_k,$$

(4)

enjoys a uniform convergence rate in all the directions of a locally linear transformed parameter space, where $0 < \mu < 1$ is a normalized step size. The preconditioner can be conveniently estimated using stochastic relative (natural) gradient descent with mini-batch size 1 \([7]\).

**B. Application to RNN Training**

1) **Dense preconditioner:** It is straightforward to apply PSGD to RNN training by stacking all the elements in $W_1$ and $W_2$ to form a single coefficient vector $\theta$. The resultant preconditioner has no sparsity. Hence, such a brutal force solution is practical only for small scaled problems with up to thousands of parameters to learn.

2) **Preconditioner with sparse structures:** For large scaled problems, it is necessary to enforce certain sparse structures on the preconditioner so that it can be stored and manipulated on computers. Supposing the dimensions of $u, x$ and $y$ are $N_u, N_x$ and $N_y$ respectively, one example is to enforce $P$ to have form

$$P = (P_2 \otimes P_1) \oplus (P_4 \otimes P_3),$$  

(5)

where the dimensions of positive definite matrices $P_1, P_2, P_3$, and $P_4$ are $N_x, N_x + N_x + 1, N_y$ and $N_x + 1$ respectively, and $\otimes$ and $\oplus$ denote Kronecker product and direct sum respectively. Algorithms for learning these $P_i, 1 \leq i \leq 4$, are detailed in \([7]\) as well. We mainly study the performance of PSGD with sparse preconditioner due to its better scalability.
III. EXPERIMENTS

We consider the pathological synthetic problems originally proposed in [4] and restudied in [2], [3]. Details of these problems can be found in [4] and the supplement of [3]. For continuous problems (outputs are continuous), mean squared error (MSE) loss is used, and for discrete problems (outputs are discrete), cross entropy loss or hinge loss is used. The same parameter settings as in [7] are used for PSGD, and no problem-specific hand tweaking is made. Specifically, the preconditioner is initialized to identity matrix, and then updated using stochastic relative gradient descent with mini-batch size 1, step size 0.01 and sampling $\delta \theta$ from Gaussian distribution $\mathcal{N}(0, \text{eps})$ element-wisely, where $\text{eps} = 2^{-52}$ is the accuracy in double precision. The recurrent matrix of RNN is initialized to a random orthogonal matrix so that neither exploding nor vanishing gradient issue is severe at the beginning, loosely comparable to setting large initial biases in the forgetting gates of LSTM [5]. Other non-recurrent weights are element-wisely initialized to random numbers drawn from distribution $\mathcal{N}(0, 0.01)$. Mini-batch size 100 and step size 0.01 are used for RNN training. Program code written in Matlab and supplemental materials revealing more detailed experimental results can be found at https://sites.google.com/site/lixilinx/home/psgd.

A. Experiment 1: PSGD vs. SGD

We consider the addition problem where a RNN is trained to predict the sum of a pair of marked, but randomly located, continuous random numbers in a sequence. For SGD, clipped stochastic gradient with clipping threshold 1 is used to address the exploding gradient issue. SGD seldom succeeds on this problem when the sequence length is no less than 100. To make it easier, sequences with length uniformly distributed in range $[50, 100]$ are used for training, hoping that SGD can learn the desired patterns from shorter sequences and then generalize them to longer ones. Starting from an identical initial guess, Fig. 1 shows three learning curves for three algorithms: SGD, PSGD with a sparse preconditioner, and PSGD with a dense preconditioner. Clearly, PSGD with a dense preconditioner converges the fastest. Its gradient norm soars to over 30 around the 2000th iteration, and then quickly backs to smaller values, so does the norm of parameter change, $\|\delta \theta\|$. PSGD with a sparse preconditioner demonstrates similar behaviors, nevertheless converges much slower. SGD performs the worst.
Fig. 1. Convergence curves of SGD and PSGD. The hidden layer has 50 neurons. A preconditioner, even a very coarse one, significantly accelerates convergence.

