Parallel Numerical Algorithms

Chapter 6 – Matrix Models
Section 6.2 – Low Rank Approximation

Edgar Solomonik

Department of Computer Science
University of Illinois at Urbana-Champaign

CS 554 / CSE 512
Outline

1. Low Rank Approximation by SVD
   - Truncated SVD
   - Fast Algorithms with Truncated SVD

2. Computing Low Rank Approximations
   - Direct Computation
   - Indirect Computation

3. Randomness and Approximation
   - Randomized Approximation Basics
   - Structured Randomized Factorization

4. Hierarchical Low-Rank Structure
   - HSS Matrix–Vector Multiplication
   - Parallel HSS Matrix–Vector Multiplication
Rank-$k$ Singular Value Decomposition (SVD)

For any matrix $A \in \mathbb{R}^{m \times n}$ of rank $k$ there exists a factorization

$$A = U D V^T$$

- $U \in \mathbb{R}^{m \times k}$ is a matrix of orthonormal left singular vectors
- $D \in \mathbb{R}^{k \times k}$ is a nonnegative diagonal matrix of singular values in decreasing order $\sigma_1 \geq \cdots \geq \sigma_k$
- $V \in \mathbb{R}^{n \times k}$ is a matrix of orthonormal right singular vectors
Truncated SVD

Given $A \in \mathbb{R}^{m \times n}$ seek its best $k < \text{rank}(A)$ approximation

$$B = \arg\min_{B \in \mathbb{R}^{m \times n}, \text{rank}(B) \leq k} (\|A - B\|_2)$$

- Eckart-Young theorem: given SVD

$$A = \begin{bmatrix} U_1 & U_2 \end{bmatrix} \begin{bmatrix} D_1 & 0 \\ 0 & D_2 \end{bmatrix} \begin{bmatrix} V_1 \\ V_2 \end{bmatrix}^T \Rightarrow B = U_1 D_1 V_1^T$$

where $D_1$ is $k \times k$.

- $U_1 D_1 V_1^T$ is the rank-$k$ truncated SVD of $A$ and

$$\|A - U_1 D_1 V_1^T\|_2 = \min_{B \in \mathbb{R}^{m \times n}, \text{rank}(B) \leq k} (\|A - B\|_2) = \sigma_{k+1}$$
Given a rank $k$ truncated SVD $A \approx UDV^T$ of $A \in \mathbb{R}^{m \times n}$ with $m \geq n$

- Performing approximately $y = Ax$ requires $O(mk)$ work
  \[ y \approx U(D(V^T x)) \]

- Solving $Ax = b$ requires $O(mk)$ work via approximation
  \[ x \approx VD^{-1}U^T b \]
Computing the Truncated SVD

Reduction to upper-Hessenberg form via two-sided orthogonal updates can compute full SVD

- Given full SVD can obtain truncated SVD by keeping only largest singular value/vector pairs
- Given set of transformations $Q_1, \ldots, Q_s$ so that $U = Q_1 \cdots Q_s$, can obtain leading $k$ columns of $U$ by computing

$$U_1 = Q_1 \left( \cdots \left( Q_s \begin{bmatrix} I \\ 0 \end{bmatrix} \right) \right)$$

- This method requires $O(mn^2)$ work for the computation of singular values and $O(mnk)$ for $k$ singular vectors
Computing the Truncated SVD by Krylov Subspace Methods

Seek $k \ll m, n$ leading right singular vectors of $A$

- Find a basis for Krylov subspace of $B = A^T A$
- Rather than computing $B$, compute products $Bx = A^T(Ax)$
- For instance, do $k' \geq k + O(1)$ iterations of Lanczos, to yield $BQ = QH$
- Left singular vectors can be obtained by working with $C = AA^T$
- This method requires $O(mnk)$ work for $k$ singular vectors
- However, $\Theta(k)$ sparse-matrix-vector multiplications are needed (high latency and low flop/byte ratio)
A matrix $A \in \mathbb{R}^{m \times n}$ is rank $k$, if for some $X \in \mathbb{R}^{m \times k}$, $Y \in \mathbb{R}^{n \times k}$ with $k \leq \min(m, n)$,

$$A = XY^T$$

If $A = XY^T$ (exact low rank factorization), we can obtain reduced SVD $A = UDV^T$ via

1. $[U_1, R] = QR(X)$
2. $[U_2, D, V] = \text{SVD}(RY^T)$
3. $U = U_1U_2$

with cost $O(mk^2)$ using an SVD of a $k \times k$ rather than $m \times n$ matrix.

