L-SR1: A Novel Second Order Optimization Method for Deep Learning

Vivek Ramamurthy
Sentient Technologies
1 California Street Suite 2300
San Francisco, CA 94111
vivek.ramamurthy@sentient.ai

Nigel Duffy
Sentient Technologies
1 California Street Suite 2300
San Francisco, CA 94111
nigel.duffy@sentient.ai

Abstract

We describe L-SR1, a new second order method to train deep neural networks. Second order methods hold great promise for distributed training of deep networks. Unfortunately, they have not proven practical. Two significant barriers to their success are inappropriate handling of saddle points, and poor conditioning of the Hessian. L-SR1 is a practical second order method that addresses these concerns. We provide preliminary experimental results showing that L-SR1 performs at least as well as several other first order methods, and better than L-BFGS, on the MNIST and CIFAR10 datasets.

1 Motivation

Second order methods hold great potential for distributing the training of deep neural networks. Due to their use of curvature information, they can often find good minima in far fewer steps than first order methods such as stochastic gradient descent (SGD). However, stochastic second order methods typically require larger mini-batches [8][10]. This is because they estimate second derivatives via differences between estimated gradients. The gradient estimates need to have less variance, so that when we take their differences, the result has low variance. As a result they provide a different trade-off between number of steps and mini-batch size than do SGD-like methods. This trade-off is interesting, because while steps must be evaluated sequentially, a mini-batch may be evaluated in parallel. Thus, second order methods present an opportunity to extract more parallelism in neural network training. In particular, when mini-batches are sufficiently large, their evaluation may be distributed.

L-BFGS [1] is perhaps the most commonly used second order method in machine learning. BFGS is a quasi-Newton method that maintains an approximation to the inverse Hessian of the function being optimized. L-BFGS is a limited memory version of BFGS that removes the requirement to explicitly store the inverse Hessian and can therefore be used practically for large scale problems. L-BFGS is typically combined with a line search technique to choose an appropriate step size at each iteration. L-BFGS has been used to good effect in convex optimization problems in machine learning, but has not found effective use in large scale non-convex problems such as deep learning.

Three critical weaknesses have been identified. First, we know that training deep neural networks involves minimizing non-convex error functions over continuous, high dimensional spaces. It has been argued that the proliferation of saddle points in these problems presents a deep and profound difficulty for quasi-Newton optimization methods [11]. Furthermore, it has been argued that curvature matrices generated in second order methods are often ill-conditioned, and these need to be carefully repaired. A variety of approaches to this have been suggested, including the use of an empirical Fisher diagonal matrix [9]. Finally, popular quasi-Newton approaches, such as L-BFGS,

require line search to make parameter updates, which requires many more gradient and/or function evaluations [1].

It is worth noting that several approaches have been proposed to overcome the weaknesses of L-BFGS. First, it has been proposed to initialize L-BFGS with a number of SGD steps. However, this greatly diminishes the potential for parallelism [8][10]. Second, it has been proposed to use "forgetting", where every few (say, for example, 5) steps, the history for L-BFGS is discarded. However, this greatly reduces the ability to use second order curvature information. Despite these proposals, L-BFGS is not commonly used for deep learning.

We propose L-SR1, a second order method that addresses each of these concerns.

2 Algorithm

We believe that it is possible to overcome saddle points using rank-one update based second order methods. The more common rank-two methods, e.g. L-BFGS, maintain a positive definite approximation to the inverse of the Hessian, by design [1]. At saddle-points, the true Hessian cannot be well approximated by a positive definite matrix, causing commonly used second order methods to go uphill [11]. On the other hand, rank-one approaches such as SR1 (Symmetric Rank One) [1] don't maintain this invariant, and so they can go downhill at saddle points. Numerical experiments [5] suggest that the approximate Hessian matrices generated by the SR1 method show faster progress towards the true Hessian than those generated by BFGS. This suggests that a limited memory SR1 method (L-SR1, if you like) would potentially outperform L-BFGS in the task of high dimensional optimization in neural network training. The building blocks needed to construct an L-SR1 method have been suggested in the literature [2][4]. To the best of our knowledge, however, there is no complete L-SR1 method previously described in the literature [1]. This prompted us to develop and test the approach, specifically in the large scale non-convex problems that arise in deep learning.

