Improving LBFGS optimizer in PyTorch: Knowledge transfer from radio interferometric calibration to machine learning

Sarod Yatawatta
ASTRON, The Netherlands Institute for Radio Astronomy,
Dwingeloo, The Netherlands.
yatawatta@astron.nl

Hanno Spreeuw and Faruk Diblen
Netherlands eScience Center,
Amsterdam, The Netherlands.
h.spreeuw,f.diblen@esciencecenter.nl

***Abstract*—We have modified the LBFGS optimizer in PyTorch based on our knowledge in using the LBFGS algorithm in radio interferometric calibration (SAGECal). We give results to show the performance improvement of PyTorch in various machine learning applications due to our improvements.**

***Index Terms*—LBFGS, Machine Learning, Radio Interferometry**

I. Summary

The limited memory Broyden-Fletcher-Goldfarb-Shanno (LBFGS) algorithm is extensively used in nonlinear optimization with a large number of unknowns. Its run time memory requirement is significantly lower than methods that require the calculation of the full Hessian, but it still gives faster convergence than first order methods such as gradient descent. Due to these reasons, LBFGS is heavily used in radio interferometric calibration (SAGECal [1]) and we have optimized it for best performance in both GPU and CPU hardware.

Apart from providing the cost function to be minimized, LBFGS also requires the gradient of the cost function, and therefore, limits the use of LBFGS in general optimization problems with minimal user input. It is possible to calculate the derivatives using numerical differencing but this would severely limit the performance. On the other hand, automatic differentiation has become widely available to general users as used in PyTorch [2]. PyTorch itself has its own implementation of LBFGS but lacks the line search functionality to calculate the optimum step size. Once the direction of descent of minimizing the cost is found, the amount of descent is determined by the step size and this is currently a fixed value in PyTorch.

There are various algorithms for line search [3] and we have opted to use exact line search with cubic interpolation in SAGECal. We have ported this line search algorithm to PyTorch for general use. As expected, using this line search method gives much better results in various test problems than keeping the step size fixed. The only drawback is the slightly higher computational cost in the line search.

This work was supported by Netherlands eScience Center (project DIRAC, grant 27016G05).

In this work we will present the implementation details of the line search in PyTorch and comparisons of its performance with a fixed step size and comparisons of other step size selection algorithms as well as its performance with other optimization algorithms used in deep learning such as stochastic gradient descent and Adam. In Fig. 1 we have shown the performance of LBFGS with fixed and variable step size (also SGD [4] and Adam [5] with learning rate 0.1) while solving a rule 110 cellular automaton problem [6], [7]. We clearly see that step size obtained by the line search method gives faster convergence than the fixed step size of 0.1.

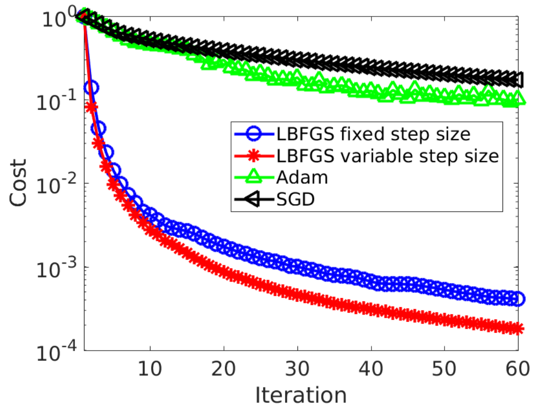

Fig. 1. Cellular automaton rule 110 cost with iteration number of LBFGS with fixed and variable step sizes, compared with Adam and SGD.

We have also modified the LBFGS algorithm to a multi-batch (stochastic) scenario (based on variance of the gradient which is estimated online [8], and Armijo line search for step size selection [9] with maximum step size determined by variance of the gradient) and unlike existing approaches [10], [11], our stochastic algorithm does not require variable batch sizes or evaluation of the gradient of individual data points within a single batch. More details about this will be presented in future work.

References

[1] S. Yatawatta, S. Kazemi, and S. Zaroubi, "GPU accelerated nonlinear optimization in radio interferometric calibration," in *Innovative Parallel Computing (InPar), 2012*, May 2012, pp. 1–6.

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

[3] R. Fletcher, *Practical Methods of Optimization; (2Nd Ed.)*, Wiley-Interscience, New York, NY, USA, 1987.

[4] Herbert Robbins and Sutton Monro, "A stochastic approximation method," *Ann. Math. Statist.*, vol. 22, no. 3, pp. 400–407, 09 1951.

[5] D. P. Kingma and J. Ba, "Adam: A Method for Stochastic Optimization," *ArXiv e-prints*, Dec. 2014.

[6] Eric W. Weisstein, "Rule 110," http://mathworld.wolfram.com/Rule110.html, Accessed: 2018-06-11.

[7] S. Yakovenko, "Rule 110," https://stackoverflow.com/questions/50621786/lbfgs-never-converges-in-large-dimensions-in-pytorch, Accessed: 2018-06-11.

[8] B. P. Welford, "Note on a method for calculating corrected sums of squares and products," *Technometrics*, vol. 4, no. 3, pp. 419–420, 1962.

[9] J. Nocedal and S. J. Wright, *Numerical Optimization*, New York USA:Springer, 1999.

[10] R. Bollapragada, D. Mudigere, J. Nocedal, H.-J. M. Shi, and P. T. P. Tang, "A Progressive Batching L-BFGS Method for Machine Learning," *ArXiv e-prints*, Feb. 2018.

[11] Nicol N. Schraudolph, Jin Yu, and Simon Gnter, "A stochastic quasi-newton method for online convex optimization," in *Proceedings of the Eleventh International Conference on Artificial Intelligence and Statistics*, Marina Meila and Xiaotong Shen, Eds., San Juan, Puerto Rico, 21–24 Mar 2007, vol. 2 of *Proceedings of Machine Learning Research*, pp. 436–443, PMLR.