Parallelizing the training of sequential and linear models

COMP 5704 Wei Li

Abstract

1. introduciton

Given the ubiquity of large-scale data solutions and the availability of low-commodity clusters and multicore systems, paralleling learning models using gradient based methods to speed them up is an obvious choice. However, such algorithms like SGD's scalability is limited by its inherently sequential nature (Recht et al., 2011). One common observation is that many parallelized solutions to those problems either result in inferior model quality(eg. divergence) or low training efficiency. (Raff et al., 2018)

One direction of researching focus on paralleling the training of large data sets that are distributed in a cluster of machines and are pre-processed in a MapReduce-like framework(Dean & Ghemawat, 2004). In such a scenario, only appropriate MapReduce-preprocessed data can be small enough to be trained parallelly in a number of machines. However, this framework's performance is often impaired by the low data reading rate due to communication and the frequent checkpointing for fault tolerance. (Recht et al., 2011)

In this type of studies, (Zinkevich et al., 2010) proposed running many instances of stochastic gradient descent on different machines and averaging their output. However, (Recht et al., 2011) shows in their experiment that this method does not outperform a serial scheme. (Dekel et al., 2012) and (Agarwal et al., 2010) also propose schemes based on the idea of averaging of gradients via a distributed protocol and achieved linear speedup.

Another stream of studies looks at paralleling multicore systems in a single workstation. Unlike cluster servers, local workstations have shared memory with low communication latency and high bandwidth. These advantages move the parallel computation bottleneck from slow communication to the frequent synchronization(or locking) – as the workers work on the same memories, they must be more careful about concurrency.

A parallel linear learning method should have two primary goals: Model Quality, that the algorithm should be able to converge to the same result as its serial counterpart and perform equally well; Training Efficiency, that as the number of processors increase, the speed of training should increase at least at the same scale. Both of the cluster training and single-machine training need to be optimized for the two goals. In terms of Training Efficiency, cluster training aims at better communication between different hosts and single-machine training aims at avoiding synchronization and locking among workers.

In the multicore scenario, a pivotal and most commonly used method is the Wild (Recht et al., 2011). Prior to the work, most locking scheme and ideas for parallelizing stochastic gradient descent is presented in (Bertsekas & Tsitsiklis, 1989). They describe, for example, using stale gradient updates computed across many processors in a master-worker setting proved convergence of those approach, but did not mention the speed of the convergence. Another locking scheme is round-robin (Zinkevich et al., 2009) where the processors are ordered and each one update the decision variable in sequence. When gradient computation is costly, the time needed by locking is dwarfed, so the method achieved linear speedup. However, the method does not scale well when gradients can be easily computed.

(Recht et al., 2011) proved and tested that in case when the data access is sparse enough, the possibility of memory overwrites are rare, and even they occurs, they introduce barely any error into the computation. They presents the Wild method which requires no locking and achieved linear speedup. They also outperforms the Round Robin method's speed by a order of magnitude in application where gradient can be effectively computed. The asynchronous strategy has become the dominant method to parallel linear models in multicore system. After Wild, many works adapted the method or use the asynchronous strategy to specific algorithms and implementation. In these adaptations a common observation is that models often diverge after many cores are added, or when the feature set is dense.(Hsieh et al., 2015)(Leblond et al., 2017)(Tran et al., 2015)(Zhao & Li, 2016). Since Wild does not scale well in multiple-CPUs system(multi-sockets) (Zhang et al., 2016) proposed Hogwild++, which uses multiple local weight vectors to avoid performance regressions of SGD on multiple CPUs systems.

This approach introduce new hyper-parameter to be tuned which consume CPU time depending on the hardware and data.

Some of the studies also look into another type of gradient method, namely the Stochastic Dual Coordinate Ascent (SDCA), which covers a wide set of possible linear models. Coordinate descent is a classical optimization technique (Bertsekas, 1995). SDCA try to solve the dual problem of risk minimization and shows faster speed than the primal solver such as SGD. (Shalev-Shwartz & Zhang, 2013) and (Chang et al., 2008) shows that solving the dual problem with SDCA is faster on large-scale data sets. SDCA is successfully implemented in a linear model packages for large data sets called LIBLINEAR (Fan et al., 2008).

After (Tran et al., 2015) implemented a parallel version of SDCA based on asynchronous idea from Wild. (Hsieh et al., 2015) developed the PASSCoDe framework for parallel implementations of the SDCA algorithm and compared Wild with Atomic and lock-based methods. Their experiment shows that Wild algorithm provide the best overall efficiency. They also found that atomic approach's performance varies significantly depending on the sparsity of the data set. Since Wild shows difficulty of converging in multiple CPUs system, (Zhang & Hsieh, 2016) developed PASSCoDe-fix to fix this problem. In the approach, although updates are performed wildly, there are convergence check at each iteration, which result in a semi-asynchronous solution. Although the adaption is successful, the method is only specific to the SDCA, and its complex implementation slows down the computation. Following the work, (Raff et al., 2018) presented a stochastic update policy (SAUS) which introduces no new tunable hyper-parameters, and is not algorithm specific. The method shows superior speedup on system with more than 8 cores and scale up nicely up to 80 cores.

