NOMAD: A Distributed Framework for Latent Variable Models

Inderjit S. Dhillon

Department of Computer Science
University of Texas at Austin

Joint work with H.-F. Yu, C.-J. Hsieh, H. Yun, and S.V.N. Vishwanathan

NIPS 2014 Workshop:
Distributed Machine Learning and Matrix Computations
Outline

- Challenges
- Matrix Completion
  - Stochastic Gradient Method
  - Existing Distributed Approaches
  - Our Solution: NOMAD-MF
- Latent Dirichlet Allocation (LDA)
  - Gibbs Sampling
  - Existing Distributed Solutions: AdLDA, Yahoo LDA
  - Our Solution: F+NOMAD-LDA
Large-scale Latent Variable Modeling

- Latent Variable Models: very useful in many applications
  - Latent models for recommender systems (e.g., MF)
  - Topic models for document corpus (e.g., LDA)

- Fast growth of data
  - Almost $2.5 \times 10^{18}$ bytes of data added each day
  - 90% of the world’s data today was generated in the past two years
Challenges

- Algorithmic as well as hardware level
  - Many effective algorithms involve fine-grain iterative computation ⇒ hard to parallelize
  - Many current parallel approaches
    - bulk synchronization ⇒ wasted CPU power when communicating
    - complicated locking mechanism ⇒ hard to scale to many machines
    - asynchronous computation using parameter server ⇒ not serializable, danger of stale parameters

- Proposed NOMAD Framework
  - access graph analysis to exploit parallelism
  - asynchronous computation, non-blocking communication, and lock-free
  - serializable (or almost serializable)
  - successful applications: MF and LDA
Matrix Factorization: Recommender Systems
# Recommender Systems

**Rating Matrix**

<table>
<thead>
<tr>
<th></th>
<th>Movie 1</th>
<th>Movie 2</th>
<th>Items</th>
<th>Movie 10</th>
<th>Movie 11</th>
</tr>
</thead>
<tbody>
<tr>
<td>Users</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Cho-Ju</td>
<td>1</td>
<td>5</td>
<td>3</td>
<td>5</td>
<td>2</td>
</tr>
<tr>
<td>Si Si</td>
<td>2</td>
<td>3</td>
<td>5</td>
<td>2</td>
<td>5</td>
</tr>
<tr>
<td>Inderjit</td>
<td>3</td>
<td>5</td>
<td>?</td>
<td>3</td>
<td></td>
</tr>
<tr>
<td>Kai-Yang</td>
<td>2</td>
<td>5</td>
<td>3</td>
<td>4</td>
<td>2</td>
</tr>
<tr>
<td>Donghyuk</td>
<td>5</td>
<td>5</td>
<td>5</td>
<td>1</td>
<td></td>
</tr>
<tr>
<td>Koje</td>
<td>5</td>
<td>1</td>
<td>1</td>
<td>2</td>
<td>4</td>
</tr>
</tbody>
</table>

Inderjit Dhillon (UT Austin.)
Matrix Factorization Approach $A \approx WH^T$

$H^T$

$W$

\begin{array}{cccccccc}
-0.07 & -0.11 & -0.53 & -0.46 & -0.06 & -0.05 & -0.53 & -0.07 \\
0.13 & -0.42 & 0.45 & 0.17 & 0.25 & -0.17 & -0.18 & 0.27 \\
-0.21 & -0.43 & -0.23 & 0.16 & 0.08 & 0.17 & 0.57 & -0.39 \\
\end{array}

\begin{array}{cccc}
-8.72 & 0.03 & -1.03 \\
-7.56 & -0.79 & 0.62 \\
-4.07 & -3.95 & 2.55 \\
-3.52 & 3.73 & -3.32 \\
-7.78 & 2.34 & 2.33 \\
-2.44 & -5.29 & -3.92 \\
-1.78 & 1.90 & -1.68 \\
\end{array}
Matrix Factorization Approach $A \approx WH^T$

