An Exploration of Optimization Algorithms for High Performance Tensor Completion

Shaden Smith\textsuperscript{1}\textsuperscript{*}, Jongsoo Park\textsuperscript{2}, and George Karypis\textsuperscript{1}

\textsuperscript{1}Department of Computer Science & Engineering, University of Minnesota
\textsuperscript{2}Parallel Computing Lab, Intel Corporation
*shaden@cs.umn.edu
Outline

Introduction & Preliminaries
   Tensor Completion
   Evaluation Criteria

Optimization Algorithms
   Alternating Least Squares
   Coordinate Descent
   Stochastic Gradient Descent

Comparison of Optimization Methods

Conclusions
Table of Contents

Introduction & Preliminaries
   Tensor Completion
   Evaluation Criteria

Optimization Algorithms

Comparison of Optimization Methods

Conclusions
Tensor introduction

- Tensors are the generalization of matrices to $\geq 3D$.
- Tensors have $N$ dimensions (or *modes*).
  - We will use dimensions $I \times J \times K$ in this talk.
Tensor completion

- Many tensors are sparse due to missing or unknown data.
  - Missing values are not treated as zero.
- Assumption: the underlying data is low rank.
- Tensor completion estimates a low rank model to recover missing entries.
  - Applications: precision healthcare, product recommendation, cybersecurity, and others.
Many tensors are sparse due to missing or unknown data.
- Missing values are not treated as zero.

Assumption: the underlying data is low rank.

Tensor completion estimates a low rank model to recover missing entries.
- Applications: precision healthcare, product recommendation, cybersecurity, and others.

The canonical polyadic decomposition (CPD) models a tensor as the summation of rank-1 tensors.
Tensor completion with the CPD

\( \mathcal{R}(i, j, k) \) is written as the inner product of \( \mathbf{A}(i,:) \), \( \mathbf{B}(j,:) \), and \( \mathbf{C}(k,:) \).

\[
\text{minimize } \mathbf{A}, \mathbf{B}, \mathbf{C} \\
L(\mathcal{R}, \mathbf{A}, \mathbf{B}, \mathbf{C}) + \lambda (||\mathbf{A}||_2^2 + ||\mathbf{B}||_2^2 + ||\mathbf{C}||_2^2)
\]
Tensor completion with the CPD

$\mathcal{R}(i, j, k)$ is written as the inner product of $A(i,:)$, $B(j,:)$, and $C(k,:)$. We arrive at a non-convex optimization problem:

$$\min_{A, B, C} \mathcal{L}(\mathcal{R}, A, B, C) + \lambda \left( \|A\|_F^2 + \|B\|_F^2 + \|C\|_F^2 \right)$$

where

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

Optimization algorithms

▶ Algorithms for matrix completion are relatively mature.
  ▶ How do their tensor adaptations perform on HPC systems?
▶ Several properties to consider when comparing algorithms:
  1. Convergence rate.
  2. Number of operations and computational intensity.
  3. Memory footprint.
  4. Parallelism!
Experimental setup

- Source code was implemented as part of SPLATT with MPI+OpenMP.
- Experiments are on the Cori supercomputer at NERSC.
  - Nodes have two sixteen-core Intel processors (Haswell).
- Experiments show a rank-10 factorization of the Yahoo Music (KDD cup) tensor.
  - 210 million user-song-month ratings.
  - More datasets and ranks in the paper.
- Root-mean-squared error (RMSE) on a test set measures solution quality:
  \[
  \text{RMSE} = \sqrt{\frac{2 \cdot \mathcal{L}(\mathcal{R}, \mathbf{A}, \mathbf{B}, \mathbf{C})}{\text{nnz}(\mathcal{R})}}
  \]
Table of Contents

Introduction & Preliminaries

Optimization Algorithms
   Alternating Least Squares
   Coordinate Descent
   Stochastic Gradient Descent

Comparison of Optimization Methods

Conclusions
Alternating least squares (ALS)

- Each row of $A$ is a linear least squares problem.
- $H_i$ is an $|\mathcal{R}(i,:,:)| \times F$ matrix:
  - $\mathcal{R}(i,j,k) \rightarrow B(j,:) \times C(k,:)$ (elementwise multiplication).
- $A(i,:) \leftarrow \left( H_i^T H_i + \lambda I \right)^{-1} H_i^T \text{vec}(\mathcal{R}(i,:,:))$. 

$$A = \begin{bmatrix} 1 & 2 & 3 \\ 4 & 5 & 6 \end{bmatrix}$$
Parallel ALS

- We impose a 1D partition on each of the factors.
- Non-zeros are then distributed according to the row partitionings.
- Only the updated rows need to be communicated.
- If mode is short, cooperatively form rows and aggregate the normal equations.
ALS evaluation

295\times relative speedup and 153\times speedup over base-ALS.

base-ALS is a pure-MPI implementation in C++ [Karlsson et al. ’15]. ALS is our MPI+OpenMP implementation with one MPI rank per node.
Coordinate descent (CCD++)

- Select a variable and update while holding all others constant.
- Rank-1 factors are updated in sequence.
Compressed sparse fiber (CSF)

- CSF is a generalization of the CSR structure for matrices.
- Paths from roots to leaves encode non-zeros.
- CSF reduces the memory bandwidth of the tensor and also structures accesses to the factors.

```
1  1  1  1  2  1  2  1  2  3
1  1  2  1  1  1  3  1  2  2
1  1  2  2  2  2  1  1  2  2
1  1  2  2  2  2  2  1  2  3
```
**CCD++ distributed-memory evaluation**

685× relative speedup and 21× speedup over base-CCD++.

