Single-Pass PCA of Large High-Dimensional Data

Wenjian Yu¹, Yu Gu¹, Jian Li¹, Shenghua Liu², Yaohang Li³

¹Department of Computer Science and Technology, Tsinghua University, Beijing 100084, China
²Institute of Computing Technology, Chinese Academy of Science
³Department of Computer Science, Old Dominion University, USA

yu-wj@tsinghua.edu.cn
• Introduction
• Technical Background
• The Single-Pass Algorithm for PCA
• Experimental Results
• Conclusion
Introduction

- Background
  - Principal component analysis (PCA)
  - An open problem: calculate PCA of large-size and high-dimensional dense data in a limited-memory computer
  - A single-pass algorithm: particularly useful / efficient, for data stored in slow memory or streaming data
  - There are single-pass PCA algorithms for SPSD matrix or low-dimensional data, but the study for the algorithm for more general matrices is not sufficient.
Introduction

- **Randomized matrix algorithm**
  - Has advantages over traditional algorithms (like SVD) (faster runtime, better parallelism, pass-efficient; suitable for large data)
  - **randQB** based on *random projection* \([1]\)
    
    \[
    A \approx QB
    \]
    
    Q captures the dominant actions of A
    Small sketch B facilitates computation
  - Applied to computing PCA \([2]\)
  - Useful for distributed PCA; excellent performance on parallel computers
  - A blocked version for rank-revealing matrix factorization \([3]\)


Our contribution

- We reconstruct the blocked randQB algorithm \cite{3} to obtain a single-pass PCA algorithm

- **Single-pass**
  - involves only one pass over specified large high-dimensional data

- **Efficiency**
  - \(O(mnk)\) time complexity and \(O(k(m+n))\) space complexity for computing k-PCA, and well adapts to parallel computing

- **Accuracy**
  - same theoretic error bounds as the randomized blocked algorithm; much less error than its counterpart

Technical Background – SVD and PCA

- Truncated singular value decomposition (SVD)
  \[ A = U \Sigma V^T \quad \Rightarrow \quad A \approx A_k = U_k \Sigma_k V_k^T \]
  - \( A_k \): rank-\( k \) approximation of \( A \) (optimal in \( l_2 \)-norm and F-norm)

- SVD and PCA are closely related
  - Suppose each row of matrix \( A \) is an observed data
  - PCA is realized through truncated SVD
  - The leading right singular vectors (\( v_i \)) of \( A \) are the principal components.
    Particularly, \( v_1 \) is the first principal component
Technical Background – Randomized SVD

- **Basic randQB scheme**
  - Produce near-optimal low-rank appr.
  - Accuracy can be improved with power iteration scheme
  - Well suit to parallel computing
  - Result has small random variance
  - Better than the column-pivoted QR

- **A single-pass variant**
  - Reduce to 1 visit of $A$
    \[ A \approx QQ^T A \tilde{Q} \tilde{Q}^T = QB\tilde{Q}^T \]
  - More approximation is included

### Algorithm 1 Basic randomized scheme for truncated SVD

<table>
<thead>
<tr>
<th>Require: $A \in \mathbb{R}^{m \times n}$, rank $k$, over-sampling parameter $s$.</th>
</tr>
</thead>
<tbody>
<tr>
<td>1: $l = k + s$;</td>
</tr>
<tr>
<td>2: $\Omega = \text{randn}(n, l)$;</td>
</tr>
<tr>
<td>3: $Q = \text{orth}(A\Omega)$;</td>
</tr>
<tr>
<td>4: $B = Q^TA$;</td>
</tr>
<tr>
<td>5: $[\tilde{U}, S, V] = \text{svd}(B)$;</td>
</tr>
<tr>
<td>6: $U = Q\tilde{U}$;</td>
</tr>
<tr>
<td>7: $U = U(:, 1 : k); V = V(:, 1 : k); S = S(1 : k, 1 : k)$;</td>
</tr>
<tr>
<td>8: return $U, S, V$.</td>
</tr>
</tbody>
</table>

### Algorithm 2 An existing single-pass algorithm

<table>
<thead>
<tr>
<th>Require: $A \in \mathbb{R}^{m \times n}$, rank parameter $k$.</th>
</tr>
</thead>
<tbody>
<tr>
<td>1: Generate random $n \times k$ matrix $\Omega$ and $m \times k$ matrix $\tilde{\Omega}$;</td>
</tr>
<tr>
<td>2: Compute $Y = A\Omega$ and $\tilde{Y} = A\tilde{\Omega}$ in a single pass over $A$;</td>
</tr>
<tr>
<td>3: $Q = \text{orth}(Y)$; $\tilde{Q} = \text{orth}(\tilde{Y})$;</td>
</tr>
<tr>
<td>4: Solve linear equation $\tilde{\Omega}^T QB = \tilde{Y}^T \tilde{Q}$ for $B$;</td>
</tr>
<tr>
<td>5: $[\tilde{U}, S, V] = \text{svd}(B)$;</td>
</tr>
<tr>
<td>6: $U = Q\tilde{U}$; $V = \tilde{Q}V$;</td>
</tr>
<tr>
<td>7: return $U, S, V$.</td>
</tr>
</tbody>
</table>
The Single-Pass PCA Algorithm

