UNIVERSITY OF CALIFORNIA
SANTA CRUZ

HYBRID-PARALLEL PARAMETER ESTIMATION FOR FREQUENTIST AND BAYESIAN MODELS

A dissertation submitted in partial satisfaction of the
requirements for the degree of

DOCTOR OF PHILOSOPHY

in

COMPUTER SCIENCE

by

Parameswaran Raman

March 2020

The Dissertation of Parameswaran Raman
is approved:

Professor Manfred K. Warmuth, Chair

Professor David P. Helmbold

Professor S.V.N. Vishwanathan

Quentin Williams
Acting Vice Provost and Dean of Graduate Studies

Table of Contents

List of Figures

List of Tables

Abstract

Hybrid-Parallel Parameter Estimation for Frequentist and Bayesian Models

by

Parameswaran Raman

Distributed algorithms in machine learning follow two main flavors: horizontal partitioning, where the data is distributed across multiple slaves and vertical partitioning, where the model parameters are partitioned across multiple machines. The main drawback of the former strategy is that the model parameters need to be replicated on every machine. This is problematic when the number of parameters is very large, and hence cannot fit in a single machine. This drawback of the latter strategy is that the data needs to be replicated on each machine, thus failing to scale to massive datasets.

The goal of this thesis is to achieve the best of both worlds by partitioning both - the data as well as the model parameters, thus enabling the training of more sophisticated models on massive datasets. In order to do so, we exploit a structure that is observed in several machine learning models, which we term as *Double-Separability*. Double-Separability basically means that the objective function of the model can be decomposed into independent sub-functions which can be computed independently. For distributed machine learning, this implies that both data and model parameters can partitioned across machines and stochastic updates for parameters can be carried out independently and without any locking. Furthermore, double-separability naturally lends itself to developing efficient asynchronous

algorithms which enable computation and communication to happen in parallel, offering further speedup.

Some machine learning models such as Matrix Factorization directly exhibit double-separability in their objective function, however the majority of models do not. My work explores techniques to reformulate the objective function of such models to cast them into double-separable form. Often this involves introducing additional auxiliary variables that have nice interpretations. In this direction, I have developed Hybrid Parallel algorithms for machine learning tasks that include *Learning to Rank and Latent Collaborative Retrieval*, *Multinomial Logistic Regression*, *Variational Inference for Mixture of Exponential Families* and *Factorization Machines*. The software resulting from this work are available for public use under an open-source license.

To my parents,

for their eternal love and support.

Acknowledgments

TODO

Chapter 1

Introduction

A wide variety of machine learning tasks can be posed as a *regularized risk-minimization* problem. That is, one would like to solve,

$$\min_{\theta} \mathcal{L}(\theta) \coloneqq \lambda \mathcal{R}(\theta) + \mathcal{F}_X^{\text{emp}}(\theta) \tag{1.1}$$

where, $\mathcal{F}_X^{\text{emp}}(\theta)$ is called the empirical data likelihood and can be often decomposed into a finite summation over the observed data X. $\mathcal{R}(\theta)$ is a regularizer that controls the complexity of the model parameters θ. λ is a hyper-parameter used to trade-off between the two terms. For ease of optimization, $\mathcal{R}(\cdot)$ is assumed to be a smooth, convex function such as $\|\theta\|_2^2$. The regularized risk-minimization framework can be easily used to generalize popular tasks in large-scale machine learning applications such as,

- *Multinomial Logistic Regression (MLR)*: We are given N data points $(\mathbf{x}_i, y_i)_{i=1,\ldots,N}$, where each data point $\mathbf{x}_i$ is a D-dimensional feature vector and y_i is the corresponding label which can take values in $\{1, \ldots, K\}$. K denotes the total number of classes. For this model, the objective function is equivalent to minimizing the

negative log probability of the target labels y_i being assigned class k, and this decomposes into $\mathcal{F}_X^{\text{emp}}(\mathbf{W}) = -\frac{1}{N}\sum_{i=1}^{N}\sum_{k=1}^{K} y_{ik}\mathbf{w}_k^T\mathbf{x}_i + \frac{1}{N}\sum_{i=1}^{N}\log\left(\sum_{k=1}^{K}\exp\left(\mathbf{w}_k^T\mathbf{x}_i\right)\right)$ and $\mathcal{R}(\mathbf{W}) = \frac{1}{2}\sum_{k=1}^{K}\|\mathbf{w}_k\|_2^2$, where $\mathbf{W} \in \mathbb{R}^{D\times K}$ are the model parameters.

- *Learning to Rank (LTR)*: For this problem, we assume a set of users $\mathcal{X}$ and a set of items $\mathcal{Y}$. The observed data comprises of the joint features for the users and items $\phi(\mathbf{x}, \mathbf{y}) \in \mathbb{R}^D$. For a small subset of user-item pairs, we observe the ratings r_{xy} e.g. on a scale of 1-5. The goal is to learn a model $\mathbf{w} \in \mathbb{R}^D$ which can rank the items $\mathbf{y} \in \mathcal{Y}$ in the order of relevance for an unknown user $\mathbf{x} \in \mathcal{X}$. We also denote $f_{\mathbf{w}}(\mathbf{x}, \mathbf{y}) = \langle \phi(\mathbf{x}, \mathbf{y}), \mathbf{w}\rangle$ as the score the model assigns to item $\mathbf{y}$ for user $\mathbf{x}$. If we assume a pair-wise loss for the ranking model which uses the logistic loss $\sigma(z) = \log(1 + \exp(-z))$ as the inner surrogate loss, then $\mathcal{F}_X^{\text{emp}}(\mathbf{w}) = \sum_{\mathbf{x}\in\mathcal{X}}\sum_{\mathbf{y}\in\mathcal{Y}} r_{xy}\sum_{\mathbf{y}'\in\mathcal{Y},\mathbf{y}'\neq\mathbf{y}}\sigma(f_{\mathbf{w}}(\mathbf{x}, \mathbf{y}) - f_{\mathbf{w}}(\mathbf{x}, \mathbf{y}'))$. The squared l-2 norm regularizer is given by $\mathcal{R}(\mathbf{w}) = \frac{1}{2}\|\mathbf{w}\|_2^2$.

- *Latent Collaborative Retrieval (LCR)*: In this setting, we aim to learn a score function $f(x, y)$ between a user x and item y without an explicit feature vector $\phi(\mathbf{x}, \mathbf{y})$, by embedding each user x and item y into a low-dimensional euclidean latent space. Observed data consists of ratings r_{xy} observed for user x and item y pairs. These ratings could be either explicit (scale of 1-5) or implicit (clicks, interaction). The score function is then defined as $f(x, y) = \langle \mathbf{U}_x, \mathbf{V}_y\rangle$, where $\mathbf{U}_x$ and $\mathbf{V}_y$ are the latent representations of the user x and item y respectively. A similar pairwise ranking loss can be defined as in the case of learning to rank. Thus, $\mathcal{F}_X^{\text{emp}}(\mathbf{U}, \mathbf{V}) = \sum_{x\in\mathcal{X}}\sum_{y\in\mathcal{Y}} r_{xy}\sum_{y'\in\mathcal{Y},y'\neq y}\sigma(f(x, y) - f(x, y'))$ and the regularizer is given by $\mathcal{R}(\mathbf{U}, \mathbf{V}) = \frac{1}{2}\left(\|\mathbf{U}\|_2^2 + \|\mathbf{V}\|_2^2\right)$. The model parameters to be learnt are $\mathbf{U} \in \mathbb{R}^{D\times K}$ and $\mathbf{V} \in \mathbb{R}^{D\times K}$.

- *Matrix Factorization (MF)*: We are given a matrix $\mathcal{A} \in \mathbb{R}^{N \times M}$ comprising of observed ratings r_{xy} for user x and item y pairs, where N is the number of users and M is the number of items. In practice, however, only a small subset of the entries in $\mathcal{A}$ are observed. We denote this set as $\Omega \subseteq \{1, \ldots, N\} \times \{1, \ldots, M\}$. For convenience, we also define Ω_i to be the set of items rated by the i-th user, and analogously $\bar{\Omega}_j$ as the set of users who have rated item j. The goal here is to predict accurately the unobserved ratings. This is accomplished by learning low-rank matrices $\mathbf{U} \in \mathbb{R}^{N \times K}$ and $\mathbf{V} \in \mathbb{R}^{M \times K}$ such that $A \approx \mathbf{U}\mathbf{V}^T$. Expressing this as a least-squares minimization problem, we can observe that, $\mathcal{F}_X^{\text{emp}}(\mathbf{U}, \mathbf{V}) = \frac{1}{2} \sum_{i,j \in \Omega} (\mathcal{A}_{ij} - \langle \mathbf{U}_i, \mathbf{V}_j \rangle)^2$. Following standard practice, a weighted squared norm regularizer can be employed to regularize the model, i.e. $\mathcal{R}(\mathbf{U}, \mathbf{V}) = \frac{1}{2} \left(\sum_{i=1}^{N} |\Omega_i| \cdot \|\mathbf{U}_i\|_2^2 + \sum_{j=1}^{M} |\bar{\Omega}_j| \cdot \|\mathbf{V}_j\|_2^2 \right)$.

- *Factorization Machines (FM)*: Factorization Machines proposed by [Rendle, 2010] combine the advantages of SVMs and factorization models. The score function in FM is parameterized using, both a linear model $\mathbf{w} \in \mathbb{R}^D$ as well as a factorized model $\mathbf{V} \in \mathbb{R}^{D \times K}$. The factorized model $\mathbf{V}$ is used to capture all pairwise interactions between the D features. The observations consist of data matrix $\mathbf{X} \in \mathbb{R}^{N \times D*}$ and corresponding labels $y \in \mathbb{R}^N$. $D*$ denotes the total number of features that include the linear interactions as well as pairwise interactions, i.e. $D* = \mathcal{O}\left(D + \frac{D^2}{2}\right)$. The score function for an observation $\mathbf{x}$ is given by $f(\mathbf{x}) = w_0 + \langle \mathbf{w}, \mathbf{x} \rangle + \sum_{j=1}^{D} \sum_{j'=1}^{D} \langle \mathbf{V}_j, \mathbf{V}_{j'} \rangle \mathbf{x}_j \mathbf{x}_{j'}$. Using this, the empirical risk term can be expressed as, $\mathcal{F}_X^{\text{emp}}(\mathbf{w}, \mathbf{V}) = \frac{1}{2} \sum_{i=1}^{N} l(f(\mathbf{x}), y_i)$ and regularizer $\mathcal{R}(\mathbf{w}, \mathbf{V}) = \frac{1}{2} \left(\|\mathbf{w}\|_2^2 + \|\mathbf{V}\|_2^2 \right)$, where $l(\cdot)$ is an appropriate loss function such as *cross-entropy* for binary classification or *squared loss* for regression.

We summarize these tasks more concisely in Table 1.1, where for each task, we list the corresponding data matrix X, model parameters θ, empirical data likelihood term $\mathcal{F}(\theta)$ and the regularizer $\mathcal{R}(\theta)$.

1.1 Challenges in distributed machine learning

Traditional optimization methods for distributed machine learning broadly fall into two categories, namely *Data parallel* and *Model parallel*.

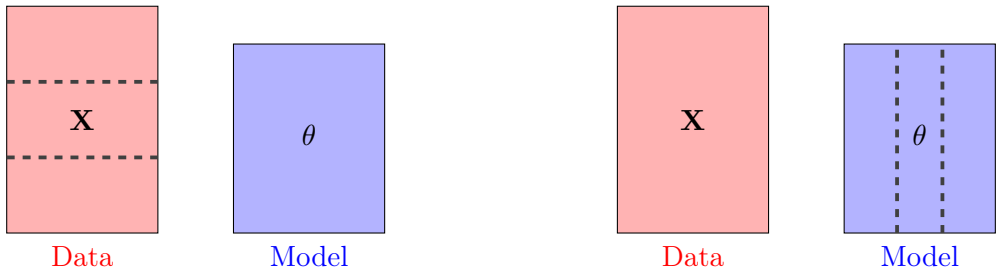
Data Parallel: The classic paradigm in distributed machine learning is to perform *data partitioning*, using, for instance, a map reduce style architecture. In other words, the data is distributed across multiple workers. At the beginning of each iteration, the master distributes a parameter vector to all the workers, who in turn use this to compute the objective function and gradient values on their part of the data and transmit it back to the master. The master aggregates the results from the workers and updates the parameters, and transmits the updates back to the workers, and the iteration proceeds. The L-BFGS optimization algorithm is used in the master to update the parameters after every iteration [Nocedal and Wright, 2006]. The main drawback of this strategy is that the model parameters need to be replicated on every machine. This is problematic when the number of classes, and consequently the number of parameters is very large, and hence cannot fit in a single machine.

Model Parallel: An orthogonal approach is to use *model partitioning*. Here, again, we use a master slave architecture but now the data is replicated across each slave. However, the model parameters are now partitioned and distributed to each machine. During each

Task	Data (X)	Model (θ)	Empirical Data Likelihood $\mathcal{F}_X^{\text{emp}}(\theta)$	Regularizer $\mathcal{R}(\theta)$
MLR	$X \in \mathbb{R}^{N \times D}$	$W \in \mathbb{R}^{D \times K}$	$-\frac{1}{N}\sum_{i=1}^{N}\sum_{k=1}^{K} y_{ik}\mathbf{w}_k^T\mathbf{x}_i + \frac{1}{N}\sum_{i=1}^{N}\log\left(\sum_{k=1}^{K}\exp\left(\mathbf{w}_k^T\mathbf{x}_i\right)\right)$	$\frac{1}{2}\sum_{k=1}^{K}\|\mathbf{w}_k\|_2^2$
LTR	$X \times Y \in \mathbb{R}^{N \times D}$	$\mathbf{w} \in \mathbb{R}^{D}$	$\sum_{\mathbf{x}\in\mathcal{X}}\sum_{\mathbf{y}\in\mathcal{Y}} r_{xy}\sum_{\mathbf{y}'\in\mathcal{Y},\mathbf{y}'\neq\mathbf{y}}\sigma\left(\langle\phi(\mathbf{x},\mathbf{y}),\mathbf{w}\rangle - \langle\phi(\mathbf{x},\mathbf{y}'),\mathbf{w}\rangle\right)$	$\frac{1}{2}\|\mathbf{w}\|_2^2$
LCR	$X \in \mathbb{R}^{N \times M}$	$U, V \in \mathbb{R}^{D \times K}$	$\sum_{x\in\mathcal{X}}\sum_{y\in\mathcal{Y}} r_{xy}\sum_{y'\in\mathcal{Y},y'\neq y}\sigma\left(\langle\mathbf{U}_x,\mathbf{V}_y\rangle - \langle\mathbf{U}_x,\mathbf{V}_{y'}\rangle\right)$	$\frac{1}{2}\left(\|\mathbf{U}\|_2^2 + \|\mathbf{V}\|_2^2\right)$
MF	$X \in \mathbb{R}^{N \times M}$	$U, V \in \mathbb{R}^{D \times K}$	$\frac{1}{2}\sum_{i,j\in\Omega}\left(\mathcal{A}_{ij} - \langle\mathbf{U}_i,\mathbf{V}_j\rangle\right)^2$	$\frac{1}{2}\left(\sum_{i=1}^{N}\lvert\Omega_i\rvert\cdot\|\mathbf{U}_i\|_2^2 + \sum_{j=1}^{M}\lvert\bar{\Omega}_j\rvert\cdot\|\mathbf{V}_j\|_2^2\right)$
FM	$X \in \mathbb{R}^{N \times D*}$	$\mathbf{w} \in \mathbb{R}^{D}, V \in \mathbb{R}^{D \times K}$	$\frac{1}{2}\sum_{i=1}^{N} l\left(f(\mathbf{x}), y_i\right)$	$\frac{1}{2}\left(\|\mathbf{w}\|_2^2 + \|\mathbf{V}\|_2^2\right)$

Table 1.1: Table to show how popular machine learning tasks - Multinomial Logistic Regression (MLR), Learning to Rank (LTR), Latent Collaborative Retrieval (LCR), Matrix Factorization (MF) and Factorization Machines (FM) fit into the regularized risk minimization framework. For each task, the table shows the corresponding data X and model θ matrices involved. Columns $\mathcal{F}_X^{\text{emp}}(\theta)$ and $\mathcal{R}(\theta)$ show the corresponding data likelihood and regularizer terms. N and D denote the number of examples and features respectively. K denotes the number of classes in case of MLR and the number of latent dimensions in case of factorization models such as MF, LCF and FM.

iteration the model parameters on the individual machines are updated, and some auxiliary variables are computed and distributed to the other workers, which use these variables in their parameter updates. See the Log-Concavity (LC) method Gopal and Yang [2013] for an example of such a strategy. The main drawback of this approach, however, is that the data needs to be replicated on each machine, and consequently it does not scale to massive datasets.



(a) **Data Parallelism** (partition data X, duplicate parameters θ).

(b) **Model Parallelism** (partition parameters θ, duplicate data X).

Figure 1.1: Two orthogonal approaches to distributing computation in machine learning.

Figure 1.1 illustrates these two approaches. In the context of Multinomial Logistic Regression (MLR), the data X is a $\mathcal{O}(N \times D)$ matrix and model θ is a $\mathcal{O}(D \times K)$ matrix, where N, D and K denote the number of examples, features and classes respectively. Interpretation of data X and model parameters θ for the other tasks can be found in Table 1.1.

Fundamental Bottleneck in Data-*only* or Model-*only* parallelism: Machine learning tasks of today run on *humongous amounts of data* involving *sophisticated models* [Weston et al., 2011], [Cheng et al., 2016], [Prabhu et al., 2018]. The memory requirements for

data and model parameters can easily exceed the capacity of a single machine in a commodity cluster. Figure 1.2 illustrates this fundamental challenge in the context of Multinomial Logistic Regression (MLR).

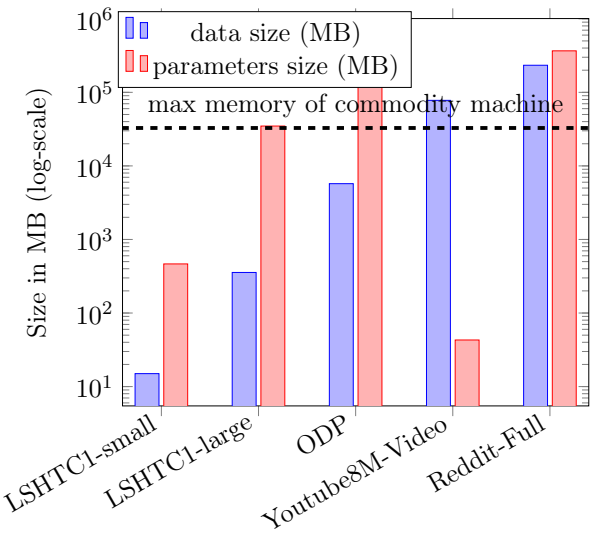


Figure 1.2: Data and Model requirements (MB) of real-world datasets for Multinomial Logistic Regression (MLR).

As depicted in the figure, real-world datasets exhibit varying storage requirements for the data and model. While the smaller datasets can be easily run on a single machine, larger datasets such as ODP and Reddit-Full are impossible to run on a commodity cluster with just traditional data or model parallel approaches. This is because ODP has a massive requirement of 355 GB for the model itself, while Reddit-Full dataset is even bigger, requiring 228 GB for the data and 358 GB for its model. For instance, if we were to run the MLR task on ODP dataset using data parallelism with a modest 4 units of parallelism (cores or threads) on a single machine, this would require a total memory footprint of 4×355 GB ≈ 1.38 TB on a single machine. Using a similar configuration to run MLR on the Reddit-Full dataset,

the per-machine memory footprint for data parallel methods is ≈ 1.39 TB and model parallel methods is ≈ 912 GB. Thus, the number of parallel units we can use on a single machine is heavily limited if we choose to replicate only *one of the data or model*. Inspired by these challenges, in this thesis we explore the following broad question,

> *How can we achieve the best of both worlds?*
>
> *i.e. Can we achieve both data as well as model parallelism simultaneously?*

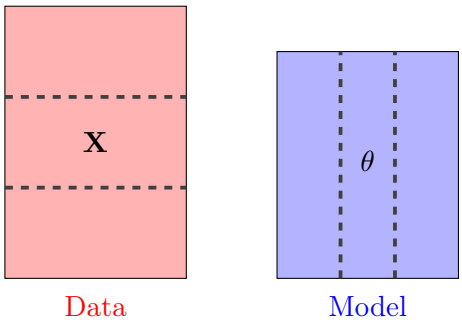


Figure 1.3: **Hybrid Parallelism** partitions both data and model parameters simultaneously without needing to duplicate either of them.

As an answer to this, we propose *Hybrid Parallelism*, which is the idea of partitioning both the data matrix X as well as the model parameters θ *simultaneously* across workers such that there is no overlap is computation among the workers. This enables us to scale to workloads with arbitrary data and model sizes.

1.2 Hybrid Parallelism

A natural question that arises is - *Can all machine learning models be made Hybrid Parallel?* To answer this question, we first introduce the notion of *Double Separability*. Double

Separability is an attractive property in the objective functions of machine learning models which makes them naturally amenable to Hybrid Parallelism.

1.2.1 Double Separability

The concept of Separability [Zhong et al., 2004] of functions is well-known in the optimization community [Tseng and Mangasarian, 2001]. Given a family of sets $\{\mathbb{S}_i\}_{i=1}^N$, a function $f : \prod_{i=1}^N \mathbb{S}_i \to \mathbb{R}$ is separable if there exist functions $f_i : \mathbb{S}_i \to \mathbb{R}$ for each $i = 1, 2, \ldots, N$ such that $f(\theta) = \sum_{i=1}^N f_i(\theta_i)$ where $\theta_i \in \mathbb{S}_i$. Extending the idea of separability [Zhong et al., 2004] of functions which is well-known in the optimization community [Tseng and Mangasarian, 2001], *Double-Separability* is formally defined as follows,

Definition 1. *Double Separability Let* $\{\mathbb{S}_i\}_{i=1}^m$ *and* $\{\mathbb{S}'_j\}_{j=1}^{m'}$ *be two families of sets of parameters. A function* $f : \prod_{i=1}^m \mathbb{S}_i \times \prod_{j=1}^{m'} \mathbb{S}'_j \to \mathbb{R}$ *is doubly separable if* $\exists\ f_{ij} : \mathbb{S}_i \times \mathbb{S}'_j \to \mathbb{R}$ *for each* $i = 1, 2, \ldots, m$ *and* $j = 1, 2, \ldots, m'$ *such that:*

$$f(\theta_1, \theta_2, \ldots, \theta_m, \theta'_1, \theta'_2, \ldots, \theta'_{m'}) = \sum_{i=1}^{m} \sum_{j=1}^{m'} f_{ij}(\theta_i, \theta'_j) \tag{1.2}$$

In simpler terms, if a function f in two set of parameters θ and θ' can be decomposed into a double-summation of sub-functions f_{ij} such that each sub-function f_{ij} depends on a pair of parameters (θ_i, θ'_j), one from each set, then the function f is said to be doubly-separable. One can think of these sub-functions f_{ij} as cells in a matrix of computations as illustrated in Figure 1.4. For each (i, j) in the diagonal blocks, f_{ij} can be computed *independently* and in *parallel* as it has no overlap in terms of access-patterns of parameters θ and θ'. One can think of the dimensions m and m' as the dimensions for data and model parallelism respectively. The values of m and m' differ depending on the machine learning

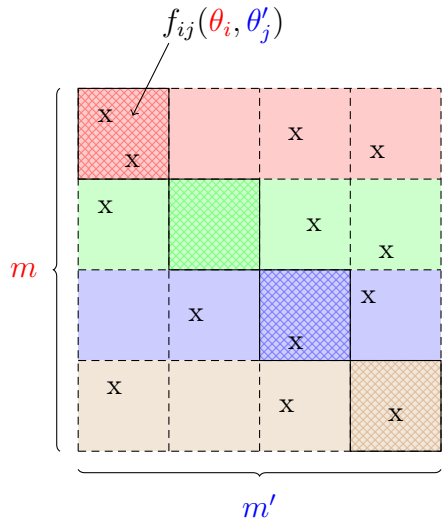


Figure 1.4: Diagram to illustrate the decomposability of doubly-separable function f. The highlighted diagonal blocks can be computed be computed independently and in parallel since they do not involve overlapping parameters θ and θ'.

task at hand.

Frequentist Models: In multinomial logistic regression (MLR) for instance, the data partitioning occurs across the examples N while the model partitioning occurs across the classes K. Thus, $m = N$ and $m' = K$. On the other hand, in factorization machines (FM), it is more reasonable to split the model across the features D. Therefore, $m = N$ and $m' = D$.

Bayesian Models: Bayesian Models can also benefit from Hybrid Parallelism. Mixture of exponential families model the observations as arising from a generative process as follows:

- Draw mixing proportions $Z_i \sim \Delta^K$, where Δ^K is a K-dimensional simplex.
- Draw observations $X_1, \ldots, X_N$ based on the mixing proportions,

$X_i|Z_i = k \sim$ ExpFamily (θ_k), where ExpFamily$(\cdot)$ refers to an exponential family distribution such as Gaussian or Multinomial distribution with parameter θ_k.

Examples of popular models that fall in this category include *Latent Dirichlet Allocation* (LDA) and *Gaussian Mixture Models* (GMM). While the data in these mixture models can be partitioned across N, the model can be partitioned across the K mixture components. Thus, $m = N$ and $m' = K$.

1.2.2 Achieving Double-Separability in objective functions

Objective functions of some machine learning models such as Matrix Factorization are *directly* doubly-separable [Yun et al., 2013]. Assuming the data matrix X is factorized into low-rank matrices $W \in \mathbb{R}^{N \times K}$ and $H \in \mathbb{R}^{M \times K}$, where N and M are the number of users and items respectively, we can express the objective function of matrix factorization as,

$$\mathcal{L}(\mathbf{w}_1, \mathbf{w}_2, \ldots, \mathbf{w}_N, \mathbf{h}_1, \mathbf{h}_2, \ldots, \mathbf{h}_M) = \frac{1}{2} \sum_{i=1}^{N} \sum_{j=1}^{M} (X_{ij} - \langle \mathbf{w}_i, \mathbf{h}_j \rangle)^2 \tag{1.3}$$

which is directly in the doubly-separable form given in eqn (1.2). This make it easy to make matrix factorization Hybrid Parallel. However, several learning tasks in machine learning are much more complicated and do not exhibit direct double-separability. They require *reformulating* the original objective function in order to be converted into a desirable form such as given in eqn (1.2).

The goal of this thesis is to study such reformulations for frequentist and bayesian models to make them hybrid parallel.

1.3 Overview of the chapters

We now sketch the contents of the main chapters of this dissertation (each chapter will have a separate more detailed introduction). The main results of Chapter 2, Chapter 3 and Chapter 4 have been published at NIPS 2014 [Yun et al., 2014a], KDD 2019 [Raman et al., 2019b] and AISTATS 2019 [Zhang et al., 2019] conferences respectively. Chapter 5 is based on the arXiV pre-print TODO. Chapters 2, 3 and 5 will appear together as a journal submission.

1.3.1 Chapter 2: Latent Collaborative Retrieval

We propose RoBiRank - a Learning to Rank algorithm inspired by Robust Binary Classification and show that it scales well on large-data. The main idea behind Robust Binary Classification is to use transformation on convex losses to help give up performance on hard to classify data points (outliers). *Firstly*, we observe that this is related to learning to rank, where we would not mind sacrificing accuracy at the bottom of the ranking list in order to gain performance at top of the list. We thus show that our ranking objective function can be viewed as a generalization of robust binary classification. *Secondly*, minimizing RoBiRank is equivalent to directly maximizing DCG (a popular evaluation metric for listwise learning to rank). As a result, RoBiRank performs really well at the top of the list. *Thirdly*, using a linearization trick on our loss allows us to obtain an unbiased stochastic gradient estimator so that our SGD optimizer becomes independent of the size of the dataset. In addition, our algorithm is parallelizable and can also be used to solve large-scale problems without any explicit features. Experimental results are shown on both medium and very large datasets.

1.3.2 Chapter 3: Multinomial Logistic Regression

We study the problem of scaling Multinomial Logistic Regression (MLR) to datasets with very large number of data points in the presence of large number of classes. At a scale where neither data nor the parameters are able to fit on a single machine, we argue that *simultaneous data and model parallelism (Hybrid Parallelism)* is inevitable. The key challenge in achieving such a form of parallelism in MLR is the log-partition function which needs to be computed *across all K classes* per data point, thus making model parallelism non-trivial.

To overcome this problem, we propose a reformulation of the original objective that exploits *double-separability*, an attractive property that naturally leads to hybrid parallelism. Our algorithm (DS-MLR) is *asynchronous and completely de-centralized*, requiring minimal communication across workers while keeping both data and parameter workloads partitioned. Unlike standard data parallel approaches, DS-MLR *avoids bulk-synchronization* by maintaining local normalization terms on each worker and accumulating them incrementally using a token-ring topology.

We demonstrate the versatility of DS-MLR under various scenarios in data and model parallelism, through an empirical study consisting of real-world datasets. In particular, to demonstrate scaling via hybrid parallelism, we created a new benchmark dataset (Reddit-Full) by pre-processing 1.7 billion reddit user comments spanning the period 2007-2015. We used DS-MLR to solve an extreme multi-class classification[1] problem of classifying 211 million data points into their corresponding subreddits. Reddit-Full is a massive data set with data occupying 228 GB and 44 billion parameters occupying 358 GB. To the best of

[1]Extreme classification is defined as multi-class / multi-label classification in the presence of very large number of examples and classes / labels.

our knowledge, no other existing methods can handle MLR in this setting.