---

base-CCD++ is a pure-MPI implementation in C++ [Karlsson et al. ’15].

CCD++ is our MPI+OpenMP implementation with two MPI ranks per node.
Stochastic gradient descent (SGD)

- Randomly select entry $\mathcal{R}(i,j,k)$ and update $A$, $B$, and $C$.
  - $O(F)$ work per non-zero.

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

$$
A(i,:) \leftarrow A(i,:) + \eta [\delta_{ijk} (B(j,:) * C(k,:)) - \lambda A(i,:)]
$$

$$
B(j,:) \leftarrow B(j,:) + \eta [\delta_{ijk} (A(i,:) * C(k,:)) - \lambda B(j,:)]
$$

$$
C(k,:) \leftarrow C(k,:) + \eta [\delta_{ijk} (A(i,:) * B(j,:)) - \lambda C(k,:)]
$$

$\eta$ is the step size; typically $O(10^{-3})$. 

Stratified SGD

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

Limitations of stratified SGD:

- There is only as much parallelism as the smallest dimension.
- Sparsely populated strata are communication bound.
Asynchronous SGD (ASGD)

- Processes overlap updates and exchange to avoid divergence.
  - Local solutions are combined via a weighted sum.
- Go Hogwild! on shared-memory systems.

Limitations of ASGD:

- Convergence suffers unless updates are frequently exchanged.
Hybrid stratified/asynchronous SGD

- Limit the number of strata to reduce communication.
- Assign multiple processes to the same stratum (called a *team*).
- Each process performs updates on its own local factors.
- At the end of a strata, updates are exchanged among the team.
Hybrid stratification combines the speed of ASGD with the stability of stratification.

Hybrid uses sixteen teams of four MPI processes.
Table of Contents

Introduction & Preliminaries

Optimization Algorithms

Comparison of Optimization Methods

Conclusions
Strong scaling

- SGD exhibits initial slowdown as strata teams are populated.
- All methods scale to (past) 1024 cores.
Convergence @ 1 core

SGD rapidly converges to a high quality solution.

Convergence is detected if the RMSE does not improve after 20 epochs.
Convergence @ 1024 cores

- ALS now has the lowest time-to-solution.
- CCD++ and SGD exhibit similar convergence rates.

Convergence is detected if the RMSE does not improve after 20 epochs.
Table of Contents

Introduction & Preliminaries

Optimization Algorithms

Comparison of Optimization Methods

Conclusions
Wrapping Up

▶ Careful attention to sparsity and data structures can give over $10 \times$ speedups.
▶ There is no “best” algorithm – it depends on your hardware architecture and problem.
  ▶ SGD: best in a serial setting.
  ▶ ALS: best in a multi-core setting or with a few nodes, but has a large memory footprint.
  ▶ CCD++: best on large-scale systems, but requires high memory-bandwidth.

http://cs.umn.edu/~splatt/
Backup Slides
Patents strong scaling

Patents is a $46 \times 240K \times 240K$ tensor with 2.9B non-zeros.
Parallel CCD++

- Shared-memory: each entry of $\mathbf{A}(\cdot, f)$ is computed in parallel.
- Distributing non-zeros with a 3D grid limits communication to the grid layers.
  - Distributing non-zeros requires $\alpha_i$ and $\beta_i$ to be aggregated.
  - Communication volume is $\mathcal{O}(IF)$ per process.
- For short modes, use a grid dimension of 1 and fully replicate the factor.
Alternating least squares (ALS)

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

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

1: $N_i \leftarrow 0^{F \times F}$, $q_i \leftarrow 0^{F \times 1}$
2: for $(i, j, k) \in \mathcal{R}(i,:,\cdot)$ do
3: $x \leftarrow B(j,:) \ast C(k,:)$
4: $N_i \leftarrow N_i + x^T x$
5: $q_i \leftarrow q_i + \mathcal{R}(i,j,k)x^T$
6: end for
7: $A(i,:) \leftarrow (N_i + \lambda I)^{-1}q_i$
Element-wise computation is an outer product formulation.

- $O(F^2)$ work with $O(F^2)$ data per non-zero.
- Instead, append $(B(j,:) \ast C(k,:))$ to a matrix $Z$.
- When $Z$ is full, do a rank-$k$ update: $N_i \leftarrow N_i + Z^T Z$.

**Algorithm 2 ALS: updating $A(i,:)$**

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

- $\mathcal{O}(F)$ work per non-zero.
- Each epoch requires $NF$ passes over the tensor.
  - Heavily dependent on memory bandwidth.

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

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

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

\[
A(i, f) \leftarrow \frac{\alpha_i}{\beta_i + \lambda}
\]
Netflix strong scaling
Communication volume on Yahoo!

Figure: Average communication volume per node on the Yahoo! dataset. CCD++ and SGD use two MPI ranks per node and ALS uses one.
Amazon strong scaling

Time per epoch (s)

Nodes

1 2 4 8 16 32

ALS
CCD++
SGD
Scaling factorization rank on 1024 cores

Figure: Effects of increasing factorization rank on the Yahoo! dataset.