Stochastic Optimization Using Nomadic Computing in Big Data Analytics

Syed Atir Raza[1], Aqsa Anwar[2] Raybal Akhtar[3], Aqsa Ali[4]

1: School of Information Technology, Minhaj University, Lahore, Pakistan
atirraza.it@mul.edu.pk
2,3 :School of Software Engineering, Minhaj University, Lahore, Pakistan
4: School of Computer Science, Minhaj University, Lahore, Pakistan

Abstract

Today's applications frequently contain datasets that are too large to fit in the main memory of a single computer. Scalable and sophisticated machine-learning methods will be required to analyze these massive datasets. Stochastic optimization and inference algorithms are two commonly used approaches. The goal of this article is to examine the NOMAD novel nomadic framework, which combines stochastic optimization and distributed computing benefits, and determine whether or not it will be useful for big data. To investigate how stochastic optimization can assist with nomadic computing and nomad algorithms in the field of big data analytics.

Keywords

Big data, Nomadic computing, Stochastic optimization, Matrix completion.

1. Introduction

Today's applications frequently contain datasets that are too large to fit in the main memory of a single computer. Scalable and sophisticated machine-learning methods will be required to analyze these massive datasets. Stochastic optimization and inference algorithms are two commonly used approaches. The goal of this article is to introduce NOMAD, a novel nomadic framework that combines stochastic optimization and distributed computing advantages, and to discover how or in what way this can be useful or how we can use it in the future. Because of their inherent sequential nature, stochastic optimization and inference algorithms have been found to be effective for large-scale machine learning. On the other hand, Map Reduce-based algorithms were discovered to suffer from the curse of the last reducer, in which slaves must wait for the slowest processor to finish before moving on to the next computational iteration. The acronym Nomad stands for non-locking stochastic multi machine framework for asynchronous and decentralized computation. In this paper, we will attempt to demonstrate how many modern machine-learning problems have a double separability property, which means that the objective function decomposes into a sum over two different variables. The framework is then demonstrated by using two concrete problems: "matrix completion" for recommended systems.

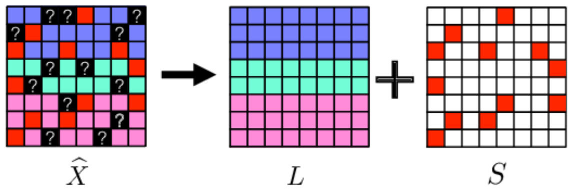
2. NOMadic Computing

Nomadic computing is an environment or information environment which is a heterogeneous assemblage of interconnected technological, social, and organizational elements that enable the physical and social mobility of computing and communication services between organizational actors both within and across organizational borders [1], [2]

2.1. Matrix Completion

The task of filling in the missing entries of a partially observed matrix. A wide range of datasets are naturally organized in matrix form".

We can understand the concept of matrix completion by a diagram:



The figure above is showing that a X matrix is recovering a low matrix L which is corrupted by both noise and outliers (as shown in illusion). [3]

In applications such as recommender systems, gene– disease interactions in bioinformatics, and link predictions in social-network analysis, biochemical or biophysical signal coming from human nervous system These inputs are taken by specialized sensors input devices, image capturing devices or scanning devices After taking inputs from these sensors, the physical data and instructions collected is then filtered and decoded using appropriate methods and finally are processed to get the desired output results. It is observed interactions between two different types of entities. For example, when users are interacting with pictures or movies, these interactions may be implicit the user who watched the movie and picture or explicit to the user reviewed and rated the movie and picture over any social network and then observed the interactions, there must infer the unobserved interactions which is a big challenge and has great practical significance because it often underlies the systems that e-commerce websites use to recommend ads, products, news articles, and movies to users as per their search history .[4].