1.3.3 Chapter 4: Mixture of Exponential Families

Mixture of exponential family models are among the most fundamental and widely used statistical models. Stochastic variational inference (SVI), the state-of-the-art algorithm for parameter estimation in such models is inherently serial. Moreover, it requires the parameters to fit in the memory of a single processor; this poses serious limitations on scalability when the number of parameters is in billions. In this work, we present extreme stochastic variational inference (ESVI), a distributed, asynchronous and lock-free algorithm to perform variational inference for mixture models on massive real world datasets. ESVI overcomes the limitations of SVI by requiring that each processor only access a subset of the data and a subset of the parameters, thus providing data and model parallelism simultaneously. Our empirical study demonstrates that ESVI not only outperforms VI and SVI in wallclock-time, but also achieves a better quality solution. To further speed up computation and save memory when fitting large number of topics, we propose a variant ESVI-TOPK which maintains only the top $k \in K$ important topics. Empirically, we found that using top 25% topics suffices to achieve the same accuracy as storing all the topics.

1.3.4 Chapter 5: Factorization Machines

Factorization Machines (FM) are powerful class of models that incorporate higher-order interaction among features to add more expressive power to linear models. They have been used successfully in several real-world tasks such as click-prediction, ranking and recommender systems. Despite using a low-rank representation for the pairwise features,

the memory overheads of using factorization machines on large-scale real-world datasets can be prohibitively high. For instance on the criteo tera dataset, assuming a modest 128 dimensional latent representation and 10^9 features, the memory requirement for the model is in the order of 1 TB. In addition, the data itself occupies 2.1 TB. Traditional algorithms for FM which work on a single-machine are not equipped to handle this scale and therefore, using a distributed algorithm to parallelize the computation across a cluster is inevitable. In this work, we propose a hybrid-parallel stochastic optimization algorithm DS-FACTO, which partitions both the data as well as parameters of the factorization machine simultaneously. Our solution is fully de-centralized and does not require the use of any parameter servers. We present empirical results to analyze the convergence behavior, predictive power and scalability of DS-FACTO.

Chapter 2

Latent Collaborative Retrieval

2.1 Introduction

Learning to rank (LTR) is a problem of ordering a set of items according to their relevances to a given context [Chapelle and Chang, 2011]. While a number of approaches have been proposed in the literature, in this chapter we provide a new perspective by showing a close connection between ranking and a seemingly unrelated topic in machine learning, namely, robust binary classification.

In robust classification [Huber, 1981], we are asked to learn a classifier in the presence of outliers. Standard models for classification such as Support Vector Machines (SVMs) and logistic regression do not perform well in this setting, since the convexity of their loss functions does not let them give up their performance on any of the data points [Long and Servedio, 2010]; for a classification model to be robust to outliers, it has to be capable of sacrificing its performance on some of the data points. We observe that this requirement is very similar to what standard metrics for ranking try to evaluate. Discounted Cumulative

Gain (DCG) [Manning et al., 2008] and its normalized version NDCG, popular metrics for learning to rank, strongly emphasize the performance of a ranking algorithm at the top of the list; therefore, a good ranking algorithm in terms of these metrics has to be able to give up its performance at the bottom of the list if that can improve its performance at the top.

In fact, we will show that DCG and NDCG can indeed be written as a natural generalization of robust loss functions for binary classification. Based on this observation we formulate RoBiRank, a novel model for ranking, which maximizes the lower bound of (N)DCG. Although the non-convexity seems unavoidable for the bound to be tight [Chapelle et al., 2008], our bound is based on the class of robust loss functions that are found to be empirically easier to optimize [Ding, 2013]. Indeed, our experimental results suggest that RoBiRank reliably converges to a solution that is competitive as compared to other representative algorithms even though its objective function is non-convex.

While standard deterministic optimization algorithms such as L-BFGS [Nocedal and Wright, 2006] can be used to estimate parameters of RoBiRank, to apply the model to large-scale datasets a more efficient parameter estimation algorithm is necessary. This is of particular interest in the context of latent collaborative retrieval [Weston et al., 2012]; unlike standard ranking task, here the number of items to rank is very large and explicit feature vectors and scores are not given.

Therefore, we develop an efficient parallel stochastic optimization algorithm for this problem. It has two very attractive characteristics: First, the time complexity of each stochastic update is independent of the size of the dataset. Also, when the algorithm is distributed across multiple number of machines, no interaction between machines is required

during most part of the execution; therefore, the algorithm enjoys near linear scaling. This is a significant advantage over serial algorithms, since it is very easy to deploy a large number of machines nowadays thanks to the popularity of cloud computing services, e.g. Amazon Web Services.

We apply our algorithm to latent collaborative retrieval task on Million Song Dataset [Bertin-Mahieux et al., 2011] which consists of 1,129,318 users, 386,133 songs, and 49,824,519 records; for this task, a ranking algorithm has to optimize an objective function that consists of $386,133 \times 49,824,519$ number of pairwise interactions. With the same amount of wall-clock time given to each algorithm, RoBiRank leverages parallel computing to outperform the state-of-the-art with a 100% lift on the evaluation metric.

2.2 Robust Binary Classification

Suppose we are given training data which consists of n data points $(x_1, y_1), (x_2, y_2), \ldots, (x_n, y_n)$, where each $x_i \in \mathbb{R}^d$ is a d-dimensional feature vector and $y_i \in \{-1, +1\}$ is a label associated with it. A linear model attempts to learn a d-dimensional parameter ω, and for a given feature vector x it predicts label $+1$ if $\langle x, \omega \rangle \geq 0$ and -1 otherwise. Here $\langle \cdot, \cdot \rangle$ denotes the Euclidean dot product between two vectors. The quality of ω can be measured by the number of mistakes it makes: $L(\omega) := \sum_{i=1}^{n} I(y_i \cdot \langle x_i, \omega \rangle < 0)$. The indicator function $I(\cdot < 0)$ is called the 0-1 loss function, because it has a value of 1 if the decision rule makes a mistake, and 0 otherwise. Unfortunately, since the 0-1 loss is a discrete function its minimization is difficult [Feldman et al., 2012]. The most popular solution to this problem in machine learning is to upper bound the 0-1 loss by an easy to optimize function [Bartlett et al., 2006].

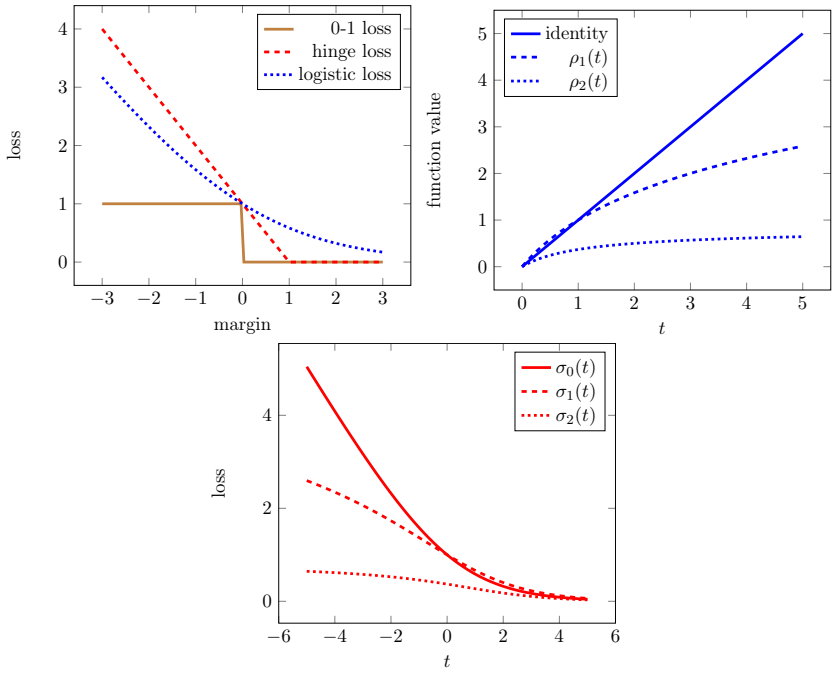


Figure 2.1: Left: Convex Upper Bounds for 0-1 Loss. Middle: Transformation functions for constructing robust losses. Right: Logistic loss and its transformed robust variants.

For example, logistic regression uses the logistic loss function $\sigma_0(t) := \log_2(1 + 2^{-t})$, to come up with a continuous and convex objective function

$$\overline{L}(\omega) := \sum_{i=1}^{n} \sigma_0(y_i \cdot \langle x_i, \omega \rangle), \tag{2.1}$$

which upper bounds $L(\omega)$. It is clear that for each i, $\sigma_0(y_i \cdot \langle x_i, \omega \rangle)$ is a convex function in ω; therefore, $\overline{L}(\omega)$, a sum of convex functions, is also a convex function which is relatively easier to optimize [Boyd and Vandenberghe, 2004]. Support Vector Machines (SVMs) on the other hand can be recovered by using the hinge loss to upper bound the 0-1 loss. Figure 2.1 (left) graphically illustrates three loss functions discussed here.

However, convex upper bounds such as $\overline{L}(\omega)$ are known to be sensitive to outliers

[Long and Servedio, 2010]. The basic intuition here is that when $y_i \cdot \langle x_i, \omega \rangle$ is a very large negative number for some data point i, $\sigma(y_i \cdot \langle x_i, \omega \rangle)$ is also very large, and therefore the optimal solution of (2.1) will try to decrease the loss on such outliers at the expense of its performance on "normal" data points.

In order to construct robust loss functions, consider the following two transformation functions:

$$\rho_1(t) := \log_2(t+1), \quad \rho_2(t) := 1 - \frac{1}{\log_2(t+2)}, \tag{2.2}$$

which, in turn, can be used to define the following loss functions:

$$\sigma_1(t) := \rho_1(\sigma_0(t)), \quad \sigma_2(t) := \rho_2(\sigma_0(t)). \tag{2.3}$$

One can see that $\sigma_1(t) \to \infty$ as $t \to -\infty$, but at a much slower rate than $\sigma_0(t)$ does; its derivative $\sigma_1'(t) \to 0$ as $t \to -\infty$. Therefore, $\sigma_1(\cdot)$ does not grow as rapidly as $\sigma_0(t)$ on hard-to-classify data points. Such loss functions are called Type-I robust loss functions by Ding [2013], who also showed that they enjoy statistical robustness properties. $\sigma_2(t)$ behaves even better: $\sigma_2(t)$ converges to a constant as $t \to -\infty$, and therefore "gives up" on hard to classify data points. Such loss functions are called Type-II loss functions, and they also enjoy statistical robustness properties [Ding, 2013].

In terms of computation, of course, $\sigma_1(\cdot)$ and $\sigma_2(\cdot)$ are not convex, and therefore the objective function based on such loss functions is more difficult to optimize. However, it has been observed in Ding [2013] that models based on optimization of Type-I functions are often empirically much more successful than those which optimize Type-II functions. Furthermore, the solutions of Type-I optimization are more stable to the choice of parameter

initialization. Intuitively, this is because Type-II functions asymptote to a constant, reducing the gradient to almost zero in a large fraction of the parameter space; therefore, it is difficult for a gradient-based algorithm to determine which direction to pursue. See Ding [2013] for more details.

2.2.1 Ranking Model via Robust Binary Classification (RoBiRank)

In this section, we will extend robust binary classification to formulate RoBiRank, a novel model for ranking.

Let $\mathcal{X} = \{x_1, x_2, \ldots, x_n\}$ be a set of contexts, and $\mathcal{Y} = \{y_1, y_2, \ldots, y_m\}$ be a set of items to be ranked. For example, in movie recommender systems $\mathcal{X}$ is the set of users and $\mathcal{Y}$ is the set of movies. In some problem settings, only a subset of $\mathcal{Y}$ is relevant to a given context $x \in \mathcal{X}$; e.g. in document retrieval systems, only a subset of documents is relevant to a query. Therefore, we define $\mathcal{Y}_x \subset \mathcal{Y}$ to be a set of items relevant to context x. Observed data can be described by a set $W := \{W_{xy}\}_{x \in \mathcal{X}, y \in \mathcal{Y}_x}$ where W_{xy} is a real-valued score given to item y in context x.

We adopt a standard problem setting used in the literature of learning to rank. For each context x and an item $y \in \mathcal{Y}_x$, we aim to learn a scoring function $f(x, y) : \mathcal{X} \times \mathcal{Y}_x \to \mathbb{R}$ that induces a ranking on the item set $\mathcal{Y}_x$; the higher the score, the more important the associated item is in the given context. To learn such a function, we first extract joint features of x and y, which will be denoted by $\phi(x, y)$. Then, we parametrize $f(\cdot, \cdot)$ using a parameter ω, which yields the linear model $f_\omega(x, y) := \langle \phi(x, y), \omega \rangle$, where, as before, $\langle \cdot, \cdot \rangle$ denotes the Euclidean dot product between two vectors. ω induces a ranking on the set of items $\mathcal{Y}_x$; we define $\text{rank}_\omega(x, y)$ to be the rank of item y in a given context x induced by ω.

Observe that $\text{rank}_\omega(x, y)$ can also be written as a sum of 0-1 loss functions (see e.g. Usunier et al. [2009]):

$$\text{rank}_\omega(x, y) = \sum_{y' \in \mathcal{Y}_x, y' \neq y} I\left(f_\omega(x, y) - f_\omega(x, y') < 0\right). \tag{2.4}$$

2.2.2 Basic LTR Model

If an item y is very relevant in context x, a good parameter ω should position y at the top of the list; in other words, $\text{rank}_\omega(x, y)$ has to be small, which motivates the following objective function [Buffoni et al., 2011]:

$$L(\omega) := \sum_{x \in \mathcal{X}} c_x \sum_{y \in \mathcal{Y}_x} v(W_{xy}) \cdot \text{rank}_\omega(x, y), \tag{2.5}$$

where c_x is an weighting factor for each context x, and $v(\cdot) : \mathbb{R}^+ \to \mathbb{R}^+$ quantifies the relevance level of y on x. Note that $\{c_x\}$ and $v(W_{xy})$ can be chosen to reflect the metric the model is going to be evaluated on (this will be discussed in Section **??**). Note that (2.5) can be rewritten using (2.4) as a sum of indicator functions. Following the strategy in Section **??**, one can form an upper bound of (2.5) by bounding each 0-1 loss function by a logistic loss function:

$$\overline{L}(\omega) := \sum_{x \in \mathcal{X}} c_x \sum_{y \in \mathcal{Y}_x} v\left(W_{xy}\right) \cdot \sum_{y' \in \mathcal{Y}_x, y' \neq y} \sigma_0\left(f_\omega(x, y) - f_\omega(x, y')\right). \tag{2.6}$$

Just like (2.1), (2.6) is convex in ω and hence easy to minimize.

2.2.3 DCG

Although (2.6) enjoys convexity, it may not be a good objective function for ranking. This is because in most applications of learning to rank, it is more important to do well

at the top of the list than at the bottom, as users typically pay attention only to the top few items. Therefore, it is desirable to *give up* performance on the lower part of the list in order to gain quality at the top. This intuition is similar to that of robust classification in Section **??**; a stronger connection will be shown below.

Discounted Cumulative Gain (DCG) [Manning et al., 2008] is one of the most popular metrics for ranking. For each context $x \in \mathcal{X}$, it is defined as:

$$\mathrm{DCG}(\omega) := c_x \sum_{y \in \mathcal{Y}_x} \frac{v\left(W_{xy}\right)}{\log_2\left(\mathrm{rank}_\omega(x, y) + 2\right)}, \tag{2.7}$$

where $v(t) = 2^t - 1$ and $c_x = 1$. Since $1/\log(t+2)$ decreases quickly and then asymptotes to a constant as t increases, this metric emphasizes the quality of the ranking at the top of the list. Normalized DCG (NDCG) simply normalizes the metric to bound it between 0 and 1 by calculating the maximum achievable DCG value m_x and dividing by it [Manning et al., 2008].

2.2.4 RoBiRank formulation

Now we formulate RoBiRank, which optimizes the lower bound of metrics for ranking in form (2.7). Observe that $\max_\omega \mathrm{DCG}(\omega)$ can be rewritten as

$$\min_\omega \sum_{x \in \mathcal{X}} c_x \sum_{y \in \mathcal{Y}_x} v\left(W_{xy}\right) \cdot \left\{1 - \frac{1}{\log_2\left(\mathrm{rank}_\omega(x, y) + 2\right)}\right\}. \tag{2.8}$$

Using (2.4) and the definition of the transformation function $\rho_2(\cdot)$ in (2.2), we can rewrite the objective function in (2.8) as:

$$L_2(\omega) := \sum_{x \in \mathcal{X}} c_x \sum_{y \in \mathcal{Y}_x} v\left(W_{xy}\right) \cdot \rho_2\left(\sum_{y' \in \mathcal{Y}_x, y' \neq y} I\left(f_\omega(x, y) - f_\omega(x, y') < 0\right)\right). \tag{2.9}$$

Since $\rho_2(\cdot)$ is a monotonically increasing function, we can bound (2.9) with a continuous function by bounding each indicator function using the logistic loss:

$$\overline{L}_2(\omega) := \sum_{x \in \mathcal{X}} c_x \sum_{y \in \mathcal{Y}_x} v\left(W_{xy}\right) \cdot \rho_2 \left(\sum_{y' \in \mathcal{Y}_x, y' \neq y} \sigma_0\left(f_\omega(x, y) - f_\omega(x, y')\right) \right). \tag{2.10}$$

This is reminiscent of the basic model in (2.6); as we applied the transformation $\rho_2(\cdot)$ on the logistic loss $\sigma_0(\cdot)$ to construct the robust loss $\sigma_2(\cdot)$ in (2.3), we are again applying the same transformation on (2.6) to construct a loss function that respects the DCG metric used in ranking. In fact, (2.10) can be seen as a generalization of robust binary classification by applying the transformation on a *group* of logistic losses instead of a single loss. In both robust classification and ranking, the transformation $\rho_2(\cdot)$ enables models to give up on part of the problem to achieve better overall performance.

As we discussed in Section **??**, however, transformation of logistic loss using $\rho_2(\cdot)$ results in Type-II loss function, which is very difficult to optimize. Hence, instead of $\rho_2(\cdot)$ we use an alternative transformation $\rho_1(\cdot)$, which generates Type-I loss function, to define the objective function of RoBiRank:

$$\overline{L}_1(\omega) := \sum_{x \in \mathcal{X}} c_x \sum_{y \in \mathcal{Y}_x} v\left(W_{xy}\right) \cdot \rho_1 \left(\sum_{y' \in \mathcal{Y}_x, y' \neq y} \sigma_0\left(f_\omega(x, y) - f_\omega(x, y')\right) \right). \tag{2.11}$$

Since $\rho_1(t) \geq \rho_2(t)$ for every $t > 0$, we have $\overline{L}_1(\omega) \geq \overline{L}_2(\omega) \geq L_2(\omega)$ for every ω. Note that $\overline{L}_1(\omega)$ is continuous and twice differentiable. Therefore, standard gradient-based optimization techniques can be applied to minimize it. As is standard, a regularizer on ω can be added to avoid overfitting; for simplicity, we use the ℓ_2-norm in our experiments.

2.2.5 Standard Learning to Rank Experiments

We conducted experiments to check the performance of RoBiRank (2.11) in a standard learning to rank setting, with a small number of labels to rank. We pitch RoBiRank against the following algorithms: RankSVM [Lee and Lin, 2013], the ranking algorithm of Le and Smola [2007] (called LSRank in the sequel), InfNormPush [Rudin, 2009], IRPush [Agarwal, 2011], and 8 standard ranking algorithms implemented in RankLib[1] namely MART, RankNet, RankBoost, AdaRank, CoordAscent, LambdaMART, ListNet and RandomForests.

We use three sources of datasets: LETOR 3.0 [Chapelle and Chang, 2011] , LETOR 4.0[2] and YAHOO LTRC [Qin et al., 2010], which are standard benchmarks for ranking (see Table **??** for summary statistics). Each dataset consists of five folds; we consider the first fold, and use the training, validation, and test splits provided. We train with different values of regularization parameter, and select one with the best NDCG on the validation dataset. The performance of the model with this parameter on the test dataset is reported. For a fair comparison, every algorithm follows exactly the same protocol and uses the same split of data. All experiments in this section are conducted on a computing cluster where each node has two 2.1 GHz 12-core AMD 6172 processors with 48 GB physical memory per node. We used implementation of the L-BFGS algorithm provided by the Toolkit for Advanced Optimization (TAO)[3] for estimating the parameter of RoBiRank. For the other algorithms, we either implemented them using our framework or used the implementations provided by the authors.

We use values of NDCG at different levels of truncation as our evaluation metric

[1] http://sourceforge.net/p/lemur/wiki/RankLib
[2] http://research.microsoft.com/en-us/um/beijing/projects/letor/letor4dataset.aspx
[3] http://www.mcs.anl.gov/research/projects/tao/index.html

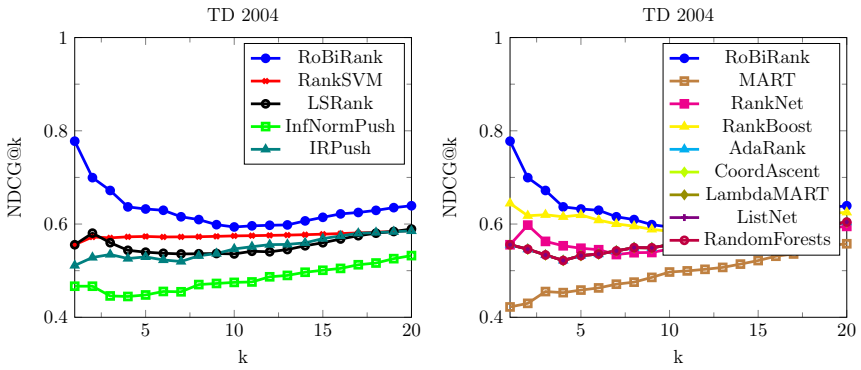


Figure 2.2: Comparison of RoBiRank with a number of competing algorithms.

[Manning et al., 2008]; see Figure 2.2. RoBiRank outperforms its competitors on most of the datasets; due to space constraints, we only present plots for the TD 2004 dataset here and other plots can be found in Appendix **??**. The performance of RankSVM seems insensitive to the level of truncation for NDCG. On the other hand, RoBiRank, which uses non-convex loss function to concentrate its performance at the top of the ranked list, performs much better especially at low truncation levels. It is also interesting to note that the NDCG@k curve of LSRank is similar to that of RoBiRank, but RoBiRank consistently outperforms at each level. RoBiRank dominates Inf-Push and IR-Push at all levels. When compared to standard algorithms, Figure 2.2 (right), again RoBiRank outperforms especially at the top of the list.

Overall, RoBiRank outperforms IRPush and InfNormPush on all datasets except TD 2003 and OHSUMED where IRPush seems to fare better at the top of the list. Compared to the 8 standard algorithms, again RobiRank either outperforms or performs comparably to the best algorithm except on two datasets (TD 2003 and HP 2003), where MART and Random Forests overtake RobiRank at few values of NDCG. We present a summary of the NDCG values obtained by each algorithm in Table **??** in the appendix. On the MSLR30K

dataset, some of the additional algorithms like InfNormPush and IRPush did not complete within the time period available; indicated by dashes in the table.

In the MSLR dataset (Figure **??** in the Appendix), despite the large number of queries and instances, RoBiRank still manages to outperform its competitors and it follows a trend as expected; having a higher NDCG@1 score, which provides an evidence that it focusses on the most important documents.

2.3 Latent Collaborative Retrieval (LCR)

2.3.1 Basic LCR model

For each context x and an item $y \in \mathcal{Y}$, the standard problem setting of learning to rank requires training data to contain feature vector $\phi(x, y)$ and score W_{xy} assigned on the x, y pair. When the number of contexts $|\mathcal{X}|$ or the number of items $|\mathcal{Y}|$ is large, it might be difficult to define $\phi(x, y)$ and measure W_{xy} for all x, y pairs. Therefore, in most learning to rank problems we define the set of *relevant* items $\mathcal{Y}_x \subset \mathcal{Y}$ to be much smaller than $\mathcal{Y}$ for each context x, and then collect data only for $\mathcal{Y}_x$. Nonetheless, this may not be realistic in all situations; in a movie recommender system, for example, for each user *every* movie is somewhat relevant.

On the other hand, implicit user feedback data is much more abundant. For example, a lot of users on Netflix would simply watch movie streams on the system but do not leave an explicit rating. By the action of watching a movie, however, they implicitly express their preference. Such data consist only of positive feedback, unlike traditional learning to rank datasets which have score W_{xy} between each context-item pair x, y. Again, we may not be

able to extract feature vectors for each x, y pair.

In such a situation, we can attempt to learn the score function $f(x, y)$ without a feature vector $\phi(x, y)$ by embedding each context and item in an Euclidean latent space; specifically, we redefine the score function to be: $f(x, y) := \langle U_x, V_y \rangle$, where $U_x \in \mathbb{R}^d$ is the embedding of the context x and $V_y \in \mathbb{R}^d$ is that of the item y. Then, we can learn these embeddings by a ranking model. This approach was introduced in Weston et al. [2012], and was called *latent collaborative retrieval*.

Adapting RoBiRank for Latent Collaborative Retrieval

In this section, we show how RoBiRank can be specialized for Latent Collaborative Retrieval. Let us define Ω to be the set of context-item pairs (x, y) which was observed in the dataset. Let $v(W_{xy}) = 1$ if $(x, y) \in \Omega$, and 0 otherwise; this is a natural choice since the score information is not available. For simplicity, we set $c_x = 1$ for every x. Now RoBiRank (2.11) specializes to:

$$\overline{L}_1(U, V) = \sum_{(x,y)\in\Omega} \rho_1 \left(\sum_{y' \neq y} \sigma_0(f(U_x, V_y) - f(U_x, V_{y'})) \right). \tag{2.12}$$

Note that now the summation inside the parenthesis of (2.12) is over all items $\mathcal{Y}$ instead of a smaller set $\mathcal{Y}_x$, therefore we omit specifying the range of y' from now on. To avoid overfitting, a regularizer is added to (2.12); for simplicity we use the Frobenius norm of U and V in our experiments.

2.3.2 Stochastic Optimization

When the size of the data $|\Omega|$ or the number of items $|\mathcal{Y}|$ is large, however, methods that require exact evaluation of the function value and its gradient will become very slow since the evaluation takes $O(|\Omega| \cdot |\mathcal{Y}|)$ computation. In this case, stochastic optimization methods are desirable [Bottou and Bousquet, 2011]; in this subsection, we will develop a stochastic gradient descent algorithm whose complexity is independent of $|\Omega|$ and $|\mathcal{Y}|$.

For simplicity, let θ be a concatenation of all parameters $\{U_x\}_{x\in\mathcal{X}}$, $\{V_y\}_{y\in\mathcal{Y}}$. The gradient $\nabla_\theta L_1(U, V)$ of (2.12) is

$$\sum_{(x,y)\in\Omega} \nabla_\theta \rho_1 \left(\sum_{y'\neq y} \sigma_0(f(U_x, V_y) - f(U_x, V_{y'})) \right).$$

Finding an unbiased estimator of the gradient whose computation is independent of $|\Omega|$ is not difficult; if we sample a pair (x, y) uniformly from Ω, then it is easy to see that the following estimator

$$|\Omega| \cdot \nabla_\theta \rho_1 \left(\sum_{y'\neq y} \sigma_0(f(U_x, V_y) - f(U_x, V_{y'})) \right) \tag{2.13}$$

is unbiased. This still involves a summation over $\mathcal{Y}$, however, so it requires $O(|\mathcal{Y}|)$ calculation. Since $\rho_1(\cdot)$ is a nonlinear function it seems unlikely that an unbiased stochastic gradient which randomizes over $\mathcal{Y}$ can be found; nonetheless, to achieve convergence guarantees of the stochastic gradient descent algorithm, unbiasedness of the estimator is necessary [Nemirovski et al., 2009].

We attack this problem by *linearizing* the objective function by parameter expansion. Note the following property of $\rho_1(\cdot)$ [Bouchard, 2007]:

$$\rho_1(t) = \log_2(t+1) \leq -\log_2 \xi + \frac{\xi \cdot (t+1) - 1}{\log 2}. \tag{2.14}$$

This holds for any $\xi > 0$, and the bound is tight when $\xi = \frac{1}{t+1}$. Now introducing an auxiliary parameter ξ_{xy} for each $(x, y) \in \Omega$ and applying this bound, we obtain an upper bound of (2.12) as

$$L(U, V, \xi) := \sum_{(x,y)\in\Omega} -\log_2 \xi_{xy} + \frac{\xi_{xy}\left(\sum_{y'\neq y} \sigma_0(f(U_x, V_y) - f(U_x, V_{y'})) + 1\right) - 1}{\log 2}. \quad (2.15)$$

Now we propose an iterative algorithm in which, each iteration consists of (U, V)-step and ξ-step; in the (U, V)-step we minimize (2.15) in (U, V) and in the ξ-step we minimize in ξ. Pseudo-code can be found in Algorithm 1.

(U, V)-step The partial derivative of (2.15) in terms of U and V can be calculated as: $\nabla_{U,V} L(U, V, \xi) := \frac{1}{\log 2} \sum_{(x,y)\in\Omega} \xi_{xy} \left(\sum_{y'\neq y} \nabla_{U,V} \sigma_0(f(U_x, V_y) - f(U_x, V_{y'}))\right)$. Now it is easy to see that the following stochastic procedure unbiasedly estimates the above gradient:

- Sample (x, y) uniformly from Ω
- Sample y' uniformly from $\mathcal{Y} \setminus \{y\}$
- Estimate the gradient by

$$\frac{|\Omega| \cdot (|\mathcal{Y}| - 1) \cdot \xi_{xy}}{\log 2} \cdot \nabla_{U,V} \sigma_0(f(U_x, V_y) - f(U_x, V_{y'})). \quad (2.16)$$

Therefore a stochastic gradient descent algorithm based on (2.16) will converge to a local minimum of the objective function (2.15) with probability one [Robbins and Monro, 1951]. Note that the time complexity of calculating (2.16) is independent of $|\Omega|$ and $|\mathcal{Y}|$. Also, it is a function of only U_x and V_y; the gradient is zero in terms of other variables.

ξ-step When U and V are fixed, minimization of ξ_{xy} variable is independent of each other and a simple analytic solution exists: $\xi_{xy} = \frac{1}{\sum_{y'\neq y} \sigma_0(f(U_x, V_y) - f(U_x, V_{y'})) + 1}$. This of course

requires $O(|\mathcal{Y}|)$ work. In principle, we can avoid summation over $\mathcal{Y}$ by taking stochastic gradient in terms of ξ_{xy} as we did for U and V. However, since the exact solution is simple to compute and also because most of the computation time is spent on (U, V)-step, we found this update rule to be efficient.