<table>
<thead>
<tr>
<th>$H^T$</th>
<th>$W$</th>
</tr>
</thead>
<tbody>
<tr>
<td>-0.07</td>
<td>-8.72</td>
</tr>
<tr>
<td>-0.11</td>
<td>-7.56</td>
</tr>
<tr>
<td>-0.53</td>
<td>-4.07</td>
</tr>
<tr>
<td>-0.46</td>
<td>-3.52</td>
</tr>
<tr>
<td>-0.06</td>
<td>-3.24</td>
</tr>
<tr>
<td>-0.05</td>
<td>2.34</td>
</tr>
<tr>
<td>-0.53</td>
<td>2.33</td>
</tr>
<tr>
<td>-0.07</td>
<td>2.44</td>
</tr>
<tr>
<td>-0.35</td>
<td>-5.29</td>
</tr>
<tr>
<td>-0.19</td>
<td>-3.92</td>
</tr>
<tr>
<td>-0.14</td>
<td>-1.78</td>
</tr>
<tr>
<td>-0.14</td>
<td>1.90</td>
</tr>
</tbody>
</table>

$$W = \begin{bmatrix}
1 & 5 & 3 & 5 & 2 \\
2 & 3 & 5 & 2 & 5 \\
2 & 5 & 3 & 4 & 2 \\
5 & 5 & 1 & 5 & 1 \\
1 & 1 & 2 & 4 & 4
\end{bmatrix}$$
Matrix Factorization Approach

\[
\min_{W \in \mathbb{R}^{m \times k}, H \in \mathbb{R}^{n \times k}} \sum_{(i,j) \in \Omega} (A_{ij} - w_i^T h_j)^2 + \lambda (\|W\|_F^2 + \|H\|_F^2),
\]

- \( \Omega = \{(i,j) \mid A_{ij} \text{ is observed}\} \)
- Regularized terms to avoid over-fitting

A transform maps users/items to latent feature space \(\mathbb{R}^k\)
- the \(i^{th}\) user \(\Rightarrow i^{th}\) row of \(W, w_i^T\),
- the \(j^{th}\) item \(\Rightarrow j^{th}\) column of \(H^T, h_j\).
- \(w_i^T h_j\): measures the interaction.
SGM: Stochastic Gradient Method

SGM update: pick \((i, j) \in \Omega\)

- \(R_{ij} \leftarrow A_{ij} - w_i^T h_j\),
- \(w_i \leftarrow w_i - \eta \left( \frac{\lambda}{|\Omega_i|} w_i - R_{ij} h_j \right)\),
- \(h_j \leftarrow h_j - \eta \left( \frac{\lambda}{|\bar{\Omega}_j|} h_j - R_{ij} w_i \right)\),

\(\Omega_i\): observed ratings of \(i\)-th row.
\(\bar{\Omega}_j\): observed ratings of \(j\)-th column.

An iteration: \(|\Omega|\) updates

- Time per update: \(O(k)\)
- Time per iteration: \(O(|\Omega|k)\), better than \(O(|\Omega|k^2)\) for ALS
Challenge: direct parallel updates $\Rightarrow$ memory conflicts.

- Multi-core parallelization
  - Hogwild [Niu 2011]
  - Jellyfish [Recht et al, 2011]
  - FPSGD** [Zhuang et al, 2013]

- Multi-machine parallelization:
  - DSGD [Gemulla et al, 2011]
  - DSGD ++ [Teflioudi et al, 2013]
Motivation

Most existing parallel approaches require

- **Synchronization** and/or
  - E.g., ALS, DSGD/JellyFish, DSGD++, CCD++
  - Computing power is wasted:
    - Interleaved computation and communication
    - Curse of the last reducer

- **Locking** and/or
  - E.g., parallel SGD, FPSGD**
  - A standard way to avoid conflict and guarantee *serializability*
  - Complicated remote locking slows down the computation
  - Hard to implement efficient locking on a distributed system

