SVD-RD: AN EFFICIENT STREAMING ALGORITHM FOR DOWNDATING THE SVD[*]

SAURAV JHA[†], NEVIL ANTO[†], ARPITA SINGH[†], GIRI PRABHAKAR[‡], RAHUL MARATHE[†], SHANKAR BALACHANDRAN[†§]

Abstract. Singular value decomposition(SVD) plays a prominent role in many modern data prediction techniques. In real time applications like video/image/document retrieval, data may arrive at a high rate and typically obsolete rows and columns of the data are no longer retained. SVD downdate is the process of incrementally decomposing a matrix when a small number of rows or columns are removed. We propose an algorithm called SVD-RD for the row downdate problem for $m \times n$ matrices where q rows are removed and SVD of the resultant matrix is desired. The algorithm requires only one pass over the data and uses asymptotically lesser space when both m and n are large, thus making it a streaming algorithm. The algorithm is error free for full rank matrices with $m < n$. For the general case, we take a user specified k-rank approximation and show that we can approximate the SVD in $O(m + n)$ time. We derive upper bounds for the error due to this approximation. We also present experimental results and show how the algorithm scales with increase in m, n, q and k. SVD-RD is several orders of magnitude faster than recomputing SVD from scratch and much faster than an existing incremental method.

Key words. SVD downdate, matrix decomposition, singular vectors, streaming algorithms, approximation

AMS subject classifications. 15A18, 65K05, 65Y20

1. Introduction.

Singular Value Decomposition (SVD) has received a lot of interest in the research community across several decades. Stewart pointed out that the stability and the low rank approximation provided by SVD makes it an ideal vehicle for many practical applications [20].

Singular Value Decomposition of a real matrix A of dimensions $m \times n$ is given by

$$A = USV^T,$$

where U and V are two unitary matrices of size $m \times m$ and $n \times n$, respectively. S is a matrix of size $m \times n$ whose diagonal entries contain the singular values of A arranged in the non-increasing order of magnitude. The columns of U and V are known as left- and right-singular vectors of A and they form an orthonormal basis for the column and row space of A respectively. Matrices U and V, being unitary, satisfy $UU^T = U^TU = I$ and $VV^T = V^TV = I$, where I is the identity matrix.

A powerful use of singular value decomposition comes is that it can be used to find a low-rank approximation for a given matrix. Low-rank approximation is achieved by retaining only the k largest singular values and their corresponding singular vectors.

$$A \simeq \mathcal{A} = \mathcal{U}\mathcal{S}\mathcal{V}^T. \tag{1.1}$$

Here, $\mathcal{S}$ is a $k \times k$ diagonal matrix which contains only the k largest singular values along the diagonal. $\mathcal{U}$ is an $m \times k$ matrix and $\mathcal{V}$ is an $n \times k$ matrix. $\mathcal{U}$ and $\mathcal{V}$ are now column-wise orthonormal matrices and not unitary matrices i.e., $\mathcal{U}^T\mathcal{U} = I$,

[†]INDIAN INSTITUTE OF TECHNOLOGY MADRAS, INDIA
[‡]SIEMENS TECHNOLOGY AND SERVICES PRIVATE LIMITED, INDIA.
[*]This work is supported by the IBM Shared University Research Grant, 2011-2014.
[§]Corresponding Author. Email shankar@cse.iitm.ac.in.

$\mathcal{V}^T\mathcal{V} = I$ but $\mathcal{U}\mathcal{U}^T \neq I$, $\mathcal{V}\mathcal{V}^T \neq I$. They contain the corresponding left- and right-singular vectors. Typically, k is chosen such that $0 < k < rank(A) \leq \min(m, n)$. Since the effective rank can potentially be reduced, low rank approximation is also called the *Reduced-SVD* of a matrix and $\mathcal{A}$ is called the reduced model of A. In the paper, we have referred to the decomposition $\mathcal{A} = \mathcal{U}\mathcal{S}\mathcal{V}^T$ as k-rank SVD of A when $k < \min(m, n)$. If k is chosen such that $k = rank(A) = \min(m, n)$, we call it the full-rank SVD.

