Parallel Numerical Algorithms
Chapter 6 – Structured and Low Rank Matrices
Section 6.3 – Numerical Optimization

Michael T. Heath and Edgar Solomonik

Department of Computer Science
University of Illinois at Urbana-Champaign

CS 554 / CSE 512
Outline

1. Alternating Least Squares
   - Quadratic Optimization
   - Parallel ALS

2. Coordinate Descent
   - Coordinate Descent
   - Cyclic Coordinate Descent

3. Gradient Descent
   - Gradient Descent
   - Stochastic Gradient Descent
   - Parallel SGD

4. Nonlinear Optimization
   - Nonlinear Equations
   - Optimization
Quadratic Optimization: Matrix Completion

Given a subset of entries

\[ \Omega \subseteq \{1, \ldots, m\} \times \{1, \ldots, n\} \]

of the entries of matrix \( A \in \mathbb{R}^{m \times n} \), seek rank-\( k \) approximation

\[
\arg\min_{W \in \mathbb{R}^{m \times k}, H \in \mathbb{R}^{n \times k}} \sum_{(i,j) \in \Omega} \left( a_{ij} - \sum_l w_{il} h_{jl} \right)^2 + \lambda (\|W\|_F + \|H\|_F)
\]

- Problems of these type studied in sparse approximation
- \( \Omega \) may be randomly selected sample subset
- Methods for this problem are typical of numerical optimization and machine learning
Alternating least squares (ALS) fixes $W$ and solves for $H$ then vice versa until convergence.

- Each step improves approximation, convergence to a minimum expected given satisfactory starting guess.
- We have a quadratic optimization problem:

\[
\arg\min_{W \in \mathbb{R}^{m \times k}} \sum_{(i,j) \in \Omega} \left( a_{ij} - \sum_l w_{il} h_{jl} \right)^2 + \lambda \|W\|_F
\]

- The optimization problem is independent for rows of $W$.
- Letting $w_i = w_i^\star$, $h_i = h_i^\star$, $\Omega_i = \{j : (i,j) \in \Omega\}$, seek

\[
\arg\min_{w_i \in \mathbb{R}^k} \sum_{j \in \Omega_i} \left( a_{ij} - w_i h_j^T \right)^2 + \lambda \|w_i\|_2
ALS: Quadratic Optimization

Seek minimizer $w_i$ for quadratic vector equation

$$f(w_i) = \sum_{j \in \Omega_i} \left(a_{ij} - w_i h_j^T\right)^2 + \lambda \|w_i\|$$

- Differentiating with respect to $w_i$ gives

$$\frac{\partial f(w_i)}{\partial w_i} = 2 \sum_{j \in \Omega_i} h_j^T \left(w_i h_j^T - a_{ij}\right) + 2\lambda w_i = 0$$

- Rotating $w_i h_j^T = h_j w_i^T$ and defining $G^{(i)} = \sum_{j \in \Omega_i} h_j^T h_j$,

$$\left(G^{(i)} + \lambda I\right) w_i^T = \sum_{j \in \Omega_i} h_j^T a_{ij}$$

which is a $k \times k$ symmetric linear system of equations
For updating each \( w_i \), ALS is dominated in cost by two steps:

1. \( G^{(i)} = \sum_{j \in \Omega_i} h_j^T h_j \)
   - dense matrix-matrix product
   - \( O(|\Omega_i|k^2) \) work
   - logarithmic depth

2. Solve linear system with \( G^{(i)} + \lambda I \)
   - dense symmetric \( k \times k \) linear solve
   - \( O(k^3) \) work
   - typically \( O(k) \) depth

Can do these for all \( m \) rows of \( W \) independently.
Let each task optimize a row $w_i$ of $W$

- Need to compute $G^{(i)}$ for each task
- Specific subset of rows of $H$ needed for each $G^{(i)}$
- Task execution is embarrassingly parallel if all of $H$ stored on each processor
Memory-Constrained Parallel ALS

May not have enough memory to replicate $H$ on all processors

- Communication required and pattern is data-dependent
- Could rotate rows of $H$ along a ring of processors
- Each processor computes contributions to the $G^{(i)}$ it owns
- Requires $\Theta(p)$ latency cost for each iteration of ALS
Updating a Single Variable

Rather than whole rows $w_i$ solve for elements of $W$, recall

$$\arg\min_{W \in \mathbb{R}^{m \times k}} \sum_{(i,j) \in \Omega} \left( a_{ij} - \sum_l w_{il} h_{jl} \right)^2 + \lambda \|W\|_F$$

- **Coordinate descent** finds the best replacement $\mu$ for $w_{it}$

$$\mu = \arg\min_{\mu} \sum_{j \in \Omega_i} \left( a_{ij} - \mu h_{jt} - \sum_{l \neq t} w_{il} h_{jl} \right)^2 + \lambda \mu^2$$