The blocked randQB algorithm \[^3\]

<table>
<thead>
<tr>
<th>function ([Q, B] = \text{randQB}_B(A, \varepsilon, b))</th>
</tr>
</thead>
<tbody>
<tr>
<td>(1) for (i = 1, 2, 3, \ldots)</td>
</tr>
<tr>
<td>(2) (\Omega_i = \text{randn}(n, b));</td>
</tr>
<tr>
<td>(3) (Q_i = \text{orth}(A\Omega_i));</td>
</tr>
<tr>
<td>(4) (Q_i = \text{orth}(Q_i - \sum_{j=1}^{i-1} Q_j Q_j^T Q_i));</td>
</tr>
<tr>
<td>(5) (B_i = Q_i^T A);</td>
</tr>
<tr>
<td>(6) (A = A - Q_i B_i);</td>
</tr>
<tr>
<td>(7) if (|A| &lt; \varepsilon) then stop</td>
</tr>
<tr>
<td>(8) end for</td>
</tr>
<tr>
<td>(9) (Q = [Q_1 \cdots Q_i]; B = [B_1^T \cdots B_i^T]^T).</td>
</tr>
</tbody>
</table>

- Mathematically equivalent to the basic randQB algorithm (Gram-Schmidt procedure)
- Iterative blocked procedure for monitoring approximation error while keeping high efficiency
- Mainly aimed at the problem of adaptive rank determination

Convert it to a pass-efficient procedure (multiplications with \(A\) moved out of loop)

Theorem 1: The \(Q\) and \(B\) obtained with Alg. 3 satisfy: \(Q\) is orthonormal and \(B = Q^T A\)

Algorithm 3 A pass-efficient blocked algorithm

<table>
<thead>
<tr>
<th>Require: (A \in \mathbb{R}^{m \times n}), rank parameter (k), block size (b).</th>
</tr>
</thead>
<tbody>
<tr>
<td>1: (Q = []; B = []);</td>
</tr>
<tr>
<td>2: (\Omega = \text{randn}(n, k));</td>
</tr>
<tr>
<td>3: (G = A\Omega);</td>
</tr>
<tr>
<td>4: (H = A^T G);</td>
</tr>
<tr>
<td>5: for (i = 1, 2, \ldots, k/b) do</td>
</tr>
<tr>
<td>6: (\Omega_i = \Omega(:, (i - 1)b + 1 : ib));</td>
</tr>
<tr>
<td>7: (Y_i = G(:, (i - 1)b + 1 : ib) - Q(B\Omega_i));</td>
</tr>
<tr>
<td>8: ([Q_i, R_i] = \text{qr}(Y_i));</td>
</tr>
<tr>
<td>9: (B_i = R_i^{-T}(H(:, (i - 1)b + 1 : ib)^T - \Omega_i^T B^T B));</td>
</tr>
<tr>
<td>10: (Q = [Q, Q_i]; B = [B^T, B_i^T]^T);</td>
</tr>
<tr>
<td>11: end for</td>
</tr>
</tbody>
</table>
The Single-Pass PCA Algorithm

- **Algorithm 3**
  - Equivalent to the randQB alg.
  - Steps 3 and 4 can be executed with only one pass over $A$
- Add re-orthogonalization steps to alleviate round-off error
- Memory cost is about $(m+2n)l$ floating numbers
- Time complexity (flop count) close to the basic randQB (Alg. 1)
- With the power scheme, accuracy can be largely improved at the cost of one more visit of $A$