For reduced-SVD, $\mathcal{A}$ can be shown to be a k-rank matrix (for $0 < k < rank(A) \leq \min(m, n)$) that best approximates A in the least square sense. Further, storing $\mathcal{U}$, $\mathcal{S}$ and $\mathcal{V}$ takes $O((m+n)k)$ space as opposed to $O(mn)$ space for A. For these two reasons, it is customary in the SVD community to use $\mathcal{A}$ instead of A, even though it is an approximation to A.

Singular value decomposition is computationally expensive. When real-time applications demand frequent recomputation of SVD, either due to arrival of new information or removal of any obsolete data, it is essential to develop simple methods to keep the decomposition current. Incremental methods to compute SVD of a matrix on removal (addition) of rows is called the SVD downdate (update) problem.

In general, let $\mathcal{A}$ be of the form

$$\mathcal{A} = \begin{bmatrix} \tilde{A} \\ L \end{bmatrix},$$

where $\tilde{A}$ is the top $m - q$ rows of $\mathcal{A}$ and L is the bottom q rows of $\mathcal{A}$. The downdate problem is defined as finding the SVD of $\tilde{A}$, given $\mathcal{U}, \mathcal{S}$ and $\mathcal{V}$. To be specific, this is the row-downdate problem. Similarly, the column-downdate problem is the problem of incrementally recomputing the SVD if columns from $\mathcal{A}$ are removed.

Many modern day applications frequently need to update or downdate the SVD. Latent Semantic Indexing [12] is one such application. It is widely used in diverse areas such as text summarization [8], image annotation with keywords [15], handling noisy data [18], hidden relationship discovery in databases [3], information retrieval in search engines and CDNA microarray data analysis [6]. It is easy to see that such systems can undergo a lot of updates/downdates over time.

SVD downdate is necessary for systems where recency of information has to be maintained when rows are removed. For instance, imagine a search engine that is unable to index a URL any more. The document retrieval mechanism should remove it from the "concept space" without incurring huge runtimes. Another use of downdate is when data streams through a system and the new data makes the old data obsolete. Such a system requires both a downdate (of stale data) and an update (of the new data) of the underlying model. An example of such a system is video analysis where old frames have to be downdated and new frames have to be updated.

The matrices are typically thin-and-tall in information retrieval problems. In video analysis applications like background subtraction, the matrices are usually short-and-fat. In both these cases, they typically allow a good low-rank approximation.

2. Background. Computing SVD incrementally has been an active area of research and it all started with the SVD update method by Berry [2]. Research work that followed Berry's method have presented algorithms that were more efficient either in terms of time or space. Specifically, applications like video and document processing required that the incoming data is not remembered forever because of space constraints. Thus, *streaming* algorithms came into existence: any incoming data is inspected only once (preferably) and the model is updated as and when new data arrives. Soon, the research community started focusing on applications where a recent "window" of data should be analyzed. This required incremental methods for not just updating the SVD but also for downdating it.

The downdate problem in the closed form was considered impossible by the research community for a while [10]. However, a series of algorithms have since been developed for SVD downdate and have been deployed in different contexts, including LSI [22] and motion detection [7]. Brand discusses the problem in a more general setup [4].

Our work is motivated because of the increase in the use of SVD in various real time applications that handle large volumes of streaming data. In the next subsection, we briefly discuss Brand's algorithm [4] and later in section 4, we compare the performance of our algorithm with it.