- **Computation using stale values**
  - E.g., Hogwild, Asynchronous SGD using parameter server
  - Lack of serializability

Q: Can we avoid both *synchronization* and *locking* but keep CPU from being *idle* and guarantee *serializability*?
Our answer: NOMAD

A: Yes, NOMAD keeps CPU and network busy simultaneously

- **Stochastic gradient** update rule
  - only a small set of variables involved

- **Nomadic token passing**
  - widely used in telecommunication area
  - avoids conflict without explicit remote locking
  - Idea: “owner computes”
  - NOMAD: multiple “active tokens” and nomadic passing

Features:

- fully asynchronous computation
- lock-free implementation
- non-blocking communication
- serializable update sequence
Access Graph for Stochastic Gradient

- Access graph $G = (V, E)$:
  - $V = \{w_i\} \cup \{h_j\}$
  - $E = \{e_{ij} : (i, j) \in \Omega\}$

- Connection to SG:
  - each $e_{ij}$ corresponds to a SG update
  - only access to $w_i$ and $h_j$

- Parallelism:
  - edges without common node can be updated in parallel
  - identify “matching” in the graph

- Nomadic Token Passing:
  - mechanism s.t. active edges always form a “matching”
  - serializability guaranteed
Nomadic Tokens for \( \{h_j\} \):

- \( n \) tokens
- \((j, h_j)\): \( O(k) \) space

Worker:

- \( p \) workers
- a computing unit + a concurrent token queue
- a block of \( W \): \( O(mk/p) \)
- a block row of \( A \): \( O(|\Omega|/p) \)
Illustration of NOMAD communication
Illustration of NOMAD communication
Illustration of NOMAD communication
Illustration of NOMAD communication
Illustration of NOMAD communication
Illustration of NOMAD communication
Illustration of NOMAD communication
Illustration of NOMAD communication
Illustration of NOMAD communication
Comparison on a Multi-core System

- On a 32-core processor with enough RAM.
- Comparison: NOMAD, FPSGD**, and CCD++.

![Graphs showing test RMSE over time for Netflix and Yahoo! datasets with different parameters and algorithms.](image-url)

Netflix, machines=1, cores=30, $\lambda = 0.05$, $k = 100$

Yahoo!, machines=1, cores=30, $\lambda = 1.00$, $k = 100$
Comparison on a Distributed System

- On a distributed system with 32 machines.
- Comparison: NOMAD, DSGD, DSGD++, and CCD++.

(100M ratings)

Netflix, machines=32, cores=4, $\lambda = 0.05$, $k = 100$

(250M ratings)

Yahoo!, machines=32, cores=4, $\lambda = 1.00$, $k = 100$
Super Linear Scaling of NOMAD-MF

Yahoo!, cores=4, $\lambda = 1.00$, $k = 100$

Inderjit Dhillon (UT Austin.)
Topic Modeling:
Latent Dirichlet Allocation
Each **topic** is a multinomial distribution over words

Each **document** is a multinomial distribution over topics

Each **word** is drawn from one of these topics

---

1 source:


Inderjit Dhillon (UT Austin.)
Graphical Model for LDA

Joint distribution

\[
Pr(\cdot) = \prod_{t=1}^{T} Pr(\phi_t \mid \beta) \prod_{i=1}^{l} Pr(\theta_i \mid \alpha) \left( \prod_{j=1}^{n_i} Pr(z_{i,j} \mid \theta_i)Pr(w_{i,j} \mid \phi_{z_{i,j}}) \right)
\]

- Pr(\phi_t \mid \beta), Pr(\theta_i \mid \alpha): Dirichlet distributions
- Pr(w \mid \phi_t), Pr(z \mid \theta_i): multinomial distributions
Inference for LDA

- Only documents are observed
- $\theta_t, \phi_t, z_{i,j}$ are latent
- Goal: infer these latent structures