The problem can be expressed mathematically as follows: Let A ∈ R^{PxQ} be an interaction matrix, where p represents the number of users and Q represents the number of items, and P > Q. Let B denote the observed entries of A, that is, (i,j). B implies that the interaction between user I and item j has an Aij value.The purpose for this is to accurately predict the unobserved entries of A. For easiness, it is defined that Bi as the set of interactions observed for the ith user, that is, Ai: = {j: (i,j) ∈ B}. Analogously, B j := {i: (i,j) ∈ B} is the set of users who have interacted with item j. Also, let a_i^T denote the ith row of A. As there is One popular model for matrix completion finds matrices W ∈ R^{PxK} and H ∈ R^{QxK} with k ≪ min(P, Q), such that A ≈ w_H^T One way to understand this model is to think of each row wiT ∈ Rk of W as a k-dimensional embedding of the user. And, each row $\boldsymbol{h_j^T}$ ∈ Rk of H is an embedding of the item in the same k-dimensional space. To predict the (i,j)th entry of A, it'll simply use ⟨wi,hj⟩, where ⟨·,·⟩ denotes the Euclidean inner product of two vectors. This model will measure goodness that how fit it is measured by a loss function, typically given by 1/2(Aij – (Gi,hj) }2. Furthermore, it is a need to enforce regularization to prevent the overfitting issue and to properly predict the unknown A entries [4][5]. So a regularize will be as :

$$\left\{ \frac{\lambda}{2} \Sigma_{i-1}^{p} + |B| - \left| \left| Wj \right| \right| + \frac{\lambda}{2} \Sigma_{j-1}^{Q} \left| B \right| . ||Hj|| \right\}$$

In the equation given above $\boldsymbol{\lambda} > 0$ is a tunable parameter and | . | is donating cardinality of a set and ||H|| is norm of $\boldsymbol{\lambda}$ vector if they all are put together this will give

$$\text{Min J (G, H)} = \frac{1}{2} \left\{ \sum_{(i\ ,j)\ e\ B} Ai,j - (Wi\ \ hj))\text{^}2 + \ regularzation\ on\ G\ and\ H \right\}$$

2.2. Stochastic optimization

In recent years, significant progress has been made in the field of nomadic computing, with one of the best examples being stochastic optimization on data to find the exact unserved data, using a technique known as Stochastic gradient descent (SGD) According to the above findings, the best and simplest technique will replace J(G,H) by the instantaneous approximation.

$$\text{J (G , H)} = \frac{1}{2} \{ \ (\ \textstyle\sum_{i,j\ \ k\ \ ,=B}^{p} \ \text{(Aij - (Gi , hj))2} + \lambda \text{ (||Gi||B + || hj||B) }\}$$

From the equation given above it can b say as an input and after this input the gradient of this objective function can simply be calculated as

△Gi J(G,H) = - (Aij – Gi , hj))hj+ LGi

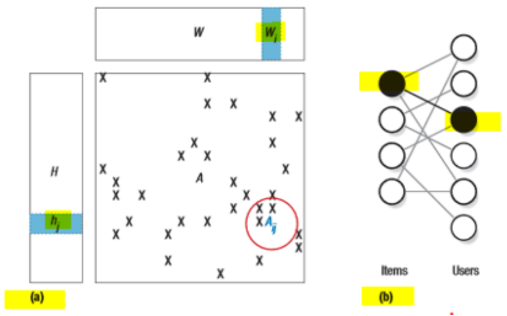
And

△hj J(G,H) = - (Aij – Gi , hj))Gj+ Lhi

This will be used to update the parameters.

Such systems, which are based on the Nomad algorithm and framework, are becoming increasingly popular these days. [4], [1]

It is found that SGD updates 1 and 2 only which require to read Wi,hj, and Aij for some (i,j) ∈ B, and to update Wi and hj (as in Figure 2). As a result, it can perform multiple SGD updates in parallel at the same time without any issue. These updates will not conflict with one another as long as they are not reading or writing the same Wi and hj values. The fundamental forms of NOMAD can be seen in this way. It is referring to a worker, which is a parallel computing unit .



Observations for stochastic optimization of the matrix-completion objective function are shown in the graph above. (a) Changing the parameters Gi and hj necessitates access to Gi,hj, and Aij while investigating function (b) A graphical representation of the same access pattern. The color black indicates that the node value is being updated, gray indicates that the node value is being read, and white indicates that the nodes are not being updated or read. [5], [4], [6], [1].