If instead $\|A - XY^T\|_2 \leq \varepsilon$ then $\|A - UDV^T\|_2 \leq \varepsilon$

So we can obtain a truncated SVD given an optimal generic low-rank approximation.
Rank-Revealing QR

If $A$ is of rank $k$ and its first $k$ columns are linearly independent

$$A = Q \begin{bmatrix} R_{11} & R_{12} \\ 0 & 0 \\ 0 & 0 \end{bmatrix}$$

where $R_{11}$ is upper-triangular and $k \times k$ and $Q = YTY^T$ with $n \times k$ matrix $Y$

- For arbitrary $A$ we need column ordering permutation $P$

  $$A = QRP$$

- **QR with column pivoting** (due to Gene Golub) is an effective method for this
  - pivot so that the leading column has largest 2-norm
  - method can break in the presence of roundoff error (see Kahan matrix), but is very robust in practice
QR with column pivoting can be used to either

- determine the (numerical) rank of \( A \)
- compute a low-rank approximation with a bounded error

performs only \( O(mnk) \) rather than \( O(mn^2) \) work for a full QR or SVD
In distributed-memory, column pivoting poses further challenges:

- Need at least one message to decide on each pivot column, which leads to $\Omega(k)$ synchronizations.
- Existing work tries to pivot many columns at a time by finding subsets of them that are sufficiently linearly independent.
- Randomized approaches provide alternatives and flexibility.
Randomization Basics

Intuition: consider a random vector $w$ of dimension $n$, all of the following holds with high probability in exact arithmetic:

- Given any basis $Q$ for the $n$ dimensional space, random $w$ is not orthogonal to any row of $Q^T$.
- Let $A = UDV^T$ where $V^T \in \mathbb{R}^{n \times k}$.
- Vector $w$ is not orthogonal to any row of $V^T$, consequently $z = V^T w$ is a random vector.
- $Aw = UDz$ is random linear combination of cols of $UD$.
- Given $k$ random vectors, i.e., random matrix $W \in \mathbb{R}^{n \times k}$.
- Columns of $B = AW$ gives $k$ random linear combinations of columns of in $UD$.
- $B$ has the same span as $U$!
Using the Basis to Compute a Factorization

If $B$ has the same span as the range of $A$

- $[Q, R] = \text{QR}(B)$ gives orthogonal basis $Q$ for $B = AW$
- $QQ^T A = QQ^T UDV^T = (QQ^T U)DV^T$, now $Q^T U$ is orthogonal and so $QQ^T U$ is a basis for the range of $A$
- so compute $H = Q^T A$, $H \in \mathbb{R}^{k \times n}$ and compute $[U_1, D, V] = \text{SVD}(H)$
- then compute $U = QU_1$ and we have a rank $k$ truncated SVD of $A$

$$A = UDV^T$$
Matrix multiplications e.g. $AW$, all require $O(mnk)$ operations.

QR and SVD require $O((m + n)k^2)$ operations.

If $k \ll \min(m, n)$ the bulk of the computation here is within matrix multiplication, which can be done with fewer synchronizations and higher efficiency than QR with column pivoting or Arnoldi.
Now let's consider the case when $A = UDV^T + E$ where $D \in \mathbb{R}^{k \times k}$ and $E$ is a small perturbation.