Algorithm 1 Serial parameter estimation algorithm for latent collaborative retrieval

η: step size
repeat
  // (U, V)-step
  repeat
    Sample (x, y) uniformly from Ω
    Sample y' uniformly from $\mathcal{Y} \setminus \{y\}$
    $U_x \leftarrow U_x - \eta \cdot \xi_{xy} \cdot \nabla_{U_x} \sigma_0(f(U_x, V_y) - f(U_x, V_{y'}))$
    $V_y \leftarrow V_y - \eta \cdot \xi_{xy} \cdot \nabla_{V_y} \sigma_0(f(U_x, V_y) - f(U_x, V_{y'}))$
  until convergence in U, V
  // ξ-step
  for $(x, y) \in \Omega$ **do**
    $\xi_{xy} \leftarrow \frac{1}{\sum_{y' \neq y} \sigma_0(f(U_x, V_y) - f(U_x, V_{y'})) + 1}$
  end for
until convergence in U, V and ξ

2.3.3 Parallelization

The linearization trick in (2.15) not only enables us to construct an efficient stochastic gradient algorithm, but also makes possible to efficiently parallelize the algorithm

across multiple number of machines.

Suppose there are p number of machines. The set of contexts $\mathcal{X}$ is randomly partitioned into mutually exclusive and exhaustive subsets $\mathcal{X}^{(1)}, \mathcal{X}^{(2)}, \ldots, \mathcal{X}^{(p)}$ which are of approximately the same size. This partitioning is fixed and does not change over time. The partition on $\mathcal{X}$ induces partitions on other variables as follows: $U^{(q)} := \{U_x\}_{x \in \mathcal{X}^{(q)}}$, $\Omega^{(q)} := \left\{(x, y) \in \Omega : x \in \mathcal{X}^{(q)}\right\}$, $\xi^{(q)} := \{\xi_{xy}\}_{(x,y) \in \Omega^{(q)}}$, for $1 \leq q \leq p$.

Each machine q stores variables $U^{(q)}$, $\xi^{(q)}$ and $\Omega^{(q)}$. Since the partition on $\mathcal{X}$ is fixed, these variables are local to each machine and are not communicated. Now we describe how to parallelize each step of the algorithm: the pseudo-code can be found in Algorithm 2.

(U, V)-step At the start of each (U, V)-step, a new partition on $\mathcal{Y}$ is sampled to divide $\mathcal{Y}$ into $\mathcal{Y}^{(1)}, \mathcal{Y}^{(2)}, \ldots, \mathcal{Y}^{(p)}$ which are also mutually exclusive, exhaustive and of approximately the same size. The difference here is that unlike the partition on $\mathcal{X}$, a new partition on $\mathcal{Y}$ is sampled for every (U, V)-step. Let us define $V^{(q)} := \{V_y\}_{y \in \mathcal{Y}^{(q)}}$. After the partition on $\mathcal{Y}$ is sampled, each machine q fetches V_y's in $V^{(q)}$ from where it was previously stored; in the very first iteration which no previous information exists, each machine generates and initializes these parameters instead. Now let us define $L^{(q)}(U^{(q)}, V^{(q)}, \xi^{(q)}) :=$

$$\sum_{(x,y) \in \Omega^{(q)}, y \in \mathcal{Y}^{(q)}} -\log_2 \xi_{xy} + \frac{\xi_{xy}\left(\sum_{y' \in \mathcal{Y}^{(q)}, y' \neq y} \sigma_0(f(U_x, V_y) - f(U_x, V_{y'})) + 1\right) - 1}{\log 2}.$$

In parallel setting, each machine q runs stochastic gradient descent on $L^{(q)}(U^{(q)}, V^{(q)}, \xi^{(q)})$ instead of the original function $L(U, V, \xi)$. Since there is no overlap between machines on the parameters they update and the data they access, every machine can progress independently of each other. Although the algorithm takes only a fraction of data into consideration at

a time, this procedure is also guaranteed to converge to a local optimum of the *original* function $L(U, V, \xi)$ according to Stratified Stochastic Gradient Descent (SSGD) scheme of Gemulla et al. [2011]. The intuition is as follows: if we take expectation over the random partition on $\mathcal{Y}$, we have $\nabla_{U,V} L(U, V, \xi) =$

$$q^2 \cdot \mathbb{E}\left[\sum_{1 \le q \le p} \nabla_{U,V} L^{(q)}(U^{(q)}, V^{(q)}, \xi^{(q)})\right], \tag{2.17}$$

while the expectation is over the selection of the partition $\left\{\mathcal{Y}^{(1)}, \mathcal{Y}^{(2)}, \ldots, \mathcal{Y}^{(p)}\right\}$. Therefore, although there is some discrepancy between the function we take stochastic gradient on and the function we actually aim to minimize, in the long run the bias will be washed out and the algorithm will converge to a local optimum of the objective function $L(U, V, \xi)$. Specifically, (2.17) ensures that Condition 7 of Theorem 1 in Gemulla et al. [2011] is satisfied, while the rest of conditions can be easily met by introducing an L_2 regularizer and thus bounding the parameter space.

The convergence can be formally proved as follows. We introduce a simplifying assumption that for each inner **repeat** loop in line ?? of Algorithm 2, each machine executes exactly the same number of updates, which we will denote by T. Let $(x^{(q),t}, y^{(q),t})$ be the t-th pair sampled in line ?? in machine q. Since updates in line ?? and ?? in each machine is independent of updates made in other machines, we can regard that every machine is simultaneously executing (reading or writing) updates. In other words, each machine p samples an unbiased stochastic gradient for $L^{(q)}(U^{(q)}, V^{(q)}, \xi^{(q)})$.

ξ-step In this step, all machines synchronize to retrieve every entry of V. Then, each machine can update $\xi^{(q)}$ independently of each other. When the size of V is very large

and cannot be fit into the main memory of a single machine, V can be partitioned as in (U, V)-step and updates can be calculated in a round-robin way.

Note that this parallelization scheme requires each machine to allocate only $\frac{1}{p}$-fraction of memory that would be required for a single-machine execution. Therefore, in terms of space complexity the algorithm scales linearly with the number of machines.

2.3.4 Experiments

In this subsection, we ask the following question: Given large amounts of computational resources, what is the best latent collaborative retrieval model (in terms of predictive performance on the test dataset) that one can produce within a given wall-clock time? Towards this end, we work with the parallel variant of RoBiRank described in Section 2.3.3. As a representative dataset we use the Million Song Dataset (MSD) Bertin-Mahieux et al. [2011], which consists of 1,129,318 users ($|\mathcal{X}|$), 386,133 songs ($|\mathcal{Y}|$), and 49,824,519 records ($|\Omega|$) of a user x playing a song y in the training dataset. The objective is to predict the songs from the test dataset that a user is going to listen to[4].

Squared frobenius norm of matrices U and V were added to the objective function (2.11) for regularization, and the entries of U and V were independently sampled uniformly from 0 to $1/\sqrt{d}$. We performed a grid-search to find the best step size parameter. Since explicit ratings are not given, NDCG is not applicable for this task; we use precision at 1 and 10 [Manning et al., 2008] as our evaluation metric.

[4]the original data also provides the number of times a song was played by a user, but we ignored this in our experiment.

Hardware: This experiment was run on a computing cluster where each machine is equipped with 2 Intel Xeon E5 processors (16 cores) and 32GB of RAM. Our algorithm is implemented in C++ and uses Intel Thread Building Blocks (TBB) to handle thread-level parallelization, and MVAPICH2 was used for machine-to-machine communication. Due to a limitation of the job scheduler on the cluster all experiments had to be stopped after 100,000 seconds.

Predictive Performance

We compare RoBiRank with a state of the art algorithm from Weston et al. [2012], which optimizes a similar objective function (2.18). We compare how fast the quality of the solution improves as a function of wall clock time. Since the authors of Weston et al. [2012] do not make available their code, we implemented their algorithm within our framework using the same data structures and libraries used by our method. Furthermore, for a fair comparison, we used the same initialization for U and V and performed an identical grid-search over the step size parameter.

It can be seen from Figure 2.3 that on a single machine the algorithm of Weston et al. [2012] is very competitive and outperforms RoBiRank. The reason for this might be the introduction of the additional ξ variables in RoBiRank, which slows down convergence. However, RoBiRank training can be distributed across processors, while it is not clear how to parallelize the algorithm of Weston et al. [2012]. Consequently, RoBiRank 32 which uses 32 machines for its computation can produce a significantly better model within the same wall clock time window.

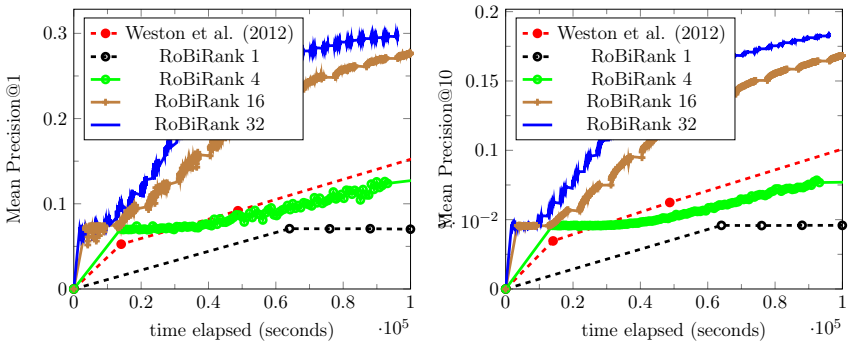


Figure 2.3: Comparison of RoBiRank and Weston et al. [2012] in terms of Mean Precision @1 (left) and Mean Precision@10 (right) when the same amount of wall-clock computation time is given.

Scalability

Next, we study the scaling behavior of RoBiRank as a function of number of machines. RoBiRank p denotes the parallel version of RoBiRank which is distributed across p machines. In Figure 2.4 we plot mean Precision@1 as a function of the number of machines $\times$ the number of seconds elapsed; this is a proxy for CPU time. If an algorithm linearly scales across multiple processors, then all lines in the figure should overlap with each other. As can be seen RoBiRank exhibits near ideal speed up when going from 4 to 32 machines[5].

2.4 Related Work

In terms of modeling, viewing ranking problems as generalization of binary classification problems is not a new idea; for example, RankSVM defines the objective function as a sum of hinge losses, similarly to our basic model (2.5) in Section 2.2.2. However, it does not directly optimize the ranking metric such as NDCG; the objective function and the metric

[5]The graph for RoBiRank 1 is hard to see because it was run for only 100,000 CPU-seconds.

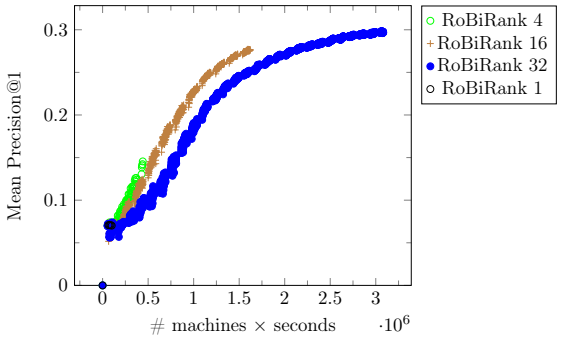


Figure 2.4: Scaling behavior of RoBiRank as the number of machines changes. Since the x-axis is now scaled by the number of machines, when we achieve linear scaling the curves should overlap with each other.

are not immediately related to each other. In this respect, our approach is closer to that of Le and Smola [2007] which constructs a convex upper bound on the ranking metric and Chapelle et al. [2008] which improves the bound by introducing non-convexity. The objective function of Chapelle et al. [2008] is also motivated by ramp loss, which is used for robust classification; nonetheless, to our knowledge the direct connection between the ranking metrics in form (2.7) (DCG, NDCG) and the robust loss (2.3) is our novel contribution. Also, our objective function is designed to specifically bound the ranking metric, while Chapelle et al. [2008] proposes a general recipe to improve existing convex bounds.

Stochastic optimization of the objective function for latent collaborative retrieval has been also explored in Weston et al. [2012]. They attempt to minimize

$$\sum_{(x,y)\in\Omega} \Phi\left(1 + \sum_{y'\neq y} I(f(U_x, V_y) - f(U_x, V_{y'}) < 0)\right), \tag{2.18}$$

where $\Phi(t) = \sum_{k=1}^{t} \frac{1}{k}$. This is similar to our objective function (2.15); $\Phi(\cdot)$ and $\rho_2(\cdot)$ are asymptotically equivalent. However, we argue that our formulation (2.15) has two major

advantages. First, it is a continuous and differentiable function, therefore gradient-based algorithms such as L-BFGS and stochastic gradient descent have convergence guarantees. On the other hand, the objective function of Weston et al. [2012] is not even continuous, since their formulation is based on a function $\Phi(\cdot)$ that is defined for only natural numbers. Also, through the linearization trick in (2.15) we are able to obtain an unbiased stochastic gradient, which is necessary for the convergence guarantee, and to parallelize the algorithm across multiple machines as discussed in Section 2.3.3. It is unclear how these techniques can be adapted for the objective function of Weston et al. [2012].

Note that Weston et al. [2012] proposes a more general class of models for the task than can be expressed by (2.18). For example, they discuss situations in which we have side information on each context or item to help learning latent embeddings. Some of the optimization techniqures introduced in Section 2.3.2 can be adapted for these general problems as well, but is left for future work.

Parallelization of an optimization algorithm via parameter expansion (2.14) was applied to a bit different problem named multinomial logistic regression [Gopal and Yang, 2013]. However, to our knowledge we are the first to use the trick to construct an unbiased stochastic gradient that can be efficiently computed, and adapt it to stratified stochastic gradient descent (SSGD) scheme of Gemulla et al. [2011]. Note that the optimization algorithm can alternatively be derived using convex multiplicative programming framework of Kuno et al. [1993]. In fact, Ding [2013] develops a robust classification algorithm based on this idea; this also indicates that robust classification and ranking are closely related.

It has been observed that while matrix completion approaches are successful in

predicting ratings of users to items, they have poor performances on predicting items users might be interested in Cremonesi et al. [2010]. It has been argued that in recommendation purposes, it is important to model what user-item pair was *not* observed as well as what was observed [Cremonesi et al., 2010, Steck, 2010].

2.5 Conclusion

In this chapter, we developed RoBiRank, a novel model on ranking, based on insights and techniques from robust binary classification. Then, we proposed a scalable and parallelizable stochastic optimization algorithm that can be applied to latent collaborative retrieval task which large-scale data without feature vectors and explicit scores have to take care of. Experimental results on both learning to rank datasets and latent collaborative retrieval dataset suggest the advantage of our approach.

As a final note, the experiments in Section 2.3.4 are arguably unfair towards WSABIE. For instance, one could envisage using clever engineering tricks to derive a parallel variant of WSABIE (*e.g.*, by averaging gradients from various machines), which might outperform RoBiRank on the MSD dataset. While performance on a specific dataset might be better, we would lose global convergence guarantees. Therefore, rather than obsess over the performance of a specific algorithm on a specific dataset, via this work we hope to draw the attention of the community to the need for developing principled parallel algorithms for this important problem.

Algorithm 2 Multi-machine parameter estimation algorithm for latent collaborative retrieval

η: step size

repeat

  `// parallel` (U, V)`-step`

  repeat

    Sample a partition $\left\{\mathcal{Y}^{(1)}, \mathcal{Y}^{(2)}, \ldots, \mathcal{Y}^{(p)}\right\}$ **for all machine** $q \in \{1, 2, \ldots, p\}$ **do in parallel**

      Fetch all $V_y \in V^{(q)}$

      repeat

        Sample (x, y) uniformly from $\left\{(x, y) \in \Omega^{(q)}, y \in \mathcal{Y}^{(q)}\right\}$

        Sample y' uniformly from $\mathcal{Y}^{(q)} \setminus \{y\}$

        $U_x \leftarrow U_x - \eta \cdot \xi_{xy} \cdot \nabla_{U_x} \sigma_0(f(U_x, V_y) - f(U_x, V_{y'}))$

        $V_y \leftarrow V_y - \eta \cdot \xi_{xy} \cdot \nabla_{V_y} \sigma_0(f(U_x, V_y) - f(U_x, V_{y'}))$

      until predefined time limit is exceeded

    end for

  until convergence in U, V

  `// parallel` ξ`-step`

  for all machine $q \in \{1, 2, \ldots, p\}$ **do in parallel**

    Fetch all $V_y \in V$

    for $(x, y) \in \Omega^{(q)}$ **do**

      $\xi_{xy} \leftarrow \frac{1}{\sum_{y' \neq y} \sigma_0(f(U_x, V_y) - f(U_x, V_{y'})) + 1}$

    end for

  end for

until convergence in U, V and ξ

Chapter 3

Multinomial Logistic Regression

3.1 Introduction

In this paper, we focus on *multinomial logistic regression* (MLR), also known as softmax regression which computes the probability of a D-dimensional data point $\mathbf{x}_i \in \{\mathbf{x}_1, \mathbf{x}_2, \ldots, \mathbf{x}_N\}$ belonging to a class $k \in \{1, 2, \ldots, K\}$. The model is parameterized by a parameter matrix $W \in \mathbb{R}^{D \times K}$. MLR is a method of choice for several real-world tasks such as Image Classification [Russakovsky et al., 2015] and Video Recommendation [Davidson et al., 2010]. It also manifests as the final output layer in Feed-Forward Deep Neural Networks [Goodfellow et al., 2016]. Therefore, it has received significant research attention [Gopal and Yang, 2013], [Yen et al., 2016]. We concern ourselves with running MLR in the presence of large number of data points N and large number of classes K - a setting which often requires distributing computation over P machines (viz. workers).

Traditional methods to perform distributed MLR typically fall into two categories: (a) *data parallel* methods such as L-BFGS [Nocedal and Wright, 2006] which partition the

data workload across P workers, however, duplicate the model workload across all workers, and (b) *model parallel* methods such as LC Gopal and Yang [2013], which partition the model workload across P workers, but need to duplicate the data across all of them. This is illustrated in Table 3.1.

	Storage per worker		Communication
	Data	Parameters	
L-BFGS	$O(\frac{ND}{P})$	$O(KD)$	$O(KD)$
LC	$O(ND)$	$O(\frac{KD}{P}) + O(N)$	$O(N)$
DS-MLR	$O(\frac{ND}{P})$	$O(\frac{KD}{P}) + O(\frac{N}{P})$	$O(\frac{KD}{P})$

Table 3.1: Memory requirements of various algorithms in MLR (N data points, D features, K classes, P workers).

The growing acclaim of machine learning is witnessing a surge of novel prediction tasks in diverse domains such as natural language, speech, image and video. These tasks not only involve *humongous amounts of data*, but also are powered by *sophisticated models*, thus demanding larger storage footprints for the model itself. Such memory requirements typically exceed the capacity of a single machine in a commodity cluster easily. Figure 1.2 illustrates this fundamental challenge in large-scale machine learning. As seen in the figure, real-world datasets exhibit varying storage requirements for the data and model. While the smaller ones are within the capacity of a single machine, larger datasets such as ODP and Reddit are impossible to run on a commodity cluster with just traditional model parallelism approaches. This is because ODP has a massive requirement of 355 GB for the model itself,

while Reddit-Full dataset is even bigger, requiring 228 GB for the data and 358 GB for its model.

One versatile method (DS-MLR): We propose a universal method which acts as a swiss army knife to get the best of both worlds. As a consequence of partitioning both data and model workloads simultaneously (*Hybrid Parallelism*) across P processors, DS-MLR is able to handle data and model parameters of varying sizes, without incurring any storage costs due to duplication. In Figures 3.4 and 3.6, we show results of running DS-MLR on three representative datasets which have large model storage requirements: (i) Despite being a modestly-sized dataset, **LSHTC-large** requires 34 GB for its model parameters. As a result, only model-parallel methods such as LC can be run on it. When compared against DS-MLR, we observed that DS-MLR is able to achieve a much faster convergence than LC as seen in the plot (ii) **ODP** is a much larger dataset requiring 355 GB for the model parameters. Even though LC could theoretically be run on it, we observed that it took an enormous amount of time to complete even a single iteration. We believe this is because LC is a second-order method and therefore its per iteration cost is significantly higher than a stochastic method such as DS-MLR (iii) Finally, **Reddit-Full** is a new benchmark dataset pushing the limits of data and model storage. Running MLR on this dataset requires 228 GB of data and 358 GB of model storage. This is a prototypical use-case where a hybrid-parallel method shines over its vanilla data/model parallel counterparts. To the best of our knowledge *no other existing methods* are able to handle such large workloads. Figure 3.6 shows that DS-MLR is able to handle this scale. However, since each iteration on this massive dataset takes 5 hours (this is excluding the time spent in data loading and initializing the optimizer), we could not

keep our experiment running beyond 10 days due to job time limitations on our HPC cluster.

Cost Analysis of using commodity hardware: Massive real-world datasets demand high memory, e.g. Reddit-Full consumes 600 GB of total memory (228 GB data, 358 GB model parameters). *When using hybrid parallelism*, one can get away with using cheap commodity hardware. Since the load for RedditFull dataset demands 50,000 compute hours per iteration, using 20 `c5.4xlarge` EC2 instances (32 GB RAM, 16 CPUs, $0.68 / hr per machine) each running 16 threads, one iteration requires 156 hours with a cost of $2,121 per iteration. On the other hand, *if we use data parallelism only (or model parallelism only)*, high-memory instances such as `x1.16xlarge` (900 GB RAM, 64 CPUs, $6.67 per hr per machine) are inevitable. A rough calculation shows that to achieve the same per iteration time, one would have to spend $111,335. This is mainly because, either data or parameters would need to be replicated across each processor, thus making it impossible to use all 64 cores. Even if we make use of a clever data or parameter sharing mechanism to avoid replication, the resulting cost comes down to $5,000 roughly, which is still twice the earlier case.

Our main contributions:

- **Hybrid Parallel reformulation for MLR**: We present DS-MLR, a novel distributed stochastic optimization algorithm that can partition both data as well as model parameters *simultaneously* (*hybrid parallel*) across its workers. DS-MLR is able to perfectly partition the workload across P workers, costing $O(\frac{ND}{P})$ storage for data and $O(\frac{KD}{P} + \frac{N}{P})$ for the model. As a result, DS-MLR can scale to arbitrarily large datasets where to the best of our knowledge, many of the existing distributed algorithms cannot

be applied since they need to either duplicate $O(KD)$ storage (data parallel methods) or $O(ND)$ storage (model parallel methods) across their workers.

- **Asynchronous and De-centralized Implementation**: To deploy DS-MLR on real-world datasets, we develop a *non-blocking* and *asynchronous* variant (DS-MLR Async), which provides further speedups in the multi-core, multi-machine setting by interleaving the computation and communication phases during every iteration. DS-MLR avoids expensive *bulk-synchronization* operations by maintaining local normalization terms on each worker and accumulating them incrementally using a token-ring topology. DS-MLR is implemented in C++ making use of intra-machine parallelism (multi-threading using Intel TBB) as well as inter-machine parallelism (using MPI).

- **Large-scale real world experiments**: We present an exhaustive empirical study running DS-MLR on real-world datasets with varying data and model footprints, showing that DS-MLR readily applies in all cases. In particular, to demonstrate applicability of DS-MLR to the scenario where *both data and model do not fit* on a single machine, we created a new benchmark dataset Reddit-Full that has data and model footprints of *228 GB and 358 GB* respectively.

3.2 Related Work

There has been a flurry of work in the past few years on developing distributed optimization algorithms for machine learning. In this section, we characterize some of this related work and put our method DS-MLR in perspective.

Data Parallelism vs Model Parallelism: The classic paradigm in distributed machine learning is to perform *data partitioning*, using, for instance, a map reduce style architecture [Chu et al., 2007] where data is distributed across multiple slaves. In each iteration, the slaves gather the parameter vector from the master, compute gradients locally and transmit them back to the master. The L-BFGS optimization algorithm [Nocedal and Wright, 2006] is typically used in the master to update the parameters after every iteration. *The main drawback of this strategy is that the model parameters need to be replicated on every machine.* For a D dimensional problem involving K classes, this demands $O(K \times D)$ storage. In many cases, this is too large to fit on a single machine. An orthogonal approach is to use *model partitioning*. Here again, a master slave architecture is used, but now, the data is replicated across each slave. The model parameters are partitioned and distributed to each machine. During each iteration, the model parameters on the individual machines are updated, and some auxiliary variables are computed and distributed to the other slaves, which use these variables in their parameter updates. See the Log-Concavity (LC) method Gopal and Yang [2013] for an example of such a strategy. *The main drawback of this approach, however, is that the data needs to be replicated on each machine*, and consequently it is not applicable when the data is too large to fit on a single machine.

Distributed Stochastic Gradient Methods: Stochastic gradient descent based approaches have proven to be very fruitful since they make frequent parameter updates and converge much more rapidly [Bottou, 2010]. Several algorithms for parallelizing SGD have been proposed in the past such as Hogwild [Recht et al., 2011], Parallel SGD [Zinkevich et al., 2010], DSGD [Gemulla et al., 2011], FPSGD [Zhuang et al., 2013] and more recently,

Parameter Server [Li et al., 2013] and Petuum [Xing et al., 2015]. Although the importance of data and model parallelism has been recognized in Parameter Server and the Petuum framework [Xing et al., 2015], to the best of our knowledge this has not been exploited in their specific instantiations such as applications to multinomial logistic regression [Xie et al., 2015]. We believe this is because [Xie et al., 2015] does not reformulate the problem the way DS-MLR does. *Several problems in machine learning are not naturally well-suited for simultaneous data and model parallelism, and therefore such reformulations are essential in uncovering a suitable structure.*

De-centralized vs Parameter Server Based: Parameter Server [Li et al., 2013] a widely popular architecture for distributed machine learning, makes use of two types of nodes: *workers* and *parameter servers*. The former is used to store the partitioned data and the latter to store the partitioned model. Workers communicate with the parameter servers and push/ pull gradient updates. Therefore, this architecture can be leveraged for hybrid parallelism (simultaneous data and model parallelism). [Xiao et al., 2017b] is one such work where parameter servers have been used to provide simultaneous data and model parallel formulation for binary regularized risk minimization problems. However parameter server has its own challenges: (1) There is an added overhead in network bandwidth arising due to communication between the layers of workers and parameter servers, (2) There is some effort required to strike the right balance between hardware efficiency and statistical efficiency while setting up the resource allocation (ratio of # of worker nodes to # of parameter servers). Adding too few parameter servers could cause the model to converge very slowly or not converge at all (poor statistical efficiency) due to insufficient rounds of synchronization. On

the other hand, adding too many of these servers to enable frequent model synchronization, could take hardware resources away from the workers (poor hardware efficiency), (3) Moreover, the optimal resource allocation of workers and parameter servers depends on several factors such as the cluster size, hardware characteristics, and the training data. These challenges have been explored in much more detail in [Watcharapichat et al., 2016] with some empirical study. The work in [Watcharapichat et al., 2016] provides a motivation towards exploring de-centralized architectures for distributed machine learning. *Our proposed method DS-MLR is a step in this direction, where at any given point of time, both data as well as model stays truly partitioned into mutually exclusive blocks across the workers. Parameter updates are directly exchanged across workers, eliminating the need for any intermediate servers.*

Asynchronous vs Synchronous: Parameter Server and HogWild [Recht et al., 2011] are asynchronous approaches. In Hogwild, parameter updates are executed in parallel using different threads under the assumption that any two serial updates are not likely to collide on the same data point when the data is sparse. *DS-MLR does not make any such assumptions. It has both synchronous and asynchronous variants and the latter is in the spirit of NOMAD* [Yun et al., 2013].

Alternating direction method of multipliers (ADMM) [Boyd et al., 2011] is another popular technique to parallelize convex optimization problems. The key idea in ADMM is to reformulate the original optimization problem by introducing redundant linear constraints. This makes the new objective easily *data parallel*. However, *ADMM suffers from a similar drawback as L-BFGS when applied to multinomial logistic regression.* The number of redundant constraints that need to be introduced are N (# data points) $\times$ K (# classes) which is a

major bottleneck to model parallelism. Moreover, the convergence rate of ADMM for MLR is known to be slow as discussed in [Gopal and Yang, 2013].

Log-Concavity (LC) method Gopal and Yang [2013] proposed a distributed *model parallel* approach to solve the multinomial logistic regression problem by linearizing the log-partition function based on its variational form [Bouchard, 2007]. However, because their formulation is only model parallel - *the entire data has to be replicated across all the workers, and a bulk-synchronization step is required per iteration to accumulate the partial models from various machines.* This is not practical for real world applications when both the data and model sizes get larger. Interestingly, we noticed that the objective function of the LC method can also be recovered from the objective function of DS-MLR (3.5).

Doubly-Separable formulations: Our reformulation in DS-MLR exploits the doubly-separable [Yun, 2014] structure in terms of global model parameters and some local auxiliary variables. Other doubly-separable methods also exist such as NOMAD [Yun et al., 2013] for matrix completion and RoBiRank [Yun et al., 2014a] for latent collaborative retrieval. NOMAD [Yun et al., 2013] is a distributed-memory, asynchronous and decentralized algorithm and RoBiRank [Yun et al., 2014a] is also a distributed-memory but synchronous algorithm.