A worker can be a thread in a shared memory system, but a machine in a distributed memory architecture. This abstraction will allow NOMAD to be presented in a unified manner. NOMAD can thus be used in a hybrid setting where multiple threads are distributed across multiple machines .

Looking further the users {1, …, m} are split into p disjoint sets S1, S2, …, Sp, SQ which are of approximately equal in size. In simple words it is an alternative strategy to split the users in such a way that each set has approximately the same number of ratings. This will include a partition of the rows of the ratings matrix A . [4],[7], [8].

It has also been discovered that once data has been partitioned and distributed to workers, it is never moved during the algorithm's execution.[4] In Matrix completion, there are two types of parameters: Wi user parameters and hj item parameters. In NOMAD, wi parameters are divided according to S1, S2, …, Sp, SQ, that is, the Qth worker stores and updates Wi for i ∈ Sq. The variables in We are divided at the start, and will never move across workers during the algorithm's execution. On the other hand, hj parameters are initially randomly divided into P partitions, and their ownership changes as the algorithm progresses. The hj variables are nomadic: at each time point, only one hj variable exists in one worker, and it moves to another worker after it is processed, regardless of other item variables. Because of the symmetry in the matrix-completion problem's formulation, one can also make Wi nomadic and partition hj. Because the number of users is usually much greater than the number of items, more communication occurs, and hj variables can be made nomadic as shown in the figure above .

3. Performed Analysis(Related work)

Two analysis done during the research to demonstrate NOMAD's performance.

3.1. Analysis on Shared Memory

Some experiments on shared memory predicted NOMAD against FPSGD, which outperformed DSGD in single machine experiments as well as CCD++.

3.2. Analysis on Distributed Memory

On the second distributed memory experiment, NOMAD was compared to DSGD,7 DSGD++,8, and CCD++. 9 DSGD and CCD++ are synchronous algorithms, while DSGD++ and FPSGD are asynchronous SGD variants. The analysis was carried out using three benchmark datasets: Netflix, Yahoo! Music, and YouTube (mentioned in table 1). The same analysis was performed on dataset partition for all algorithms in each experiment. Because the goal here was to compare optimization algorithms, very little parameter tuning was done. For example, by employing the same regularization parameter λ across all datasets [9]. The dimension of the latent space is set to 100 i.e. K by default. Which started all algorithms with the same parameters: each W and H entry was set by sampling a uniformly random variable in the range independently.

By comparing the results in terms of root mean square error (RMSE) on the test set, defined as :[4]

$$\sqrt{\frac{\sum_{(i,k\ \ ,k.\ \ \ B\ \ test)}\left(A-(Wi,hj)\right)^{2}}{|Btest|}}$$

However, when each of the experiment performed following things observed.

Note: "The NOMAD algorithm. Each x denotes an observed entry in the interaction matrix A. The ownership of data and variables is shown by different colors. Small rectangles in the middle denote each machine's active regions."

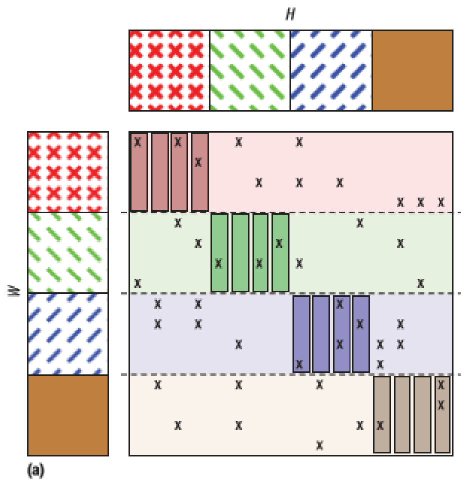


Figure a

Figure (a) is showing Initial assignment of matrices W and H. Each worker processes only the diagonal active area.[4][9]