- $E$ may be noise in data or numerical error.
- To obtain a basis for $U$ it is insufficient to multiply by random $B \in \mathbb{R}^{n \times k}$, due to influence of $E$.
- However, oversampling, for instance $l = k + 10$, and random $B \in \mathbb{R}^{n \times l}$ gives good results.
- A Gaussian random distribution provides particularly good accuracy.
- So far the dimension of $B$ has assumed knowledge of the target approximate rank $k$, to determine it dynamically generate vectors (columns of $B$) one at a time or a block at a time, which results in a provably accurate basis.
The cost of the randomized algorithm for is

\[ T_p^{\text{MM}}(m, n, k) + T_p^{\text{QR}}(m, k, k) \]

which means that the work is \( O(mnk) \) and the algorithm is well-parallelizable.

This assumes we factorize the basis by QR and SVD of \( R \).
Fast Algorithms via Pseudo-Randomness

We can lower the number of operations needed by the randomized algorithm by generating $B$ so that $AB$ can be computed more rapidly.

- Generate $W$ as a pseudo-random matrix

$$ B = DFR $$

- $D$ is diagonal with random elements
- $F$ can be applied to a vector in $O(n \log(n))$ operations
  - e.g. DFT or Hadamard matrix $H_{2n} = \begin{bmatrix} H_n & H_n \\ H_n & -H_n \end{bmatrix}$
- $R$ is $p \approx k$ columns of the $n \times n$ identity matrix
- Computes $AB$ with $O(mn \log(n))$ operations (if $m > n$)
Instead of matrix multiplication, apply $m$ FFTs of dimension $n$

- Each FFT is independent, so it suffices to perform a single transpose

- So we have the following overall cost

\[
O\left(\frac{mn \log(n)}{p} \cdot \gamma\right) + T_p\text{all-to-all}(mn/p) + T_p^{QR}(m, k, k)
\]

assuming $m > n$

- This is lower with respect to the unstructured/randomized version, however, this idea does not extend well to the case when $A$ is sparse
Consider two-way partitioning of vertices of a graph

The connectivity within each partition is given by a block diagonal matrix

\[
\begin{bmatrix}
A_1 & \\
& A_2
\end{bmatrix}
\]

If the graph is nicely *separable* there is little connectivity between vertices in the two partitions

Consequently, it is often possible to approximate the off-diagonal blocks by low-rank factorization

\[
\begin{bmatrix}
A_1 & U_1 D_1 V_1^T \\
U_2 D_2 V_2^T & A_2
\end{bmatrix}
\]

Doing this recursively to \(A_1\) and \(A_2\) yields a matrix with hierarchical low-rank structure
HSS Matrix, Two Levels

Hierarchical semi-separable (HSS) matrix, space padded around each matrix block, which are uniquely identified by dimensions and color.
HSS Matrix, Three Levels
HSS Matrix Formal Definition

- The \( l \)-level HSS factorization is described by
  \[
  \mathcal{H}_l(A) = \begin{cases} 
  \{U, V, T_{12}, T_{21}, A_{11}, A_{22}\} & : l = 1 \\
  \{U, V, T_{12}, T_{21}, \mathcal{H}_{l-1}(A_{11}), \mathcal{H}_{l-1}(A_{22})\} & : l > 1 
  \end{cases}
  \]

- The low-rank representation of the diagonal blocks is given by
  \[
  A_{21} = \bar{U}_2 T_{21} \bar{V}_1^T, \quad A_{12} = \bar{U}_1 T_{12} \bar{V}_2^T
  \]
  where for \( a \in \{1, 2\} \),

  \[
  \bar{U}_a = U_a(\mathcal{H}_l(A)) = \begin{cases} 
  U_a & : l = 1 \\
  U_1(\mathcal{H}_{l-1}(A_{aa})) & : l > 1 
  \end{cases}
  \]

  \[
  \bar{V}_a = V_a(\mathcal{H}_l(A)) = \begin{cases} 
  V_a & : l = 1 \\
  V_1(\mathcal{H}_{l-1}(A_{aa})) & : l > 1 
  \end{cases}
  \]
We now consider computing $y = Ax$.