3.3 Multinomial Logistic Regression

Consider training data of the form $(\mathbf{x}_i, y_i)_{i=1,\ldots,N}$ where $\mathbf{x}_i \in \mathbb{R}^d$ is a d-dimensional feature vector and $y_i \in \{1, 2, \ldots, K\}$ is a label associated with it; K denotes the number of class labels. Let $y_{ik} = I(y_i = k)$ denote the membership of data point $\mathbf{x}_i$ to class k. The

probability that $\mathbf{x}_i$ belongs to class k is given by:

$$p(y = k|\mathbf{x}_i) = \frac{\exp(\mathbf{w}_k^T \mathbf{x}_i)}{\sum_{j=1}^{K} \exp(\mathbf{w}_j^T \mathbf{x}_i)}, \tag{3.1}$$

where $W = \{\mathbf{w}_1, \mathbf{w}_2, \ldots, \mathbf{w}_K\}$ denotes the parameter vector for each of the K classes. Using the negative log-likelihood of (3.1) as a loss function, the L2-regularized objective function of MLR can be written as:

$$L_1(W) = \frac{\lambda}{2} \sum_{k=1}^{K} \|\mathbf{w}_k\|^2 - \frac{1}{N} \sum_{i=1}^{N} \sum_{k=1}^{K} y_{ik} \mathbf{w}_k^T \mathbf{x}_i + \frac{1}{N} \sum_{i=1}^{N} \log \left(\sum_{k=1}^{K} \exp(\mathbf{w}_k^T \mathbf{x}_i) \right), \tag{3.2}$$

where λ is the regularization hyper-parameter. Table 5.1 summarizes these notations. Optimizing the above objective function (3.2) when the number of classes K is large, is extremely challenging as computing the *log partition function* involves summing up over a large number of classes. In addition, it couples the class level parameters $\mathbf{w}_k$ together, making it difficult to distribute computation. In this section, we present an alternative formulation for MLR, to address this challenge.

3.4 Doubly-Separable Multinomial Logistic Regression (DS-MLR)

In this section, we present a reformulation of the MLR problem, which is closer in spirit to dual-decomposition methods [Boyd and Vandenberghe, 2004]. We begin by first rewriting (3.2) as,

$$L_1(W) = \frac{\lambda}{2} \sum_{k=1}^{K} \|\mathbf{w}_k\|^2 - \frac{1}{N} \sum_{i=1}^{N} \sum_{k=1}^{K} y_{ik} \mathbf{w}_k^T \mathbf{x}_i - \frac{1}{N} \sum_{i=1}^{N} \log \frac{1}{\sum_{k=1}^{K} \exp(\mathbf{w}_k^T \mathbf{x}_i)}, \tag{3.3}$$

Symbol	Definition
N	total number of observations
D	total number of dimensions
K	total number of classes
$x = \{\mathbf{x}_1, \ldots, \mathbf{x}_N\}, \quad \mathbf{x}_i \in \mathbb{R}^D$	data points
$y = \{y_1, \ldots, y_N\}, \quad y_i \in \{1, 2, \ldots, K\}$	class the data point x_i belongs to (i.e. label)
$W = \{\mathbf{w}_1, \ldots, \mathbf{w}_K\}, \quad \mathbf{w}_k \in \mathbb{R}^D$	parameters of the model
$a = \{a_1, \ldots, a_N\}, \quad a_i \in \mathbb{R}$	auxiliary variables mapping one-one to the observations
$b = \{b_1, \ldots, b_N\}, \quad b_i \in \mathbb{R}$	auxiliary variables used to represent $\log a_i$ for convenience
$y_{ik} = I(y_i = k)$	Indicator variable denoting the membership of data point x_i to class k
λ	regularization hyper-parameter
η	learning rate hyper-parameter

Table 3.2: Notations for Multinomial Logistic Regression

This can be expressed as a constrained optimization problem,

$$L_1(W, A) = \frac{\lambda}{2} \sum_{k=1}^{K} \|\mathbf{w}_k\|^2 - \frac{1}{N} \sum_{i=1}^{N} \sum_{k=1}^{K} y_{ik} \mathbf{w}_k^T \mathbf{x}_i - \frac{1}{N} \sum_{i=1}^{N} \log a_i, \tag{3.4}$$

$$\text{s.t.} \quad a_i = \frac{1}{\sum_{k=1}^{K} \exp(\mathbf{w}_k^T \mathbf{x}_i)}, \quad i = 1, 2, \ldots N$$

where $A = \{a_i\}_{i=1,\ldots,N}$.

Observe that this resembles dual-decomposition methods of the form:

$\min_{x,z} f(x) + g(z)$ s.t. $Ax + Bz = c$, where f and g are convex functions. In our objective function (3.4), the decomposable functions are $f(W)$ and $g(A)$ respectively. Introducing Lagrange multipliers, $\beta_i, \quad i = 1, 2 \ldots N$, we obtain the equivalent unconstrained minimax

problem [Boyd and Vandenberghe, 2004],

$$L_2(W, A, \beta) = \frac{\lambda}{2}\sum_{k=1}^{K}\|\mathbf{w}_k\|^2 - \frac{1}{N}\sum_{i=1}^{N}\sum_{k=1}^{K} y_{ik}\mathbf{w}_k^T\mathbf{x}_i - \frac{1}{N}\sum_{i=1}^{N}\log a_i \\ + \frac{1}{N}\sum_{i=1}^{N}\sum_{k=1}^{K}\beta_i\, a_i \exp(\mathbf{w}_k^T\mathbf{x}_i) - \frac{1}{N}\sum_{i=1}^{N}\beta_i \tag{3.5}$$

It is known that dual-decomposition methods can reliably find a stationary point, therefore the solution obtained by our method is also globally optimal. The updates for the primal variables W, A and dual variable β can be written as follows:

$$W_k^{t+1} \leftarrow \underset{W_k}{\operatorname{argmin}}\, L_2(W_k, a^t, \beta^t), \tag{3.6}$$

$$a_i^{t+1} \leftarrow \underset{a_i}{\operatorname{argmin}}\, L_2(W_k^{t+1}, a_i^t, \beta_i^t), \tag{3.7}$$

$$\beta_i^{t+1} \leftarrow \beta_i^t + \rho\left(a_i^{t+1}\sum_{k=1}^{K}\exp\left({w_k^T}^{t+1}x_i\right) - 1\right) \tag{3.8}$$

Here, W_k^{t+1} and a_i^{t+1} can be obtained by any black-box optimization procedure, while β_i^{t+1} is updated via dual-ascent using a step-length ρ. Intuitively, the dual-ascent update of β penalizes any violation of the constraint in problem (3.4). Next, we make the following interesting observations in these updates:

- **Update for a_i^{t+1}**: When (3.7) is solved to optimality, a_i admits an exact closed-form solution given by,

$$a_i = \frac{1}{\beta_i \sum_{k=1}^{K}\exp(\mathbf{w}_k^T\mathbf{x}_i)}, \tag{3.9}$$

- **Update for β_i^{t+1}**: As a consequence of the above exact solution for a_i, the dual-ascent update for β_i is no longer needed, since the penalty is always zero during such a

projection if β_i is set to a constant equal to 1.

- **Update for W_k^{t+1}**: This is the only update that we need to handle numerically.

$L_2(W, A)$ can be first written in this form,

$$L_2(W, B) = \sum_{i=1}^{N} \sum_{k=1}^{K} \left(\frac{\lambda}{2N} \|\mathbf{w}_k\|^2 - \frac{1}{N} y_{ik} \mathbf{w}_k^T \mathbf{x}_i - \frac{1}{NK} b_i \right.$$
$$\left. + \frac{1}{N} \exp(\mathbf{w}_k^T \mathbf{x}_i + b_i) - \frac{1}{NK} \right) \tag{3.10}$$

where we denote $b_i = \log(a_i)$ for convenience and $B = \{b_i\}_{i=1,\ldots,N}$. The objective function is now *doubly-separable* [Yun, 2014] since,

$$L_2(w_1, \ldots, w_K, b_1, \ldots, b_N) = \sum_{i=1}^{N} \sum_{k=1}^{K} f_{ki}(\mathbf{w}_k, b_i) \tag{3.11}$$

where

$$f_{ki}(\mathbf{w}_k, b_i) = \frac{\lambda}{2N} \|\mathbf{w}_k\|^2 - \frac{y_{ik} \mathbf{w}_k^T \mathbf{x}_i}{N} + \frac{\exp(\mathbf{w}_k^T \mathbf{x}_i + b_i)}{N} - \frac{b_i}{NK} - \frac{1}{NK} \tag{3.12}$$

Obtaining such a form for the objective function is key to achieving simultaneous data and model parallelism. It is worth pointing out that such an objective function can also be derived using the variational form for the log-partition function [Bouchard, 2007].

Stochastic Optimization: Minimizing $L_2(W, B)$ involves computing the gradients of eqn (3.10) w.r.t. $\mathbf{w}_k$ which is often computationally expensive. Instead, one can compute *stochastic gradients* [Robbins and Monro, 1951] which are computationally cheaper than the exact gradient, and perform stochastic updates as follows:

$$\mathbf{w}_k \leftarrow \mathbf{w}_k - \eta_t K \left(\lambda \mathbf{w}_k - y_{ik} \mathbf{x}_i + \exp(\mathbf{w}_k^T \mathbf{x}_i + b_i) \mathbf{x}_i \right) \tag{3.13}$$

where η_t is the learning rate for $\mathbf{w}_k$ in the t-th iteration.

Our reformulation DS-MLR in eqn (3.10) offers some key advantages namely,:

1. The objective function $L_2(W, B)$ splits as summations over N data points and K classes. Therefore, each term in the stochastic updates only depends one data point i and one class k. We exploit this to achieve hybrid parallelism.

2. We are able to update the variational parameters b_i in closed-form, avoiding noisy stochastic updates. This improves our overall convergence.

3. Our formulation lends itself nicely to an asynchronous implementation. Section 3.5.2 describes this in more detail.

3.5 Distributing the Computation of DS-MLR

3.5.1 DS-MLR Synchronous

We first describe the distributed DS-MLR Synchronous algorithm in Algorithm 3. The data and parameters are distributed among the P processors as illustrated in Figure 3.1 where the row-blocks and column-blocks represent data $X^{(p)}$ and weights $W^{(p)}$ on each local processor respectively. The algorithm proceeds by running T iterations in parallel on each of the P workers arranged in a ring network topology.

Each iteration consist of $2P$ inner-epochs. During the first P inner-epochs, each worker sends/receives its parameters $W^{(p)}$ to/from the adjacent machine and performs stochastic $W^{(p)}$ updates using the block of data $X^{(p)}$ and parameters $W^{(p)}$ that it owns. The second P inner-epochs are used to pass around the $W^{(p)}$ to compute the $b^{(p)}$ exactly using

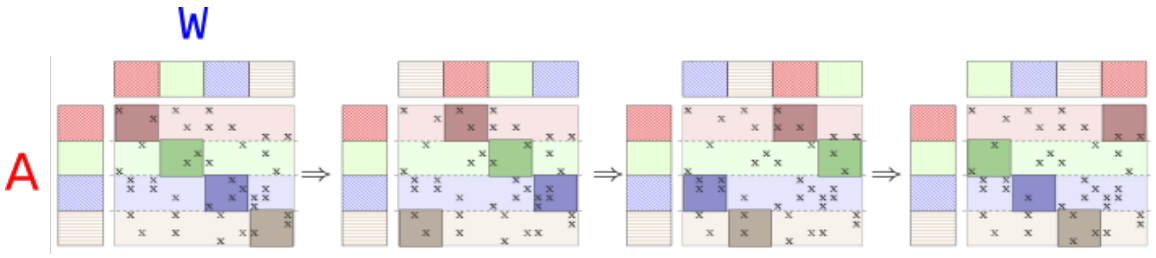


Figure 3.1: $P = 4$ inner-epochs of distributed SGD. Each worker updates mutually-exclusive blocks of data and parameters as shown by the dark colored diagonal blocks [Gemulla et al., 2011].

(3.9).

3.5.2 DS-MLR Asynchronous

The performance of DS-MLR Sync can be significantly improved by performing computation and communication in parallel. Thanks to the double-separable nature of our objective function (3.10), this can be easily achieved by applying the NOMAD algorithm Yun et al. [2013]. The entire DS-MLR Async algorithm is described in Algorithm 4.

The algorithm begins by distributing the data and parameters among P workers in the same fashion as in the synchronous version. However, here we also maintain P worker queues. Initially the parameters $W^{(p)}$ are distributed uniformly at random across the queues. The workers subsequently can run their updates in parallel as follows: each one pops a parameter $\mathbf{w}_k$ out the queue, updates it stochastically and pushes it into the queue of the next worker. Simultaneously, each worker also records the partial sum (the local contribution of each worker towards the global normalization constant $\sum_{k=1}^{K} \exp\left(\mathbf{w}_k^T \mathbf{x}_i\right)$ that is required for updating the variational parameters. This process repeats until K updates have been

Algorithm 3 DS-MLR Synchronous

K: # classes, P: # workers, T: total outer iterations, t: outer iteration index, s: inner epoch index

$W^{(p)}$: weights per worker, $b^{(p)}$: variational parameters per worker

Initialize $W^{(p)} = 0$, $b^{(p)} = \frac{1}{K}$

for all $p = 1, 2, \ldots, P$ in parallel **do**

  for all $t = 1, 2, \ldots, T$ **do**

    for all $s = 1, 2, \ldots, P$ **do**

      Send $W^{(p)}$ to worker on the right

      Receive $W^{(p)}$ from worker on the left

      Update $W^{(p)}$ stochastically using (3.13)

    end for

    for all $s = 1, 2, \ldots, P$ **do**

      Send $W^{(p)}$ to worker on the right

      Receive $W^{(p)}$ from worker on the left

      Compute partial sums

    end for

    Update $b^{(p)}$ exactly (3.9) using the partial sums

  end for

end for

made which is equivalent to saying that each worker has updated every parameter $\mathbf{w}_k$. Following this, the worker updates all its variational parameters $b^{(p)}$ exactly using the partial sums (3.9). For simplicity of explanation, we restricted Algorithm 4 to P workers on a

Algorithm 4 DS-MLR Asynchronous

K: total # classes, P: total # workers, T: total outer iterations, $W^{(p)}$: weights per worker

$b^{(p)}$: variational parameters per worker, queue$[P]$: array of P worker queues

Initialize $W^{(p)} = 0$, $b^{(p)} = \frac{1}{K}$ `//Initialize parameters`

for $k \in W^{(p)}$ **do**

  Pick q uniformly at random

  queue$[q]$.push$((k, \mathbf{w}_k))$ `//Initialize worker queues`

end for

`//Start P workers`

for all $p = 1, 2, \ldots, P$ in parallel **do**

  for all $t = 1, 2, \ldots, T$ **do**

    repeat

      $(k, \mathbf{w}_k) \leftarrow$ queue[p].pop()

      Update $\mathbf{w}_k$ stochastically using (3.13)

      Compute partial sums

      Compute index of next queue to push to: $\hat{q}$

      queue$[\hat{q}]$.push$((k, \mathbf{w}_k))$

    until # of updates is equal to K

    Update $b^{(p)}$ exactly (3.9) using the partial sums

  end for

end for

single-machine. However, in our actual implementation, there are multiple threads running on a single machine in addition to multiple machines sharing the load across the network. Therefore, in this setting, each worker (thread) first passes around the parameter $\mathbf{w}_k$ across all the threads on its machine. Once this is completed, the parameter is tossed onto the queue of the first thread on the next machine.

3.6 Convergence Analysis

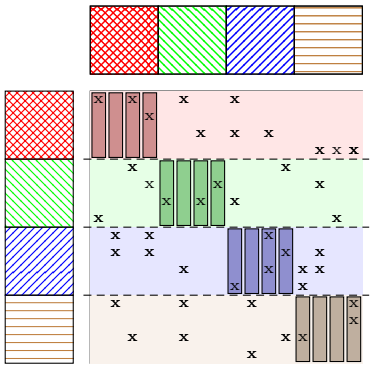
In this section, we present the convergence analysis of DS-MLR. The flavor of stochasticity we use in DS-MLR is *sampling without replacement* [Shamir, 2016], which is also popularly known as *Incremental Gradient Descent* [Nedić and Bertsekas, 2001], [Bertsekas, 2011] and is found to converge faster[1] in practice than vanilla *sampling with replacement* SGD [HaoChen and Sra, 2018]. For the asynchronous case, we make an additional assumption which is a sufficient condition to characterize gradient delays. Such a condition has been widely used to prove convergence of asynchronous SGD algorithms as discussed in [Zhang et al., 2018]. Theorem 1 presents the rate for the synchronous version of DS-MLR.

Theorem 1. *Suppose all* $\|\mathbf{x}_i\| \leq r$ *for a constant* $r > 0$. *Let the step size* η_t *in* (3.13) *decay at the rate of* $\frac{\eta_0}{\sqrt{t}}$ *where* η_0 *is a carefully tuned hyper-parameter. Then, under standard assumptions of smoothness, strong convexity, lipschitz hessian and bounded gradients,*

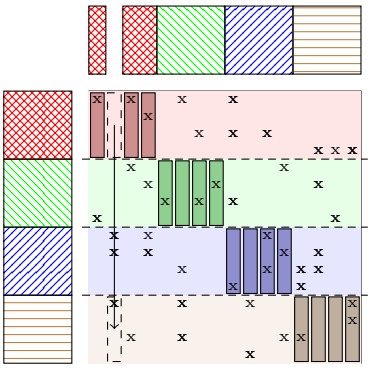
$$\mathbb{E}[\|\mathbf{w}_k^{t,n} - \mathbf{w}_k^*\|^2] \leq \frac{\|\mathbf{w}_k^{1,n} - \mathbf{w}_k^*\|^2 + \frac{2\eta_0^2 M_2}{\eta_0 M_1 - 1}}{\sqrt{t}}, \quad \forall t = \{1, 2, \ldots, T\} \tag{3.14}$$

where $\mathbf{w}_k^{t,n}$ *is value of parameter vector* $\mathbf{w}_k$ *at outer and inner iterations indexed by* t *and* n

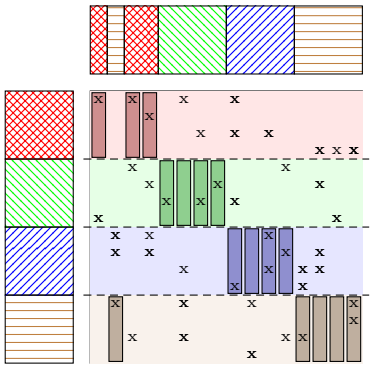
[1] [Bertsekas, 2011] outlines exact conditions under which Incremental Gradient Descent converges namely: diminishing step sizes and choosing indices in a cyclic order, and re-shuffling at the end of cycle. Our implementation of DS-MLR follows these guidelines closely.



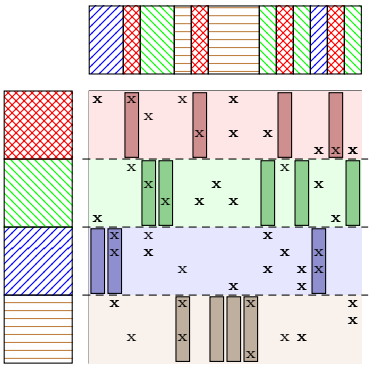
(a) Initial assignment of W and X. Each worker works only on the diagonal active area in the beginning.



(b) After a worker finishes processing column k, it sends the corresponding item parameter $\mathbf{w}_k$ to another worker. Here, $\mathbf{w}_2$ is sent from worker 1 to 4.



(c) Upon receipt, the column is processed by the new worker. Here, worker 4 can now process column 2 since it owns the column.



(d) During the execution of the algorithm, the ownership of the global parameters (weight vectors) $\mathbf{w}_k$ changes.

Figure 3.2: Illustration of the communication pattern in DS-MLR Async algorithm. Parameter vector $\mathbf{w}_k$ is exchanged in a de-centralized manner across workers without the use of any parameter servers [Li et al., 2013].

respectively, $\mathbf{w}_k^*$ *is the optimal solution,* $\mathbf{x}_i$ *denotes the data point,* $M_1 = nC_4$ *and* $M_2 = n^2C_5$ *(where constants* C_4 *and* C_5 *depend on* L *and* μ*).*

Proof is available in Appendix **??**. The key steps in our analysis are as follows:

- *First*, for the t-th iteration, we introduce a random variable R^t to absorb the effects of re-shuffling the indices within the epoch by closely following results in [HaoChen and Sra, 2018].
- *Second*, to account for the delay and staleness in updates for $\mathbf{w}_k$ in the inner-iterations, we prove and make use of Lemma **??** to bound the staleness in ∇f_i.
- *Finally*, we prove our main result using a proof by induction (see Lemma **??**) argument revealing the $\frac{1}{\sqrt{t}}$ rate.

Remark: Our analysis can also be easily adapted to prove $\frac{1}{t}$ rate using step-size of $\frac{\eta_0}{t}$. However, in practice, we found $\frac{\eta_0}{\sqrt{t}}$ to be slightly more stable. Using an assumption on boundedness of the delay, we can use of results in [Zhang et al., 2018] to achieve $\frac{1}{\sqrt{t}}$ rates for DS-MLR Async for a suitable diminishing step-size sequence.

3.7 Experiments

In our empirical study, we analyze the behavior of DS-MLR Async by running it on several real-world datasets of varying scale. Table 5.2 provides a summary of their characteristics.

Hardware: All single-machine experiments were run on a cluster with the configuration of two 8-core Intel Xeon-E5 processors and 32 GB memory per node. For multi-machine

Dataset	# instances	# features	#classes	data (train + test)	parameters	density (% nnz)
CLEF	10,000	80	63	9.6 MB + 988 KB	40 KB	100
NEWS20	11,260	53,975	20	21 MB + 14 MB	9.79 MB	0.21
LSHTC1-small	4,463	51,033	1,139	11 MB + 4 MB	465 MB	0.29
LSHTC1-large	93,805	347,256	12,294	258 MB + 98 MB	34 GB	0.049
ODP	1,084,404	422,712	105,034	3.8 GB + 1.8 GB	355 GB	0.0533
YouTube8M-Video	4,902,565	1,152	4,716	59 GB + 17 GB	43 MB	100
Reddit-Small	52,883,089	1,348,182	33,225	40 GB + 18 GB	358 GB	0.0036
Reddit-Full	211,532,359	1,348,182	33,225	159 GB + 69 GB	358 GB	0.0036

Table 3.3: Characteristics of the datasets used

multi-core, we used Intel vLab Knights Landing (KNL) cluster with node configuration of Intel Xeon Phi 7250 CPU (64 cores, 200GB memory), connected through Intel Omni-Path (OPA) Fabric.

Baselines and Implementation Details: We implemented DS-MLR in C++ using MPI for communication across nodes and Intel TBB for concurrent queues and multi-threading. Although, there exist numerous data and model parallel methods, we use L-BFGS and LC Gopal and Yang [2013] as representative baselines. To make the comparison fair, we re-implemented the LC method in C++ and MPI using ALGLIB for inner optimization. For the L-BFGS baseline, we used the TAO solver (from PETSc).

Reproducibility: The hyper-parameter values and node configuration used in our experiments are in Table 5.2. Code and scripts required for reproducing the experiments are readily available for download from https://bitbucket.org/params/dsmlr. The repository

includes instructions to compile and run the code and scripts to launch the jobs on a HPC cluster with similar capability as ours.

3.7.1 Comparison with other methods

SMALL SCALE DATASETS:

For this experiment, we compare DS-MLR, L-BFGS and the LC methods on small scale datasets **CLEF, NEWS20, LSHTC1-small** which can easily fit in the memory of a single machine and therefore require no parallelism. In such scenarios, a second order methods

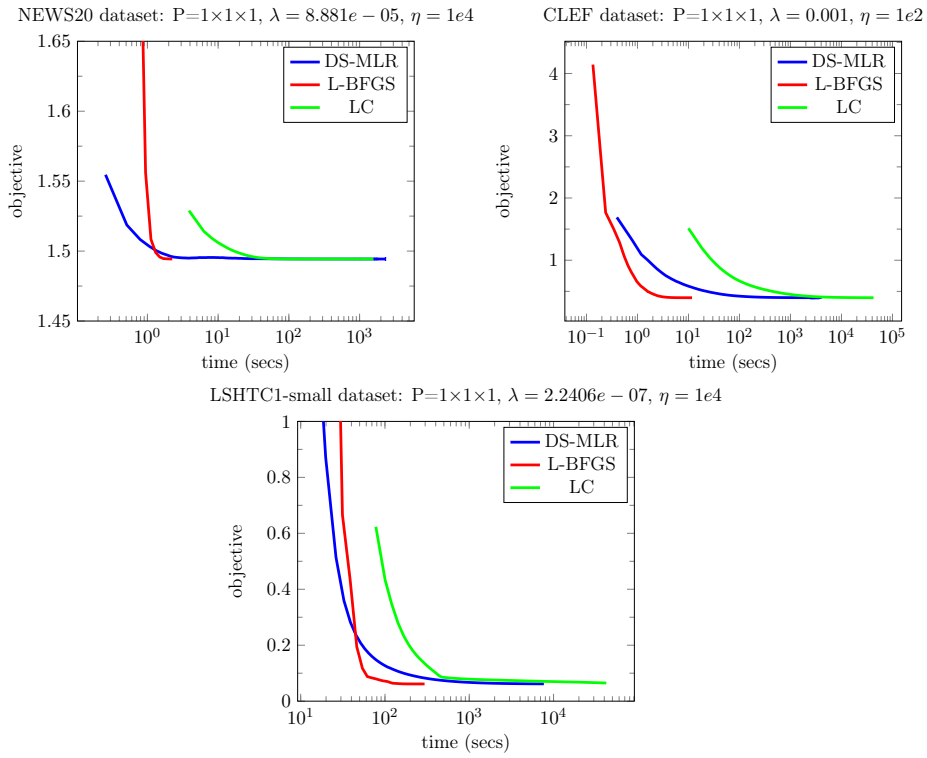


Figure 3.3: **Data and Model both fit in memory**. In each plot, P=N×M×T denotes that there are N nodes each running M mpi tasks, with T threads each. λ and η refer to regularization and learning-rate.

such as L-BFGS are theoretically expected to out-perform stochastic methods due their superior quadratic convergence rates. In our experiments, we observed that it is indeed the case. When comparing DS-MLR against LC (which is our model parallel baseline method) we found that DS-MLR consistently shows a faster decrease in objective value compared to LC on all three datasets: NEWS20, LSHTC1-small and CLEF. LC stalls towards the end and progresses very slowly as seen in the plots. Figure 3.3 shows the progress of objective function as a function of time for DS-MLR, L-BFGS and LC on the three datasets.

LARGE SCALE DATASETS:

LSHTC1-large: L-BFGS requires all its parameters to fit on one machine and is therefore not suited for model parallelism (even a modest dataset such as LSHTC1-large requires $\approx$ 4.2 billion parameters or $\approx$ 34GB). Thus, parallelizing L-BFGS would involve duplicating 34 GB of parameters across all its processors. We ran both DS-MLR and LC using 48 workers. Figure 3.4 (left) shows how the objective function changes vs time for DS-MLR and LC. As can be seen, DS-MLR out performs LC by a wide-margin despite the advantage LC has by duplicating data across all its processors.

ODP: We ran DS-MLR on ODP dataset [2] which has a huge model parameter size of 355 GB. For this experiment we used 20 nodes $\times$ 1 mpi task $\times$ 260 threads. The progress in decreasing the objective function value is shown in Figure 3.4 (right). LC method being a second-order method has a very high per-iteration cost and it takes an enormous amount of time to finish even a single iteration.

[2] https://github.com/JohnLangford/vowpal_wabbit/tree/master/demo/recall_tree/odp

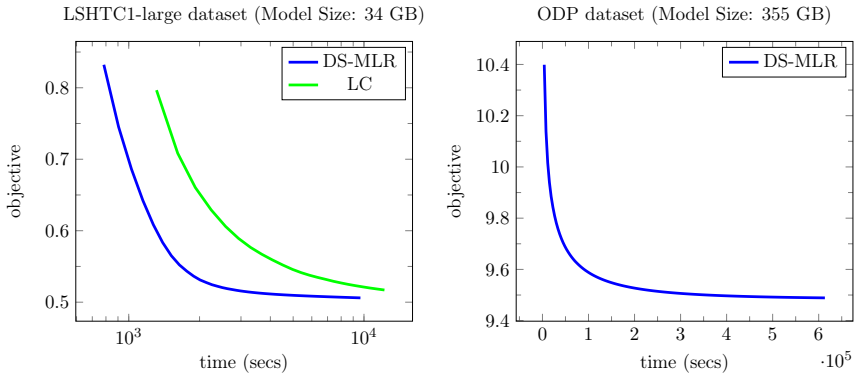


Figure 3.4: **Data fits and Model does not fit**.

YouTube8M-Video: This dataset was created by pre-processing the publicly available dataset of youtube video embeddings [3] into a multi-class classification dataset consisting of 4,716 classes and 1,152 features. Since it was created from features derived from embeddings, it is a dense dataset. We used the configuration of 4 nodes × 1 mpi tasks ×

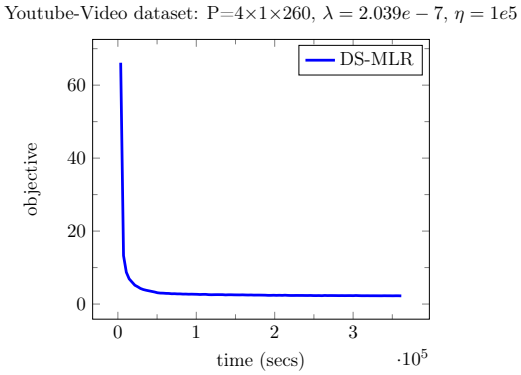


Figure 3.5: **Data does not fit and Model fits**.

260 threads to run DS-MLR on this dataset and we observed a fast convergence as shown in Figure 3.5. This is likely because DS-MLR being non-blocking and asynchronous in nature runs at its peak performance on a dense dataset such as YouTube8M-Video, since the number

[3] https://research.google.com/youtube8m/

of non-zeros in the data remains uniform across all its workers.

Reddit datasets: In this sub-section, we demonstrate the capability of DS-MLR to solve a multi-class classification problem of massive scale, on a new benchmark dataset *RedditFull* which we created out of 1.7 billion reddit user comments spanning the period 2007-2015. Our aim is to classify a particular reddit comment into a suitable sub-reddit.

