DS222: Assignment 2(Multiclass Classification Using Parameter Server)

Kapil Pathak [1]

Abstract

In this report, the results of implementations of multilabel and multiclass logistic classification for DBPedia dataset have been discussed. First implementation is in local set up while other implementation involves distributed tensorflow with synchronous SGD, Asynchronous SGD and Stale Synchronous SGD with parameter server framework.

1. Local Settings

In the local settings, each document from the data-set is embedded into a vector 100 dimension using doc2vec document embedding (Le & Mikolov, 2014)from gensim library. This 100 dimensional embedding are classified into 50 classes using logistic regression. Here 50 binary classifiers have been implemented for individual classes and training accuracy as well as losses have been recorded for an individual classifier.

Here each document with given labels is fed as positive example to classifiers corresponding to each label and fed as an negative example to other classifiers. Here, every document may have multilabels. We treat that document positive for every label and negative for other labels. Formula for loss function is referred from (Bishop, 2006) and it can be give as follows.

$$Loss = \sum_{n=1}^{N}\sum_{k=1}^{K} y_{n,k}\log t_{n,k} + \lambda||\mathbf{W}||^{\mathbf{2}} \qquad (1)$$

Here total number of examples N and total K classes in the classification. This settings gives 32.13 percent accuracy on training dataset, 22.24 percent accuracy on testing dataset while 24.28 percent on development set. This implementation involves very screwed distribution corresponding to positive and negative instances. This might be the reason for lower accuracy values. The learning rate was kept 0.1 with L2-regularization constant as 0.1 initially.

Training graph can be obtained with constant learning rate as following. The local implementation consists of hand-coded SGD. For training, it takes 4532.90643 seconds for 700 epochs on full DBPedia dataset. For testing, it takes 3697.0546 seconds for 1000 epochs.

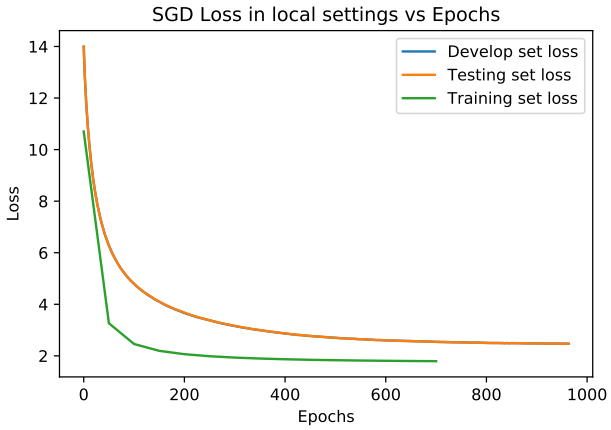

Figure 1. Here training loss graph can be seen as overlapped with development set loss graph and is saturated below testing loss.

1.1. Comparison of different learning schemes

In this section, we will perform different learning schemes for our SGD implementation. Here three schemes are considered as constant learning scheme 0.015, exponentially increasing learning scheme with factor 1.08 and exponentially decreasing learning scheme with factor 0.92. Here after every update learning rate increases or decreases by given factor. Here the rate of convergence for both increasing and decreasing learning schemes is better than that of learning with constant learning rate. From the graph, it can be see that decreasing learning rate is performing better in terms of the rate of convergence and loss after saturation. Also towards the end of epoch cycle, loss with increasing rate increases as it reaches near global optimum, learning agent begins to oscillate near local minima and the loss diverges. This effect can be seen from following figure2.

*Equal contribution [1]Department of Computer Science and Automation, Indian Institute of Science Bangalore. Correspondence to: Kapil Pathak <kapilpathak@iisc.ac.in>.

Proceedings of the 35^{th} International Conference on Machine Learning, Stockholm, Sweden, PMLR 80, 2018.