- With $\mathcal{H}_1(A)$ we would just compute:
  
  $y_1 = A_{11}x_1 + U_1(T_{12}(V_2^T x_2))$ and
  
  $y_2 = A_{22}x_2 + U_2(T_{21}(V_1^T x_1))$

- For general $\mathcal{H}_l(A)$ perform up-sweep and down-sweep:
  
  - up-sweep computes $w = \begin{bmatrix} \bar{V}_1^Tx_1 \\ \bar{V}_2^Tx_2 \end{bmatrix}$ at every tree node
  
  - down-sweep computes a tree sum of $\begin{bmatrix} \bar{U}_1T_{12}w_2 \\ \bar{U}_2T_{21}w_1 \end{bmatrix}$
The up-sweep is performed by using the nested structure of $\bar{V}$

$$w = \mathcal{W}(\mathcal{H}_l(A), x) = \begin{cases} \begin{bmatrix} V_1^T & 0 \\ 0 & V_2^T \end{bmatrix} x & : l = 1 \\ \begin{bmatrix} V_1^T & 0 \\ 0 & V_2^T \end{bmatrix} \begin{bmatrix} \mathcal{W}(\mathcal{H}_{l-1}(A_{11}), x_1) \\ \mathcal{W}(\mathcal{H}_{l-1}(A_{22}), x_2) \end{bmatrix} & : l > 1 \end{cases}$$
HSS Matrix–Vector Multiplication, Down-Sweep

Use \( w = \mathcal{W}(\mathcal{H}_l(A), x) \) from the root to the leaves to get

\[
y = Ax = \begin{bmatrix} U_1 T_{12} w_2 \\ U_2 T_{21} w_1 \end{bmatrix} + \begin{bmatrix} A_{11} x_1 \\ A_{22} x_2 \end{bmatrix} = \begin{bmatrix} \bar{U}_1 & 0 \\ 0 & \bar{U}_2 \end{bmatrix} \begin{bmatrix} 0 & T_{12} \\ T_{21} & 0 \end{bmatrix} w + \begin{bmatrix} A_{11} & 0 \\ 0 & A_{22} \end{bmatrix} x
\]


- using the nested structure of \( \bar{U}_a \) and \( v = \begin{bmatrix} U_1 & 0 \\ 0 & U_2 \end{bmatrix} \begin{bmatrix} 0 & T_{12} \\ T_{21} & 0 \end{bmatrix} w, \)

\[
y_a = \begin{bmatrix} U_1(\mathcal{H}_{l-1}(A_{aa})) & 0 \\ 0 & U_2(\mathcal{H}_{l-1}(A_{aa})) \end{bmatrix} v_a + A_{aa} x_a \text{ for } a \in \{1, 2\}
\]

which gives the down-sweep recurrence

\[
y = Ax + z = \mathcal{Y}(\mathcal{H}_l(A), x, z) = \begin{cases} 
\begin{bmatrix} U_1 q_1 \\ U_2 q_2 \end{bmatrix} + \begin{bmatrix} A_{11} x_1 \\ A_{22} x_2 \end{bmatrix} : l = 1 \\
\mathcal{Y}(\mathcal{H}_{l-1}(A_{11}), x_1, U_1 q_1) \\
\mathcal{Y}(\mathcal{H}_{l-1}(A_{22}), x_2, U_2 q_2) \end{cases} : l > 1
\]

where \( q = \begin{bmatrix} 0 & T_{12} \\ T_{21} & 0 \end{bmatrix} w + z \)
We can express the \( n \)-element prefix sum \( y(i) = \sum_{j=1}^{i-1} x(j) \) as

\[
y = Lx \quad \text{where} \quad L = \begin{bmatrix} L_{11} & 0 \\ L_{21} & L_{22} \end{bmatrix} = \begin{bmatrix} 0 & 0 & \cdots & 0 \\ 1 & 0 & \cdots & \vdots \\ \vdots & \vdots & \ddots & \vdots \\ 1 & \cdots & 1 & 0 \end{bmatrix}
\]