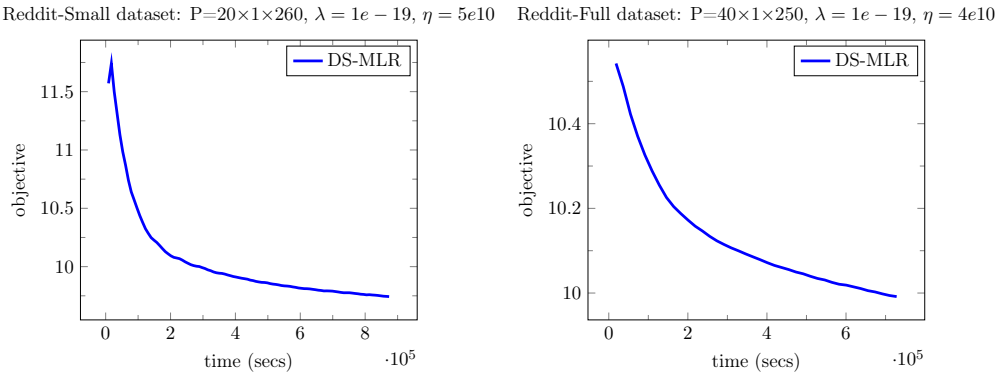


Figure 3.6: **Data does not fit and Model does not fit**.

The data and model parameters occupy 228 GB and 358 GB respectively. Therefore, both L-BFGS and LC cannot be applied here. We also created a smaller subset of this dataset *Reddit-Small* by sub-sampling around 50 million data points. The result of running DS-MLR on these two datasets are shown in Figure 3.6.

3.7.2 Predictive performance

In this section, we plot the cumulative distribution function (CDF) of ranks of test labels. This is a proxy for the *precision@k* curve and gives a more closer indication of the predictive performance of a multinomial classification algorithm. In Figure 3.7, we plot the precision obtained after the first 5 iterations (denoted by dashed lines), and after the

end of optimization (denoted by solid lines). As seen, DS-MLR performs competitively well compared to other methods in all datasets, and in general tends to give a good accuracy within the first 5 iterations. *Using roughly $\frac{1}{4}$ top-k classes was enough to get a predictive performance of around 95% in all datasets.*

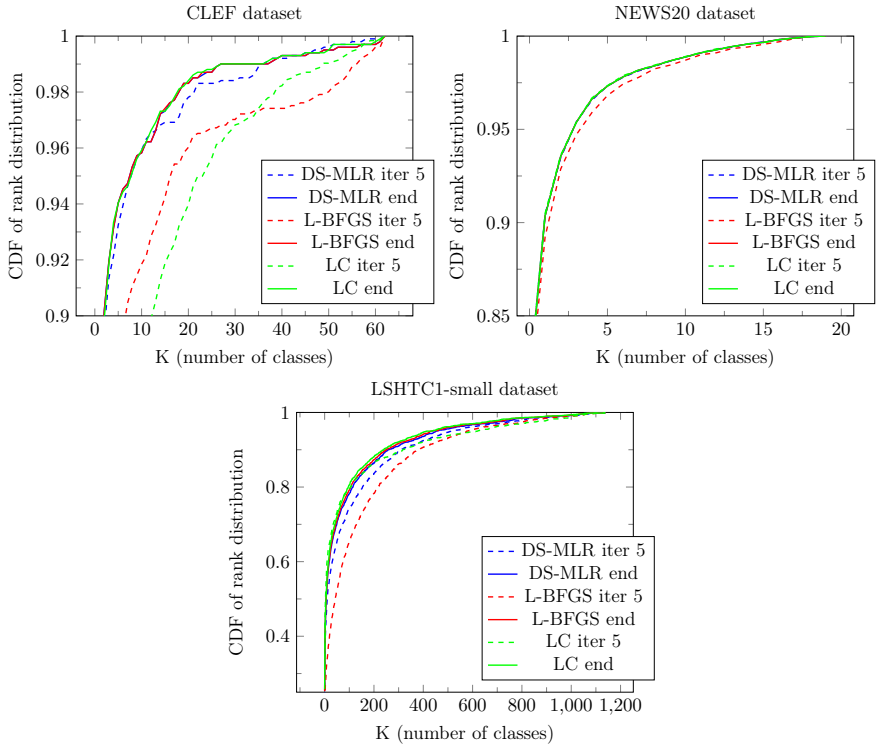


Figure 3.7: Cumulative distribution function (CDF) of predictive ranks of the test labels for three sample datasets. DS-MLR performs competitively well within the first 5 iterations. Using roughly $\frac{1}{4}$ top-k classes was enough to get a predictive performance of around 95% in all datasets.

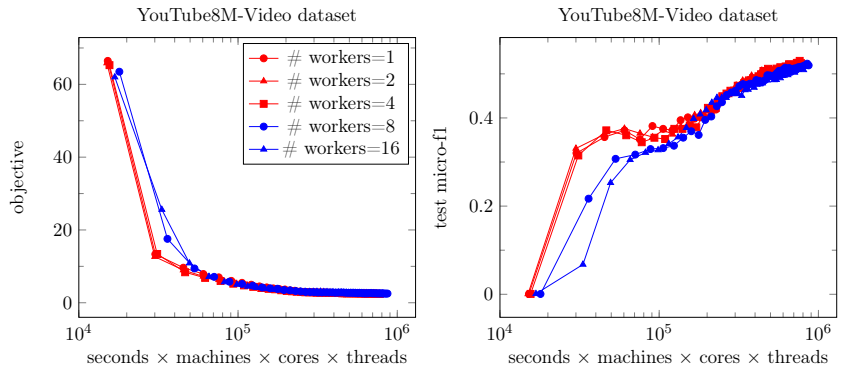


Figure 3.8: Scalability analysis of DS-MLR on YouTube8M-Video dataset: Change in objective function and test f1-score vs computation time varying the # of workers (machines).

3.7.3 Scalability

In Figures 3.8 and 3.9, we analyze the scaling behavior of DS-MLR under the settings of multi-machine and multi-thread parallelism. We picked a dataset for each of these scenarios: YouTube8M-Video and LSHTC1-large respectively. We plot the rate of change in objective function as well as the f-score as the number of workers (# machines × # cores × # threads) is varied. For YouTube8M-Video dataset, we vary the number of machines as $1, 2, 4, 8, 16$. For LSHTC1-large, DS-MLR can handle this dataset on a single machine, therefore, we simply vary the number of threads on a single machine (as a single mpi task) as $1, 2, 4, 8, 16, 20$. In an ideal scenario with linear scaling, we would expect all the figures to overlap with each other. From the plot we observe that multi-thread behavior is pretty close to the ideal behavior while in multi-machine case there is some slowdown with 8 and 10 workers. This is most likely due to the communication and network overheads in the cluster.

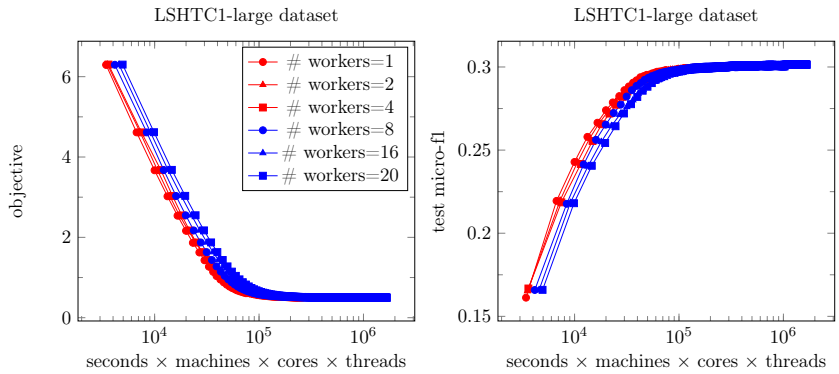


Figure 3.9: Scalability analysis of DS-MLR on LSHTC1-large dataset - Change in objective function and test f1-scores vs computation time varying the # of workers (threads).

3.8 Conclusion

In this paper, we present a novel distributed stochastic optimization algorithm DS-MLR to solve multinomial logistic regression problems having large number of examples and classes. By exploiting double-separability, we present a reformulation that is hybrid parallel (both data and model parallel simultaneously). DS-MLR is able to perfectly partition the workload across P workers, costing $O(\frac{ND}{P})$ storage for data and $O(\frac{KD}{P} + \frac{N}{P})$ for the model. As a result, DS-MLR can scale to arbitrarily large datasets. DS-MLR is fully de-centralized unlike the parameter-server architecture. Parameter updates are directly exchanged asynchronously across workers, eliminating the need for any intermediate servers. We provide empirical results showing DS-MLR applies to all regimes of distributed machine learning, especially the case where both data and model sizes exceed the memory capacity of a single machine. To show this, we created a benchmark dataset (Reddit-Full) to run extreme multi-class classification with 228 GB data and 358 GB parameters. Future directions of work include topics such as extreme multi-label classification [Agrawal et al., 2013], [Jain

et al., 2016] and log-linear parameterization for undirected graphical models which exhibit similar computational challenges.

Chapter 4

Mixture of Exponential Families

4.1 Introduction

Mixture of exponential family models generalize a wide collection of popular latent variable models such as *Latent Dirichlet Allocation (LDA)*, *Gaussian Mixture Models (GMM)*, and *Mixed Membership Stochastic Block Models (MM-SBM)*. In recent years, variational inference (VI) has emerged as a powerful technique for parameter estimation in these Bayesian models Wainwright and Jordan [2008], Blei et al. [2016]. One attractive property of VI is that it reduces parameter estimation to the task of optimizing a objective function, often with a well defined "structure". This opens up the possibility of bringing to bear mature tools from optimization to tackle massive problems. Traditionally, VI in mixture models involves alternating between updating *global variables* and *local variables*. Both these operations involve accessing all the data points. Large datasets are usually stored on disk, and the cost of accessing every datapoint to perform updates is prohibitively high.

The first approach to tackle this, is to divide the data across multiple machines and

use a distributed framework such as map-reduce to aggregate the computations [Neiswanger et al., 2015]. The second approach, is to exploit the underlying structure of the optimization problem to reduce the number of iterations (and therefore the corresponding data access) [Hoffman et al., 2013]. The key observation here is that the optimization problem corresponding to the local variables is *separable*, that is, it can be written as a sum of functions, where each function only depends on one data point. Therefore, one can use stochastic optimization to update the local variables. Moreover, in Stochastic Variational Inference (SVI) [Hoffman et al., 2013], even before one pass through the dataset, the global variables are updated multiple times, and therefore the model parameters converge rapidly towards their final values. The argument is similar in spirit to how stochastic optimization outperforms batch algorithms for maximum aposteriori (MAP) estimation Bottou and Bousquet [2011]. Consequently, SVI enabled applying variational inference to datasets with millions of documents such as *Nature* and *NewYork Times* Hoffman et al. [2013], which could not be handled before.

what fits in memory	*Data, Parameters*	*Parameters*	*Data*	*None*
Distributed-VI	✓	✓	✗	✗
SVI	✓	✗	✗	✗
ESVI	✓	✓	✓	✓

Table 4.1: Applicability of the three bayesian inference algorithms - Variational Inference (VI), Stochastic Variational Inference (SVI) and Extreme Stochastic Variational Inference (ESVI) to common scenarios in distributed machine learning.

With the advent of the big-data era, we now routinely deal with industry-scale problems involving billions of documents and tokens. Such massive datasets pose another

challenge, which, unfortunately neither VI nor SVI are able to address; namely, the set of parameters is so large that *all the parameters do not fit* on a single processor[1]. For instance, if we have D dimensional data and K mixture components, then the parameter size is $O(DK)$. If D is of the order of millions and K is in the 100's or 1000's, modest numbers by todays standards, the parameter size is a few 100s of GB (see our experiments in Section 5.5).

In this paper, we propose a new framework, Extreme Stochastic Variational Inference (ESVI) to address these challenges. The main contributions of this paper are:

1. We develop a novel approach to achieve *simultaneous data and model parallelism* in *mixture models* by exploiting the following key idea: instead of updating all the K coordinates of a local variable and then updating all K global variables, we propose updating a small subset of the local variables and the corresponding global variables (see Theorem 2 for proof of correctness). The global variables are exchanged across the processors, and this ensures mixing (see Section 4.4.2 for more details). This seemingly simple idea has some powerful consequences. It allows multiple processors to *simultaneously perform parameter updates independently.*

2. Using a classic owner-computes paradigm, we make ESVI *asynchronous* and *lock-free*, and thus avoid expensive bulk synchronization between processors. We present an extensive empirical study to evaluate the performance of ESVI by applying it to GMM and LDA models on several large real-world datasets. We find that ESVI *outperforms VI and SVI both in terms of time as well as the quality of solutions obtained.* For practitioners, ESVI offers the advantage of *not having to tune a learning rate*, since it

[1]The discussion in this paper applies to the shared memory, distributed memory, as well as hybrid settings, and therefore we will use the term processor to denote either a thread or a machine.

makes closed form parameter updates unlike SVI.

3. We develop a variant ESVI-LDA-TOPK to *speed up computation and save memory when fitting large number of topics*. Empirically, we found that using the most frequent 25% of the topics was enough to obtain the same level of accuracy as storing all the topics.

To the best of our knowledge there is no existing algorithm for VI that sports these desirable properties. Although, in principle, ESVI is applicable even when data and/or model parameters fit in memory, it truly shines for massive datasets where both model and data parallelism are essential.

The rest of the paper is structured as follows: We present an exhaustive study of related work in Section 4.2. We briefly review VI and SVI in Section 4.3. We present our new algorithm ESVI in Section 4.4, and discuss its advantages. Empirical evaluation is presented in Section 5.5 and Section 4.6 concludes the paper.

4.2 Related Work

Recent research on VI has focused on extending it to non-conjugate models [Wang and Blei, 2013] and developing variants that can scale to large datasets such as SVI [Hoffman et al., 2013]. Other than the fact that SVI is inherently serial, it also suffers from another drawback: storage of the entire $D \times K$ matrix θ on a single machine. On the other hand, our method, ESVI, exhibits model parallelism; each processor only needs to store $1/P$ fraction of θ. Black-box variational inference (BBVI) [Ranganath et al., 2014] generalizes SVI beyond conditionally conjugate models. The paper proposes a more generic framework by

observing that the expectation in the ELBO can be exploited directly to perform stochastic optimization. We view this line of work as complementary to our research. It would be interesting to verify if an ESVI like scheme can also be applied to BBVI.

There has been a flurry of work in the past few years in developing data-parallel distributed methods for Approximate Bayesian Inference. One such popular work includes a classic Map-Reduce style inference algorithm [Neiswanger et al., 2015], where the data is divided across several worker nodes and each of them perform VI updates in parallel until a final synchronization step during which the parameters from the slaves are combined to produce the final result. This method suffers from the well-known *curse of the last reducer*, that is, a single slow machine can dramatically slow down the performance. *ESVI does not suffer from this problem*, because our asynchronous and lock-free updates avoid bulk synchronization altogether.

[Broderick et al., 2013] presents an algorithm that applies VI to the streaming setting by performing asynchronous Bayesian updates to the posterior as batches of data arrive continuously, which is similar in spirit to Hogwild [Recht et al., 2011]. Their approach uses a parameter server to enable asynchronous local updates. Unlike ESVI, their work cannot guarantee that - (a) each worker works on the latest parameters, (b) the global parameters are all parallely updated. In [Archambeau and Ermis, 2015] the authors present *Incremental Variational Inference* which is also a distributed variational inference algorithm, however it is also only data-parallel. Besides, it requires tuning of a step-size and sequential access of global parameters. *ESVI avoids these drawbacks.*

A number of data-parallel approaches exist in the Exact Bayesian Inference literature

as well. [Hasenclever et al., 2017] is a distributed MCMC based approach where workers perform MCMC updates locally and these are aggregated by maintaining a posterior server. [Yu et al., 2015a] proposed a distributed asynchronous algorithm for parameter estimation in LDA Blei et al. [2003]. However, the algorithm is specialized to collapsed Gibbs sampling for LDA, and it is unclear how to extend it to other, more general, mixture models. *ESVI in contrast is a purely VI based method and provides model-parallelism in addition to data-parallelism.*

Somewhat close to our ESVI-TOPK approach is *Memoized Online Variational Inference for DP Mixture Models* [Hughes and Sudderth, 2013]. This paper describes the application of Expectation Truncation to mixture models. In their L-sparse method, unused dimensions are set to zero and used dimensions are shifted. In ESVI-TOPK, unused dimensions are averaged (1-sum of used dimensions).

Another related line of work is *Sparse EM* [Neal and Hinton, 1998]. There are some high-level similarities to ESVI in that both the methods update a subset of latent variables at any given time while keeping others frozen. However there are some crucial differences: (a) Sparse EM is not a parallel algorithm while ESVI is, (b) Sparse EM needs to iterate between sparse EM update and full EM update (to select active dimensions occasionally) while each ESVI worker's job queue will continuously distribute Z's dimensions to ensure a good mixing, (c) Sparse EM selects active dimensions based on values of Z, while ESVI is designed to ensure the active dimensions of each worker is an unbiased sample of all dimensions.

Since the coordinate-ascent algorithm in VI can be formulated as a message passing scheme applied to general graphical models, we believe ESVI is also related to Variational

Message Passing [Winn and Bishop, 2005]. This connection could be made more concrete if we assume a Mixture Model setup in both cases. Both the d-VMP algorithm (Algorithm 2 in [Masegosa et al., 2017]) and ESVI de-couple the global parameters to make the updates scalable, however they differ in some fundamental aspects. d-VMP defines a disjoint partitioning of the global parameters based on their markov-blankets. In contrast, ESVI completely decentralizes the global parameter updates by requiring that the local variables (or assignment vector z_i) need to only satisfy local summation constraints (as discussed in Theorem 2 in Section 4.4). As a side-note, the local updates in d-VMP algorithm do not seem to be de-coupled across the mixture components, whereas this holds true in the case of ESVI.

4.3 Parameter Estimation for Mixture of Exponential Families

4.3.1 Generative Model

The following data generation scheme underlies a mixture of exponential family model (Table 4.2 defines the notations):

Prior:

$$p(\pi|\alpha) = \text{Dirichlet}(\alpha) \tag{4.1}$$

$$p(\theta_k|n_k, \nu_k) = \exp\left(\langle n_k \cdot \nu_k, \theta_k \rangle - n_k \cdot g(\theta_k) - h(n_k, \nu_k)\right) \tag{4.2}$$

where, n_k and ν_k are the parameters of the *conjugate prior* $p(\theta_k|n_k, \nu_k)$.

Symbol	Definition
N	total number of observations
D	total number of dimensions
K	total number of mixture components
Δ_K	K-dimensional simplex
$x = \{x_1, \ldots, x_N\}, \quad x_i \in \mathbb{R}^D$	observations
$z = \{z_1, \ldots, z_N\}, \quad z_i \in \Delta_K$	the component data point x_i was drawn from (local variable)
$\theta = \{\theta_1, \ldots, \theta_K\}, \quad \theta_k \in \mathbb{R}^D$	sufficient statistics of the exponential family distribution (global variable)
$\pi \in \Delta_K$	mixing coefficients (global variable)
$\tilde{z} = \{\tilde{z}_1, \ldots, \tilde{z}_N\}, \quad \tilde{z}_i \in \Delta_K$	variational parameter for z (local variable)
$\tilde{\theta} = \left\{\tilde{\theta}_1, \ldots, \tilde{\theta}_K\right\}, \quad \tilde{\theta}_k \in \mathbb{R}^D$	variational parameter for θ (global variable)
$\tilde{\pi} \in \Delta_K$	variational parameter for π (global variable)

Table 4.2: Notations for Mixture of Exponential Family Model. $\sim$ denotes variational parameters.

Likelihood:

$$p\left(z_i|\pi\right) = \text{Multinomial}\left(\pi\right) \tag{4.3}$$

$$p\left(x_i|z_i,\theta\right) = \exp\left(\left\langle\phi\left(x_i,z_i\right),\theta_{z_i}\right\rangle - g\left(\theta_{z_i}\right)\right) \tag{4.4}$$

where, ϕ denotes the sufficient statistics. Observe that $p\left(\theta_k|n_k,\nu_k\right)$ is *conjugate* to $p\left(x_i|z_i=k,\theta_k\right)$, while $p\left(\pi|\alpha\right)$ is *conjugate* to $p\left(z_i|\pi\right)$.

Joint Distribution:

$$p\left(x,\pi,z,\theta|\alpha,n,\nu\right) = p\left(\pi|\alpha\right)\cdot\prod_{k=1}^{K}p\left(\theta_k|n_k,\nu_k\right)\cdot\prod_{i=1}^{N}p\left(z_i|\pi\right)\cdot p\left(x_i|z_i,\theta\right) \tag{4.5}$$

4.3.2 Variational Inference and Stochastic Variational Inference

The goal of inference is to estimate $p\left(\pi,z,\theta|x,\alpha,n,\nu\right)$. This however, involves an intractable marginalization. Therefore, variational inference [Blei et al., 2016] approximates this distribution with a fully factorized distribution [2]:

$$q\left(\pi,z,\theta|\tilde{\pi},\tilde{z},\tilde{\theta}\right) = q\left(\pi|\tilde{\pi}\right)\cdot\prod_{i=1}^{N}q\left(z_i|\tilde{z}_i\right)\cdot\prod_{k=1}^{K}q\left(\theta_k|\tilde{\theta}_k\right). \tag{4.6}$$

Note that $\tilde{z}_i\in\Delta_K$ and $z_{i,k}=q\left(z_i=k|\tilde{z}_i\right)$. Moreover, each of the factors in the variational distribution is assumed to belong to the same exponential family as their full conditional counterparts in (4.5). The variational parameters are estimated by maximizing the following evidence lower-bound (ELBO) [Blei et al., 2016]:

$$\begin{aligned}\mathcal{L}\left(\tilde{\pi},\tilde{z},\tilde{\theta}\right) = {}& \mathbb{E}_{q\left(\pi,z,\theta|\tilde{\pi},\tilde{z},\tilde{\theta}\right)}\left[\log p\left(x,\pi,z,\theta|\alpha,n,\nu\right)\right] \\ & -\mathbb{E}_{q\left(\pi,z,\theta|\tilde{\pi},\tilde{z},\tilde{\theta}\right)}\left[\log q\left(\pi,z,\theta|\tilde{\pi},\tilde{z},\tilde{\theta}\right)\right]\end{aligned} \tag{4.7}$$

[2] A $\sim$ over a symbol is used to denote that it is a parameter of the variational distribution. See Table 5.1.

VI performs coordinate ascent updates on $\mathcal{L}$ by optimizing each set of variables, one at a time.

Update for $\tilde{\pi}_k$:

$$\tilde{\pi}_k = \alpha + \sum_{i=1}^{N} \tilde{z}_{i,k} \tag{4.8}$$

Update for $\tilde{\theta}_k$: The components of $\tilde{\theta}_k$ namely $\tilde{n}_k$ and $\tilde{\nu}_k$ are updated as follows:

$$\tilde{n}_k = n_k + N_k \tag{4.9}$$

$$\tilde{\nu}_k = n_k \cdot \nu_k + N_k \cdot \bar{x}_k \tag{4.10}$$

where $N_k = \sum_{i=1}^{N} \tilde{z}_{i,k}$ and $\bar{x}_k = \frac{1}{N_k} \sum_{i=1}^{N} \tilde{z}_{i,k} \cdot \phi(x_i, k)$.

Update for $\tilde{z}_i$: Let u_i be a K dimensional vector whose k-th component is given by

$$u_{i,k} = \psi(\tilde{\pi}_k) - \psi\left(\sum_{k'=1}^{K} \tilde{\pi}_{k'}\right) + \left\langle \phi(x_i, k), \mathbb{E}_{q(\theta_k|\tilde{\theta}_k)}[\theta_k] \right\rangle - \mathbb{E}_{q(\theta_k|\tilde{\theta}_k)}[g(\theta_k)] \tag{4.11}$$

$$\tilde{z}_{i,k} = \frac{\exp(u_{i,k})}{\sum_{k'=1}^{K} \exp(u_{i,k'})} \tag{4.12}$$

It has to be noted that the summation term $\psi\left(\sum_{k'=1}^{K} \tilde{\pi}_{k'}\right)$ cancels out during the $\tilde{z}_{i,k}$ update in (4.12).

The VI algorithm [Wainwright and Jordan, 2008, Blei et al., 2016] iteratively updates all the local variables $\tilde{z}_i$ before updating the global variables $\tilde{\pi}_k$ and $\tilde{\theta}_k$ (see Algorithm 5). In contrast, SVI [Hoffman et al., 2013], updates $\tilde{z}_i$ corresponding to one data point x_i, followed by updating the global parameters $\tilde{\pi}$ and $\tilde{\theta}$ (see Algorithm 6).

Algorithm 5 VI

for $i = 1, \ldots, N$ **do**

  Update $\tilde{z}_i$ using (4.12)

end for

for $k = 1, \ldots K$ **do**

  Update $\tilde{\pi}_k$ using (4.8)

  Update $\tilde{\theta}_k$ using (4.9) and (4.10)

end for

Algorithm 6 SVI

Generate step size sequence $\eta_t \in (0, 1)$

Pick an $i \in \{1, \ldots, N\}$ uniformly at random

  Update $\tilde{z}_i$ using (4.12)

  for $k = 1, \ldots K$ **do**

    Update $\tilde{\pi}_k \leftarrow (1 - \eta_t)\tilde{\pi}_k + \eta_t \left(\alpha + N \cdot \tilde{z}_{i,k}\right)$

    $\hat{\tilde{\theta}}_k = \{n_k + N \cdot \tilde{z}_i,\ n_k \cdot \nu_k + N \cdot \tilde{z}_{i,k} \cdot \phi\left(x_i, k\right)\}$

    Update $\tilde{\theta}_k \leftarrow (1 - \eta_t)\tilde{\theta}_k + \eta_t \hat{\tilde{\theta}}_k$

  end for

4.4 Extreme Stochastic Variational Inference (ESVI)

Algorithm 7 ESVI

1: Sample $i \in \{1, \ldots, N\}$

2: Select $\mathcal{K} \subset \{1, \ldots, K\}$

3: Update $\tilde{z}_{i,k}$ for all $k \in \mathcal{K}$ (see below)

4: Update $\tilde{\pi}_k$ for all $k \in \mathcal{K}$ using (4.8)

5: Update $\tilde{\theta}_k$ for all $k \in \mathcal{K}$ using (4.9), (4.10)

Algorithm 7 illustrates our proposed updates in ESVI. Before we discuss why this update is advantageous for parallelization, let us first study why updating a subset of coordinates of $\tilde{z}_i$ is justified. Plugging in (4.5) and (4.6) into (4.7), and restricting our attention to the terms in the above equation which depend on z_i, substitute (4.4), $\tilde{z}_{i,k} = q(z_i = k|\tilde{z}_i)$ and $\mathbb{E}_{q(\pi|\tilde{\pi})}[\log p(z_i = k|\pi)] = \psi(\tilde{\pi}_k) - \psi\left(\sum_{k'=1}^{K} \tilde{\pi}_{k'}\right)$, to obtain the following objective function,

$$\mathcal{L}\left(\tilde{z}_i|\tilde{\pi}, \tilde{\theta}\right) = \sum_{k=1}^{K} \tilde{z}_{i,k} \cdot \left(\psi(\tilde{\pi}_k) - \psi\left(\sum_{k'=1}^{K} \tilde{\pi}_{k'}\right)\right) + \sum_{k=1}^{K} \tilde{z}_{i,k} \cdot \Big(\Big\langle \phi(x_i, k), \mathbb{E}_{q(\theta_k|\tilde{\theta}_k)}[\theta_k] \Big\rangle - \mathbb{E}_{q(\theta_k|\tilde{\theta}_k)}[g(\theta_k)] - \log \tilde{z}_{i,k}\Big). \tag{4.13}$$

Now using the definition of $u_{i,k}$ in (4.11), one can compactly rewrite the above objective function as

$$\mathcal{L}\left(\tilde{z}_i|\tilde{\pi}, \tilde{\theta}\right) = \sum_{k=1}^{K} \tilde{z}_{i,k} \cdot (u_{i,k} - \log \tilde{z}_{i,k}). \tag{4.14}$$

Moreover, to ensure that $\tilde{z}_{i,k}$ is a valid distribution, one needs to enforce the

following constraints:

$$\sum_k \tilde{z}_{i,k} = 1, \quad 0 \leq \tilde{z}_{i,k} \leq 1. \tag{4.15}$$

Theorem 2 shows that one can find a closed form solution to maximizing (4.14) even if we restrict our attention to a subset of coordinates.