B. Experiment 2: Performance on Pathological Synthetic Problems

We consider the four groups of pathological synthetic problems in [4]. The first group includes the addition, multiplication, and XOR problems; the second group includes the 2-bit and 3-bit temporal order problems; the third group only has the random permutation problem; and the fourth group are the 5-bit and 20-bit noiseless memorization problems. Totally we have eight problems. Detailed experimental results are recorded in supplement. In the addition and multiplication problems, RNN needs to memorize continuous random numbers with certain precision for many steps. In the 2-bit and 3-bit temporal order problems, RNN needs to memorize both widely separated binary bits and their order. The XOR problem challenges both RNN and LSTM training since the problem cannot be decomposed into smaller ones. In the random
permutation problem, RNN is taught to predict random unpredictable symbols, except the one at the end of sequence, leading to extremely noisy gradients. On the contrary, all symbols in the 5-bit and 20-bit memorization problems, except those information carrying bits, can be trivially predicted, but are not task related, thus diluting the importance of task related gradient components.

We follow the experimental configurations in [3], [4] so that the results can be compared. The results reported in [2] could be biased because according to the descriptions in [2], for most problems, RNN is trained on sequences with length uniformly distributed in range [50, 200]. This considerably facilitates the training since RNN has chances to learn the desired patterns from short sequences and then to generalize them to long ones, as shown in Experiment 1. We follow the configurations in [3], [4] to ensure that there is no short time lag training exemplar to facilitate learning.

Except that the noiseless memorization problems use hinge loss, all the other discrete problems use cross entropy loss as in [3]. The two losses should be comparable. The only reason to choose the hinge loss for the noiseless memorization problems is to reduce computational time. We use BPTT for gradient calculation. Thus for the hinge loss, once a symbol is correctly predicted with margin larger than a threshold, there is no need to do BPTT to calculate the gradient for that symbol. For the cross entropy loss, we need to do as many BPTT steps as the number of symbols in the sequence. Thus using hinge loss could save a lot of BPTT steps. The original hinge loss is highly non-differentiable. We propose a soft hinge loss defined as

$$
\ell = \left( \sum_{i=1, i \neq j}^{N_y} \left[ \max(y_i + 1 - y_j, 0) \right]^p + \epsilon \right)^{\frac{1}{p}},
$$

where $j$ is the desired class label, $p > 0$, and $\epsilon \geq 0$. It reduces to the standard hinge loss, $\max_{i \neq j} (y_i + 1 - y_j, 0)$, when $\epsilon = 0$ and $p = \inf$. In our implementation, we choose $\epsilon = 1$ and have tried $p = 10$ and $p = 5$. Note that unlike the cross entropy loss, the soft hinge loss might yield ill-conditioned Hessian and cause numerical difficulty.

Among these eight problems, the 5-bit memorization problem is special in the way that it only has 32 distinct input sequences. Hence we set its mini-batch size to 32. Then the gradient is exact, no longer stochastic. PSGD applies to deterministic optimization as well, but extra cares
need to be taken to prevent the arising of an ill-conditioned Hessian since PSGD is essentially a second-order optimization algorithm. Note that both the hinge loss and the cross entropy loss are only sensitive to the differences among $y_i, 1 \leq i \leq N_y$. Thus $W_2$ only needs to have $(N_y - 1)(N_x + 1)$ degrees of freedom, and its extra $N_x + 1$ degrees of freedom cause singular Hessians in the whole parameter space. We remove the extra $N_x + 1$ degrees of freedom in $W_2$ by constraining it to have zero sum in all columns.

We would like to point out that gradient noise in stochastic gradient naturally regularizes the preconditioner estimation as shown in [7]. Hence we have no need to remove those extra $N_x + 1$ degrees of freedom in $W_2$ in the other five discrete problems. Still, it should be beneficial to remove any nuisance parameter in PSGD, although we do not study this topic in detail.

Only the PSGD with sparse preconditioner is tested. The maximum allowed number of iteration is $10^5$. Table I summarizes the failure rate results. Note that RNN training may take a long time, and for certain cases, we have not finished five runs when this paper is prepared. We compare them with the results reported in [3]. Since only a few runs are conducted, neither the result here nor the one in [3] has statistical significance. Hence we would like to compare the maximum sequence length that an algorithm can handle without failure. This criterion favors the results reported in [3] as only four runs are conducted there, while PSGD carries five runs for each sequence length. These results are summarized in Table II. From table II, we observe that PSGD outperforms Hessian-free optimization with Tikhonov damping on the multiplication, XOR, 2-bit temporal order and 5-bit memorization problems. PSGD outperforms Hessian-free optimization with structural damping on the multiplication, 2-bit temporal order and random permutation problems. Overall speaking, PSGD may outperform Hessian-free optimization with either Tikhonov damping or structural damping, and can be comparable to the best performance achieved by both versions of Hessian-free optimization.

