Stochastic SVD on Hadoop

Shannon Quinn

(with thanks to Gunnar Martinsson and Nathan Halko of UC Boulder, and Joel Tropp of CalTech)
Lecture breakdown

• Part I
  – Stochastic SVD

• Part II
  – Distributed stochastic SVD
Part I: Stochastic SVD
Basic goal

• Matrix $A$
  – Find a low-rank approximation of $A$
  – Basic dimensionality reduction

$$\|A - QQ^*A\| < \epsilon$$
Basic algorithm

- **INPUT:** $A, k, p$
- **OUTPUT:** $Q$

1. Draw Gaussian $n \times (k + p)$ test matrix $\Omega$
2. Form product $Y = A\Omega$
3. Orthogonalize columns of $Y \Rightarrow Q$
Basic evaluation

$$\mathbb{E} \left\| A - Q Q^T A \right\|_2 \leq \left( 1 + \sqrt{\frac{k}{p-1}} \right) \sigma_{k+1} + \frac{e \sqrt{k+p}}{p} \cdot \left( \sum_{j > k} \sigma_j^2 \right)^{1/2}$$

$$\leq \left[ 1 + \frac{4 \sqrt{k+p}}{p-1} \cdot \sqrt{\min\{m, n\}} \right] \sigma_{k+1}$$

$$= C \cdot \sigma_{k+1}.$$
Approximating the SVD

1. Form $k \times n$ matrix $B = Q^T A$
2. Compute SVD of $B = \hat{U} \Sigma \hat{V}^T$
3. Compute singular vectors $U = Q \hat{U}$
Empirical Results

• 1000x1000 matrix
Power iterations

• Affects decay of eigenvalues / singular values

\[ Y = (A A^*)^q A \Omega \]
Empirical Results

Approximation error $e_k$

Estimated Eigenvalues $\lambda_j$

- "Exact" eigenvalues
- $\lambda_j$ for $q = 3$
- $\lambda_j$ for $q = 2$
- $\lambda_j$ for $q = 1$
- $\lambda_j$ for $q = 0$
Empirical Results

Approximation error $e_k$

- Minimal error (est)
- $q = 0$
- $q = 1$
- $q = 2$
- $q = 3$

Estimated Singular Values $\sigma_j$

Magnitude

$k$

$0$ $20$ $40$ $60$ $80$ $100$

$10^0$ $10^1$ $10^2$

$j$

$0$ $20$ $40$ $60$ $80$ $100$

$10^0$ $10^1$ $10^2$
Part II: Distributed SSVD
Algorithm Overview

Algorithm 4.3: Stochastic Singular Value Decomposition

Given an \( m \times n \) matrix \( A \), a target rank \( k \), an oversampling parameter \( p \), and a number of power iterations \( q \), the following algorithm computes an approximate rank \( k \) singular value decomposition \( A \approx UV^T \).

Draw an \( n \times k + p \) random matrix \( \Omega \).
Form the product \( Y = A\Omega \).
Orthogonalize the columns of \( Y \rightarrow Q \).
\[ \text{for } i = 1..q \]
\[ \quad \text{Form the product } Y = AA^TQ. \]
\[ \quad \text{Orthogonalize the columns of } Y \rightarrow Q. \]
\[ \text{end} \]
Form the projection \( B = Q^TA \).
Compute the factorization \( \tilde{U} \Sigma^2 \tilde{U}^T = BB^T \).
Solve \( \tilde{V}^T = \Sigma^{-1} \tilde{U}^TB \).
Set \( U = Q\tilde{U}(:,1:k) \).
Set \( V = \tilde{V}(:,1:k) \).
**SSVD Primitives**

- Matrix-vector multiplication: \( y = Ax \)

**Algorithm 4.5: Matrix Multiplication \( Ax \)**

This algorithm forms the product \( y = Ax \) assuming \( A \) is stored in row major format.

*Map*

- Iterate \( A_{row} \)
  
  \[ y_{row} = \langle A_{row}, x \rangle \]

  output \( y \)

- (midterm, anyone?)
SSVD Primitives

• Matrix-matrix multiplication: \( y = A^T A x \)

**Algorithm 4.5: Matrix Multiplication \( A^T A x \)**

This algorithm forms the product \( y = A^T A x \) assuming \( A \) is stored in row major format.

