An Exploration of Optimization Algorithms for High Performance Tensor Completion

Shaden Smith\(^1\), Jongsoo Park\(^2\), and George Karypis\(^1\)

\(^1\)University of Minnesota

\(^2\)Parallel Computing Lab, Intel Corporation
Tensor Introduction

CPD (or: CANDECOMP/PARAFAC)

- Given: tensor $\mathcal{R}^{I \times J \times K}$ and desired rank $F$
- Compute: low-rank matrices $A^{I \times F}, B^{J \times F}, C^{K \times F}$
- Element-wise: $\mathcal{R}(i, j, k) \approx \sum_{f=1}^{F} A(i, f)B(j, f)C(k, f)$
Tensor Storage - Compressed Sparse Fiber (CSF)

(Smith & Karypis ’15)

- Values are stored in the leaves (not shown).
- Modes are recursively compressed.
  - Compression naturally exposes opportunities for operation savings.

\[
\begin{bmatrix}
  i & j & k & l \\
  1 & 1 & 2 \\
  1 & 1 & 3 \\
  1 & 2 & 1 \\
  1 & 2 & 2 \\
  2 & 2 & 1 \\
  2 & 2 & 3 \\
  2 & 2 & 2 \\
\end{bmatrix}
\rightarrow
\begin{array}{c}
  i \\
  j \\
  k \\
  l \\
\end{array}
\begin{array}{c}
  1 \\
  2 \\
  2 \\
\end{array}
\begin{array}{c}
  1 \\
  2 \\
  1 \\
\end{array}
\begin{array}{c}
  1 \\
  2 \\
  1 \\
\end{array}
\begin{array}{c}
  2 \\
  2 \\
  3 \\
\end{array}
\begin{array}{c}
  3 \\
  3 \\
  1 \\
\end{array}
\begin{array}{c}
  1 \\
  1 \\
  3 \\
\end{array}
\begin{array}{c}
  2 \\
  1 \\
  2 \\
\end{array}
\begin{array}{c}
  2 \\
  2 \\
\end{array}
\]
Objective

- We only want to model *observed* entries (non-zeros).
  - A least-squares objective would predict zeros!
- The objective is a combination of the prediction ability (the *loss*) and regularization terms (to prevent overfitting).
  - Regularization is controlled by $\lambda$, a user-specified parameter.

$$\minimize_{A,B,C} \underbrace{\mathcal{L}(\mathcal{R}, A, B, C)}_{\text{Loss}} + \lambda \underbrace{\left( \|A\|_F^2 + \|B\|_F^2 + \|C\|_F^2 \right)}_{\text{Regularization}}$$

$$\mathcal{L}(\mathcal{R}, A, B, C) = \frac{1}{2} \sum_{\text{nnz}(\mathcal{R})} \left( \mathcal{R}(i, j, k) - \sum_{f=1}^{F} A(i, f)B(j, f)C(k, f) \right)^2$$
Challenges

Optimization Algorithms

- Optimization algorithms for matrix completion are relatively mature
  - How do they adapt to tensors?
- We must consider multiple properties when comparing algorithms:
  1. Number of operations
  2. Convergence rate
  3. Computational intensity
  4. Parallelism

Tensor Properties

- Most matrix optimization algorithms parallelize over the many rows and columns (e.g., users and items).
- Many domains have a mix of short and long modes.
  - Context-aware recommender systems will have orders of magnitude fewer contexts than users or items.
Table of Contents

1 Introduction

2 Alternating Least Squares

3 Stochastic Gradient Descent

4 Coordinate Descent

5 Experiments

6 Conclusions
Alternating Least Squares (ALS)

Problem Formation

- Hold $B$ and $C$ constant, solving for $A$ convex.
- Each row of $A$ is a linear least squares problem.
- $H_i$ is an $|\mathcal{R}(i,:,::)| \times F$ matrix:
  - $\mathcal{R}(i,j,k) \rightarrow B(j,:) \ast C(k,:)$. (element-wise multiplication)
- $A(i,:) \leftarrow (H_i^T H_i + \lambda I)^{-1} H_i^T \text{vec}(\mathcal{R}(i,:,::))$
- $O(F^2)$ work per non-zero.
Alternating Least Squares (ALS)

Shao ’12, Karlsson ’15

- Normal equations $N_i = H_i^T H_i$ are formed one non-zero at a time.
- $H_i^T \text{vec}(\mathcal{R}(i,:,:))$ is similarly accumulated into a vector $q_i$.