References

Agarwal, A., Wainwright, M. J., and Duchi, J. C. Distributed dual averaging in networks. In *Advances in Neural Information Processing Systems*, pp. 550–558, 2010.

Asanovic, K., Bodik, R., Catanzaro, B., Gebis, J., Husbands, P., Keutzer, K., Patterson, D., Plishker, W., Shalf, J., Williams, S., and Yelick, K. The landscape of parallel computing research: A view from berkeley. 2006.

Bertsekas, D. and Tsitsiklis, J. Parallel and distributed computation: Numerical methods. 1989.

Bertsekas, D. P. Nonlinear programming. *Journal of the Operational Research Society*, 48:334, 1995.

Chang, K.-W., Hsieh, C.-J., and Lin, C. Coordinate descent method for large-scale l2-loss linear support vector machines. *J. Mach. Learn. Res.*, 9:1369–1398, 2008.

De, S. and Goldstein, T. Efficient distributed sgd with variance reduction. *2016 IEEE 16th International Conference on Data Mining (ICDM)*, pp. 111–120, 2016.

Dean, J. and Ghemawat, S. Mapreduce: Simplified data processing on large clusters. 2004.

Dekel, O., Gilad-Bachrach, R., Shamir, O., and Xiao, L. Optimal distributed online prediction using mini-batches. *The Journal of Machine Learning Research*, 13:165–202, 2012.

Fan, R.-E., Chang, K.-W., Hsieh, C.-J., Wang, X.-R., and Lin, C. Liblinear: A library for large linear classification. *J. Mach. Learn. Res.*, 9:1871–1874, 2008.

Fuller, S. and Millett, L. I. The future of computing performance: Game over or next level? 2014.

Gan, W., Lin, C.-W., Fournier-Viger, P., Chao, H.-C., and Yu, P. S. A survey of parallel sequential pattern mining. *ACM Transactions on Knowledge Discovery from Data (TKDD)*, 13:1 – 34, 2019.

Hsieh, C.-J., Yu, H.-F., and Dhillon, I. Passcode: Parallel asynchronous stochastic dual co-ordinate descent. *ArXiv*, abs/1504.01365, 2015.

Leblond, R., Pedregosa, F., and Lacoste-Julien, S. Asaga: asynchronous parallel saga. In *Artificial Intelligence and Statistics*, pp. 46–54. PMLR, 2017.

Nesterov, Y. Efficiency of coordinate descent methods on huge-scale optimization problems. *SIAM J. Optim.*, 22: 341–362, 2012.

Raff, E., Hamilton, B. A., and Sylvester, J. Linear models with many cores and cpus: A stochastic atomic update scheme. *2018 IEEE International Conference on Big Data (Big Data)*, pp. 65–73, 2018.

Recht, B., Ré, C., Wright, S. J., and Niu, F. Hogwild: A lock-free approach to parallelizing stochastic gradient descent. In *NIPS*, 2011.

Ruder, S. An overview of gradient descent optimization algorithms. *ArXiv*, abs/1609.04747, 2016.

Saha, A. and Tewari, A. On the nonasymptotic convergence of cyclic coordinate descent methods. *SIAM J. Optim.*, 23:576–601, 2013.

Shalev-Shwartz, S. and Zhang, T. Stochastic dual coordinate ascent methods for regularized loss. *J. Mach. Learn. Res.*, 14:567–599, 2013.

Tran, K., Hosseini, S., Xiao, L., Finley, T., and Bilenko, M. Scaling up stochastic dual coordinate ascent. In *Proceedings of the 21th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, pp. 1185–1194, 2015.

Zhang, H. and Hsieh, C.-J. Fixing the convergence problems in parallel asynchronous dual coordinate descent. *2016 IEEE 16th International Conference on Data Mining (ICDM)*, pp. 619–628, 2016.

Zhang, H., Hsieh, C.-J., and Akella, V. Hogwild++: A new mechanism for decentralized asynchronous stochastic gradient descent. *2016 IEEE 16th International Conference on Data Mining (ICDM)*, pp. 629–638, 2016.

Zhao, S.-Y. and Li, W.-J. Fast asynchronous parallel stochastic gradient descent: A lock-free approach with convergence guarantee. In *AAAI*, pp. 2379–2385, 2016.

Zinkevich, M., Smola, A., and Langford, J. Slow learners are fast. In *NIPS*, 2009.

Zinkevich, M., Weimer, M., Smola, A., and Li, L. Parallelized stochastic gradient descent. In *NIPS*, 2010.