**Map**

Iterate \( A_{row} \)

\[
y_{partial} = \langle A_{row}, x \rangle \cdot A_{row}^T \]

output \( y_{partial} \)

**Reduce**

\[
y = \sum y_{partial} \]

output \( y \)
Matrix-matrix multiplication

• Very clever use of map/reduce

\[ Cx = \sum_{i=1}^{M} A_{i}^{T} (A_{i}x) \]
\[ = \sum_{i=1}^{M} v^{(i)} \]
\[ = y \]

• Each Mapper outputs: \( < j, v^{(i)}_{j} > \)
SSVD Primitives

- Distributed orthogonalization: \( Y = A\Omega \)
  - Givens rotation
    \[
    G(i, j, \theta) = \begin{bmatrix}
    1 & \cdots & 0 & \cdots & 0 & \cdots & 0 \\
    \vdots & \ddots & \vdots & \ddots & \vdots & \ddots & \vdots \\
    0 & \cdots & c & \cdots & -s & \cdots & 0 \\
    \vdots & \ddots & \vdots & \ddots & \vdots & \ddots & \vdots \\
    0 & \cdots & s & \cdots & c & \cdots & 0 \\
    \vdots & \ddots & \vdots & \ddots & \vdots & \ddots & \vdots \\
    0 & \cdots & 0 & \cdots & 0 & \cdots & 1
    \end{bmatrix}
    \]
  - Streaming QR
    - Sliding window
  - Merge factorizations
    1. Merge R
    2. Merge \( Q^T \)
Algorithm 4.7: Stochastic Singular Value Decomposition

The ssvd algorithm produces rank $k$ matrices $U, V, \Sigma$ that form an approximate singular value decomposition of matrix $A$.

1. Q-JOB
2. Bt-JOB
   \begin{algorithmic}
   \For {$i = 1..q$}
   3. ABt-JOB
   4. Bt-JOB
   \EndFor
   \end{algorithmic}

   Serial step:
   \begin{align*}
   &\text{compute } \tilde{U}\Sigma^2\tilde{U}^T = BB^T
   \end{align*}

4. U-JOB
5. V-JOB
Algorithm 4.7.1: Q-job

This algorithm forms the product $Y = A\Omega$, performs the streaming QR factorization and the first level merge.

Map

Iterate $A_{row}$

$Y_{row} = A_{row} \cdot \Omega$

$Y_{row} \rightarrow streamingQR$

output $Q_i^{r\times \ell}$ and $R_i^{\ell\times \ell}$.

$merge\{Q_i, R_i\}_{i=1}^{z}$

output $Q_{s\times\ell} R^{\ell\times\ell}$
2: $B^T$-job

Algorithm 4.7.2: $B^T$-job

This algorithm completes the second level merge and computes the product $B^T = A^T Q$. Optionally, partial sums of $BB^T$ are formed and output to disk.

Map

$Q_i \leftarrow$ merge $\tilde{Q}_i, \tilde{R}_1, \ldots, \tilde{R}_M$

output $Q_i$ as block of final $Q$.

Iterate $A_{row}$

$B^T_{\text{partial}} = (A_{row})^T \cdot Q_{row}$

output $B^T_{\text{partial}}$

Reduce

$B^T = \sum B^T_{\text{partial}}$

Option

$(BB^T)_{\text{partial}} = B_iB_i^T$
Algorithm 4.7.3: ABt-job

This algorithm computes the product $AB^T$, performs the streaming QR factorization and the first level merge.

**Map**

Iterate $A_{row}$

$$A_{block}(i,:) = A_{row}$$

foreach $B_{row}^T$

$$Y_{partial} = A_{block}(:,j) \cdot B_{row}^T$$

output $Y_{partial}$

**Reduce**

$$Y = \sum Y_{partial}$$

$Y \rightarrow$ streamingQR

output $Q_i^{r \times \ell}$ and $R_i^{\ell \times \ell}$

$merge\{Q_i, R_i\}_{i=1}^z$

output $Q^{s \times \ell} R^{\ell \times \ell}$
4: U-job

Algorithm 4.7.4: U-job

This algorithm computes the rank k factor $U$ of the singular value decomposition.

**Map**

**Iterate** $Q_{row}$

$U_{row} = Q_{row} \tilde{U}$

**output** $U_{row}$
Algorithm 4.7.5: V-job

This algorithm computes the rank $k$ factor $V$ of the singular value decomposition.

Map

Iterate $B_{row}^T$

$$V_{row} = B_{row}^T \tilde{U} \Sigma^{-1}$$

output $V_{row}$
## Mahout SSVD Parameters

<table>
<thead>
<tr>
<th>parameter</th>
<th>default</th>
<th>description</th>
</tr>
</thead>
<tbody>
<tr>
<td>rank (-k)</td>
<td>none</td>
<td>decomposition rank</td>
</tr>
<tr>
<td>oversampling (-p)</td>
<td>15</td>
<td>oversampling</td>
</tr>
<tr>
<td>powerIter (-q)</td>
<td>0</td>
<td>number of additional power iterations</td>
</tr>
<tr>
<td>blockHeight (-r)</td>
<td>10,000</td>
<td>Y block height (must be ( &gt; (k + p) ))</td>
</tr>
<tr>
<td>outerProdBlockHeight (-oh)</td>
<td>30,000</td>
<td>block height of outer products during multiplication, increase for sparse input</td>
</tr>
<tr>
<td>abtBlockHeight (-abth)</td>
<td>200,000</td>
<td>block height of ( Y_i ) in ( AB^T ) multiplication, increase for extremely sparse inputs</td>
</tr>
<tr>
<td>reduceTasks (-t)</td>
<td>1</td>
<td>number of reduce tasks (where applicable)</td>
</tr>
<tr>
<td>minSplitSize (-s)</td>
<td>-1</td>
<td>minimum split size</td>
</tr>
</tbody>
</table>
Block height