Algorithm 1 ALS: updating $A(i,:)$

1: $N_i \leftarrow 0^{F \times F}$  
2: $q_i \leftarrow 0^{F \times 1}$  
3: for $(i,j,k) \in \mathcal{R}(i,:,:)$ do  
4:   $x \leftarrow B(j,:) \ast C(k,:)$  
5:   $N_i \leftarrow N_i + x^T x$  
6:   $q_i \leftarrow q_i + \mathcal{R}(i,j,k)x^T$  
7: end for  
8: $A(i,:) \leftarrow (N_i + \lambda I)^{-1} q_i$
ALS - Parallelism

**Shared-Memory (Shao ’12)**
- Least squares problems are solved in batches of size $B = \mathcal{O}(100)$.
- Each core independently accumulates the $B$ sets of $N_i$ and $q_i$.
- Corresponding $N_i$ and $q_i$ are aggregated.
- Finally, the $B$ inversions and updates are performed in parallel.

**Distributed-Memory (Karlsson ’15)**
- Non-zeros can be distributed in any fashion.
- All $N_i$ and $q_i$ aggregated (MPI_Allreduce).
  - $\mathcal{O}(IF^2)$ data communicated per process.
- Processes evenly divide the inversions and then exchange updates (MPI_Allgather).
Contributions - Shared Memory

Tensor Representations

- Storing multiple representations of $\mathcal{R}$ allows us to parallelize over rows of $\mathbf{A}$, $\mathbf{B}$, and $\mathbf{C}$.
  - No parallel reductions or synchronization required.
  - Each core only requires $O(F^2)$ intermediate storage.
- If mode is short, use method of (Shao ’12) with a single batch of size equal to the dimension of that mode.

BLAS-3 Formulation

- Element-wise computation is an outer product formulation.
  - $F^2$ work with $F^2$ data per non-zero.
- Instead, we store $(\mathbf{B}(j,:) \ast \mathbf{C}(k,:))$ into rows of a thread-local buffer $\mathbf{Z}$.
  - When $\mathbf{Z}$ is full, do a rank-$k$ update: $\mathbf{N}_i \leftarrow \mathbf{N}_i + \mathbf{Z}^T \mathbf{Z}$. 
Contributions - Distributed Memory

Coarse-Grained Decomposition (following Shin & Kang ’14)

- Avoid communicating normal equations by using separate 1D decompositions of $A$, $B$, and $C$.
- Each process owns all necessary non-zeros and only needs to exchange the updated factor rows.
- If mode is short, use method of (Karlsson ’15) with MPI_Allreduce.

George Karypis, karypis@cs.umn.edu
# Table of Contents

1. Introduction

2. Alternating Least Squares

3. Stochastic Gradient Descent

4. Coordinate Descent

5. Experiments

6. Conclusions
Stochastic Gradient Descent (SGD)

Problem Formulation

- Randomly select entry $\mathcal{R}(i,j,k)$ and update rows of $A$, $B$, and $C$.
  - $O(F)$ work per non-zero.
- $\eta$ is the step size; typically $O(10^{-3})$.

$$
\delta \leftarrow \mathcal{R}(i,j,k) - \sum_{f=1}^{F} A(i,f)B(j,f)C(k,f)
$$

$$
A(i,:) \leftarrow A(i,:) + \eta \left[ \delta (B(j,:) * C(k,:)) - \lambda A(i,:) \right],
$$

$$
B(j,:) \leftarrow B(j,:) + \eta \left[ \delta (A(i,:) * C(k,:)) - \lambda B(j,:) \right],
$$

$$
C(k,:) \leftarrow C(k,:) + \eta \left[ \delta (A(i,:) * B(j,:)) - \lambda C(k,:) \right].
$$
SGD - Stratification

Beutel ’14

- *Strata* identify independent blocks of non-zeros.
- Each stratum is processed in parallel.

Limitation of Stratification

- There is only as much parallelism as the smallest dimension.
- Sparsely populated strata are communication bound.
Contributions - SGD

Problem Relaxation: Cheat!