Posterior Inference for LDA

Task: $Pr(\theta_i, \phi_t, z_{i,j} \mid \{d_i\}, \alpha, \beta)$
- Given
  - a corpus of documents $\{d_i : i = 1, \ldots, N\}$, $\alpha, \beta$
  - each document $d_i = \{w_{i,j} : j = 1, \ldots, n_i\}$
- Exact inference for $z_{i,j}, \theta_i, \phi_t$
  - Intractable
  - Latent variables are dependent when conditioned on data

Approximate Inference approaches:

- Variational Methods
  - See [Blei et al, 2003]
  - an optimization approach
  - runs faster
  - but generates biased results

- Gibbs Samplings
  - See [Griffiths & Steyvers, 2004]
  - an MCMC approach
  - more accurate
  - but slower with a vanilla implementation

Goal: Design a scalable Gibbs sampler for LDA
Gibbs Sampling for LDA [Griffiths & Steyvers, 2004]

- Count matrices for topic assignment \( \{z_{i,j}\} \):
  - \( n_{dt} \): \# words of document \( d \) assigned to topic \( t \)
  - \( n_{wt} \): \# of times word \( w \) assigned to topic \( t \)
  - \( n_t := \sum_w n_{wt} = \sum_d n_{dt} \)

- Gibbs Sampling Step
  1. choose \( w := w_{i,j} \) with old assignment \( t_o := z_{i,j} \) of document \( d := d_i \)
  2. Decrease \( n_{dt_o}, n_{wt_o}, n_{t_o} \) by 1
  3. Resample a new assignment \( t_n := z_{i,j} \) according to

\[
Pr(z_{i,j} = t) \propto \frac{(n_{dt} + \alpha)(n_{wt} + \beta)}{n_t + \bar{\beta}}, \quad \forall t = 1, \ldots, T.
\]

  4. Increase \( n_{dt_n}, n_{wt_n}, n_{t_n} \) by 1

- Constants
  - \( J \): vocabulary size
  - \( \bar{\beta} = \beta \times J \)
Access Pattern for Gibbs Sampling

$n_t$ Topics

Words

$n_{wt}$

$n_{dt}$ Docs

$Z_{ij}$
## Multinomial Sampling Techniques for $\mathbf{p} \in \mathbb{R}_+^T$

<table>
<thead>
<tr>
<th></th>
<th>Initialization</th>
<th>Generation</th>
<th>Parameter Update</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Time</td>
<td>Space</td>
<td>Time</td>
</tr>
<tr>
<td>LSearch</td>
<td>$\Theta(T)$</td>
<td>$\Theta(1)$</td>
<td>$\Theta(T)$</td>
</tr>
<tr>
<td>BSearch</td>
<td>$\Theta(T)$</td>
<td>$\Theta(1)$</td>
<td>$\Theta(\log T)$</td>
</tr>
<tr>
<td>Alias Method</td>
<td>$\Theta(T)$</td>
<td>$\Theta(T)$</td>
<td>$\Theta(1)$</td>
</tr>
<tr>
<td>F+tree Sampling</td>
<td>$\Theta(T)$</td>
<td>$\Theta(1)$</td>
<td>$\Theta(\log T)$</td>
</tr>
</tbody>
</table>

- **LSearch**
  - maintain $c_T = \mathbf{p}^T \mathbf{1}$
  - linear search
  - $\Theta(1)$ update

- **BSearch**
  - maintain $\mathbf{c} = \text{cumsum}(\mathbf{p})$
  - binary search
  - no support for update

- **Alias Method**
  - Alias table
  - construction has **some overhead**
  - no support for updates

- **F+tree**
  - a variant of Fenwick tree
  - construction has **low overhead**
  - logarithmic time for sampling and update
Construction in $\Theta(T)$ time

$p = [0.3, 1.5, 0.4, 0.3]^T$
F+Tree: Sampling

- Multinomial sampling in $\Theta(\log T)$ time
- Initial $u$: a uniformly number drawn from $[0, F[1])$
F+Tree: Update