Theorem 2. *For $2 \leq K' \leq K$, let $\mathcal{K} \subset \{1, \ldots, K\}$ be s.t. $|\mathcal{K}| = K'$. For any $C > 0$, the problem*

$$\begin{aligned} \max_{z_i \in \mathbb{R}^{K'}} \quad & \mathcal{L}_{\mathcal{K}} = \sum_{k \in \mathcal{K}} \tilde{z}_{i,k} \cdot u_{i,k} - \tilde{z}_{i,k} \cdot \log \tilde{z}_{i,k} \\ s.t. \quad & \sum_{k \in \mathcal{K}} \tilde{z}_{i,k} = C \quad \text{and} \quad 0 \leq \tilde{z}_{i,k}, \end{aligned} \tag{4.16}$$

has the closed form solution:

$$\tilde{z}^*_{i,k} = C \frac{\exp\left(u_{i,k}\right)}{\sum_{k' \in \mathcal{K}} \exp\left(u_{i,k'}\right)}, \text{ for } k \in \mathcal{K}. \tag{4.17}$$

Proof We prove that $\tilde{z}^*_i$ is a stationary point by checking the KKT conditions for (4.16). Let $h\left(\tilde{z}_i\right) = \left(\sum_{k \in \mathcal{K}} \tilde{z}_{i,k}\right) - C$ and $g_k\left(\tilde{z}_i\right) = -z_{i,k}$. It is clear that $\tilde{z}^*_i$ satisfies the primal feasibility. Now consider KKT multipliers:

$$\lambda = \log \frac{C}{\sum_{k' \in \mathcal{K}} \exp\left(u_{i,k'}\right)}, \text{ and } \mu_k = 0.$$

We have

$$\begin{aligned}
\nabla_k \mathcal{L}_{\mathcal{K}}\left(\tilde{z}_i^*\right) &= u_{i,k} - \log(\tilde{z}_{i,k}^*) - 1 \\
&= u_{i,k} - \left(u_{i,k} + \log \frac{C}{\sum_{k' \in \mathcal{K}} \exp(u_{i,k'})}\right) \\
&= \log \frac{C}{\sum_{k' \in \mathcal{K}} \exp(u_{i,k'})} \\
\lambda \nabla_k h(\tilde{z}_i^*) &= \log \frac{C}{\sum_{k' \in \mathcal{K}} \exp(u_{i,k'})} \\
\mu_k \nabla_k g_{k'}(\tilde{z}_i^*) &= 0.
\end{aligned}$$

Then it is easy to verify that $\nabla_k \mathcal{L}_{\mathcal{K}}(\tilde{z}_i^*) = \lambda = \lambda \nabla_k h(\tilde{z}_i^*)$. Thus, $\tilde{z}_i^*$ satisfies the stationarity condition:

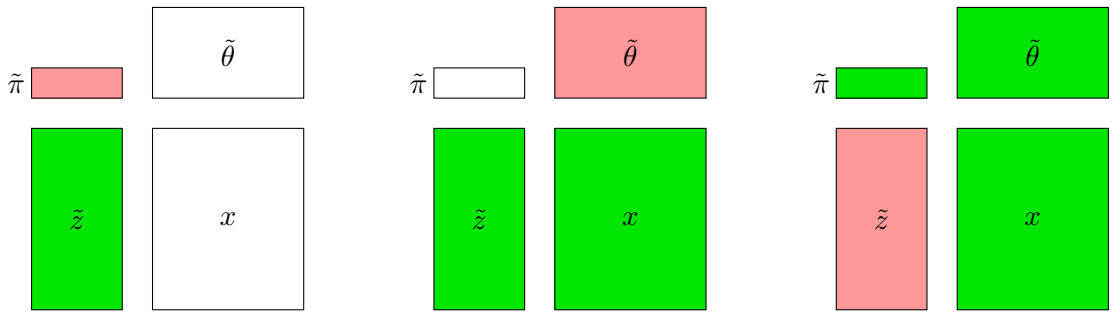
$$\nabla \mathcal{L}_{\mathcal{K}}(\tilde{z}_i^*) = \lambda \nabla h(\tilde{z}_i^*) + \sum_{k=1}^{K} \mu_k \nabla g_k(\tilde{z}_i^*).$$

Due to choice of $\mu_k = 0$, complementary slackness and dual feasibility are also satisfied. Thus, $\tilde{z}_i^*$ is the optimal solution to (4.16). ■

Theorem 2 suggests the following strategy: start with a feasible $\tilde{z}_i$, pick, say, a pair of coordinates $\tilde{z}_{i,k}$ and $\tilde{z}_{i,k'}$ and let $\tilde{z}_{i,k} + \tilde{z}_{i,k'} = C$. Solve (4.16), which has the closed form solution (4.17). Clearly, if $\tilde{z}_i$ satisfied constraints (4.15) before the update, it will continue to satisfy the constraints even after the update. On the other hand, the conditional ELBO (4.14) increases as a result of the update. Therefore, ESVI is a valid coordinate ascent algorithm for improving the ELBO (4.7).

4.4.1 Access Patterns

In this section we compare the access patterns of variables in the three algorithms to gain a better understanding of their abilities to be parallelized efficiently. In VI, the updates for $\tilde{\pi}$ and $\tilde{\theta}$ requires access to all $\tilde{z}_i$, while update to $\tilde{z}_i$ requires access to $\tilde{\pi}$ and all $\tilde{\theta}_k$. Refer to Figure 4.1 for an illustration.



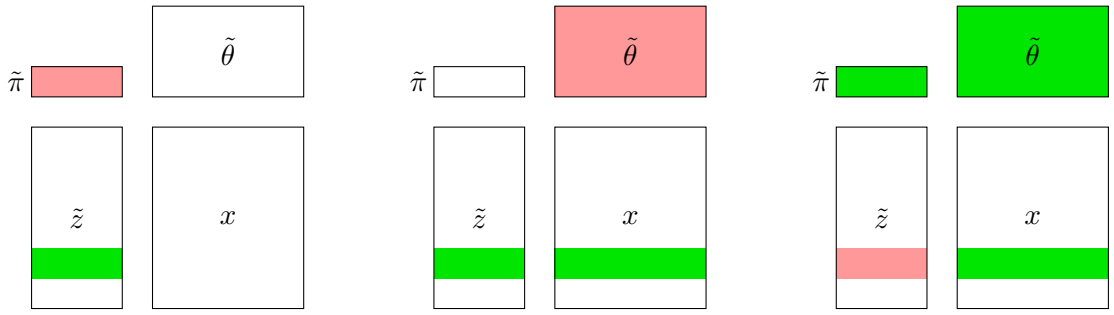
(a) $\tilde{\pi}$ update (b) $\tilde{\theta}$ update (c) $\tilde{z}$ update

Figure 4.1: Access pattern of variables during Variational Inference (VI) updates. Green indicates that the variable or data point is being read, while red indicates that the variable is being updated.

On the other hand, in case of SVI (see Figure 4.2), the access pattern is somewhat different. The updates for $\tilde{\pi}$ and $\tilde{\theta}$ require access to only the $\tilde{z}_i$ that was updated, however the update to $\tilde{z}_i$ still requires access to $\tilde{\pi}$ and all the $\tilde{\theta}_k$. This is a crucial bottleneck to model parallelism.

Bottleneck to Model Parallelism: The local variable $\tilde{z}_i$ needs to be normalized in order to be maintained on the k-dimensional simplex Δ_k after the update (4.12). This is the primary bottleneck to model parallelism in both VI and SVI, since this requires access to all K components. In ESVI, we propose a novel way to overcome this barrier, leading to

completely independent local and global variable updates.



(a) $\tilde{\pi}$ update (b) $\tilde{\theta}$ update (c) $\tilde{z}$ update

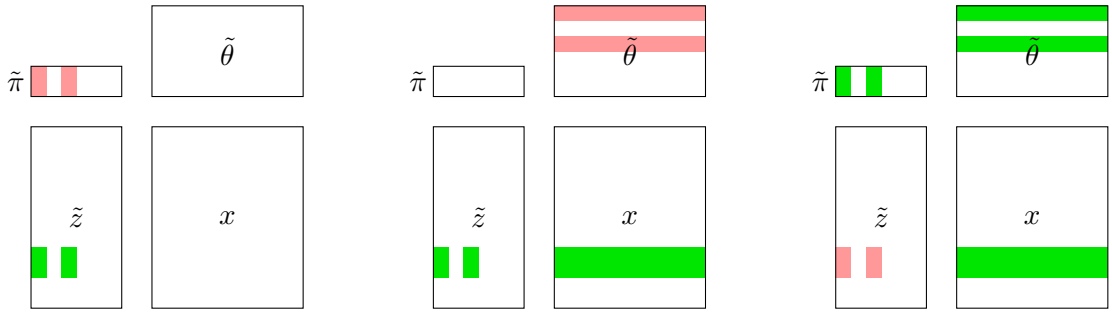
Figure 4.2: Access pattern of variables during Stochastic Variational Inference (SVI) updates. Green indicates that the variable or data point is being read, while red indicates that the variable is being updated.

The following access pattern of ESVI allows multiple processors to access and update mutually exclusive subsets of coordinates $\mathcal{K}$ independently (See Figure 4.3 for an illustration):

- The update for $\tilde{\pi}$ (4.8) requires access to the coordinates $\tilde{z}_{i,k}$ for $k \in \mathcal{K}$.
- The update for $\tilde{\theta}$ (4.9) and (4.10) requires access to $\tilde{z}_{i,k}$ for $k \in \mathcal{K}$.
- The update to $\tilde{z}_{i,k}$ for $k \in \mathcal{K}$ requires access to $\tilde{\pi}_k$ and $\tilde{\theta}_k$ for $k \in \mathcal{K}$.

4.4.2 Parallelization

In this sub-section, we describe the parallel asynchronous algorithm of ESVI. Let P denote the number of processors, and let $\mathcal{I}_p \subset \{1, \ldots, N\}$ denote indices of the data points owned by processor p. $\tilde{z}_i$ for $i \in \mathcal{I}_p$ are local variables assigned to processor p. The global variables are split across the processors. Let $\mathcal{K}_p \subset \{1, \ldots, K\}$ denote the indices of

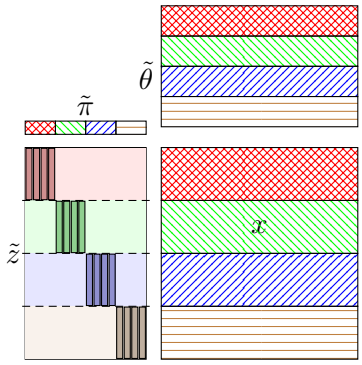


(a) $\tilde{\pi}$ update (b) $\tilde{\theta}$ update (c) $\tilde{z}$ update

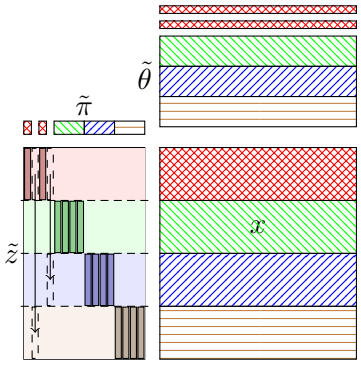
Figure 4.3: Access pattern during ESVI updates. Green indicates the variable or data point being read, while Red indicates it being updated.

the rows of $\tilde{\theta}$ currently residing in processor p. Then processor p can update any $\tilde{z}_{i,k}$ for $i \in \mathcal{I}_p$ and $k \in \mathcal{K}_p$. Finally, we need to address the issue of how to communicate $\tilde{\theta}_k$ across processors. For this, we follow the asynchronous communication scheme outlined by [Yun et al., 2014b] and [Yu et al., 2015b]. Figure 5.3 is an illustration of how this works pictorially. We partition the data and the corresponding $\tilde{z}_{1:N}$ variables across the processors. Each processors maintains its own queue. Once partitioned, the $\tilde{z}$ variables never move. On the other hand, the $\tilde{\theta}$ variables move nomadically[3] between processors. Each processor performs ESVI updates using the current subset of $\tilde{\theta}$ variables that it currently holds. Then the variables are passed on to the queue of another randomly chosen processor as shown in the second sub-figure in Figure 5.3. It is this nomadic movement of the $\tilde{\theta}$ variables that ensures proper mixing and convergence. The complete algorithm for parallel-ESVI is outlined in Algorithm 8.

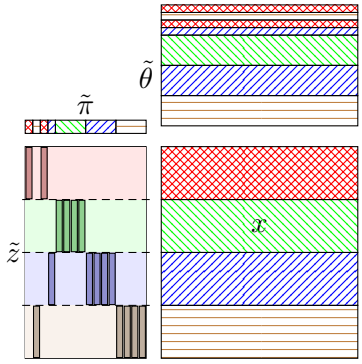
[3]Nomadic movement [Yun et al., 2014b] refers to the distributed setup, where the ownership of parameters rapidly keeps changing after every update. Figure 5.3 illustrates this.



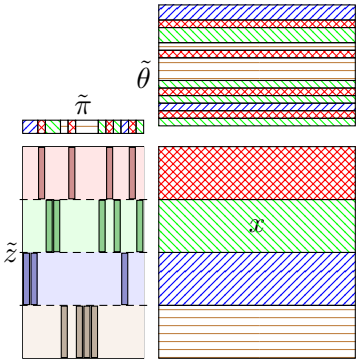
(a) Initial assignment of $\tilde{\theta}$ and x. We plot diagonal initialization while in real case random initialization is used.



(b) Worker 1 finishes processing $\{2, 4\} \in \mathcal{K}_1$, it sends them over to a random worker. Here, $\tilde{\theta}_2$ is sent from worker 1 to 4 and $\tilde{\theta}_4$ from 1 to 3.



(c) Upon receipt, the column is processed by the new worker. Here, worker 4 can now operate on $\tilde{\theta}_2$ and 3 on $\tilde{\theta}_4$



(d) During the execution of the algorithm, the ownership of the global parameters $\tilde{\theta}_k$ changes.

Figure 4.4: Illustration of the communication pattern in ESVI (asynchronous) algorithm. Parameters of same color are in memory of the same worker. Horizontal and Vertical lines indicate the two directions of partitioning data and parameters. Data x is partitioned horizontally along N and vertically along D. Local parameter $\tilde{z}$ is partitioned horizontally along N and vertically along K. Global parameters - $\tilde{\pi}$ is partitioned vertically along K, and $\tilde{\theta}$ is partitioned horizontally along K and vertically along D. $\tilde{\pi}$ and $\tilde{\theta}$ are nomadically exchanged.

4.4.3 Comparison and Complexity

ESVI updates are stochastic w.r.t. the coordinates, however the update in each coordinate is exact using (4.17). In contrast, SVI stochastically samples the data and performs inexact or noisy updates and does not guarantee each step to be an ascent step. Moreover, given a $N \times D$ dataset and fixing K clusters, by simple calculation we can see that to update all $\tilde{z}_{ik}$ once, VI requires $O(DK)$ updates on $\tilde{\theta}$, while SVI needs $O(NDK)$ and parallel ESVI needs $O(PDK)$.

4.5 Experiments

In our experiments[4], we compare our proposed **ESVI-GMM** and **ESVI-LDA** methods against VI and SVI. To handle large number of topics in LDA, we also implemented a more efficient version **ESVI-LDA-TOPK**. We use real-world datasets of varying scale as described in Table 5.2. We used a large-scale parallel computing platform with node configuration of 20 Intel Xeon E5-2680 CPUs and 256 GB memory. We implemented ESVI-LDA in C++ using MPICH, OpenMP and Intel TBB. For Distributed-VI and SVI implementations, we modified the authors' original code in C[5]. More details on the parameter settings and update equations are available in Appendix **??**, **??**, **??**.

4.5.1 ESVI-GMM

We first compare ESVI-GMM with SVI and VI in the Single Machine Single thread setting. We use a TOY dataset which consists of $N = 29,983$ data points, $D = 128$

[4]Our code and scripts will be made publicly available.

[5]http://www.cs.princeton.edu/~blei/lda-c/. Distributed-VI was implemented in Map-Reduce style

Algorithm 8 Parallel-ESVI Algorithm

P: total number of workers, T: maximum computing time

$\mathcal{I}_p$: data points owned by worker p,

$\mathcal{K}_p$: global parameters owned by worker p (concurrent queue)

Initialize global parameters $\tilde{\theta}^0$, $\tilde{\pi}^0$

for worker $p = 1 \dots P$ asynchronously **do**

  while Stop criteria not satisfied **do**

    Pick a subset $\mathbf{k}_s \subset \mathcal{K}_p$

    for All data point $i \in \mathcal{I}_p$ **do**

      for $k \in \mathbf{k}_s$ **do**

        Compute $\tilde{z}^*_{ik}$ using (4.17)

        $\tilde{\pi}_k += \tilde{z}^*_{ik} - \tilde{z}_{ik}$

        $\tilde{n}_k += \tilde{z}^*_{ik} - \tilde{z}_{ik}$

        $\tilde{\nu}_k += (\tilde{z}^*_{ik} - \tilde{z}_{ik}) \times \phi(x_i, k)$

        $\tilde{z}_{ik} \leftarrow \tilde{z}^*_{ik}$

        Pick a random worker p' and send $\tilde{\pi}_k$ and $\tilde{\theta}_k$, push k to $\mathcal{K}_{p'}$

      end for

    end for

  end while

end for

dimensions and the AP-DATA dataset which consists of $N = 2,246$ data points, $D = 10,473$ dimensions. In both cases, we set the number components $K = 256$. We plot the performance of the methods (ELBO) as a function of time. ESVI-GMM outperforms SVI and VI by quite

Dataset	# documents	# vocabulary	#words
AP-DATA	2,246	10,473	912,732
NIPS	1,312	12,149	1,658,309
Enron	37,861	28,102	6,238,796
Ny Times	298,000	102,660	98,793,316
PubMed	8,200,000	141,043	737,869,083
UMBC-3B	40,599,164	3,431,260	3,013,004,127

Table 4.3: Data Characteristics

some margin. For Multi Machine case, we use the NIPS and NY Times datasets and only compare against VI (SVI does not apply; it needs to update all its K global parameters which is infeasible when K is large). Although typically these datasets do not demand running on multiple machines, out intention here is to demonstrate scalability to very large number of components ($K = 1024$) and dimensions, which is typically the case in large scale text datasets with millions of word count features. Traditionally, GMM inference methods have not been able to handle such a scale. Figure 4.5 clearly indicates that ESVI-GMM is able to outperform VI by a good margin.

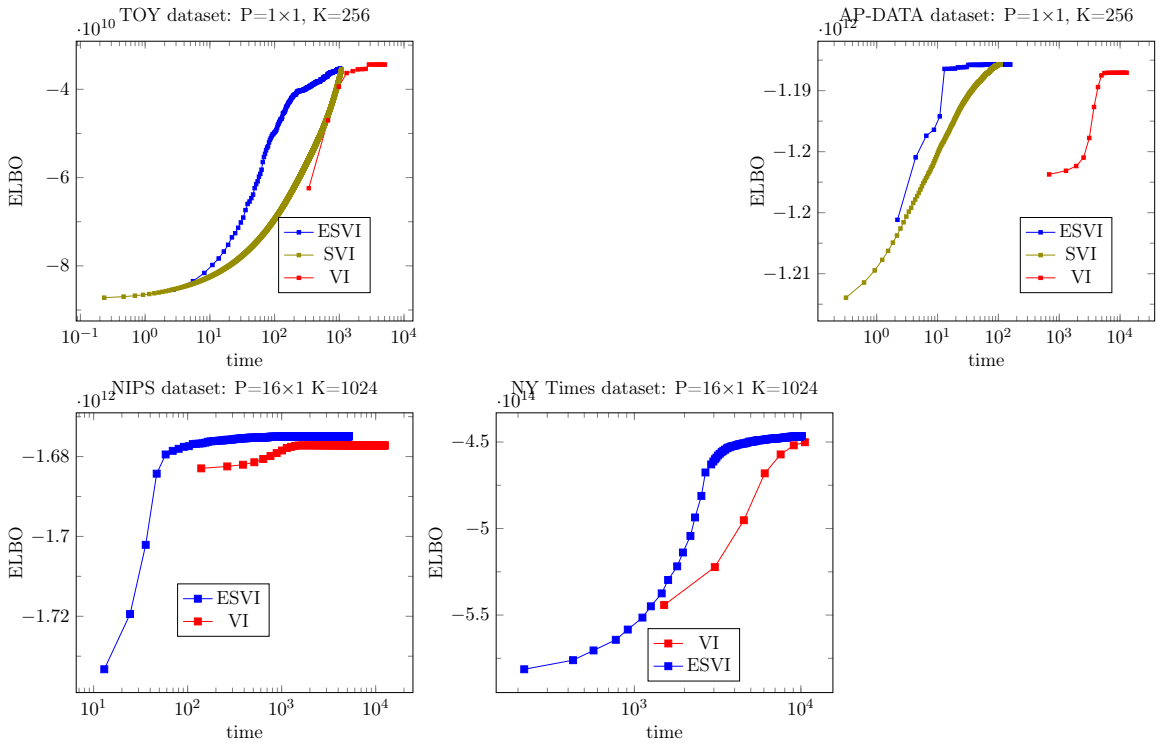


Figure 4.5: Comparison of ESVI-GMM, SVI and VI. $P = N \times n$ denotes N machines each with n threads.

4.5.2 ESVI-LDA

Single Machine Single thread

We compare serial versions of the methods on Enron and NY Times datasets which are medium sized and fit on a single-machine. On both datasets, we run with single machine and single thread. For Enron, we set # of topics $K = 8, 16, 20, 32, 64, 128, 256$. For NY Times, we set $K = 8, 16, 32, 64$. To keep the plots concise, we only show results with $K = 16, 64, 128$ in Figure 4.6 (two left-most plots). ESVI-LDA performs better than VI and SVI in both the datasets for all values of K. In our experiments, x-axis is in log-scale.

Single Machine Multi Core

We evaluate the performance of distributed ESVI-LDA against a map-reduce based distributed implementation of VI, and the streaming SVI method [Broderick et al., 2013]. We vary the number of cores as $4, 8, 16$. This is shown in Figure 4.6 (two right-most plots). For Enron dataset, we use $K = 128$ and for NY Times dataset, we use $K = 64$. ESVI-LDA outperforms VI and SVI consistently in all scenarios. In addition, we observe that both the methods benefit reasonably when we provide more cores to the computation. We observe that ESVI-LDA-TOPK, which stores only top $1/4$-th of K topics performs the best on both datasets.

Multi Machine Multi Core

We stretch the limits of ESVI-LDA method and compare it against distributed VI on large datasets: PubMed and UMBC-3B. UMBC-3B is a massive dataset with 3 billion tokens and a vocabulary of 3 million. We use 32 nodes and 16 cores, and fit $K = 128$ topics. As the results in Figure 4.7 demonstrate, ESVI-LDA achieves a better solution than distributed VI in all cases. On the largest dataset UMBC-3B, ESVI-LDA is also much faster than VI. In PubMed, VI has a slight initial advantage, however eventually ESVI progresses much faster towards a better ELBO. ESVI-LDA-TOPK is particularly better than the other two methods on both the datasets, especially on PubMed.

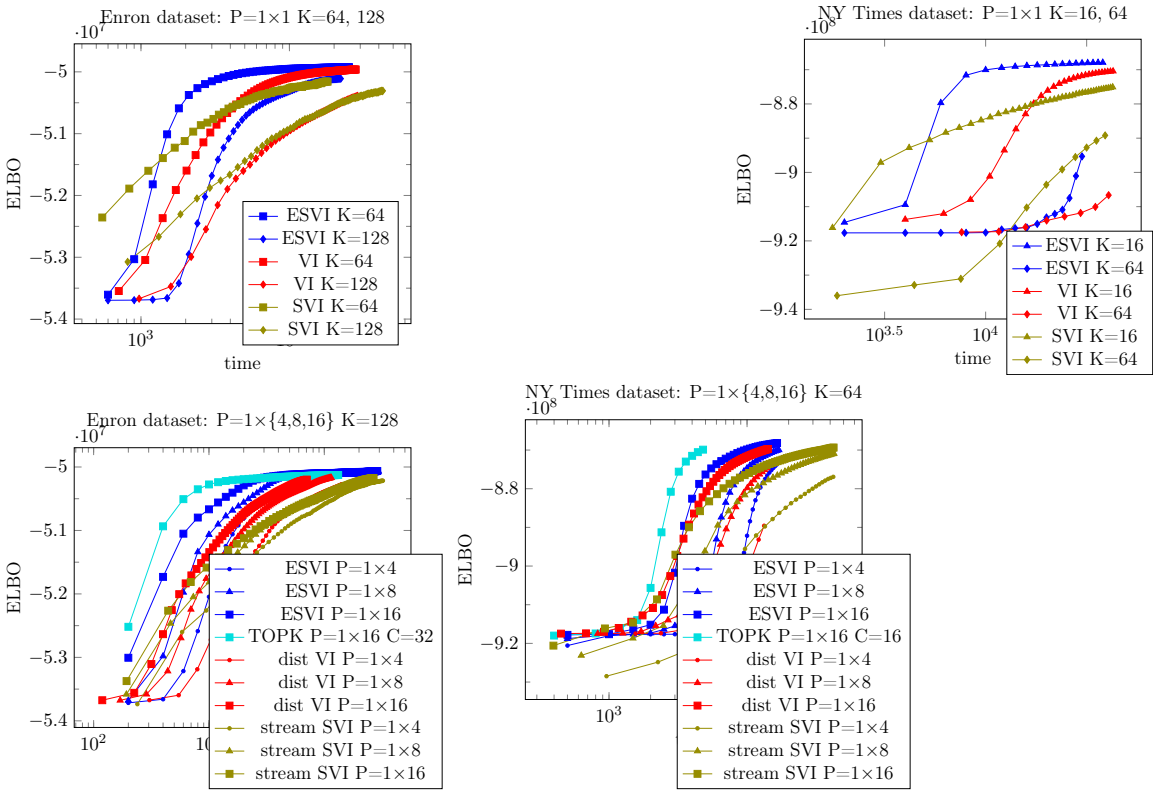


Figure 4.6: Single Machine experiments for ESVI-LDA (Single and Multi core). TOPK refers to our ESVI-TOPK method. $P = N \times n$ denotes N machines each with n threads.

Predictive Performance

We evaluate the predictive performance of ESVI-LDA comparing against distributed VI on Enron and NY Times datasets on multiple cores. As shown in Figure 4.8, ESVI typically reaches comparable test perplexity scores as VI but in much shorter wallclock time. On the Enron dataset, VI reaches a perplexity score of 9.052773 after a time of 1576.32 secs. ESVI, matches this perplexity in just 440 secs, and goes down further to a final score of 8.58929 in 2991 secs. On NY Times, VI starts off at 33.171374 and reaches 16.460541 in 14309 secs. ESVI matches this perplexity in roughly 2000 secs.

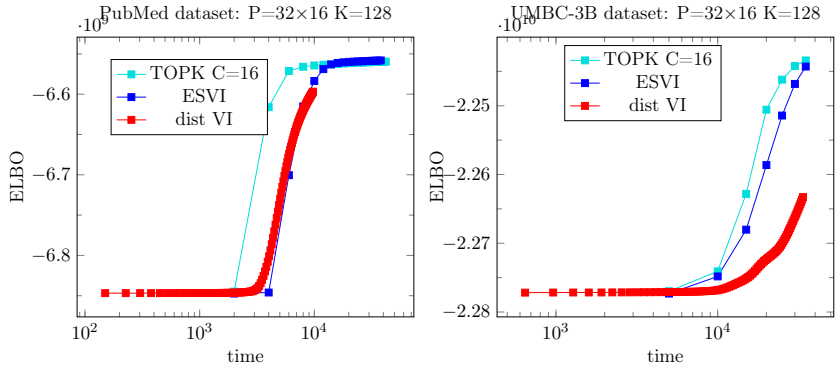


Figure 4.7: Multi Machine Multi Core experiments for ESVI-LDA. TOPK is our ESVI-TOPK method

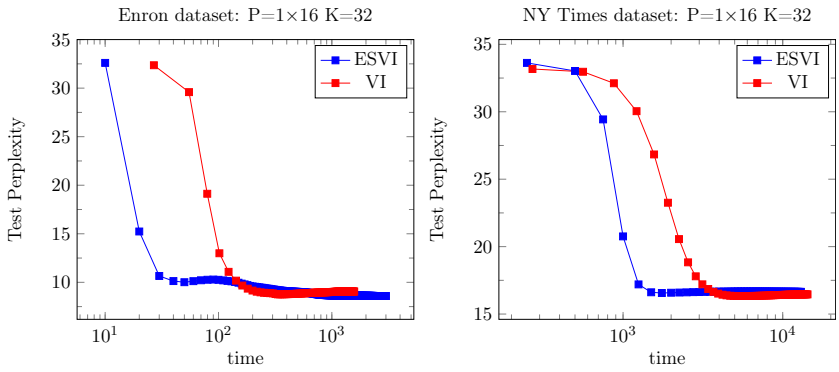


Figure 4.8: Predictive Performance of ESVI-LDA

4.5.3 Handling large number of topics in ESVI-LDA

In VI for LDA, the linear dependence of the model size on K prevents scaling to large K due to memory limitations. Our **ESVI-LDA-TOPK** approach addresses this: instead of storing all K components of the assignment parameter, we only store the most important top-k topics, which we denote using C. Using a min-heap of size C, we maintain only $C \ll K$ topics and we get performance very close to storing all the topics with much

lesser memory footprint.

Vary C (cutoff for K), Fix K

While the approximation of ELBO must get more accurate as $C \to K$, there might exist a choice of $C << K$, which gives a good enough approximation. This will give us a significant boost in speed. On Enron dataset, we varied C as $1, 8, 32, 64, 128$ with true $K = 128$ as our baseline. On NY Times dataset, we varied C as $1, 4, 16, 32, 64$ with the true $K = 64$ as baseline. As we expected, setting the cutoff to a value too low leads to very slow convergence. However, it is interesting to note that at a cut off value of roughly $\frac{K}{4}$ (32 on Enron and 16 on NY Times), we get a good result on par with the baseline. On the larger datasets - PubMed and UMBC-3B, setting $C = 16$ was enough to achieve a similar ELBO as the baseline. (See Figure 4.9).

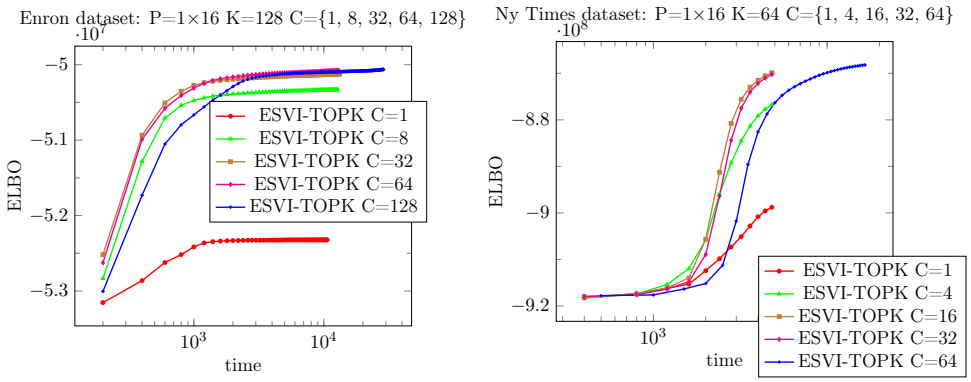


Figure 4.9: Effect of varying C in ESVI-LDA-TOPK

Fix C (cutoff for K), Scale to large K

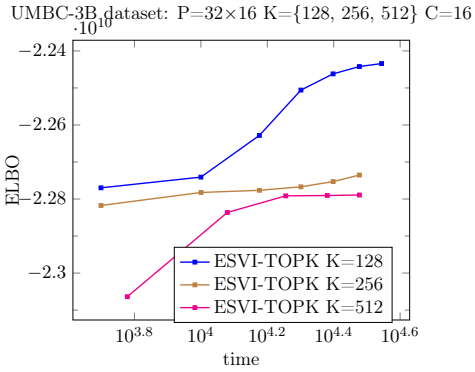


Figure 4.10: Effect of varying K by fixing C

By tuning C, ESVI-LDA-TOPK can be run on large number of topics such as $K = 256$ and $K = 512$ on the largest dataset UMBC-3B (See Figure 4.10).