- Shared-memory: go Hogwild! and allow race conditions.
- Distributed-memory: limit the number of strata to reduce communication and handle short modes.
- Assign multiple processes to the same stratum (called a *team*).
- Each performs updates on its own versions of the factors.
- At the end, the updates are exchanged among the team.
Coordinate Descent (CCD++)

- Rank-1 factors are updated in sequence.
- $\mathcal{O}(F)$ work per non-zero (same as SGD).
CCD++ - Parallelism

Distributed-Memory (Karlsson '15, Shin '15)

- Each entry of $A(:, f)$ is computed in parallel.
  - Distributing non-zeros requires $\alpha_i$ and $\beta_i$ to be aggregated.
  - Communication volume is $O(IF)$ per process.
- All $\delta_{ijk}$ can be maintained in a residual tensor.
  - All updates are totally parallel - no communication needed.

\[
\delta_{ijk} \leftarrow \mathcal{R}(i, j, k) - \sum_{f=1}^{F} A(i, f)B(j, f)C(k, f)
\]

\[
\alpha_i \leftarrow \sum_{\mathcal{R}(i,:,:)} \delta_{ijk} (B(j, f)C(k, f))
\]

\[
\beta_i \leftarrow \sum_{\mathcal{R}(i,:,:)} (B(j, f)C(k, f))^2
\]

\[
A(i, f) \leftarrow \frac{\alpha_i}{\lambda + \beta_i}
\]
Contributions - Shared Memory

**CSF Formulation**

- Column-wise methods require $F$ passes over the sparse tensor.
  - CCD++ requires a high memory bandwidth.
- CSF shrinks the memory footprint of the tensor and structures memory accesses.
  - Fewer operations and a reduced memory bandwidth.
- One example is during residual computation:

\[
\mathbf{v} \leftarrow \mathbf{A}(i,:) \ast \mathbf{B}(j,:), \\
\delta_{ijk} \leftarrow \mathbf{R}(i, j, k) - \sum_{f=1}^{F} \mathbf{v}(f)C(k, f), \\
\delta_{ijk'} \leftarrow \mathbf{R}(i, j, k') - \sum_{f=1}^{F} \mathbf{v}(f)C(k', f).
\]
Contributions - Distributed Memory

Medium-Grained Decomposition (Smith & Karypis ’16)

- Distributing non-zeros over a grid limits communication to the grid layer.
- For short modes, we use a grid dimension of one and fully replicate the factor.
  - Non-zeros are still distributed and processed in parallel.
Table of Contents

1 Introduction

2 Alternating Least Squares

3 Stochastic Gradient Descent

4 Coordinate Descent

5 Experiments

6 Conclusions
### Experimental Setup

#### Tensor Dataset
- We use Yahoo! Music ratings from the 2011 KDD Cup.
- 1M users $\times$ 625K songs $\times$ 133 months with 210M ratings.
- More datasets in paper (SC’16)

#### Computing Environment
- All experiments performed on the Cori supercomputer at NERSC.
- Nodes have two sixteen-core Intel processors (Haswell).
- Implemented as part of open source library SPLATT.
  - Written in C with hybrid MPI+OpenMP parallelism.
Strong Scaling - Rank 10

![Graph showing time per epoch vs. number of cores for ALS, SGD, and CCD++ methods.](image)

- **ALS**
- **SGD**
- **CCD++**

George Karypis, karypis@cs.umn.edu

High Performance Tensor Completion 19 / 23
Benchmarking - Rank 10

- base-ALS and base-CCD++ from Karlsson ’15 (C++ and MPI).

![Graph showing time per epoch vs number of cores for base-ALS, splatt-ALS, base-CCD++, and splatt-CCD++ algorithms.](image-url)
Convergence @ 1 core

- Convergence is detected if the RMSE was not improved after 20 epochs.
Convergence @ 1024 cores

- Convergence is detected if the RMSE was not improved after 20 epochs.
Table of Contents

1 Introduction

2 Alternating Least Squares

3 Stochastic Gradient Descent

4 Coordinate Descent

5 Experiments

6 Conclusions
Conclusions

Optimization Algorithms

- We scale ALS, SGD, and CCD++ to (past?) 1024 cores.
- SGD is best for small-scale systems.
- ALS is more expensive but shows fastest convergence at scale.
- CCD++ strong scales best.
  - Maybe overtake ALS convergence at larger scale?

Release

- Paper to appear in SC’16
- Pre-print and source code to come next month:
  - http://cs.umn.edu/~splatt/