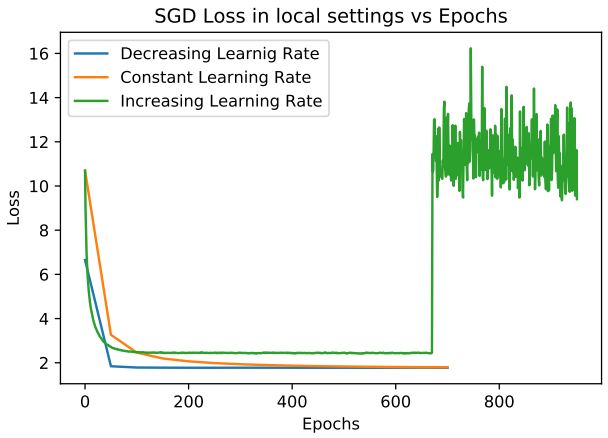

Figure 2. Here, the rate of convergence for increasing learning rate is more in initial epochs. But the loss diverges after 650 epochs

2. Distributed Settings

In the distributed settings all hyperparamters are freezed with learning rate to 0.01 and regularization to 0.1. This part of the assignment was done using (Ten)distributed tensorflow which generates model graph on each worker node and parameter server is used to communicate gradients and updated parameters from and to worker nodes respectively. This process can be performed in various manners such as Bulk Synchronous Processing (BSP) SGD, Synchronous SGD etc. In this part of the assignment, the analysis has been performed on multiclass classification of DBPedia dataset with BSP SGD, Synchronous SGD and Stale Synchronous SGD.

2.1. Busk Synchronous Parallel SGD

In Bulk Synchronous Parallel (BSP), the model alternate between computation phase and communication phase or also called synchronized barrier. Each worker accumulates the gradients for all parameters until the next 'superstep' and the effect of the computations of one worker node is invisible to other worker nodes until the next superstep occurs. Beacuse of the cleaner separation of computation and communication phase, BSP scheme often enjoys correctness of parallel machine learning algorithms.

In (Abadi et al., 2015)Tensorflow API, *SyncReplicasOptimizer* performs similar job to BSP. Here all accumulators, each for individual parameter performs the job of gradients average. After each global step, the accumulator checks the staleness of all gradients and drops the gradients before those. A drawback of this scheme is that the throughput of individual worker is severely affected as each worker needs to wait until the next superstep occurs.

The training time and testing time taken for BSP execution is 948.3753 seconds and 834.2394 seconds respectively for full DBPedia dataset for 1000 epochs. Here 2 nodes are serving as a parameter server while 2 nodes are serving as worker nodes. The loss vs epochs graphs for worker0 and worker1 are given in figure3 and figure4 respectively.

Figure 3. Here, the rate of convergence for increasing learning rate is more in initial epochs. But the loss diverges after 650 epochs

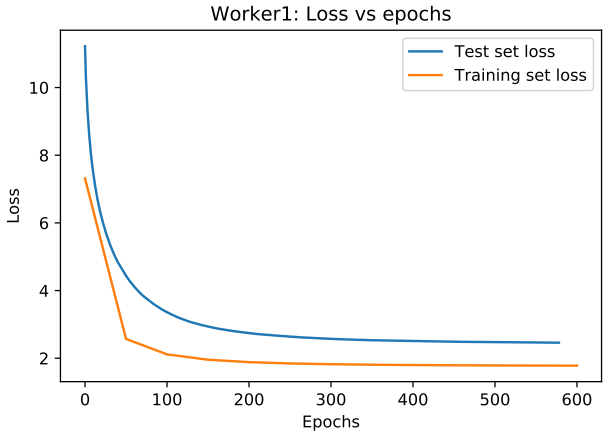

Figure 4. Here, the rate of convergence for increasing learning rate is more in initial epochs. But the loss diverges after 650 epochs

2.2. Asynchronous SGD

Unlike BSP,given in (Xing et al., 2016), in asynchronous SGD, any worker never waits for any superstep or other worker to catch synchronize the computations. Each worker computes gradients and communicates with parameter server separately throughout the course of each iteration. This scheme achieves near-ideal P-fold throughput in each

iteration. But as there is no synchronization among the co-workers,the staleness of the gradients causes slower convergence rate per iteration than BSP scheme. If the staleness is not bounded, it may even cause incorrect gradient calculations and wrong results. This is a drawback of asynchronous SGD.