4.6 Conclusion

In this paper, we propose Extreme Stochastic Variational Inference (ESVI), a distributed, asynchronous and lock-free algorithm to perform inference for mixture of exponential families. ESVI exhibits simultaneous model and data parallelism, allowing us to handle real-world datasets with large number of documents as well as learn sufficiently large number of parameters. For practitioners, we show how to use ESVI to fit GMM and LDA models on large scale real-world datasets consisting of millions of terms and billions of documents. In our empirical study, ESVI outperforms VI and SVI, and in most cases achieves a better quality solution. ESVI framework is very general and can be extended to several other latent variable models such as Mixed-Membership Stochastic Block Models.

Chapter 5

Factorization Machines

5.1 Introduction

Factorization Machines (FM), introduced by [Rendle, 2010] are powerful class of models which combine the benefits of polynomial regression and computational benefits of low-rank latent variable models such as matrix factorization. They offer a principled and flexible framework to model a variety of machine learning tasks. With the suitable feature representation, they can be used to model tasks ranging from regression, classification, learning to rank, collaborative filtering to temporal models. This makes FM a universal workhorse for predictive modeling as well as building ranking and recommender systems.

Factorization machines present a novel way to represent the higher-order interactions using low-rank latent embeddings of the features. A second-order FM thus requires a model storage of $\mathcal{O}(K \times D)$, where K is the number of latent dimensions for a feature and D is the number of features. While this is substantially smaller than the dense parameterization of polynomial regression, which would require $\mathcal{O}\left(D^2\right)$ storage, we notice that training the

FM model is still fraught will computational challenges. For example, consider the Criteo click logs dataset [Labs, 2014]. Running FM on this dataset even with a modest latent representation of $K = 128$ and 10^9 features would easily require memory in the order of 1 GB for the model parameters. In addition, the data itself occupies 2.1 TB. Such loads are impossible to run using a single-machine algorithm and demands developing distributed algorithms which can partition the workload (both data and parameters simultaneously) over a cluster of workers.

In this work, we propose DS-FACTO (Doubly-Separable Factorization Machines), a novel hybrid-parallel [Raman et al., 2019a] stochastic algorithm to scale factorization machines to arbitrarily large workloads. DS-FACTO is based on the NOMAD framework [Yun et al., 2014c] and fully de-centralizes the data as well as model parameters across the workers into mutually exclusive blocks. The key contributions of this work are as follows:

- We propose a Hybrid-Parallel algorithm for factorization machines which can partition both the data as well as model parameters simultaneously across the workers.
- DS-FACTO follows a fully de-centralized peer-only topology. This avoids the use of parameter servers and associated bottlenecks in a centralized master-slave topology [Watcharapichat et al., 2016].
- DS-FACTO is asynchronous and therefore can communicate parameters among the other workers while performing parameter updates.
- In our empirical study, we observe that, despite some delays arising from asynchronicity and staleness in parameter updates, DS-FACTO yields good predictive performance compared to other existing methods.

Outline: The rest of the chapter is organized as follows. Section 5.2 studies the related work and Section 5.3 provides some background for factorization machines. In Section 5.4, we introduce DS-FACTO and describe our hybrid-parallelization approach. Section 5.5 is devoted to empirical study comparing DS-FACTO to some standard baselines and studying its scaling behavior. Finally, Section 5.6 concludes the chapter.

5.2 Related Work

Context-aware recommender systems: There has been lot of work in using factorization machines as the basis to build recommender systems which take into account the user context for its feature representation. Fast context-aware recommendations with factorization machines [Rendle et al., 2011] discusses feature parameterizations that can incorporate diverse types of context information, and also provide numerical pre-computation techniques to optimize the factorization machine model faster. Gradient Boosting factorization machines [Cheng et al., 2014] borrows ideas from gradient-boosting to select only a subset of the pairwise feature interactions to provide more accurate context representations.

Click Through Rate (CTR) prediction: Field-aware factorization machines (FFM) [Juan et al., 2016] make use of a set of latent vector for each feature which is dependent on the context of the features with which the pairwise interactions are computed. This is in contrast with vanilla factorization machines where each features is always provided a single latent vector. Empirically, they observe that FFMs achieve better predictive performance for CTR prediction tasks compared to vanilla FMs and polynomial regression models.

[Punjabi and Bhatt, 2018] study robustness of factorization machines and design

robust counterparts for factorization machines and field-aware factorization machines using robust optimization principles. [Guo et al., 2016] propose a novel pairwise ranking model using factorization machines which incorporates implicit feedbacks with content information for the task of personalized ranking. [Hong et al., 2013] propose Co-Factorization Machines (CoFM), which can deal with multiple aspects of the dataset where each aspect makes use of a separate FM model. CoFM is able to predict user decisions and modeling user interests through content simultaneously. There has also been ample work in extending factorization machines using advances in neural networks. Neural Factorization machines [He and Chua, 2017] proposes a variant of FM to apply non-linear activation functions on the pairwise feature interactions. Likewise, Attentional Factorization Machines [Xiao et al., 2017a] is a method to improve vanilla FM models by learning the importance of each feature interaction from data using a neural attention network. DeepFM [Guo et al., 2017] proposes a new neural network architecture for factorization machines that involves training a deep component and an FM component jointly. In a completely different vein, [Blondel et al., 2015] proposes a convex formulation of factorization machines based on the nuclear norm and propose an efficient two-block coordinate descent algorithm to optimize the model. [Blondel et al., 2016] discusses how higher-order feature interactions can be used to build more general factorization machine models. [Rendle, 2013] discusses how factorization machines can be used on relational data and scaled to large datasets.

Scalability: There is very limited work in the direction of developing distributed algorithms for factorization machines. LibFM [Rendle, 2012], is one of the most popular implementations of FM that is based on the original paper [Rendle, 2010], however, it is

limited to a single-machine. Later, [Li et al., 2016] proposed DiFacto, which is a popular distributed algorithm for Factorization Machines based on the parameter server framework [Li et al., 2014]. Parameter Server uses a network of workers and servers to partition the data among the workers (and optionally, the model among the servers) and makes use of the message passing interface (MPI) as the communication paradigm. Di-Facto also proposes strategies to adaptively penalize the model parameters based on the frequency of the observed feature values. As a result, DiFacto is able to handle larger workloads than LibFM and also achieves a good predictive performance. [Zhong et al., 2016] is another work in this direction which also uses parameter server to provide a distributed algorithm for factorization machines - the key difference being that it uses Hadoop as the distributed framework instead of MPI. Finally, [Sun et al., 2014] also uses Map-Reduce to parallelize factorization machines.

5.3 Background and Preliminaries

The main goal of predictive modeling is to estimate a function $f : \mathbb{R}^D \rightarrow Y$ which can take as input a real valued feature vector in D dimensions and produce a corresponding output. We call such a function f the score function. The output set Y can take values depending on the task at hand. For example, in the case of regression $Y \in \mathbb{R}$, while for classification tasks, Y takes a positive/negative label such as $\{+1, -1\}$. In supervised settings, it is also assumed that there is a training dataset consisting of N examples and their corresponding labels $(\mathbf{x}_i, y_i)_{i=1,\ldots,N}$, where each $\mathbf{x}_i$ is a D-dimensional feature vector.

5.3.1 Polynomial Regression

In this work we are concerned with score functions which can compute second-order feature interactions[1] (also known as pairwise features) in the model. One simple way to accomplish this is by using the *Polynomial Regression* model which computes the following score function,

$$f(\mathbf{x}_i) = w_0 + \sum_{j=1}^{D} w_j \, x_{ij} + \sum_{j=1}^{D} \sum_{j'=j+1}^{D} w_{jj'} \, x_{ij} \, x_{ij'} \tag{5.1}$$

where, $\mathbf{x}_i \in \mathbb{R}^D$ is an example from the dataset $\mathbf{X} \in \mathbb{R}^{N \times D}$, and the parameters of the model are $w_0 \in \mathbb{R}, \quad \mathbf{w} \in \mathbb{R}^D, \quad \mathbf{W} \in \mathbb{R}^{D \times D}$. w_j denotes j-th dimension of $\mathbf{w}$ and $w_{jj'}$ denotes (i, j)-th entry of $\mathbf{W}$.

The model equation of polynomial regression in (5.1) has a few drawbacks. Real-world datasets are often heavily sparse, which means that all pairwise feature values are unlikely to be observed. Therefore, there is not enough information in the data to be modeled by using a weight for every pairwise feature. Such a parameterization reduces the predictive performance. Moreover, the weight matrix for pairwise features occupies $\mathcal{O}\left(D^2\right)$ storage which can be a limitation when the number of dimensions are high.

5.3.2 Factorization Machines (FM)

Factorization machines propose a different way to parameterize pairwise interaction between features to overcome the limitations of polynomial regression. FM aims to learn a latent embedding for every feature such that the pairwise interaction between any two features can be parameterized using the dot product of the corresponding latent embeddings.

[1] The techniques described in this chapter also apply to models that compute higher-order feature interactions, however, for simplicity purposes we assume this simplistic setting.

As a result of this parameterization, FM models work well even when the dataset is extremely sparse, since they only rely on first-order feature values being observed in the data. The score function for factorization machines is computed as,

$$f(\mathbf{x}_i) = w_0 + \sum_{j=1}^{D} w_j \, x_{ij} + \sum_{j=1}^{D} \sum_{j'=j+1}^{D} \langle \mathbf{v}_j, \mathbf{v}_{j'} \rangle \, x_{ij} \, x_{ij'} \tag{5.2}$$

where, the model parameters are $w_0 \in \mathbb{R}, \quad \mathbf{w} \in \mathbb{R}^D, \quad \mathbf{V} \in \mathbb{R}^{D \times K}$. $\mathbf{v}_j \in \mathbb{R}^K$ denotes the j-th dimension of $\mathbf{V}$ (latent embedding for the j-th feature).

Naive computation of the score function in FM seems to require $\mathcal{O}\left(KD^2\right)$. However, using a simple rewrite described below [Rendle, 2010], the score function can be computed in $\mathcal{O}(KD)$.

$$\begin{aligned}
\sum_{j=1}^{D} \sum_{j'=j+1}^{D} \langle \mathbf{v}_j, \mathbf{v}_{j'} \rangle \, x_{ij} \, x_{ij'} &= \frac{1}{2} \sum_{j=1}^{D} \sum_{j'=1}^{D} \langle \mathbf{v}_j, \mathbf{v}_{j'} \rangle \, x_{ij} \, x_{ij'} - \frac{1}{2} \sum_{j=1}^{D} \langle \mathbf{v}_j, \mathbf{v}_j \rangle \, x_{ij} \, x_{ij} \\
&= \frac{1}{2} \sum_{j=1}^{D} \sum_{j'=1}^{D} \sum_{k=1}^{K} v_{jk} \, v_{j'k} \, x_{ij} \, x_{ij'} - \frac{1}{2} \sum_{j=1}^{D} \sum_{k=1}^{K} v_{jk} \, v_{jk} \, x_{ij} \, x_{ij} \\
&= \frac{1}{2} \sum_{k=1}^{K} \left\{ \left(\sum_{j=1}^{D} v_{jk} x_{ij} \right) \left(\sum_{j'=1}^{D} v_{j'k} x_{ij'} \right) - \sum_{j=1}^{D} v_{jk}^2 x_{ij}^2 \right\} \\
&= \frac{1}{2} \sum_{k=1}^{K} \left\{ \left(\sum_{d=1}^{D} v_{dk} x_{id} \right)^2 - \sum_{j=1}^{D} v_{jk}^2 x_{ij}^2 \right\}
\end{aligned} \tag{5.3}$$

In the sums over j, only nnz (x_j) have to be summed up. Plugging this rewrite (5.3) into the original model equation (5.2), we obtain its simplified form,

$$f(\mathbf{x}_i) = w_0 + \sum_{j=1}^{D} w_j \, x_{ij} + \frac{1}{2} \sum_{k=1}^{K} \left\{ \left(\sum_{d=1}^{D} v_{dk} x_{id} \right)^2 - \sum_{j=1}^{D} v_{jk}^2 x_{ij}^2 \right\} \tag{5.4}$$

The complete normalized objective function for Factorization Machine model can now be written as,

$$\mathcal{L}(\mathbf{w}, \mathbf{V}) = \quad \frac{1}{N} \sum_{i=1}^{N} l\left(f(\mathbf{x}_i), y_i\right) + \frac{\lambda_w}{2} \left(\|\mathbf{w}\|_2^2 \right) + \frac{\lambda_v}{2} \left(\|\mathbf{V}\|_2^2 \right) \tag{5.5}$$

where,

- $f(\mathbf{x}_i)$ is given by (5.4)
- λ_w and λ_v are used to regularize the parameters $\mathbf{w}$ and $\mathbf{V}$
- $l(\cdot)$ is an appropriate loss function depending on the task at hand (e.g. cross-entropy for binary classification, squared loss for regression).

Symbol	Definition
N	total number of observations
D	total number of dimensions
K	total number of latent factors
$\mathbf{X} = \{\mathbf{x}_1, \ldots, \mathbf{x}_N\}, \quad \mathbf{x}_i \in \mathbb{R}^D$	observations (data points)
$\mathbf{y} = \{y_1, \ldots, y_N\}$	observed labels for the observation $\mathbf{x}_i$. For regression $y_i \in \mathbb{R}$. For classification $y_i \in \{+1, -1\}$
$\mathbf{w} \in \mathbb{R}^D$, $\mathbf{V} \in \mathbb{R}^{D \times K}$	parameters of the model
$G = \{g_1, \ldots, g_N\}, \quad A \in \mathbb{R}^{N \times K}$	auxiliary variables used in computing the parameter updates
λ_w, λ_v	regularization hyper-parameters for $\mathbf{w}$ and $\mathbf{V}$ respectively
η	learning rate hyper-parameter

Table 5.1: Notations for Factorization Machines

Optimization: The objective function in (5.5) can be optimized by any gradient based procedure such as gradient descent. Taking the derivatives of $\mathcal{L}(\mathbf{w}, \mathbf{V})$ with respect to the parameters $\mathbf{w}$ and $\mathbf{V}$ we obtain the following updates for gradient descent,

$$w_0^{t+1} \leftarrow w_0^t - \eta \cdot N \tag{5.6}$$

$$\begin{aligned} w_j^{t+1} &\leftarrow w_j^t - \eta \sum_{i=1}^{N} \nabla_{w_j} l_i \left(\mathbf{w}, \mathbf{V}\right) + \lambda_w \; w_j^t \\ &= w_j^t - \eta \sum_{i=1}^{N} G_i^t \cdot \nabla_{w_j} f\left(\mathbf{x}_i\right) + \lambda_w \; w_j^t \\ &= w_j^t - \eta \sum_{i=1}^{N} G_i^t \cdot x_{ij} + \lambda_w \; w_j^t \end{aligned} \tag{5.7}$$

$$\begin{aligned} v_{jk}^{t+1} &\leftarrow v_{jk}^t - \eta \sum_{i=1}^{N} \nabla_{v_{jk}} l_i \left(\mathbf{w}, \mathbf{V}\right) + \lambda_v \; v_{jk}^t \\ &= v_{jk}^t - \eta \sum_{i=1}^{N} G_i^t \cdot \nabla_{v_{jk}} f\left(\mathbf{x}_i\right) + \lambda_v \; v_{jk}^t \\ &= v_{jk}^t - \eta \sum_{i=1}^{N} G_i^t \cdot \left\{ x_{ij} \left(\sum_{d=1}^{D} v_{dk}^t \cdot x_{id} \right) - v_{jk}^t \; x_{ij}^2 \right\} + \lambda_v \; v_{jk}^t \end{aligned} \tag{5.8}$$

where, the multiplier G_i^t involves computing the score function using the parameter values of $\mathbf{w}$ and $\mathbf{V}$ at the t-th iteration as follows,

$$G_i^t = \begin{cases} f\left(x_i\right) - y_i, & \text{if squared loss (regression)} \\ \frac{-y_i}{1+\exp(y_i \cdot f_i(x_i))}, & \text{if logistic loss (classification)} \end{cases} \tag{5.9}$$

The term $\sum_{d=1}^{D} v_{dk} x_{id}$ requires synchronization across all D dimensions and can be pre-computed. Also, in practice, we only need to sum over the non-zero entries per dimension $\text{nnz}\left(x_i\right)$. We will denote this synchronization term succinctly as a_{ik},

$$a_{ik} = \sum_{d=1}^{D} v_{dk}^t \cdot x_{id} \tag{5.10}$$

5.4 Doubly-Separable Factorization Machines (DS-FACTO)

In this section, we describe our proposed distributed optimization algorithm for factorization machines DS-FACTO, which is based on double-separability of functions. We begin by first studying the parameter updates in factorization machines more closely.

5.4.1 Stochastic Optimization

Based on the updates described in (5.7) and (5.8), one can take stochastic gradients across $\sum_{i=1}^{N}$ and obtain update rules as follows,

$$w_0^{t+1} \leftarrow w_0^t - \eta \cdot 1 \tag{5.11}$$

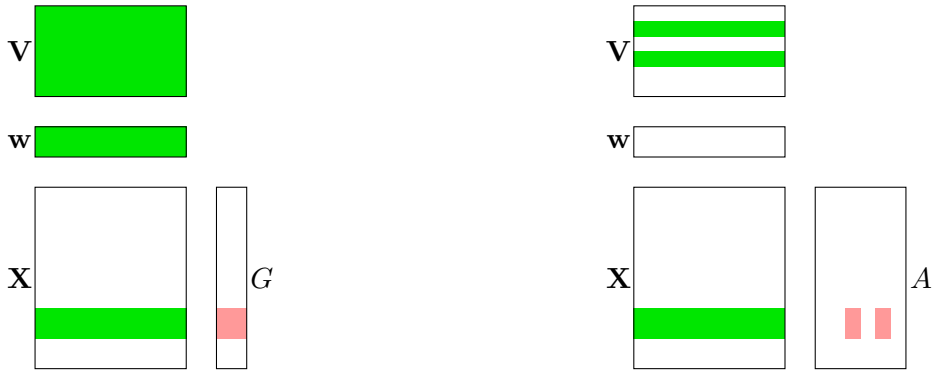
$$w_j^{t+1} \leftarrow w_j^t - \eta\, G_i^t \cdot x_{ij} + \lambda_w\, w_j^t \tag{5.12}$$

$$v_{jk}^{t+1} \leftarrow v_{jk}^t - \eta\, G_i^t \cdot \left\{x_{ij}\, a_{ik} - v_{jk}^t\, x_{ij}^2\right\} + \lambda_v\, v_{jk}^t \tag{5.13}$$

The above equations show that updates to w_j, V_{jk} require accessing only the j-th dimension of the i-th example, except the terms G_i^t and a_{ik} which involve a summation over all dimensions $j = 1, \ldots, D$, and therefore require bulk synchronization at iteration t.

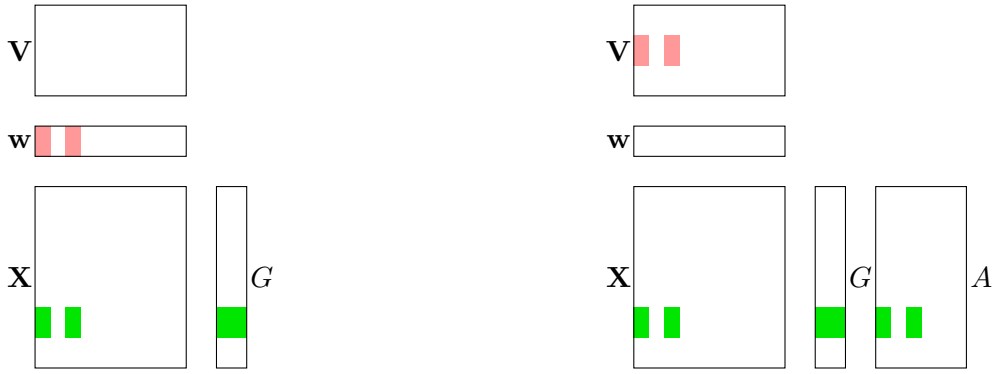
5.4.2 Distributing the computation in FM updates

The synchronization terms G_i^t and a_i are the main bottleneck in developing distributed algorithms for factorization machines that are model parallel. In this section, we study the access patterns of data $\mathbf{X}$ and parameters $\mathbf{w}$, $\mathbf{V}$ during the stochastic gradient descent (SGD) updates. Figures 5.1 and 5.2 provide a visual illustration. Updating w_j and v_{jk} depends on computing G_i and a_{ik} respectively (see Figure 5.1), which unfortunately require synchronization across all the dimensions $j = 1, \ldots, D$ (see Figure 5.2).



(a) computing G_i (b) computing a_{ik}

Figure 5.2: Access pattern of parameters while computing G and A. Green indicates the variable or data point being read, while Red indicates it being updated. Observe that computing both G and A requires accessing all the dimensions $j = 1, \ldots, D$. This is the main synchronization bottleneck.



(a) w_j update (b) v_{jk} update

Figure 5.1: Access pattern of parameters while updating w_j and v_{jk}. Green indicates the variable or data point being read, while Red indicates it being updated. Updating w_j requires computing G_i and likewise updating v_{jk} requires computing a_{ik}.

Handling the synchronization terms G and A: In a distributed machine learning system, there are two popular ways to synchronize parameters,

- In a centralized distributed framework following a Map-Reduce paradigm (e.g. parameter server), it is common to perform a *Reduce* step where all the workers transmit their copies of the parameters to the server which combines them and transmits them back to each worker.
- Another popular way to synchronize using the *All-Reduce* paradigm, where all workers transmit their parameter copies to each other.

Both of the approaches described above perform *Bulk Synchronization* which essentially means they make use of a *barrier step* where every worker waits for all other workers to finish their execution. Performing bulk synchronization at every iteration to compute G and A is a huge computational bottleneck.

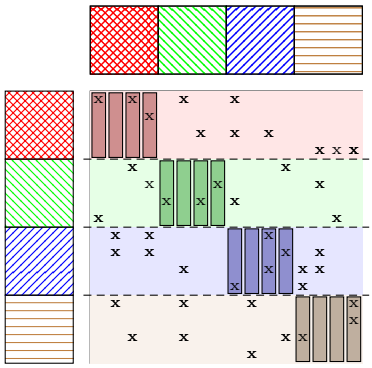
To resolve this, we propose a different paradigm for synchronization termed as *incremental synchronization* [Raman et al., 2019a] which avoids bulk synchronization altogether. The key idea behind incremental synchronization is simple - instead of computing the exact summation or dot product for the synchronization step, we propose computing it incrementally using partial sums. This can be done easily when the workers are arranged in ring topology and follow DSGD style communication (synchronous) or NOMAD style communication (asynchronous).

Handling the staleness in computing synchronization terms G and A: Since the stochastic updates modify the parameter values of w_j and v_{jk} on each worker, the older values of G_i and a_{ik} will no longer be up-to-date. This causes some staleness which

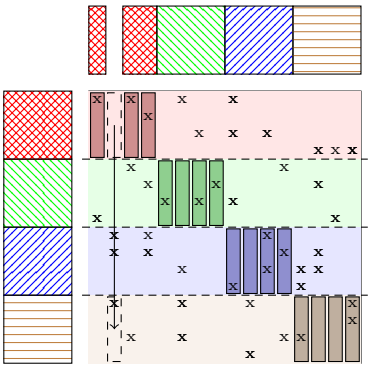
can slow down convergence significantly. To resolve this, we re-compute G and A after the update step, running an additional set of inner-epochs over all examples N and dimensions D. We observed that this re-computation is very important for such a hybrid-parallel scheme to convergence correctly.

5.4.3 Algorithm

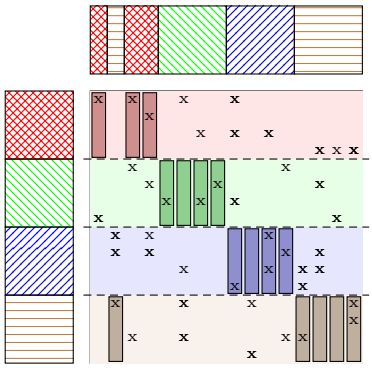
Algorithm 9 presents a basic outline of DS-FACTO which uses the NOMAD framework [Yun et al., 2014b] for asynchronous communication. The algorithm begins by distributing the data $\mathbf{X}$ and parameters $\{\mathbf{w}, \mathbf{V}\}$ among P workers as illustrated in Figure (CITE) where the row-blocks represent $\mathbf{X}^{(p)}$ and column-blocks represent parameters $\left\{\mathbf{w}^{(p)}, \mathbf{V}^{(p)}\right\}$ on each local worker respectively. In order to periodically communicate parameter updates across workers, we also maintain P worker queues. The parameters $\{\mathbf{w}, \mathbf{V}\}$ are initially distributed uniformly at random across the queues. Each worker can then perform its update in parallel as follows: (1) pops a parameter $(k, \{w_j, \mathbf{v}_j\})$ out of the queue, (2) updates w_j and v_{jk} stochastically using (5.12) and (5.13) respectively, (3) pushes the updated parameter set into the queue of the next worker. Once D rounds of updates have been performed (which is equivalent to saying each worker has updated parameters corresponding to every dimension $j \in \{1, \ldots, D\}$), we perform an additional round of communication across the P workers to compute the auxiliary variables $G^{(p)}$ and $A^{(p)}$ using the freshest copy of the parameters.



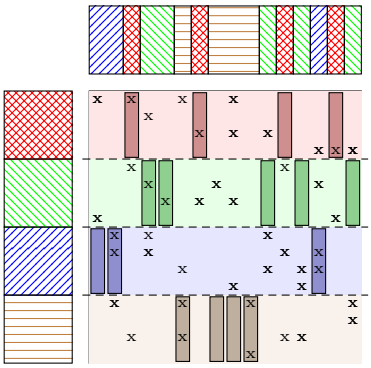
(a) Initial assignment of parameters $\{\mathbf{w}, \mathbf{V}\}$ and $\mathbf{X}$. Each worker works only on the diagonal active area in the beginning.



(b) After a worker finishes processing column j, it sends the corresponding parameter set $\{w_j, \mathbf{v}_j\}$ to another worker. Here, $\{w_2, \mathbf{v}_2\}$ is sent from worker 1 to 4.



(c) Upon receipt, the column is processed by the new worker. Here, worker 4 can now process column 2 since it owns the column.



(d) During the execution of the algorithm, the ownership of the global parameters $\{w_j, \mathbf{v}_j\}$ changes.

Figure 5.3: Illustration of the communication pattern in DS-FACTO algorithm. Parameters $\{w_j, \mathbf{v}_j\}$ are exchanged in a de-centralized manner across workers without the use of any parameter servers [Li et al., 2013].

Algorithm 9 DS-FACTO Asynchronous

D: total # dimensions, P: total # workers, T: total outer iterations

$\left\{\mathbf{w}^{(p)}, \mathbf{V}^{(p)}\right\}$: parameters per worker, $\left\{G^{(p)}, A^{(p)}\right\}$: auxiliary variables per worker

queue[P]: array of P worker queues

Initialize $\mathbf{w}^{(p)} = 0$, $\mathbf{V}^{(p)} \sim \mathcal{N}(0, 0.01)$ `//Initialize parameters`

for $j \in \left\{\mathbf{w}^{(p)}, \mathbf{V}^{(p)}\right\}$ **do**

  Pick q uniformly at random

  queue[q].push($k, \{w_j, \mathbf{v}_j\}$) `//Initialize worker queues`

end for

`//Start P workers`

for all $p = 1, 2, \ldots, P$ in parallel **do**

  for all $t = 1, 2, \ldots, T$ **do**

    repeat

      $(k, \{w_j, \mathbf{v}_j\}) \leftarrow$ queue[p].pop()

      Update w_j and v_{jk} stochastically using (5.12) and (5.13)

      Compute index of next queue to push to: $\hat{q}$

      queue[$\hat{q}$].push($k, \{w_j, \mathbf{v}_j\}$)

    until # of updates is equal to D

    repeat

      $(k, \{w_j, \mathbf{v}_j\}) \leftarrow$ queue[p].pop()

      Compute $G^{(p)}$ and $A^{(p)}$ using (5.4) and (5.10)

    until # of rounds is equal to D

  end for

end for

Although the algorithm snippet in Algorithm 9 assumes a restricted setting consisting of P workers, in practice DS-FACTO uses multiple threads on multiple machines. In such a scenario, each worker (thread) first passes around the parameter set across all its threads on its machine. Once this is completed, the parameter set is tossed onto the queue of the first thread on the next machine.

5.5 Experiments

In our empirical study, we evaluate DS-FACTO to examine the following,

- Convergence behavior and predictive performance.
- Scaling behavior as the number of threads and cores are varied.

Datasets:

Dataset	N	D	K
diabetes	513	8	4
housing	303	13	4
ijcnn1	49,990	22	4
realsim	50,616	20,958	16
url	2,396,130	3,231,961	16

Table 5.2: Dataset Characteristics.