Two other insights make L-SR1 practical by removing the requirement for a line search and addressing the conditioning problem. First, we replace the line search with a trust region approach. While L-BFGS using line search is well studied, recently, an L-BFGS method that uses a trust-region framework has also been proposed [6]. Second, we combine L-SR1 with batch normalization. Batch normalization is a technique of normalizing inputs to layers of a neural network, used to address a phenomenon known as *internal covariate shift* during training [12]. Our hypothesis is that batch normalization may cause parameters of a neural network to be suitably scaled so that the Hessian becomes better conditioned. We tested this hypothesis empirically and outline the results below.

Our algorithm is synthesized as follows. We take the basic SR1 algorithm described in [1] (Algorithm 6.2), and represent the relevant input matrices using the limited-memory representations described in [2]. The particular limited-memory representations used in the algorithm vary, depending on whether we use trust region or line search methods as subroutines to make parameter updates, as does some of the internal logic. In the Appendix, we provide a brief primer on line search and trust region methods, as well as on quasi-Newton methods and their limited memory variants, to capture the core novelty of our approach. Additional details about the algorithm may be found in the pseudocode that is also provided in the Appendix. The algorithm we develop is general enough to work with any line search or trust region method. While we tested the algorithm with line search approaches described in [7], and with the trust region approach described in [3], in this paper, we focus our experimental investigations on using the trust region approach, and the advantage that provides over using other first and second order optimization methods.

3 Experiments

In the following, we summarize the results of training standard neural networks on the MNIST and CIFAR10 datasets using our approach, and benchmarking the performance with respect to other first and second order methods. We compared our L-SR1 (with trust region) approach, with Nesterovs Accelerated Gradient Descent (NAG), L-BFGS with (and without) forgetting every 5 steps, default SGD, AdaDelta, and SGD with momentum. For each approach, and for each dataset, we considered

[1] The reference [3] describes an approach to solve the trust region subproblem encountered in an L-SR1 method, but does not describe the L-SR1 method itself.

the case where our networks had batch normalization layers within them, and the case where they did not. The parameters of the networks were randomly initialized. In all cases, the networks were trained for 20 epochs. Finally, all experiments were repeated 10 times to generate error bars. The plots below show the variation of test loss with number of epochs. Here, test loss refers to the misclassification error rate on the test dataset.

3.1 MNIST

We considered the LeNet5 architecture in this case, which comprised 2 convolutional layers, followed by a fully connected layer and an outer output layer. Each convolutional layer was followed by a max-pooling layer. In the case where we used batch-normalization, each convolutional and fully connected layer was followed by a spatial batch normalization layer. We used a mini-batch size of 20 for the first order methods like NAG, SGD, AdaDelta and SGD with momentum, and a mini-batch size of 400 for L-SR1 and L-BFGS. The memory size was set to 5 for both L-SR1 and L-BFGS. Further details on the network architecture and other parameter settings are provided in the Appendix.