2.1. Related work. Gu and Eisenstat [9] were one of the earliest to consider the downdate problem. Given the SVD of A, they solved the single-row downdate problem by re-orthogonalizing the left sub-space using SVD of another intermediate matrix that is derived from the deleted row. The complexity of single row downdate is $O((m+n)n^2)$ and the algorithm does not suppport q-row update directly. Park and Huffel [17] proposed a single row downdate algorithm which was inspired by incremental algorithms for downdating the QR decomposition. They produce a bidiagonal reduction of the downdated matrix followed by a diagonalization step. They calculate only the singular values and the right singular vectors deeming calculation of left vectors computationally expensive. Witter and Berry [23, 22] proposed a downdate method for the LSI problem by repeatedly applying Givens rotations to an intermediate matrix that uses the left singular vectors and the singular values. The non-zero entries in the matrix are "chased out" to a bi-diagonal matrix which is further decomposed to get the final SVD. The complexity is $O((m+n+k)k^2)$ and can support downdate of only one row at a time. There are several works that model the downdate problem as an update problem with a forgetting factor [5, 13, 19, 16]. The downdated entries are not removed truly but progressively "forgotten" using the forgetting factor. It has been shown that downdating is better than forgetting in non-stationary environments [11]. Also, downdating requires a fixed size of entries to be stored. More recently, Zhang et al. [24] proposed a downdate algorithm in which they observed that the intermediate matrices in [9] are Cauchy-like matrices and used an approximation for these matrices. The approximation allowed them to do efficient multiplication and the algorithm takes $O(mnr)$ steps for some moderate constant r.

2.1.1. Brand's method for SVD downdate. We now briefly discuss Brand's method [4] which can perform both update and downdate of a SVD. Let A be an $m \times n$ real matrix and the k-rank SVD of A be $\mathcal{A} = \mathcal{U}\mathcal{S}\mathcal{V}^T$. Brand proposed a model for incrementally computing the SVD of $\mathcal{A} + XY^T$, where X and Y are arbitrary matrices of rank c and have the dimensions $m \times c$ and $n \times c$ respectively. The algorithm can further be specialized from the general identity to perform SVD downdate. If $\mathcal{A}$ is of the form

$$\mathcal{A} = \begin{bmatrix} \tilde{A} \\ \mathbf{a}_m \end{bmatrix},$$

where $\mathbf{a_m}$ is the last row of $\mathcal{A}$, Brand's method performs the downdate of $\mathbf{a_m}$ from $\mathcal{A}$ by performing the incremental SVD on $\mathcal{A} + XY^T$, where

$$X = \begin{bmatrix} 0 \\ . \\ . \\ 0 \\ -1 \end{bmatrix},$$
$$Y = \mathbf{a}_m{}^T.$$

Here, X and Y are two vectors of size $m \times 1$ and $n \times 1$ respectively. Two matrices P and Q are computed such that

$$\begin{aligned} \mathcal{A} + [\mathbf{0}, \cdots, \mathbf{0}, \mathbf{-1}]^T \times \mathbf{a}_m &= \mathcal{U}\mathcal{S}\mathcal{V}^T + [\mathbf{0}, \cdots, \mathbf{0}, \mathbf{-1}]^T \times \mathbf{a}_m \\ &= [\mathcal{U}\ P]K[\mathcal{V}\ Q]^T, \end{aligned} \tag{2.1}$$

where $[U\ P]$ and $[V\ Q]$ are column-wise orthonormal. K is a small and highly structured matrix. By performing SVD on K to get U', S' and V', the equation above can be rewritten as

$$\mathcal{A} + [\mathbf{0}, \cdots, \mathbf{0}, \mathbf{-1}]^T \times \mathbf{a}_m = ([\mathcal{U}\ P]U')S'([\mathcal{V}\ Q]V')^T$$

which is the SVD of $\tilde{A}$. It is worthwhile to note that the algorithm needs to remember the row ($\mathbf{a}_m$) which has to be downdated which implies that the original matrix entries have to be stored.

By deferring the (left and right) subspace updation after every downdate, Brand's method keeps the downdate cost to $O((m+n)q^2 + k(m+n)q + (k+q)^3)$ for a q-row downdate. For single row downdate ($q = 1$), cost of downdate is $O((m+n)k + k^3)$. If subspaces also need to be updated, the complexity of algorithm is $O((m+n+k+q)(k+q)^2)$. For small values of k and q, the complexity of Brand's method is $O(m+n)$. The method is proposed for $k \leq \sqrt{\min(m,n)}$ and is general enough to handle a few other forms of additive modification.