- \( L \) is an \( \mathcal{H} \)-matrix since \( L_{21} = \mathbf{1}_n \mathbf{1}_n^T = [1 \cdots 1]^T [1 \cdots 1] \)
- \( L \) also has rank-1 HSS structure, in particular

\[
\mathcal{H}_l(L) = \begin{cases} \left\{ \begin{bmatrix} 1_2, 1_2, [0] \end{bmatrix}, \begin{bmatrix} 1 \end{bmatrix}, \begin{bmatrix} 0 \end{bmatrix}, \begin{bmatrix} 0 \end{bmatrix} \right\} & : l = 1 \\
\left\{ \begin{bmatrix} 1_4, 1_4, [0] \end{bmatrix}, \begin{bmatrix} 1 \end{bmatrix}, \mathcal{H}_{l-1}(L_{11}), \mathcal{H}_{l-1}(L_{22}) \right\} & : l > 1 
\end{cases}
\]

so each \( U, V, \bar{U}, \bar{V} \) is a vector of 1s, \( T_{12} = [0] \) and \( T_{21} = [1] \)
We can use the HSS structure of $L$ to compute the prefix sum of $x$

- recall that the up-sweep recurrence has the general form

$$ w = \mathcal{W}(\mathcal{H}_l(A), x) = \begin{cases} 
    V_1^T 0 & \mathcal{W}(\mathcal{H}_{l-1}(A_{11}), x_1) \\
    0 V_2^T & \mathcal{W}(\mathcal{H}_{l-1}(A_{22}), x_2) 
\end{cases} 
$$

- for the prefix sum this becomes

$$ w = \mathcal{W}(\mathcal{H}_l(L), x) = \begin{cases} 
    x & : l = 1 \\
    1 1 0 0 & \mathcal{W}(\mathcal{H}_{l-1}(L_{11}), x_1) \\
    0 0 1 1 & \mathcal{W}(\mathcal{H}_{l-1}(L_{22}), x_2) 
\end{cases} 
$$

- so the up-sweep computes $w = \begin{bmatrix} S(x_1) \\ S(x_2) \end{bmatrix}$ where $S(y) = \sum_i y_i$
Prefix Sum HSS Down-Sweep

The down-sweep has the general structure

\[
y = \mathcal{Y}(\mathcal{H}_l(A), x, z) = \begin{cases}
  \begin{bmatrix} U_1 & 0 \\ 0 & U_2 \end{bmatrix} q + \begin{bmatrix} A_{11} & 0 \\ 0 & A_{22} \end{bmatrix} x : l = 1 \\
  \mathcal{Y}(\mathcal{H}_{l-1}(A_{11}), x_1, U_1 q_1) \quad : l > 1 \\
  \mathcal{Y}(\mathcal{H}_{l-1}(A_{22}), x_2, U_2 q_2) 
\end{cases}
\]

where \( q = \begin{bmatrix} 0 \\ T_{21} \\ T_{12} \end{bmatrix} \mathcal{W}(\mathcal{H}_l(A), x) + z \), for the prefix sum

\[
\begin{bmatrix} 0 \\ T_{21} \\ 0 \end{bmatrix} \mathcal{W}(\mathcal{H}_l(L), x) = \begin{bmatrix} 0 & 0 \\ 1 & 0 \end{bmatrix} \begin{bmatrix} S(x_1) \\ S(x_2) \end{bmatrix} = \begin{bmatrix} 0 \\ S(x_1) \end{bmatrix} = q - z
\]

\[
y = \mathcal{Y}(\mathcal{H}_l(L), x, z) = \begin{cases}
  \begin{bmatrix} z_1 \\ x_1 + z_2 \end{bmatrix} : l = 1 \\
  \mathcal{Y}(\mathcal{H}_{l-1}(L_{11}), x_1, 1_2 z_1) \\
  \mathcal{Y}(\mathcal{H}_{l-1}(L_{22}), x_2, 1_2 (S(x_1) + z_2)) : l > 1
\end{cases}
\]