<table>
<thead>
<tr>
<th>blockHeight (-r)</th>
<th>1250</th>
<th>2500</th>
<th>5000</th>
<th>10000</th>
<th>20000</th>
</tr>
</thead>
<tbody>
<tr>
<td>Q-job per map</td>
<td>9</td>
<td>9</td>
<td>12</td>
<td>28</td>
<td>44</td>
</tr>
<tr>
<td></td>
<td>3.6</td>
<td>4</td>
<td>6</td>
<td>13</td>
<td>23</td>
</tr>
</tbody>
</table>
# Power iterations

<table>
<thead>
<tr>
<th>phase</th>
<th>time</th>
</tr>
</thead>
<tbody>
<tr>
<td>Q-job</td>
<td>4</td>
</tr>
<tr>
<td>Bt-job</td>
<td>16</td>
</tr>
<tr>
<td>ABt-job</td>
<td>12</td>
</tr>
<tr>
<td>U-job</td>
<td>2</td>
</tr>
<tr>
<td>V-job</td>
<td>2</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>q</th>
<th>time</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>26</td>
</tr>
<tr>
<td>1</td>
<td>52</td>
</tr>
<tr>
<td>2</td>
<td>80</td>
</tr>
<tr>
<td>3</td>
<td>107</td>
</tr>
</tbody>
</table>
Comparison to Lanczos

Algorithm 4.8.2: Lanczos SVD

Given an \( m \times n \) matrix \( A \) and a desired rank \( k \), Lanczos SVD computes the \( k \) dimensional eigen-decomposition, \( V \Sigma^2 V^T \), of \( A^T A \) which yields the right singular vectors \( V \) and singular values \( \Sigma \) of \( A \).

\[
q = A^T A \omega \\
q_1 = q / \|q\| \\
\text{while } i \leq k \text{ do} \\
\hspace{1em} \text{MapReduce step: Algorithm 4.5} \\
\hspace{2em} q = A^T A q_i \\
\hspace{1em} \text{Serial step:} \\
\hspace{2em} \text{for } j = 1..i \\
\hspace{3em} q \leftarrow q - (q_j, q) q_j \\
\hspace{3em} q_{i+1} = q / \|q\| \\
\hspace{2em} \text{collect: } \alpha, \beta \\
\hspace{1em} \text{end while} \\
\hspace{1em} \text{compute } X \Lambda X^T = T \\
\hspace{1em} V = Q X \\
\hspace{1em} \sigma_i = \sqrt{\lambda_i}
\]
Comparison to Lanczos
Comparison to Lanczos
Comparison to Lanczos
Datasets

- Wikipedia-all

<table>
<thead>
<tr>
<th>Execution times</th>
<th>minutes</th>
</tr>
</thead>
<tbody>
<tr>
<td>Q-job</td>
<td>4</td>
</tr>
<tr>
<td>Bt-job</td>
<td>49</td>
</tr>
<tr>
<td>ABt-job</td>
<td>127</td>
</tr>
<tr>
<td>U-job</td>
<td>1.5</td>
</tr>
<tr>
<td>V-job</td>
<td>7</td>
</tr>
<tr>
<td>$q = 0$</td>
<td>61</td>
</tr>
<tr>
<td>$q = 1$</td>
<td>235</td>
</tr>
</tbody>
</table>

- Wikipedia-MAX

<table>
<thead>
<tr>
<th>phase</th>
<th>time</th>
</tr>
</thead>
<tbody>
<tr>
<td>Q-job</td>
<td>20</td>
</tr>
<tr>
<td>Bt-Job</td>
<td>365</td>
</tr>
<tr>
<td>ABt-job</td>
<td>571</td>
</tr>
<tr>
<td>U-job</td>
<td>8</td>
</tr>
<tr>
<td>V-job</td>
<td>14</td>
</tr>
<tr>
<td>total</td>
<td>1335</td>
</tr>
</tbody>
</table>
That’s SSVD!
Resources

• Randomized methods for computing the SVD of very large matrices

• Randomized methods for computing low-rank approximations of matrices
  – https://amath.colorado.edu/faculty/martinss/Pubs/2012_halko_dissertation.pdf

• SSVD on Mahout