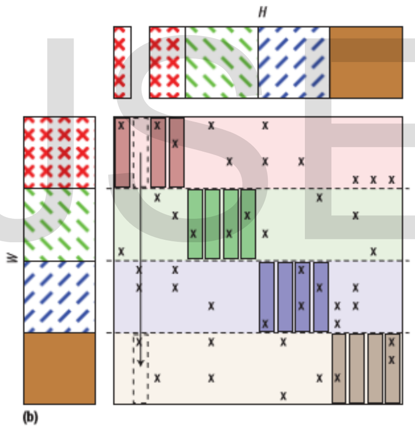


Figure b

Figure (b) is showing that once a worker finishes processing column j, it will send the corresponding item parameter hj to another worker. Here, h2 is sent from worker 1 to worker 4.[9][4]

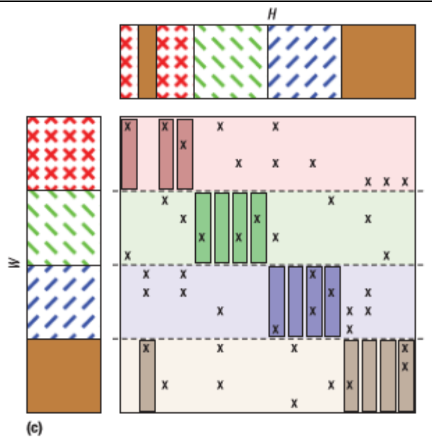


Figure (c) is showing that upon receipt, the column processed by the new worker. Here, worker 4 can now process column 3 because it owns the column. [4][1].

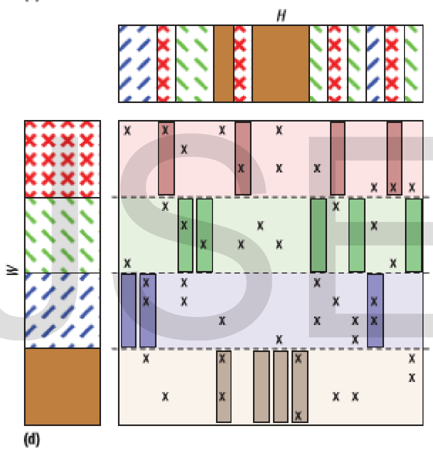


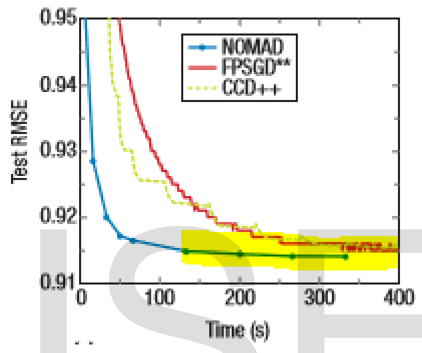
Figure (d) is showing that during the algorithm's execution, ownership of the hj item parameters can be change and it changed as shown in fugure.[4][9][1]

3.3. Shared Memory Analysis

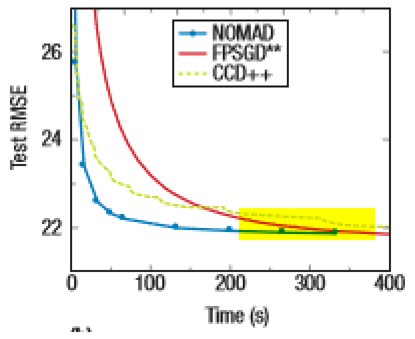
NOMAD, FPSGD**, and CCD++ were compared on a single machine with 30 computational cores using stochastic gradient descent on three main datasets.

Table No 1

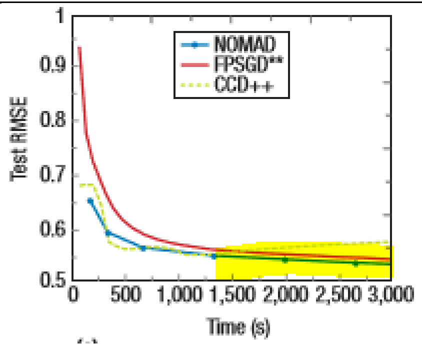
Dataset Name	Rows	Columns	Non Zeros
YouTube	23,60,784	3272	772448
Yahoo Music	1652090	514241	252600
Netflix	3659633	27780	1016646