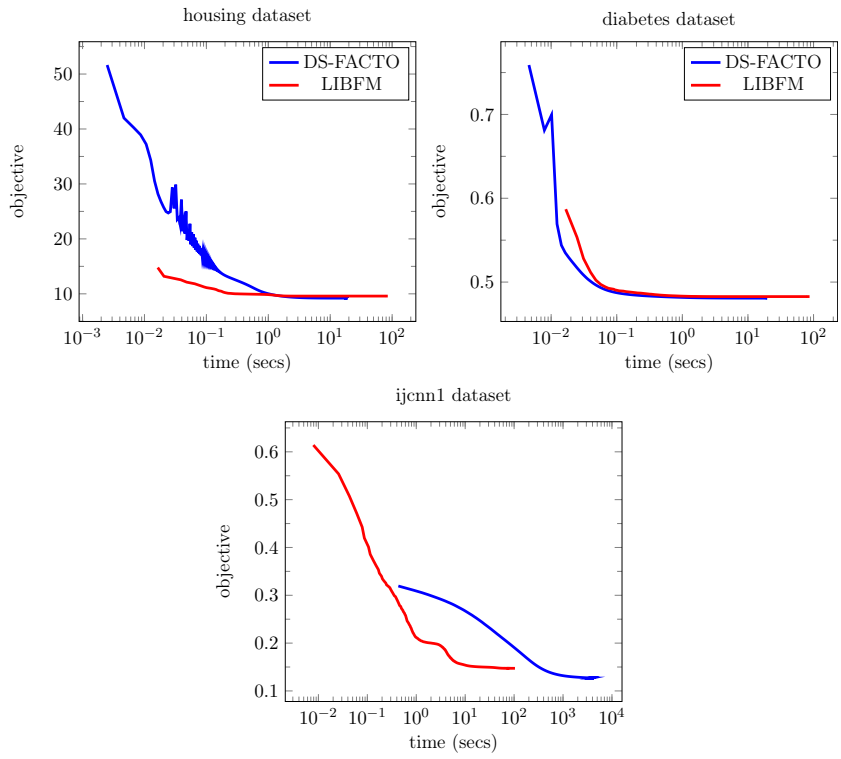


Figure 5.4: Convergence behavior of DS-FACTO on **diabetes**, **housing** and **ijcnn1** datasets.

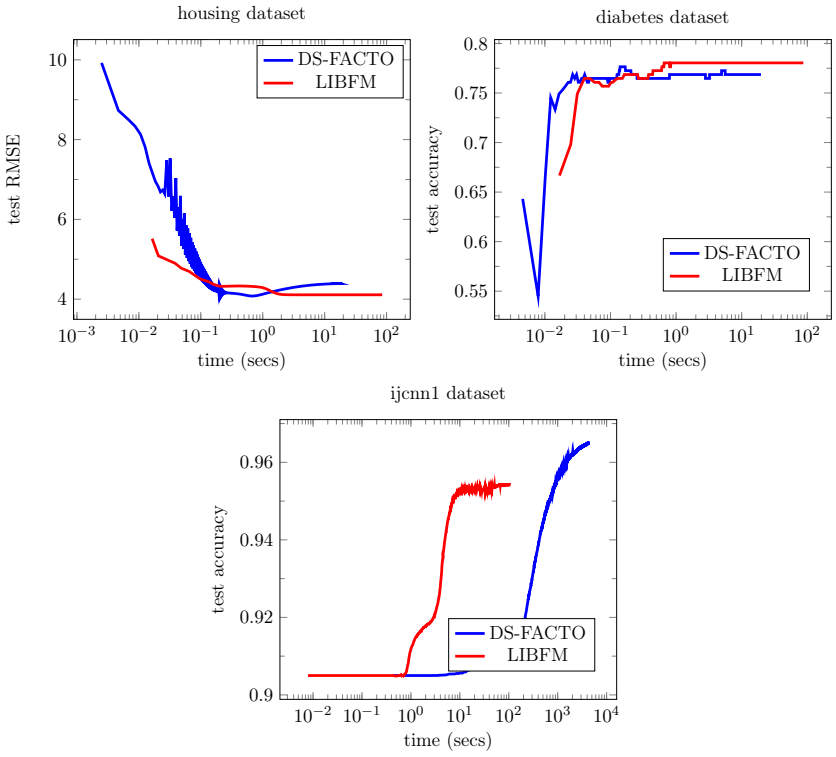


Figure 5.5: Predictive Performance - Test RMSE (Regression) and Test Accuracy (Classification) of DS-FACTO on **diabetes**, **housing** and **ijcnn1** datasets.

5.5.1 Convergence and Predictive Performance

5.5.2 Scalability

Below we present scalability results.

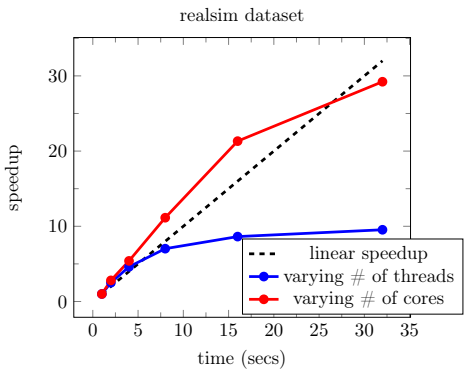


Figure 5.6: Scalability of DS-FACTO as # of threads, cores are varied as 1, 2, 4, 8, 16, 32.

5.6 Conclusion

In this chapter, we presented DS-FACTO, a distributed stochastic optimization algorithm for factorization machines which is hybrid-parallel, i.e. it can partition both data as well as model parameters in a de-centralized manner across workers. In order to circumvent the bulk-synchronization required in computing the gradients for the parameter updates, we make use of local auxiliary variables to maintain partial sums of the synchronization terms and update them in a post-update step. We analyze the behavior of DS-FACTO in terms of convergence and scalability on several real-world datasets. The data partitioning scheme and distributed parameter update strategy used in DS-FACTO is very general and can be easily adapted to scale other variants of factorization machines models such field-aware factorization

machines and factorization machines for context-aware recommender systems. We believe these are promising future directions to pursue.

Chapter 6

Conclusions and future work

In this chapter, we summarize the contributions of this thesis and discuss directions for future research.

6.1 Contributions

Distributed approaches to training machine learning models typically fall into two categories namely, *Data Parallel* where the data is partitioned across multiple workers while the model is replicated, and *Model Parallel* where the model is partitioned across workers while the data is replicated. Both of these approaches fail to handle scenarios when the data or model sizes exceed the storage capacity of a single worker, which is not unreasonable to expect with current explosion in data driven internet and mobile applications. In this work, we argue that *Hybrid Parallel* methods are necessary to overcome these limitations of traditional distributed machine learning algorithms. Hybrid Parallelism partitions both the data as well as model parameters simultaneously across the workers. Since neither data nor

model is replicated on any worker, such methods can scale to arbitrarily large workloads. In order to make machine learning tasks hybrid-parallel, we identify an essential property in their objective functions known as Double-Separability. Double-Separability allows the objective function of the machine learning task to be decomposed into sub-functions each of which depend on non-overlapping blocks of data and model parameters. This allows parameter estimation to be massively parallelized without any locks on the individual units of parallelism.

While some tasks in machine learning such as Matrix Factorization are naturally doubly-separable (and thus can be made Hybrid Parallel), most of them are not and require some work to be cast into such a desirable form. By analyzing the read-write access patterns in the parameter updates, we propose novel reformulations for the following frequentist and bayesian models to make them hybrid-parallel. Subsequently, we discuss how to build synchronous and asynchronous optimization methods for these hybrid-parallel formulations by making use of the DSGD [Gemulla et al., 2011] and NOMAD [Yun et al., 2013] frameworks respectively.

Latent Collaborative Retrieval: In Latent Collaborative Retrieval, we formulate a pairwise learning to rank loss function to rank a set of candidate items for a particular user. In this setting, both the number of users as well as items can be very large and therefore computing all pairwise scores across the items can easily become prohibitively high. To solve this, we need distributed optimization methods which can partition both the users as well as items parameters across workers. In Section 3, we show how to obtain an unbiased stochastic optimization algorithm for this setting by reformulating the original pairwise loss function.

Multinomial Logistic Regression: Multinomial Logistic Regression involves predicting the probability that an observation belongs to one of the K categories and is optimized by minimizing the negative log probability of an observation belonging to a particular class. The fundamental bottleneck to model parallelism in this setting is the log-partition function which needs to be computed across all K classes per observation. In Section 3, we present a reformulation to convert the original objective function into a doubly-separable form and hence can be made hybrid-parallel.

Mixture of Exponential Families: Mixture of Exponential Family models encompass a wide variety of probabilistic models which can be used to model N observations arising from K different component distributions. The state of the art method for performing inference in mixture models - Stochastic Variational Inference (SVI) is inherently serial in that it replicates the $\mathcal{O}(K \times D)$ parameters (where D is the dimension of the observations) on every worker even in the distributed setting. This poses limitations on the number of components that can the model can be trained on when K and D are very large. In Chapter 4, we address this fundamental bottleneck by proposing a novel sequence of local and global updates which involve no overlap in the parameter access patterns. As a result, variational inference can be made Hybrid-Parallel and scaled to arbitrarily large values of N and K.

Factorization Machines: Factorization Machines present a systematic way to combine first-order features with second-order (pairwise) features in a model. In order to make more effective use of the pairwise features, they propose using a factorized weight matrix. In this setting, the storage complexity for the model scales as $\mathcal{O}(K \times D)$, where K is the number of latent dimensions for a feature and D is the total number of features.

For e.g. the cost of storage becomes high on massive datasets such as Criteo Tera logs, even with $K = 128$, $D = 10^9$. This motivates the need to partition not only the data but also the model. We observe that the objective function of factorization machines is amenable to Hybrid-Parallelism with some book keeping to handle the synchronization terms. In Chapter 5, we analyze the access pattern of parameters in more detail and propose a strategy to obtain Hybrid Parallelism.

6.2 Future Work

The work in Chapter 3, DS-MLR can be naturally extended to the multi-label setting to scale to large number of data points and labels. In addition, by linearizing the log-partition function, one can potentially develop Hybrid-Parallel algorithms for a variety of log-linear models since they share similar computational structure as the optimization problem for multinomial logistic regression.

Another intriguing direction to explore making use of techniques proposed in this thesis to develop hybrid-parallel training algorithms for Deep Neural Networks. There has been some work in this direction in the past. [Carreira-Perpinan and Wang, 2014] propose introducing auxiliary variables in the objective functions of deep neural networks to de-couple the parameter estimation.

The work in Chapter 4, ESVI can also be extended to other types of mixture of exponential family models such as *Mixed Membership Stochastic Block Models* which are generative models to discover communities in graphs such as social networks.

Finally, although the work in this thesis has focussed on parametric bayesian models,

parameter estimation for large-scale non-parametric bayesian models still remains challenging. Examples of such models include - *Dirichlet Process Mixture Models* and *Pitman-Yor Mixture Models* which exhibit similar computational structure and scaling challenges. It would be interesting to explore how techniques from this thesis can be leveraged to make these models Hybrid-Parallel.

Bibliography

S. Agarwal. The infinite push: A new support vector ranking algorithm that directly optimizes accuracy at the absolute top of the list. In *SDM*, pages 839–850. SIAM, 2011.

R. Agrawal, A. Gupta, Y. Prabhu, and M. Varma. Multi-label learning with millions of labels: Recommending advertiser bid phrases for web pages. In *Proceedings of the 22nd international conference on World Wide Web*, pages 13–24. ACM, 2013.

C. Archambeau and B. Ermis. Incremental variational inference for latent dirichlet allocation. *arXiv preprint arXiv:1507.05016*, 2015.

P. L. Bartlett, M. I. Jordan, and J. D. McAuliffe. Convexity, classification, and risk bounds. *Journal of the American Statistical Association*, 101(473):138–156, 2006.

T. Bertin-Mahieux, D. P. Ellis, B. Whitman, and P. Lamere. The million song dataset. In *Proceedings of the 12th International Conference on Music Information Retrieval (ISMIR 2011)*, 2011.

D. P. Bertsekas. Incremental gradient, subgradient, and proximal methods for convex optimization: A survey. *Optimization for Machine Learning*, 2010(1-38):3, 2011.

D. Blei, A. Ng, and M. Jordan. Latent Dirichlet allocation. *Journal of Machine Learning Research*, 3:993–1022, Jan. 2003.

D. M. Blei, A. Kucukelbir, and J. D. McAuliffe. Variational inference: A review for statisticians. *arXiv preprint arXiv:1601.00670*, 2016.

M. Blondel, A. Fujino, and N. Ueda. Convex factorization machines. In *Joint European Conference on Machine Learning and Knowledge Discovery in Databases*, pages 19–35. Springer, 2015.

M. Blondel, A. Fujino, N. Ueda, and M. Ishihata. Higher-order factorization machines. In *Advances in Neural Information Processing Systems*, pages 3351–3359, 2016.

L. Bottou. Large-scale machine learning with stochastic gradient descent. In *Proceedings of COMPSTAT'2010*. Springer, 2010.

L. Bottou and O. Bousquet. The tradeoffs of large-scale learning. *Optimization for Machine Learning*, page 351, 2011.

G. Bouchard. Efficient bounds for the softmax function, applications to inference in hybrid models. 2007.

S. Boyd and L. Vandenberghe. *Convex Optimization*. Cambridge University Press, Cambridge, England, 2004.

S. Boyd, N. Parikh, E. Chu, B. Peleato, and J. Eckstein. Distributed optimization and statistical learning via the alternating direction method of multipliers. *Foundations and Trends® in Machine Learning*, 3(1):1–122, 2011.

T. Broderick, N. Boyd, A. Wibisono, A. C. Wilson, and M. I. Jordan. Streaming variational bayes. In *Advances in Neural Information Processing Systems*, pages 1727–1735, 2013.

D. Buffoni, P. Gallinari, N. Usunier, and C. Calauzènes. Learning scoring functions with order-preserving losses and standardized supervision. In *Proceedings of the 28th International Conference on Machine Learning (ICML-11)*, pages 825–832, 2011.

M. Carreira-Perpinan and W. Wang. Distributed optimization of deeply nested systems. In *Artificial Intelligence and Statistics*, pages 10–19, 2014.

O. Chapelle and Y. Chang. Yahoo! learning to rank challenge overview. *Journal of Machine Learning Research-Proceedings Track*, 14:1–24, 2011.

O. Chapelle, C. B. Do, C. H. Teo, Q. V. Le, and A. J. Smola. Tighter bounds for structured estimation. In *Advances in neural information processing systems*, pages 281–288, 2008.

C. Cheng, F. Xia, T. Zhang, I. King, and M. R. Lyu. Gradient boosting factorization machines. In *Proceedings of the 8th ACM Conference on Recommender systems*, pages 265–272. ACM, 2014.

H.-T. Cheng, L. Koc, J. Harmsen, T. Shaked, T. Chandra, H. Aradhye, G. Anderson, G. Corrado, W. Chai, M. Ispir, et al. Wide & deep learning for recommender systems. In *Proceedings of the 1st workshop on deep learning for recommender systems*, pages 7–10. ACM, 2016.

C.-T. Chu, S. K. Kim, Y.-A. Lin, Y. Yu, G. Bradski, K. Olukotun, and A. Y. Ng. Map-reduce for machine learning on multicore. In *Advances in neural information processing systems*, pages 281–288, 2007.

P. Cremonesi, Y. Koren, and R. Turrin. Performance of recommender algorithms on top-n recommendation tasks. In *Proceedings of the fourth ACM conference on Recommender systems*, pages 39–46. ACM, 2010.

J. Davidson, B. Liebald, J. Liu, P. Nandy, T. Van Vleet, U. Gargi, S. Gupta, Y. He, M. Lambert, B. Livingston, et al. The youtube video recommendation system. In *Proceedings of the fourth ACM conference on Recommender systems*, pages 293–296. ACM, 2010.

N. Ding. *Statistical Machine Learning in T-Exponential Family of Distributions*. PhD thesis, PhD thesis, Purdue University, West Lafayette, Indiana, USA, 2013.

V. Feldman, V. Guruswami, P. Raghavendra, and Y. Wu. Agnostic learning of monomials by halfspaces is hard. *SIAM Journal on Computing*, 41(6):1558–1590, 2012.

R. Gemulla, E. Nijkamp, P. J. Haas, and Y. Sismanis. Large-scale matrix factorization with distributed stochastic gradient descent. In *Conference on Knowledge Discovery and Data Mining*, pages 69–77, 2011.

I. Goodfellow, Y. Bengio, and A. Courville. *Deep Learning*. MIT Press, 2016. http://www.deeplearningbook.org.

S. Gopal and Y. Yang. Distributed training of large-scale logistic models. In *Proceedings of the 30th International Conference on Machine Learning (ICML-13)*, pages 289–297, 2013.

H. Guo, R. Tang, Y. Ye, Z. Li, and X. He. Deepfm: a factorization-machine based neural network for ctr prediction. *arXiv preprint arXiv:1703.04247*, 2017.

W. Guo, S. Wu, L. Wang, and T. Tan. Personalized ranking with pairwise factorization machines. *Neurocomputing*, 214:191–200, 2016.

J. Z. HaoChen and S. Sra. Random shuffling beats sgd after finite epochs. *arXiv preprint arXiv:1806.10077*, 2018.

L. Hasenclever, S. Webb, T. Lienart, S. Vollmer, B. Lakshminarayanan, C. Blundell, and Y. W. Teh. Distributed bayesian learning with stochastic natural gradient expectation propagation and the posterior server. *Journal of Machine Learning Research*, 18(106):1–37, 2017.

X. He and T.-S. Chua. Neural factorization machines for sparse predictive analytics. In *Proceedings of the 40th International ACM SIGIR conference on Research and Development in Information Retrieval*, pages 355–364. ACM, 2017.

M. D. Hoffman, D. M. Blei, C. Wang, and J. Paisley. Stochastic variational inference. *Journal of Machine Learning Research*, 14:1303–1347, 2013.

L. Hong, A. S. Doumith, and B. D. Davison. Co-factorization machines: modeling user interests and predicting individual decisions in twitter. In *Proceedings of the sixth ACM international conference on Web search and data mining*, pages 557–566. ACM, 2013.

P. J. Huber. *Robust Statistics*. John Wiley and Sons, New York, 1981.

M. C. Hughes and E. Sudderth. Memoized online variational inference for dirichlet process mixture models. In *Advances in Neural Information Processing Systems*, pages 1133–1141, 2013.

H. Jain, Y. Prabhu, and M. Varma. Extreme multi-label loss functions for recommendation, tagging, ranking & other missing label applications. In *Proceedings of the 22nd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, pages 935–944. ACM, 2016.

Y. Juan, Y. Zhuang, W.-S. Chin, and C.-J. Lin. Field-aware factorization machines for ctr prediction. In *Proceedings of the 10th ACM Conference on Recommender Systems*, pages 43–50. ACM, 2016.

T. Kuno, Y. Yajima, and H. Konno. An outer approximation method for minimizing the product of several convex functions on a convex set. *Journal of Global Optimization*, 3(3): 325–335, September 1993.

C. Labs. Criteo terabyte click logs. http://labs.criteo.com/downloads/download-terabyte-click-logs, 2014.

Q. V. Le and A. J. Smola. Direct optimization of ranking measures. Technical Report 0704.3359, arXiv, April 2007. http://arxiv.org/abs/0704.3359.

C.-P. Lee and C.-J. Lin. Large-scale linear ranksvm. *Neural Computation*, 2013. To Appear.

M. Li, L. Zhou, Z. Yang, A. Li, F. Xia, D. G. Andersen, and A. Smola. Parameter server for distributed machine learning. In *Big Learning NIPS Workshop*, 2013.

M. Li, D. G. Andersen, J. W. Park, A. J. Smola, A. Ahmed, V. Josifovski, J. Long, E. J. Shekita, and B.-Y. Su. Scaling distributed machine learning with the parameter server. In *11th {USENIX} Symposium on Operating Systems Design and Implementation ({OSDI} 14)*, pages 583–598, 2014.

M. Li, Z. Liu, A. J. Smola, and Y.-X. Wang. Difacto: Distributed factorization machines. In *Proceedings of the Ninth ACM International Conference on Web Search and Data Mining*, pages 377–386. ACM, 2016.

P. Long and R. Servedio. Random classification noise defeats all convex potential boosters. *Machine Learning Journal*, 78(3):287–304, 2010.

C. D. Manning, P. Raghavan, and H. Schütze. *Introduction to Information Retrieval*. Cambridge University Press, 2008. URL http://nlp.stanford.edu/IR-book/.

A. R. Masegosa, A. M. Martinez, H. Langseth, T. D. Nielsen, A. Salmerón, D. Ramos-López, and A. L. Madsen. Scaling up bayesian variational inference using distributed computing clusters. *International Journal of Approximate Reasoning*, 88:435–451, 2017.

R. M. Neal and G. E. Hinton. A view of the em algorithm that justifies incremental, sparse, and other variants. In *Learning in graphical models*, pages 355–368. Springer, 1998.

A. Nedić and D. Bertsekas. Convergence rate of incremental subgradient algorithms. In *Stochastic optimization: algorithms and applications*, pages 223–264. Springer, 2001.

W. Neiswanger, C. Wang, and E. Xing. Embarrassingly parallel variational inference in nonconjugate models. *arXiv preprint arXiv:1510.04163*, 2015.

A. Nemirovski, A. Juditsky, G. Lan, and A. Shapiro. Robust stochastic approximation approach to stochastic programming. *SIAM Journal on Optimization*, 19(4):1574–1609, 2009.

Y. Nesterov. *Introductory lectures on convex optimization: A basic course*, volume 87. Springer Science & Business Media, 2013.

J. Nocedal and S. J. Wright. *Numerical Optimization*. Springer Series in Operations Research. Springer, 2nd edition, 2006.

Y. Prabhu, A. Kag, S. Harsola, R. Agrawal, and M. Varma. Parabel: Partitioned label trees for extreme classification with application to dynamic search advertising. In *Proceedings of the 2018 World Wide Web Conference*, pages 993–1002. International World Wide Web Conferences Steering Committee, 2018.

S. Punjabi and P. Bhatt. Robust factorization machines for user response prediction. In *Proceedings of the 2018 World Wide Web Conference*, pages 669–678. International World Wide Web Conferences Steering Committee, 2018.

T. Qin, T.-Y. Liu, J. Xu, and H. Li. Letor: A benchmark collection for research on learning to rank for information retrieval. *Information Retrieval*, 13(4):346–374, 2010.

P. Raman, S. Srinivasan, S. Matsushima, X. Zhang, H. Yun, and S. Vishwanathan. Scaling multinomial logistic regression via hybrid parallelism. In *Proceedings of the 25th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining*, pages 1460–1470. ACM, 2019a.

P. Raman, S. Srinivasan, S. Matsushima, X. Zhang, H. Yun, and S. Vishwanathan. Scaling multinomial logistic regression via hybrid parallelism. In *Proceedings of the 25th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining*, pages 1460–1470. ACM, 2019b.

R. Ranganath, S. Gerrish, and D. M. Blei. Black box variational inference. In *aistats*, 2014.

B. Recht, C. Re, S. Wright, and F. Niu. Hogwild: A lock-free approach to parallelizing stochastic gradient descent. In *Advances in Neural Information Processing Systems*, pages 693–701, 2011.

S. Rendle. Factorization machines. In *Data Mining (ICDM), 2010 IEEE 10th International Conference on*, pages 995–1000. IEEE, 2010.

S. Rendle. Factorization machines with libFM. *ACM Trans. Intell. Syst. Technol.*, 3(3): 57:1–57:22, May 2012. ISSN 2157-6904.

S. Rendle. Scaling factorization machines to relational data. In *Proceedings of the VLDB Endowment*, volume 6, pages 337–348. VLDB Endowment, 2013.

S. Rendle, Z. Gantner, C. Freudenthaler, and L. Schmidt-Thieme. Fast context-aware recommendations with factorization machines. In *Proceedings of the 34th international ACM SIGIR conference on Research and development in Information Retrieval*, pages 635–644. ACM, 2011.

H. E. Robbins and S. Monro. A stochastic approximation method. *Annals of Mathematical Statistics*, 22:400–407, 1951.

C. Rudin. The p-norm push: A simple convex ranking algorithm that concentrates at the top of the list. *The Journal of Machine Learning Research*, 10:2233–2271, 2009.

O. Russakovsky, J. Deng, H. Su, J. Krause, S. Satheesh, S. Ma, Z. Huang, A. Karpa-

thy, A. Khosla, M. Bernstein, et al. Imagenet large scale visual recognition challenge. *International Journal of Computer Vision*, 115(3):211–252, 2015.

O. Shamir. Without-replacement sampling for stochastic gradient methods. In *Advances in Neural Information Processing Systems*, pages 46–54, 2016.

H. Steck. Training and testing of recommender systems on data missing not at random. In *Proceedings of the 16th ACM SIGKDD international conference on Knowledge discovery and data mining*, pages 713–722. ACM, 2010.

H. Sun, W. Wang, and Z. Shi. Parallel factorization machine recommended algorithm based on mapreduce. In *2014 10th International Conference on Semantics, Knowledge and Grids*, pages 120–123. IEEE, 2014.

P. Tseng and C. O. L. Mangasarian. Convergence of a block coordinate descent method for nondifferentiable minimization. *J. Optim Theory Appl*, pages 475–494, 2001.

N. Usunier, D. Buffoni, and P. Gallinari. Ranking with ordered weighted pairwise classification. In *Proceedings of the International Conference on Machine Learning*, 2009.

M. J. Wainwright and M. I. Jordan. Graphical models, exponential families, and variational inference. *Foundations and Trends in Machine Learning*, 1(1 – 2):1–305, 2008.

C. Wang and D. M. Blei. Variational inference in nonconjugate models. *Journal of Machine Learning Research*, 14(Apr):1005–1031, 2013.

P. Watcharapichat, V. L. Morales, R. C. Fernandez, and P. Pietzuch. Ako: Decentralised deep

learning with partial gradient exchange. In *Proceedings of the Seventh ACM Symposium on Cloud Computing*, pages 84–97. ACM, 2016.

J. Weston, S. Bengio, and N. Usunier. Wsabie: Scaling up to large vocabulary image annotation. In *Proceedings of the Twenty-Second international joint conference on Artificial Intelligence-Volume Volume Three*, pages 2764–2770. AAAI Press, 2011.

J. Weston, C. Wang, R. Weiss, and A. Berenzweig. Latent collaborative retrieval. *arXiv preprint arXiv:1206.4603*, 2012.

J. Winn and C. M. Bishop. Variational message passing. *Journal of Machine Learning Research*, 6(Apr):661–694, 2005.

J. Xiao, H. Ye, X. He, H. Zhang, F. Wu, and T.-S. Chua. Attentional factorization machines: Learning the weight of feature interactions via attention networks. *arXiv preprint arXiv:1708.04617*, 2017a.

L. Xiao, A. W. Yu, Q. Lin, and W. Chen. Dscovr: Randomized primal-dual block coordinate algorithms for asynchronous distributed optimization. *arXiv preprint arXiv:1710.05080*, 2017b.

P. Xie, J. K. Kim, Y. Zhou, Q. Ho, A. Kumar, Y. Yu, and E. P. Xing. Distributed machine learning via sufficient factor broadcasting. *CoRR*, 2015.

E. P. Xing, Q. Ho, W. Dai, J. K. Kim, J. Wei, S. Lee, X. Zheng, P. Xie, A. Kumar, and Y. Yu. Petuum: a new platform for distributed machine learning on big data. *Big Data, IEEE Transactions on*, 2015.

I. E.-H. Yen, X. Huang, P. Ravikumar, K. Zhong, and I. Dhillon. Pd-sparse : A primal and dual sparse approach to extreme multiclass and multilagrawal2013multoabel classification. In *Proceedings of The 33rd International Conference on Machine Learning*, pages 3069–3077, 2016.

H.-F. Yu, C.-J. Hsieh, H. Yun, S. Vishwanathan, and I. S. Dhillon. A scalable asynchronous distributed algorithm for topic modeling. In *Proceedings of the 24th International Conference on World Wide Web*, pages 1340–1350. International World Wide Web Conferences Steering Committee, 2015a.

H.-F. Yu, C.-J. Hsieh, H. Yun, S. Vishwanathan, and I. S. Dhillon. A scalable asynchronous distributed algorithm for topic modeling. In *WWW*, 2015b.

H. Yun. *Doubly Separable Models*. PhD thesis, Purdue University West Lafayette, 2014.

H. Yun, H.-F. Yu, C.-J. Hsieh, S. Vishwanathan, and I. Dhillon. Nomad: Non-locking, stochastic multi-machine algorithm for asynchronous and decentralized matrix completion. 2013.

H. Yun, P. Raman, and S. Vishwanathan. Ranking via robust binary classification. In *Advances in Neural Information Processing Systems*, 2014a.

H. Yun, H.-F. Yu, C.-J. Hsieh, S. Vishwanathan, and I. Dhillon. Nomad: Non-locking, stochastic multi-machine algorithm for asynchronous and decentralized matrix completion. *Proceedings of the VLDB Endowment*, 7(11):975–986, 2014b.

H. Yun, H.-F. Yu, C.-J. Hsieh, S. Vishwanathan, and I. Dhillon. Nomad: Non-locking,

stochastic multi-machine algorithm for asynchronous and decentralized matrix completion. *Proceedings of the VLDB Endowment*, 7(11):975–986, 2014c.

J. Zhang, P. Raman, S. Ji, H.-F. Yu, S. Vishwanathan, and I. Dhillon. Extreme stochastic variational inference: Distributed inference for large scale mixture models. In K. Chaudhuri and M. Sugiyama, editors, *Proceedings of Machine Learning Research*, volume 89 of *Proceedings of Machine Learning Research*, pages 935–943. PMLR, 2019.

X. Zhang, J. Liu, and Z. Zhu. Taming convergence for asynchronous stochastic gradient descent with unbounded delay in non-convex learning. *arXiv preprint arXiv:1805.09470*, 2018.

E. Zhong, Y. Shi, N. Liu, and S. Rajan. Scaling factorization machines with parameter server. In *Proceedings of the 25th ACM International on Conference on Information and Knowledge Management*, pages 1583–1592. ACM, 2016.

W. Zhong, J. Liu, M. Xue, and L. Jiao. A multiagent genetic algorithm for global numerical optimization. *Systems, Man, and Cybernetics, Part B: Cybernetics, IEEE Transactions on*, 34(2):1128–1141, 2004.

Y. Zhuang, Y.-C. Juan, and C.-J. Lin. A fast parallel stochastic gradient method for matrix factorization in shared memory systems. 2013.

M. Zinkevich, M. Weimer, L. Li, and A. J. Smola. Parallelized stochastic gradient descent. In *Advances in Neural Information Processing Systems*, pages 2595–2603, 2010.