- The solution is given by

$$\mu = \frac{\sum_{j \in \Omega_i} h_{jt} \left( a_{ij} - \sum_{l \neq t} w_{il} h_{jl} \right)}{\lambda + \sum_{j \in \Omega_i} h_{jt}^2}$$
For $\forall (i, j) \in \Omega$ compute elements $r_{ij}$ of

$$R = A - WH^T$$

so that we can optimize via

$$\mu = \frac{\sum_{j \in \Omega_i} h_{jt} \left( a_{ij} - \sum_{l \neq t} w_{il} h_{jl} \right)}{\lambda + \sum_{j \in \Omega_i} h_{jt}^2} = \frac{\sum_{j \in \Omega_i} h_{jt} \left( r_{ij} + w_{it} h_{jt} \right)}{\lambda + \sum_{j \in \Omega_i} h_{jt}^2}$$

after which we can update $R$ via

$$r_{ij} \leftarrow r_{ij} - (\mu - w_{it}) h_{jt} \quad \forall j \in \Omega_i$$

both using $O(|\Omega_i|)$ operations
Cyclic Coordinate Descent (CCD)

- Updating $w_i$ costs $O(|Ω_i|k)$ operations with coordinate descent rather than $O(|Ω_i|k^2 + k^3)$ operations with ALS.
- By solving for all of $w_i$ at once, ALS obtains a more accurate solution than coordinate descent.
- Coordinate descent with different update orderings:
  - *Cyclic coordinate descent (CCD)* updates all columns of $W$ then all columns of $H$ (ALS-like ordering).
  - *CCD++* alternates between columns of $W$ and $H$.
  - All entries within a column can be updated concurrently.
Yu, Hsieh, Si, and Dhillon 2013 propose using a row-blocked layout of $H$ and $W$

- They keep track of a corresponding $m/p$ and $n/p$ rows and columns of $A$ and $R$ on each processor (using twice the minimal amount of memory)

- Every column update in CCD++ is then fully parallelized, but an allgather of each column is required to update $R$

- The complexity of updating all of $W$ and all of $H$ is then

$$T_p(m, n, k) = \Theta(kT_p^{\text{allgather}}(m+n) + \gamma Q_1(m, n, k)/p)$$

$$= O(\alpha k \log p + \beta (m+n)k + \gamma |\Omega|k/p)$$
Gradient-Based Update

ALS minimizes $w_i$, gradient descent methods only improve it

- Recall that we seek to minimize

$$f(w_i) = \sum_{j \in \Omega_i} \left(a_{ij} - w_i h_j^T\right)^2 + \lambda ||w_i||$$

and use the partial derivative

$$\frac{\partial f(w_i)}{\partial w_i} = 2 \sum_{j \in \Omega_i} h_j^T \left(w_i h_j^T - a_{ij}\right) + 2\lambda w_i = 2\left(\lambda w_i - \sum_{j \in \Omega_i} r_{ij} h_j\right)$$

- \textit{Gradient descent} method updates

$$w_i = w_i - \eta \frac{\partial f(w_i)}{\partial w_i}$$

where parameter $\eta$ is our step-size
Stochastic Gradient Descent (SGD)

Stochastic gradient descent (SGD) performs fine-grained updates based on a component of the gradient

- Again the full gradient is

\[
\frac{\partial f(w_i)}{\partial w_i} = 2 \left( \lambda w_i - \sum_{j \in \Omega_i} r_{ij} h_j \right) = 2 \sum_{j \in \Omega_i} \lambda w_i / |\Omega_i| - r_{ij} h_j
\]

- SGD selects random \((i, j) \in \Omega\) and updates \(w_i\) using \(h_j\)

\[
w_i \leftarrow w_i - \eta \left( \lambda w_i / |\Omega_i| - r_{ij} h_j \right)
\]

- SGD then updates \(r_{ij} = a_{ij} - w_i^T h_j\)

- Each update costs \(O(k)\) operations
Asynchronous SGD

Parallelizing SGD is easy aside from ensuring concurrent updates do not conflict

- Asynchronous shared-memory implementations of SGD are popular and achieve high performance
- For sufficiently small step-size, inconsistencies among updates (e.g. duplication) are not problematic statistically
- Asynchronicity can slow down convergence
Blocked SGD

Distributed blocking SGD introduces further considerations

- Associate a task with updates on a block
- Can define $p \times p$ grid of blocks of dimension $m/p \times n/p$
- Diagonal/superdiagonals/subdiagonals of blocks updated independently, so $p$ tasks can execute concurrently
- Assuming $\Theta(|\Omega|/p^2)$ updates are performed on each block, the execution time for $|\Omega|$ updates is

$$T_p(m, n, k) = \Theta(\alpha p \log p + \beta \min(m, n)k + \gamma|\Omega|k/p)$$
Nonlinear Equations

Potential sources of parallelism in solving nonlinear equation $f(x) = 0$ include:

- Evaluation of function $f$ and its derivatives in parallel
- Parallel implementation of linear algebra computations (e.g., solving linear system in Newton-like methods)
- Simultaneous exploration of different regions via multiple starting points (e.g., if many solutions are sought or convergence is difficult to achieve)
Sources of parallelism in optimization problems include

- Evaluation of objective and constraint functions and their derivatives in parallel
- Parallel implementation of linear algebra computations (e.g., solving linear system in Newton-like methods)
- Simultaneous exploration of different regions via multiple starting points (e.g., if global optimum is sought or convergence is difficult to achieve)
- Multi-directional searches in direct search methods
- Decomposition methods for structured problems, such as linear, quadratic, or separable programming
References


References


References – Parallel Optimization