ROC for YouTube (λ = 1.00, k = 100)



ROC for Yahoo music (λ = 0.05, k = 100)

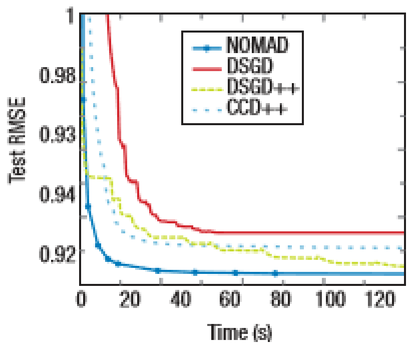


ROC for Netflix (λ = 0.01, k = 100). RMSE: root mean square error.[10]

In the Shared memory experiment (see above ROC curves), the number of cores is fixed to 30 and the performance of NOMAD with CCD++ and FPSGD** is compared because the current FPSGD** implementation in LibMF only reports CPU execution time, which is divided by the number of threads and used as a proxy for wall-clock time. On Netflix (table.1, d), NOMAD not only converged to a slightly better quality solution than the other methods (NOMAD RMSE was 0.914, versus 0.916 for FPSGD** and CCD++), but it also rapidly reduced the RMSE from the start. Yahoo! Music (Mentioned in ROC Yahoo Music), NOMAD converged to a slightly worse solution than FPSGD** (RMSE 21.894 versus 21.853, respectively), but the initial convergence was faster when compared to Netflix. The difference in RMSE was smaller on YouTube (Figure YouTube ROC), but NOMAD still outperformed the other two methods. CCD++'s initial speed on YouTube was comparable to NOMAD, but its quality in terms of test RMSE began to decrease after 1500 due to heavy load, with user behavior playing a role.[3],[10],[4],[11][12],[13], [14],[4],[15], [16], [17],[15].

4. Distributed Memory Analysis

This was a comparison of Nomad, DSGD, DSGD++, and CCD++ on a high-performance computing cluster using four computing threads and two communication threads per machine for three datasets Findings are below:



In the above mentioned figure the observation for the above mentioned datasets were as:

- Netflix (machines = 32, λ = 0.05, k = 100
- Yahoo Music (machines = 32, λ = 1.00, k = 100)
- YouTube (machines = 64, λ = 0.01, k = 100)

For the distributed memory experiment (above-mentioned ROC), four computation threads are used per machine, with 32 fixed machines and 64 for YouTube, and the performance of NOMAD is compared to DSGD, DSGD++, and CCD++. It was discovered that NOMAD converged much faster than its competitors on Netflix and YouTube. Not only was its initial convergence faster, but it also found a higher quality solution. On Yahoo Music, all four methods performed similarly because the cost of network communication relative to the size of the data is much higher and more expensive. While Netflix and YouTube have 5,575 and 68,635 nonzero ratings per item, respectively, Yahoo Music has only 404 ratings per uploaded item. It was discovered that when Yahoo Music was divided equally among 32 machines, each item only received an average of 10 ratings per machine. [4] As a result, the cost of sending and receiving item parameter vector hj for one item j across the network was greater than the cost of performing SGD updates on the ratings of the items that were locally stored inside the machine. As a result, the cost of network communication dominated the overall execution time of the algorithms, and there was little difference in their convergence speeds. [14],[4],[15], [16], [17],[15].

5 Conclusion

It has been concluded that this novel NOMAD framework can tackle matrix completion and inference in LAD(latent direct allocation). This framework is currently not fault tolerant in case if one worker fails, there is no way to recover which is definitely a drawback of this framework. There is a need of active working to extend NOMAD to other machine-learning problems, in case to obtain encouraging results for training support vector machines and logistic regression and relational databases. There is a need to find a way to minimize nomadic variables movement based on the data distribution. This frame work will not reliable for short lived battery systems because of limited resources and databases.