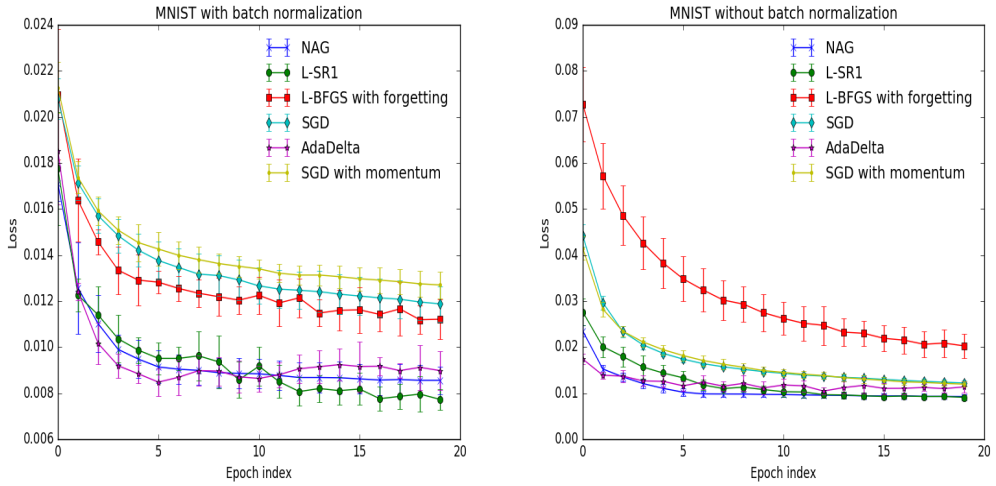

Figure 1: Variation of test loss with number of epochs, on the MNIST dataset, with and without batch normalization. Note that the scales on the y-axes are different.

3.2 CIFAR10

We considered a slight modification to the 'LeNet5' architecture described above. We used a mini-batch size of 96 for NAG, SGD, AdaDelta and SGD with momentum. The other mini-batch sizes and memory sizes for L-SR1 and L-BFGS were as above. Further details on the network architecture and other parameter settings are provided in the Appendix.

3.3 Discussion

Our experiments suggest the following:

- L-SR1 performs as well as, or slightly better than all the first order methods on both the MNIST and CIFAR10 datasets, with or without batch normalization.
- L-SR1 is substantially better than L-BFGS in all settings, with or without forgetting.
- Forgetting appears to be necessary in order to get L-BFGS to work. Without forgetting, the approach appears to be stuck where it is initialized. For this reason, the plots for L-BFGS without forgetting have not been included.

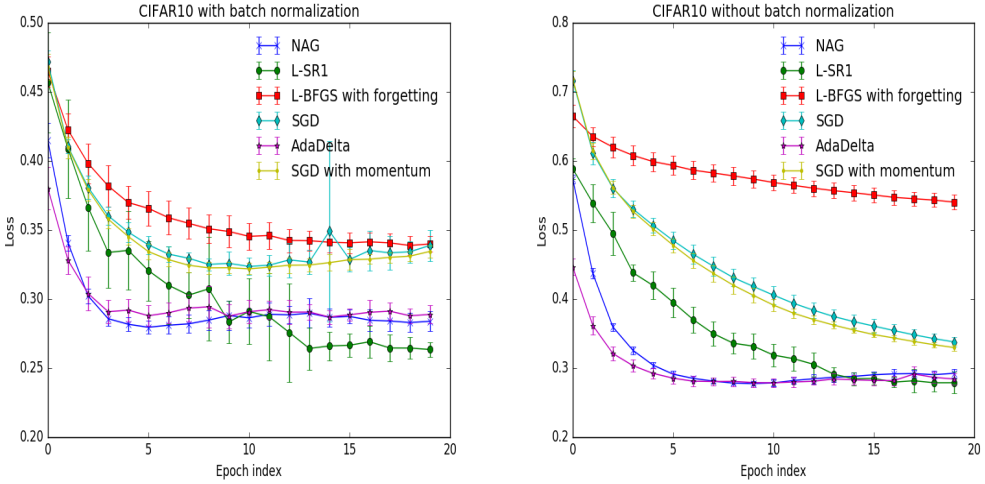

Figure 2: Variation of test loss with number of epochs, on the CIFAR10 dataset, with and without batch normalization. Note that the scales on the y-axes are different.

- Batch normalization appears to improve the performance of all approaches, particularly the early performance of second order approaches like L-SR1 and L-BFGS.