In asynchronous SGD implementation, *AdagradOptimizer* is used for update. Also a node works as a parameter server while 2 nodes are working as worker nodes. The training time taken by asynchronous SGD is 869.3475 seconds for worker0 and 936.0925 for worker1 for the whole DBPedia dataset for 3000 epochs.

The training loss variation for both workers is given in figure5. Another point to be noticed from the figure5 is

Figure 5. The speed difference of both graphs in terms of convergence can be observed as worker1 is running faster than worker0

that number epochs to reduce the loss to the same value as BSP scheme is more in asynchronous SGD case. This observation also justfies the slower convergence rate per iteration in asynchronous SGD.

2.3. Stale Synchronous Parallel (SSP)

In case of asynchronous SGD, as unbounded staleness may cause incorrect results, Stale Synchronous Parallel (SSP) allows us to overcome this issue by bounding the staleness for particular s. For given staleness s, SSP allows a faster worker go ahead in terms of iterations than its other co-workers until it is s iterations apart. The local model parameters become stale as each worker receives partial updates.

ML algorithms under SSP bridging model find advantages of both BSP and asynchronous SGD as it increases throughput per iteration compared to BSP and avoid unbounded staleness enhancing convergence rate.

This part of the assignment needs more debugging.

Acknowledgements

All codes required for this assignment have been uploaded on Github. I also aknowledge Mr.Vikram Bhatt for helping me in concepts clearing and implementation in distributed settings. Apart from that, I took help from https://github.com/tmulc18/Distributed-TensorFlow-Guide

References

TensorFlow. URL https://www.tensorflow.org/api{_}docs/python/tf/train/SyncReplicasOptimizer.

Abadi, Martín, Agarwal, Ashish, Barham, Paul, Brevdo, Eugene, Chen, Zhifeng, Citro, Craig, Corrado, Greg S., Davis, Andy, Dean, Jeffrey, Devin, Matthieu, Ghemawat, Sanjay, Goodfellow, Ian, Harp, Andrew, Irving, Geoffrey, Isard, Michael, Jia, Yangqing, Jozefowicz, Rafal, Kaiser, Lukasz, Kudlur, Manjunath, Levenberg, Josh, Mané, Dan, Monga, Rajat, Moore, Sherry, Murray, Derek, Olah, Chris, Schuster, Mike, Shlens, Jonathon, Steiner, Benoit, Sutskever, Ilya, Talwar, Kunal, Tucker, Paul, Vanhoucke, Vincent, Vasudevan, Vijay, Viégas, Fernanda, Vinyals, Oriol, Warden, Pete, Wattenberg, Martin, Wicke, Martin, Yu, Yuan, and Zheng, Xiaoqiang. TensorFlow: Large-scale machine learning on heterogeneous systems, 2015. URL http://tensorflow.org/. Software available from tensorflow.org.

Bishop, Christopher M. *Pattern Recognition and Machine Learning (Information Science and Statistics)*. Springer-Verlag, Berlin, Heidelberg, 2006. ISBN 0387310738.

Le, Quoc and Mikolov, Tomas. Distributed representations of sentences and documents. In *Proceedings of the 31st International Conference on International Conference on Machine Learning - Volume 32*, ICML'14, pp. II–1188–II–1196. JMLR.org, 2014. URL http://dl.acm.org/citation.cfm?id=3044805.3045025.

Xing, Eric P, Ho, Qirong, Xie, Pengtao, and Wei, Dai. Strategies and principles of distributed machine learning on big data. *Engineering*, 2(2):179–195, 2016.