Initially the prefix \( z = 0 \) and it will always be the case that \( z_1 = z_2 \)
The down-sweep and the up-sweep perform small dense matrix–vector multiplications at each recursive step

- Let’s assume $k$ is the dimension of the leaf blocks and the rank at each level (number of columns in each $U_a, V_a$)

- The work for both the down-sweep and up-sweep is

  \[ Q(n, k) = 2Q(n/2, k) + O(k^2 \cdot \gamma), \quad Q(k, k) = O(k^2 \cdot \gamma) \]

  \[ Q(n, k) = O(nk \cdot \gamma) \]

- The depth of the algorithm scales as $D = \Theta(\log(n))$ for fixed $k$
If we assign each tree node to a single processor for the first $\log_2(p)$ levels, and execute a different leaf subtree with a different processor

\[
T_p(n, k) = 2T_{p/2}(n, k) + O(k^2 \cdot \gamma + k \cdot \beta + \alpha) \\
= O((nk/p + k^2 \log(p)) \cdot \gamma + k \log(p) \cdot \beta + \log(p) \cdot \alpha)
\]
Synchronization-Efficient HSS Multiplication

- The leaf subtrees can be computed independently
  \[ T_{p}^{\text{leaf-subtrees}}(n, k) = O(nk/p \cdot \gamma + k \cdot \beta + \alpha) \]

- Focus on up-sweep and down-sweep with \( \log_2(p) \) levels

- Executing the root subtree sequentially takes time
  \[ T_{p}^{\text{root-subtree}}(pk, k) = O(pk^2 \cdot \gamma + pk \cdot \beta + \alpha) \]

- Instead have \( p^r \) (\( r < 1 \)) processors compute subtrees with \( p^{1-r} \) leaves, then recurse on the \( p^r \) roots
  \[ T_{p}^{\text{rec-tree}}(pk, k) = T_{p^r}^{\text{rec-tree}}(p^r k, k) + O(p^{1-r} k^2 \cdot \gamma + p^{1-r} k \cdot \beta + \alpha) \]
  \[ = O(p^{1-r} k^2 \cdot \gamma + p^{1-r} k \cdot \beta + \log_{1/r}(\log(p)) \cdot \alpha) \]
Focus on the top tree with $p$ leaves (leaf subtrees)

Assign each processor a unique path from a leaf to the root

Given $w = \mathcal{W}(\mathcal{H}_l(A), x)$ at every node each processor can compute a down-sweep path in the subtree independently

For the up-sweep, realize that the tree applies a linear transformation, so can sum the results computed in each path

For each tree node, there is a contribution from every processor assigned a leaf of the subtree of the node

Therefore, there are $p - 1$ sums of a total of $O(p \log(p))$ contributions, for a total of $O(kp \log(p))$ elements

Do these with $\min(p, k \log_2(p))$ processors, each obtaining $\max(p, k \log_2(p))$ contributions, so

$$T_{p}^{\text{root-paths}}(k) = O(k^2 \log(p) \cdot \gamma + (k \log(p) + p) \cdot \beta + \log(p) \cdot \alpha)$$
Consider multiplication $C = AB$ where $A \in \mathbb{R}^{n \times n}$ is HSS and $B \in \mathbb{R}^{n \times b}$.

- Let's consider the case that $p \leq b \ll n$.
- If we assign each processor all of $A$, each can compute a column of $C$ simultaneously.
- This requires a prohibitive amount of memory usage.
  - Perform leaf-level multiplications, processing $n/p$ rows of $B$ with each processor (call intermediate $\bar{C}$).
  - Transpose $\bar{C}$ and apply $\log_2(p)$ root levels of HSS tree to columns of $\bar{C}$ independently.
- This algorithm requires replication only of the root $O(\log(p))$ levels of the HSS tree, $O(pb)$ data.
- For large $k$ or larger $p$ different algorithms may be desirable.
References


References