**Algorithm 4** A single-pass algorithm for computing PCA

```
Require: $A \in \mathbb{R}^{m \times n}$, rank parameter $k$, block size $b$.
1: $Q = []; B = [];$
2: Choose $l = tb$, which is slightly larger than $k$;
3: $\Omega = \text{randn}(n, l); G = [];$ Set $H$ to an $n \times l$ zero matrix:
4: while $A$ is not completely read through do
5:    Read next few rows of $A$ into RAM, denoted by $a$;
6:    $g = a\Omega; \ G = [G; \ g];$
7:    $H = H + a^Tg;$
8: end while
9: for $i = 1, 2, \ldots, l$ do
10:   $\Omega_i = \Omega(:, (i-1)b+1 : ib);$
11:   $Y_i = G(:, (i-1)b+1 : ib) - Q(B\Omega_i);$
12:   $[Q_i, R_i] = \text{qr}(Y_i);$
13:   $[Q_i, R_i] = \text{qr}(Q_i - Q(Q_i^T Q_i));$
14:   $R_i = R_i R_i;$
15:   $B_i = R_i^T (H(:, (i-1)b+1 : ib)^T - Y_i^T Q B - \Omega_i^T B^T B);$
16:   $Q = [Q, Q_i]; B = [B^T, B_i^T]^T;$
17: end for
18: [U, S, V] = svd(B);
19: $U = QU;$
20: $U = U(:, 1 : k); V = V(:, 1 : k); S = S(1 : k, 1 : k);$
21: return $U, S, V.$
```
Experimental Results

- Five types of test matrices
  - Specified singular value spectrum with various decaying trend
  - For type 1 and 2 matrices, the singular value decays asymptotically slow

- Accuracy of computed singular value
  - A 3000x3000 matrix for each type

![Graphs showing experimental results](image-url)
Experimental Results

- **Accuracy of the principal components**
  - Test on type 1 matrix (with *slowly-decayed* $\sigma_{ii}$’s); smaller error for other matrices
  - For $v_1$, only $2.8 \times 10^{-5}$ difference in $l_\infty$-norm
  - For the first 10 principal components, the correlation coefficients are calculated: 0.9993 ~1 even for the 10th component

- **Runtime comparison (200,000x200,000 matrices)**
  - Each matrix stored as a 149 GB hard-disk file
  - Alg. 4 is 2X faster than Alg. 1; more accurate than Alg. 2

<table>
<thead>
<tr>
<th>Matrix</th>
<th>$k$</th>
<th>$t_{read}$</th>
<th>$t_{PCA}$</th>
<th>$max_{err}$</th>
<th>$t_{read}$</th>
<th>$t_{PCA}$</th>
<th>$max_{err}$</th>
<th>$t_{read}$</th>
<th>$t_{PCA}$</th>
<th>$max_{err}$</th>
</tr>
</thead>
<tbody>
<tr>
<td>Type1</td>
<td>16</td>
<td>2390</td>
<td>2607</td>
<td>1.7e-3</td>
<td>1186</td>
<td>1404</td>
<td>2.2e-2</td>
<td>1206</td>
<td>1426</td>
<td>1.8e-3</td>
</tr>
<tr>
<td>Type1</td>
<td>20</td>
<td>2420</td>
<td>2616</td>
<td>9e-4</td>
<td>1198</td>
<td>1380</td>
<td>1.6e-1</td>
<td>1217</td>
<td>1413</td>
<td>1.2e-3</td>
</tr>
<tr>
<td>Type1</td>
<td>24</td>
<td>2401</td>
<td>2593</td>
<td>1e-3</td>
<td>1216</td>
<td>1400</td>
<td>1.5e-1</td>
<td>1216</td>
<td>1414</td>
<td>1.2e-3</td>
</tr>
<tr>
<td>Type2</td>
<td>12</td>
<td>2553</td>
<td>2764</td>
<td>5e-4</td>
<td>1267</td>
<td>1477</td>
<td>3e-2</td>
<td>1276</td>
<td>1490</td>
<td>5e-4</td>
</tr>
<tr>
<td>Type3</td>
<td>24</td>
<td>2587</td>
<td>2777</td>
<td>1e-5</td>
<td>1312</td>
<td>1500</td>
<td>1.7e-3</td>
<td>1310</td>
<td>1502</td>
<td>2e-5</td>
</tr>
</tbody>
</table>

Experiment on a computer with two 12-core Xeon CPUs and 32GB memory

Test on a $10^4 \times 10^4$ matrix: Our Alg. 4 is over 300X faster than svd/svds
Experimental Results

- A test of real data
  - Face images from the FERET [4]
  - As in [2], construct a 102,042x392,216 matrix (150GB file on hard disk)
  - Compute 50 eigenfaces on the machine with 24 CPU cores
  - Runtime of our algorithm: ~ 24 minutes

• A single-pass PCA algorithm for large and high-dimensional data
• Only one pass over data matrix, providing that the matrix is stored in a row-major format
• Comparable accuracy to existing randomized algorithm; much less error than an existing single-pass algorithm
• Experiments demonstrate the algorithm’s effectiveness for large-size high-dimensional data (~150 GB disk file), in terms of runtime and memory usage

The codes of the proposed algorithm and experimental data are shared on: https://github.com/WenjianYu/rSVD-single-pass
Thank You!