METHOD: A General Optimization Framework for Fast, Scalable Machine Learning

Abhimanu Kumar[*]
School of Computer Science
Carnegie Mellon University
Pittsburgh, PA
abhimank@cs.cmu.edu

Alex Beutel[*]
School of Computer Science
Carnegie Mellon University
Pittsburgh, PA
abeutel@cs.cmu.edu

Qirong Ho
School of Computer Science
Carnegie Mellon University
Pittsburgh, PA
qho@cs.cmu.edu

Eric P. Xing
School of Computer Science
Carnegie Mellon University
Pittsburgh, PA
epxing@cs.cmu.edu

ABSTRACT

How can Facebook scalably model their billion-node social network? How can Twitter efficiently run a toipc model over billions of tweets? "Big data" today does not just mean more data to mine but also more things to understand - more users, more documents, and more images. Thus, modern machine learning and data mining faces the challenge of scaling our classic models to both use all of the data available as well as build larger and more complex models of our world.

In this paper we describe METHOD, a general framework for performing fast machine learning on huge models and big data. Our system uses a combination of recent insights in stochastic learning theory to enable our system innovations. In particular, our system offers three contributions: **(1) Scalability:** Our system distributes both the data and model across the cluster of machines, allowing the system to scale to terrabytes of data as well as models with billions of parameters, beyond the size that can fit in memory for each machine. **(2) Versatility:** Our system supports synchronized parameter updates, such as normalizing across the cluster. As a result, we can fit a variety of machine learning models, including topic models, dictionary learning, and mixed membership stochastic block models. **(3) Speed:** We use a modification of stochastic gradient descent, called Always-On SGD, to continously improve our model fit, even if their are slow workers that would otherwise obstruct model synchronization and slow computation. Finally, we demonstrate that with our new approah, METHOD can fit ML models at an unprecedented scale and faster than prior work.

[*]The first two authors contributed equally to this work.

1. INTRODUCTION

alex-comment: **NOTE: The paper outline is still *very* rough, and I expect to re-order many sections, largely in attempt to keep the focus on the system.**

2. MODEL AND DATA ABSTRACTION

In designing any large scale machine learning system, one of the most important design decisions is the abstraction for the data and problem model. Recent distributed ML systems focus primarily on the abstraction for the data, such as GraphLab's [?] view of each data point as a node. Here we take a different approach - partitioning our data and the model we are building of the data simultaneously. As we will go on to demonstrate, focusing on both is advantageous for both scalability and speed.

2.1 Block structure in relational data

For many classic machine learning problems, we have relational data between two sets of items, and we are asking questions about these items. To be a bit more concrete, consider the classic topic modeling question: given a large set of documents, with what probability is each document in a given set of topics, and with what probability does a word represent a given topic? In this case, a data point is the number of times a given word was used in a given document. Because of the intrinsic relational nature of the data (here between words and documents), partitioning our data intelligently results in a clean partitioning of our model of topics for the words and documents.

More specifically, for any given subset of the documents and subset of the words, there is a unique set of data points that are applicable to items from both sets. If we view our data as a very sparse matrix, partitioning the documents and the words results in the data matrix also being partitioned into blocks, as seen in Figure **??**. Under this partitioning, all data points in block $\mathcal{B}_{i,j}$ describe a relationship between a word from set $\mathcal{S}_i^{(1)}$ and document from set $\mathcal{S}_j^{(2)}$. We focus our computation around these blocks of data.

An interesting property of these blocks is that for some blocks, such as $\mathcal{B}_{i,j}$ and $\mathcal{B}_{i',j'}$ where $i \neq i'$ and $j \neq j'$, we see that the blocks are *independent*. That is, a data point from $\mathcal{B}_{i,j}$ does not describe any relations to words in $\mathcal{S}_{i'}^{(1)}$ or documents in $\mathcal{S}_{j'}^{(2)}$ and vice versa for data in $\mathcal{B}_{i',j'}$. As has been shown in previous stochastic learning literature [**?**] and used in simpler data mining problems [**?**, **?**], we can now process

2.2 Distributed Parallel Processing

3. LEARNING

3.1 Sampling based Learning

3.2 Gradient based learning

3.3 Stochastic learning

4. ANSWERING COMPLEX QUESTIONS

5. RUNNING OVER BARRIERS

6. IMPLEMENTATION

How do we implement this in Hadoop

7. APPLICATIONS

7.1 Latent dirichlet allocation (LDA)

7.2 Dictionary Learning

7.3 Mixed membership stochastic block models

8. EXPERIMENTS

We compare our METHOD to PSGD (data partition), DSGD+ (sync barrier) and GraphLab over scalability, speed of convergence and convergence quality. METHOD performs best among all four approaches discussed over all three application sets and evaluation criteria.

9. RELATED WORK

10. REFERENCES

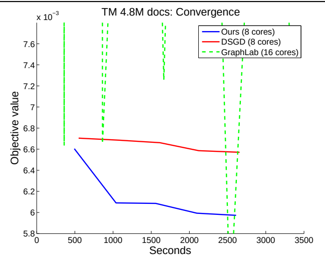
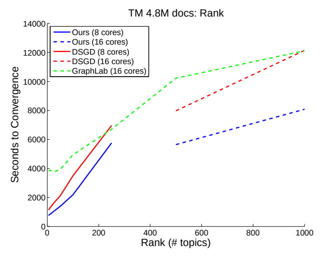
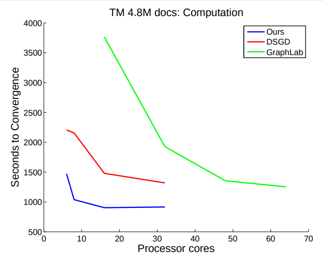
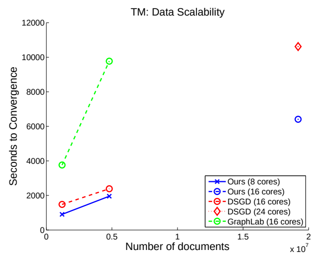
Topic Modeling			
Convergence Plots	# of Topics	# of Processors	# of Docs
			
Dictionary Learning			
Convergence Plots	# of Dictionary Bases	# of Processors	# of Images
TODO			
Mixed Membership Network Decomposition			
Convergence Plots	# of Network Roles	# of Processors	# of Network Nodes
TODO			

Figure 1: Convergence (Left) and scalability (in rank, processor cores and data size) of all methods, on topic modeling, dictionary learning and mixed-membership network decomposition. The convergence plot reveals the solution trajectory of each method, revealing pathological behavior such as oscillation. The scalability plots show how each method fares as the problem rank, number of processor cores, and data size is increased.