2.2. Our contributions. We propose a novel streaming algorithm called SVD-RD (SVD-**R**ow **D**owndate) for the SVD row downdate problem. Given a $k-$rank approximation of a $m \times n$ matrix, we present a template with which q rows can be downdated. Our technique replaces the expensive operation of SVD on a $(m-q) \times n$ with the SVD on a smaller $k \times k$ matrix. For the case where $m < n$ and $k = m$, the technique has no approximation errors. For the typical case where $k < \min(m,n)$, the technique is not error-free.

The basic idea behind SVD-RD is to compute SVD of the system $\mathcal{A}$ but with zeroes replacing all the entries of the downdated rows. However, this leads to remembering the entries of $\mathcal{A}$. We show by appropriate approximations, how SVD-RD is "streaming" and how it allows us to forget the entries of $\mathcal{A}$. Our algorithm needs only one pass over the original entries and uses asymptotically less space than recomputing the SVD from scratch.

The rest of the paper is organized as follows:

1. In section 3.2, we present our mathematical technique, called SVD-RD, to downdate the SVD of an $m \times n$ matrix with rank k. We discuss two cases, one specifically for full-rank matrices that have $m < n$ and another for the general case.

2. In section 3.4, we analyze the runtime and space requirements of SVD-RD.
3. In section 4, we explain the implementation details and present experimental results. We also show that the algorithm, though asymptotically similar to computing the SVD from scratch[1] and to Brand's method, is quite efficient in practice when m
lln. If $m = O(n)$ or $m > n$, a good $k-$rank approximation can be used in SVD-RD to retain the runtime benefits at the loss of accuracy.
4. The reduced-rank version is not approximation-free. In section 4.3, we derive upper bounds on the approximation error.

3. SVD-RD.

In this section we first show a template of SVD-RD, a new "streaming" algorithm for performing row downdate. We then use the template and present two cases: i) short-and-fat matrices of full row rank (i.e. $m < n$ and $k = m$) and ii) all other combinations of matrix sizes and rank used for approximation. For the first case, we show that the algorithm is approximation-free. SVD-RD for the second case is capable of providing good runtime benefits.

3.1. Notations.

The notations used in this section are listed in Table 1. The table also gives the dimensions of each matrix and a brief description of the matrix. The last two columns contain the SVD of the corresponding matrix, and the dimensions of the left-subspace, the singular value matrix and the right-subspace for each one of them. We use MATLAB/OCTAVE style notation for matrix indices wherever appropriate.

Table 1: Matrices used in section 3 and their notations.

Matrix	Dimensions	Description	SVD	Dimensions of the matrices
A	$m \times n$	Original matrix		
$\mathcal{A}$	$m \times n$	k-rank approximation to A	$\mathcal{U}\mathcal{S}\mathcal{V}^T$	$m \times k$, $k \times k$ and $n \times k$
$\tilde{A}$	$(m-q) \times n$	Top $(m-q)$ rows of $\mathcal{A}$	$\tilde{U}\tilde{S}\tilde{V}^T$	$(m-q) \times k$, $k \times k$ and $n \times k$
L	$q \times n$	Last q rows of $\mathcal{A}$	-	-
B	$m \times m$	An intermediate matrix	XYZ^T	All are of size $k \times k$

3.2. Row downdate.

Let A be a real matrix of size $m \times n$ and $\mathcal{A} = \mathcal{U}\mathcal{S}\mathcal{V}^{\mathcal{T}}$ be the k-rank singular value decomposition of A with $0 < k \leq \min(m, n)$. Here, $\mathcal{U}$ and $\mathcal{V}$ are column-wise orthonormal matrices of size $m \times k$ and $n \times k$ respectively. Let $\mathcal{A}$ be of the form