References:

1. Ma, C., Wang, K., Chi, Y., & Chen, Y. (2019). Implicit regularization in nonconvex statistical estimation: Gradient descent converges linearly for phase retrieval, matrix completion, and blind deconvolution. *Foundations of Computational Mathematics*, 1–182.
2. Zhuang, Y., Chin, W.-S., Juan, Y.-C., & Lin, C.-J. (2013). A fast parallel SGD for matrix factorization in shared memory systems. *Proceedings of the 7th ACM Conference on Recommender Systems*, 249–256.
3. Yusuke, D. O. I. (n.d.). *An Analysis of the Name Cache E ect on the Nomadic Computing*.
4. Yu, H.-F., Hsieh, C.-J., Yun, H., Vishwanathan, S. V. N., & Dhillon, I. (2016). Nomadic Computing for Big Data Analytics. *Computer*, *49*(4), 52–60.
5. Sallinen, S., Satish, N., Smelyanskiy, M., Sury, S. S., & Ré, C. (2016). High performance parallel stochastic gradient descent in shared memory. *2016 IEEE International Parallel and Distributed Processing Symposium (IPDPS)*, 873–882. IEEE.
6. Ribeiro, M. H., Ottoni, R., West, R., Almeida, V. A. F., & Meira Jr, W. (2020). Auditing radicalization pathways on youtube. *Proceedings of the 2020 Conference on Fairness, Accountability, and Transparency*, 131–141.
7. Papadamou, K., Papasavva, A., Zannettou, S., Blackburn, J., Kourtellis, N., Leontiadis, I., … Sirivianos, M. (2020). Disturbed YouTube for kids: Characterizing and detecting inappropriate videos targeting young children. *Proceedings of the International AAAI Conference on Web and Social Media*, *14*, 522–533.
8. Ma, C., Wang, K., Chi, Y., & Chen, Y. (2019). Implicit regularization in nonconvex statistical estimation: Gradient descent converges linearly for phase retrieval, matrix completion, and blind deconvolution. *Foundations of Computational Mathematics*, 1–182.
9. Lyytinen, K., & Yoo, Y. (2002). Research commentary: the next wave of nomadic computing. *Information Systems Research*, *13*(4), 377–388.
10. Lian, X., Huang, Y., Li, Y., & Liu, J. (2015). Asynchronous parallel stochastic gradient for nonconvex optimization. *Advances in Neural Information Processing Systems*, 2737–2745.
11. Lee, D., Oh, J., & Yu, H. (2020). OCam: Out-of-core coordinate descent algorithm for matrix completion. *Information Sciences*, *514*, 587–604.
12. KAMRAN, M., SHAHID, M. I., JAHNGEER, M. N., BAIG, H. K., KHAN, N., & MUKHLIS, M. (n.d.). *NETFLIX MOVIES RECOMMENDER SYSTEMS USING MATRIX FACTORIZATION TECHNIQUES*.
13. Guo, R., Zhang, F., Wang, L., Zhang, W., Lei, X., Ranjan, R., & Zomaya, A. (2020). BaPa: A Novel Approach of Improving Load Balance in Parallel Matrix Factorization for Recommender Systems. *IEEE Transactions on Computers*.
14. Gemulla, R., Nijkamp, E., Haas, P. J., & Sismanis, Y. (2011). Large-scale matrix factorization with distributed stochastic gradient descent. *Proceedings of the 17th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, 69–77.
15. Feng, J., Xu, H., & Yan, S. (2013). Online robust PCA via stochastic optimization. *Advances in Neural Information Processing Systems*, 404–412.
16. Cheng, Y.-L., Wang, Y.-J., Kao, W.-Y., Chen, P.-H., Huo, T.-I., Huang, Y.-H., … Lin, H.-C. (2013). Inverse association between hepatitis B virus infection and fatty liver disease: a large-scale study in populations seeking for check-up. *PloS One*, *8*(8).
17. Bishop, S. (2020). Algorithmic experts: Selling algorithmic lore on Youtube. *Social Media+ Society*, *6*(1), 2056305119897323.

18. Biccari, U., Navarro-Quiles, A., & Zuazua, E. (2020). Stochastic optimization methods for the simultaneous control of parameter-dependent systems. *ArXiv Preprint ArXiv:2005.04116*.