- Update in $\Theta(\log T)$ time
- $p_3 \leftarrow p_3 + \delta$
Decomposition of $p$

\[
p_t = \frac{(n_{dt} + \alpha)(n_{wt} + \beta)}{n_t + \beta}, \quad \forall t = 1, \ldots, T.
\]

\[
= \beta \left( \frac{n_{dt} + \alpha}{n_t + \beta} \right) + n_{wt} \left( \frac{n_{dt} + \alpha}{n_t + \beta} \right).
\]

\[\text{(1)}\]

- $p = \beta q + r$
  - two-level sampling for $p$
  - $q$ is dense
    - only 2 entries ($q_{t_o}$, $q_{t_n}$) change for each Gibbs step in the same document
    - use F+Tree for $q$
  - $r$ is sparse
    - nonzero entries: $T_w := \{t : n_{tw} \neq 0\}$
    - entire $r$ changes for each Gibbs step
    - use BSearch for $r$
- Can also work on word-by-word update sequence
**F+LDA: Alternative Decomposition**

- **Word-by-word** Gibbs sampling sequence
- **Decomposition of** $p$

\[ p_t = \frac{(n_{dt} + \alpha)(n_{wt} + \beta)}{n_t + \beta}, \quad \forall t = 1, \ldots, T. \]

\[ = \alpha \left( \frac{n_{wt} + \beta}{n_t + \beta} \right) + n_{dt} \left( \frac{n_{wt} + \beta}{n_t + \beta} \right). \quad (2) \]

- **$p = \alpha q + r$**
- **$q$: slight changes for this sequence $\Rightarrow$ use F+Tree**
- **$r$: $|T_d := \{t : n_{dt} \neq 0\}|$ nonzeros $\Rightarrow$ use BSearch**
Comparison to Other LDA Sampling

<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>Decomposition</td>
<td>( \alpha \left( \frac{n_{wt} + \beta}{n_t + \beta} \right) + n_{dt} \left( \frac{n_{wt} + \beta}{n_t + \beta} \right) )</td>
<td>( \beta \left( \frac{n_{dt} + \alpha}{n_t + \beta} \right) + n_{wt} \left( \frac{n_{dt} + \alpha}{n_t + \beta} \right) )</td>
<td>( \frac{\alpha \beta}{n_t + \beta} + \beta \left( \frac{n_{dt} + \alpha}{n_t + \beta} \right) + n_{wt} \left( \frac{n_{dt} + \alpha}{n_t + \beta} \right) )</td>
<td>( \alpha \left( \frac{n_{wt} + \beta}{n_t + \beta} \right) + n_{dt} \left( \frac{n_{wt} + \beta}{n_t + \beta} \right) )</td>
</tr>
<tr>
<td>Structure</td>
<td>F+tree BSearch</td>
<td>F+tree BSearch</td>
<td>LSearch LSearch</td>
<td>Alias LSearch</td>
</tr>
<tr>
<td>Fresh samples</td>
<td>Yes</td>
<td>Yes</td>
<td>Yes</td>
<td>No</td>
</tr>
<tr>
<td>Initialization</td>
<td>( \Theta(\log T) )</td>
<td>( \Theta(\frac{</td>
<td>T_d</td>
<td>}{T}) )</td>
</tr>
<tr>
<td>Sampling</td>
<td>( \Theta(\log T) )</td>
<td>( \Theta(\log</td>
<td>T_d</td>
<td>) )</td>
</tr>
</tbody>
</table>

- **F+LDA**: word-by-word faster than doc-by-doc for large \( I \)
  - \( |T_d| \) bounded by \( n_i \), but \( |T_w| \) approaches to \( T \)
  - per Gibbs step cost: \( \rho_F \log T + \rho_B |T_d| \)

- **SparseLDA**:
  - per Gibbs step cost: \( \Theta(T + |T_d| + |T_w|) \)
  - the first \( \Theta(T) \) rarely happens but \( |T_w| \to T \) for large \( I \)