Note that in these experiments we compare performance between the algorithms per epoch. To reiterate, we use mini-batch sizes of 20 and 96 for the first order methods for the MNIST and CIFAR experiments respectively, while we use a mini-batch size of 400 for the second order methods in all cases. This means that the second order methods take fewer steps per epoch than the first order methods due to their larger mini-batch sizes. This clearly illustrates the trade-off involved and shows the potential for parallel and distributed variants.

4 Conclusions

In this paper, we have described L-SR1, a new second order method to train deep neural networks. Our preliminary experiments suggest that this approach compares quite favorably with first order, and other second order methods. Our experiments also appear to validate our intuition about the ability of L-SR1, to overcome key challenges associated with second order methods. These challenges include inappropriate handling of saddle points, and poor conditioning of the Hessian. Second order methods hold great promise for distributed training of deep networks, so we see our work as an important step toward that goal.

Our results also suggest that a deeper exploration of L-SR1 is in order. A couple of interesting next steps are as follows. We intend to use this approach to train deeper networks such as residual networks, and study its performance. Furthermore, it would be interesting to experiment with the hyperparameters of L-SR1, such as memory size, minibatch size, and the increase and decrease factors of the trust-region radius, to gain more intuition about their effect on the performance of L-SR1.

Acknowledgments

We would like to thank Jennifer Erway and Roummel Marcia for providing us timely clarifications on their OBS method for solving L-SR1 trust region subproblems, and also for sharing their MATLAB code with us.

References

[1] Nocedal, J. & Wright, S. J. (2006) *Numerical Optimization.* Springer-Verlag, New York, 2nd edition.

[2] Byrd, R. H., Nocedal, J. & Schnabel, R. B. (1994) Representations of quasi-Newton matrices and their use in limited-memory methods. *Mathematical Programming*, **63**:129-156.

[3] Brust, J., Erway, J. B. & Marcia, R. F. (2015) On Solving L-SR1 Trust-Region Subproblems. *http://arxiv.org/pdf/1506.07222v3.pdf*.

[4] Khalfan, H., Byrd, R. H. & R. B. Schnabel. (1993) A Theoretical and Experimental Study of the Symmetric Rank One Update. *SIAM Journal on Optimization*, **3**(1):1-24.

[5] Conn, A. R., Gould, N. I. M. & Toint, Ph. L. (1991) Convergence of quasi-Newton matrices generated by the symmetric rank one update. *Mathematical Programming*, **50**:177-195.

[6] Burke, J. V., Wiegman, A. & Xu, L. (2008) Limited memory BFGS updating in a trust-region framework. *Working paper*.

[7] Dennis Jr., J. E. & Schnabel, R. B. (1983) *Numerical methods for unconstrained optimization and nonlinear equations.* Prentice-Hall.

[8] Dean, J., Corrado, G. S., Monga, R., Chen, K., Devin, M., Le, Q. V., Mao, M. Z., Ranzato, M. A., Senior, A., Tucker, P., Yang, K. & Ng, A. Y. (2012) Large Scale Distributed Deep Networks. *Advances in Neural Information Processing Systems*, pp. 1223-1231.

[9] Martens, J. (2016) Second-Order Optimization for Neural Networks. *Ph.D. thesis, Graduate Department of Computer Science, University of Toronto*.

[10] Le, Q. V., Ngiam, J., Coates, A., Lahiri, A., Prochow, B. & Ng. A. Y. (2011) On Optimization Methods for Deep Learning. *Proceedings of the 28th International Conference on Machine Learning*, Bellevue, WA, USA.

[11] Dauphin, Y., Pascanu, R., Gulcehre, C., Cho, K., Ganguli, S. & Bengio, Y. (2014) Identifying and attacking the saddle point problem in high-dimensional non-convex optimization. *https://arxiv.org/abs/1406.2572v1.pdf*.

[12] Ioffe, S. & Szegedy, C. (2015) Batch Normalization: Accelerating Deep Network Training by Reducing Internal Covariate Shift. *https://arxiv.org/pdf/1502.03167v3.pdf*.