IV. CONCLUSIONS AND DISCUSSIONS

Preconditioned stochastic gradient descent (PSGD) is a general and simple learning algorithm, and requires little tuning effort. On challenging optimization problems, PSGD may converge much faster than stochastic gradient descent (SGD), although it does need to evaluate the stochastic gradient twice at each iteration and update the preconditioner once. The time spent on preconditioner updating is negligible when the preconditioner has certain sparse structures.
TABLE I
PSGD’s failure rate on eight problems with four sequence lengths. The results are shown as (number of failed runs)/(number of total runs).

|                  | 30  | 50  | 100 | 200 |
|------------------|-----|-----|-----|-----|
| Addition         | 0/5 | 0/5 | 0/5 | 2/5 |
| Multiplication   | 0/5 | 0/5 | 0/5 | 0/5 |
| XOR              | 0/5 | 0/5 | 3/5 | 1/1 |
| 2-bit temporal order | 0/5 | 0/5 | 0/5 | 3/4 |
| 3-bit temporal order | 0/5 | 0/5 | 0/5 | 1/1 |
| Random permutation | 0/5 | 0/5 | 0/5 | 0/5 |
| 5-bit memorization | 0/5 | 0/5 | 0/5 | 0/5 |
| 20-bit memorization | 0/5 | 0/5 | 0/3 | 0/0 |

TABLE II
Maximum sequence length without failure

|                  | HF/Tikhonov | HF/structural | PSGD |
|------------------|-------------|---------------|------|
| Addition         | 100         | 100           | 100  |
| Multiplication   | 100         | 100           | 200  |
| XOR              | 30          | 50            | 50   |
| 2-bit temporal order | 50      | 50            | 100  |
| 3-bit temporal order | 100       | 100           | 100  |
| Random permutation | 200        | 100           | 200  |
| 5-bit memorization | < 30      | 200           | 200  |
| 20-bit memorization | 30       | 100           | n/a  |

When parallel processing is available, the two gradients can be evaluated simultaneously. Hence, compared with SGD, PSGD could considerably shorten the training time.

We have applied PSGD to recurrent neural network (RNN) training on typical pathological synthetic problems which fails SGD in most cases, and found that PSGD may outperform Hessian-free optimization in terms of maximum sequence length without failure. We do not formally compare the computational complexity between PSGD and Hessian-free optimization. PSGD is virtually as simple as SGD, while each iteration of Hessian-free optimization composes of many steps of line search in conjugate directions, and a huge mini-batch size is unavoidable. Thus intuitively, PSGD is simpler, and a better choice in practice.
Unlike many traditional second-order optimization algorithms which assume positive definite Hessian, PSGD is designed for both convex and non-convex optimizations. This might explains its superior performance even its implementation is just slightly more complicated than SGD. Practically, PSGD naturally damps gradient noise, allowing the use of small mini-batch sizes to reduce computational complexity, while many off-the-shelf algorithms require a large mini-batch size for accurate gradient and cost function evaluations to facilitate line search. Furthermore, PSGD is easier to use since its step size is normalized, saving the trouble of step size selection by either hand tweaking or step size searching algorithms. The preconditioner can have flexible forms, providing trade off room between performance and complexity. These properties make PSGD an attractive alternative to SGD and many other optimization algorithms.

REFERENCES

[1] P. J. Werbos, “Backpropagation through time: what it does and how to do it,” IEEE Proc., vol. 78, no. 10, pp. 1550–1560, Oct. 1990.

[2] R. Pascanu, T. Mikolov, and Y. Bengio, “On the difficulty of training recurrent neural networks,” arXiv:1211.5063, 2012.

[3] J. Martens and I. Sutskever, “Learning recurrent neural networks with Hessian-free optimization,” In Proc. of the 28th ICML, 2011.

[4] S. Hochreiter and J. Schmidhuber, “Long short-term memory,” Neural computation, vol. 9, no.8, pp. 1735–1780, 1997.

[5] K. Greff, R. K. Srivastava, J. Koutnk, B. R. Steunebrink, and J. Schmidhuber, “LSTM: a search space odyssey,” arXiv:1503.04069, 2015.

[6] N. Schraudolph, “Fast curvature matrix-vector products for second-order gradient descent,” Neural Computation, vol. 14, no. 7, pp. 1723–1738, 2002.

[7] X.-L. Li, “Preconditioned stochastic gradient descent,” arXiv:1512.04202, 2015.