- **AliasLDA**:
  - per Gibbs step cost: \( \rho_A |T_d| + \#MH \)
  - \( \rho_A \approx 3 \times \rho_B \): construction overhead of Alias table
  - If \( (\rho_A - \rho_B) |T_d| > \rho_F \log T \Rightarrow \) AliasLDA slower than F+LDA
  - say \( |T_d| \approx 100 \), F+LDA still faster for \( T < 2^{50} \)
Comparison of various sampling methods

- Single machine, single thread
- y-axis: speedup over normal $O(T)$ multinomial sampling
- $T = 1024$
- Enron: 38K docs with 6M tokens
- NyTimes: 0.3M docs with 100M tokens
Access Pattern for Gibbs Sampling

- $n_t$: Topics
- $n_{dt}$: Docs
- $n_{wt}$: Words
- $Z_{ij}$
Access Graph for Gibbs Sampling

- $G = (V, E)$: a hyper graph
  
  $V = \{d_i\} \cup \{w_j\} \cup \{s\}$
  
  $E = \{e_{ij} = (d_i, w_j, s)\}$

- Connection to Gibbs sampling
  
  $\begin{align*}
  (d_i)_t &:= n_{d_it},
  (w_j)_t &:= n_{wjt},
  (s)_t &:= nt
  \end{align*}$
  
  each $e_{ij}$: a Gibbs step for word $w_j$ in $d_i$
  
  access to $(d_i, w_j, s)$

- Parallelism: more challenging
  
  all edges incident to $s$
  
  all $(s)_t$ are large in general
  
  $\Rightarrow$ slightly stale $s$ is fine for accuracy
  
  duplicate $s$ for parallelism
Nomadic Tokens for $w_j$

Nomadic Tokens for
$\{w_j : j = 1, \ldots, J\}$:
- $J$ tokens
- $(j, w_j)$: $O(T)$ space

Worker:
- $p$ workers
- a computing unit + a concurrent token queue
- a subset of $\{d_i\}$: $O(IT/p)$
- “x”: an occurrence of a word
- bigger rectangle: a subset of corpus
- smaller rectangle: a unit subtask
Nomadic Token for $s$: Circular Delta Update

- Single global $s$
  - travels among machines as a messenger
  - broadcasts local delta updates
- Every machine $p$: $(s_p, \bar{s})$
  - $s_p$: local working copy
  - $\bar{s}$: snapshot version of global $s$

\[
s \leftarrow s + (s_3 - \bar{s}) \\
\bar{s} \leftarrow s \\
s_3 \leftarrow s
\]
Comparison on a single multi-core machine

- On a machine with a 20-core processor
- Comparison: F+NOMAD LDA, Yahoo! LDA
- PubMed: 9M docs with 700M tokens
- Amazon: 30M docs with 1.5B tokens

[Graphs showing performance comparison between F+NOMAD LDA and Yahoo! LDA on PubMed and Amazon datasets]
Comparison on a Multi-machine System

- 32 machines, each with a 20-core processor, $T = 1024$
- Comparison: F+NOMAD LDA, Yahoo! LDA
- Amazon: 30M docs with 1.5B tokens
- UMBC: 40M docs with 1.5B tokens
Scaling of NOMAD-LDA

- Each machine has 20 cores
- Multi-machine setting:
  - 18 cores for sampling
  - 2 cores for communication
Conclusions

- NOMAD framework uses nomadic tokens to provide
  - Asynchronous computation
  - Non-blocking communication
  - Lock-free implementation
  - Serializable or near Serializable

- Recommender System: Matrix factorization
  - scalable parallel stochastic gradient
  - Serializability guarantee

- Topic Modeling: Latent Dirichlet Allocation
  - Logarithmic F+tree sampling
  - Efficient Gibbs Sampling
  - Duplicated nomadic tokens for the common node
  - Outperforms Yahoo! LDA