$$\mathcal{A} = \begin{bmatrix} \tilde{A} \\ L \end{bmatrix}, \tag{3.1}$$

where L is the matrix containing last q rows of $\mathcal{A}$. We want to downdate the last q rows of $\mathcal{A}$. The row-downdate problem is to find three matrices $\tilde{U}, \tilde{S}$ and $\tilde{V}$ (of size $(m-q) \times k$, $k \times k$ and $n \times k$ respectively), that form the SVD of $\tilde{A}$. Mathematically, we want to find $\tilde{U}$, $\tilde{S}$ and $\tilde{V}$ such that

$$\tilde{A} = \tilde{U}\tilde{S}\tilde{V}^T. \tag{3.2}$$

Let $\mathcal{U}$ also be partitioned to be of the form

$$\mathcal{U} = \begin{bmatrix} \mathcal{U}_1 \\ D \end{bmatrix}, \tag{3.3}$$

[1]We call computing SVD from scratch "SVD recomputation" or simply "recomputation" hereafter.

where D is the last $(m-q)$ rows of $\mathcal{U}$. Observing that $\mathcal{A}\mathcal{V} = \mathcal{U}\mathcal{S}$, we have $L\mathcal{V} = D\mathcal{S}$. Further, by the definition of SVD of $\mathcal{A}$, $\mathcal{S} = \mathcal{U}^T\mathcal{A}\mathcal{V}$. We now rewrite $\mathcal{S}$ in terms of D.

$$\begin{aligned}
\mathcal{S} &= \mathcal{U}^T\mathcal{A}\mathcal{V} \\
&= \mathcal{U}^T \begin{bmatrix} \tilde{A} \\ L \end{bmatrix} \mathcal{V} \\
&= \mathcal{U}^T \begin{bmatrix} \tilde{A} \\ \mathbf{0}_{q\times n} \end{bmatrix} \mathcal{V} + \mathcal{U}^T \begin{bmatrix} \mathbf{0}_{(m-q)\times n} \\ L \end{bmatrix} \mathcal{V} \\
&= \mathcal{U}^T \begin{bmatrix} \tilde{A} \\ \mathbf{0} \end{bmatrix} \mathcal{V} + [\mathcal{U}_1^T \; D^T] \begin{bmatrix} \mathbf{0} \\ L \end{bmatrix} \mathcal{V} \\
&= \mathcal{U}^T \begin{bmatrix} \tilde{A} \\ \mathbf{0} \end{bmatrix} \mathcal{V} + D^T L\mathcal{V} \\
&= \mathcal{U}^T \begin{bmatrix} \tilde{A} \\ \mathbf{0} \end{bmatrix} \mathcal{V} + D^T D\mathcal{S}
\end{aligned}$$

We define B, a matrix of size $k \times k$, as

$$\begin{aligned}
B &\doteq \mathcal{U}^T \begin{bmatrix} \tilde{A} \\ \mathbf{0}_{q\times n} \end{bmatrix} \mathcal{V} && (3.4) \\
&= S - D^T D\mathcal{S}. && (3.5)
\end{aligned}$$

Multiplying (3.4) by $\mathcal{U}$ from the left and by $\mathcal{V}^T$ from the right, and by swapping the sides of the equality, we get

$$\mathcal{U}\mathcal{U}^T \begin{bmatrix} \tilde{A} \\ \mathbf{0} \end{bmatrix} \mathcal{V}\mathcal{V}^T = \mathcal{U}B\mathcal{V}^T. \tag{3.6}$$

Since $\mathcal{V}$ forms an orthonormal basis for the range of $\mathcal{A}^T$, it forms the orthonormal basis for $\tilde{A}^T$ as well. Now, $\tilde{A}\mathcal{V}\mathcal{V}^T = (\mathcal{V}\mathcal{V}^T\tilde{A}^T)^T$ is in fact the transpose of the projection of $\tilde{A}^T$ in to the space spanned by $\mathcal{V}$. Note that the space spanned by $\tilde{A}^T$ is the subspace of the space spanned by $\mathcal{V}$, which is the same as the space spanned by $\mathcal{A}^T$. Therefore, we will always have $\tilde{A}\mathcal{V}\mathcal{V}^T = \tilde{A}$. Thus,

$$\mathcal{U}\mathcal{U}^T \begin{bmatrix} \tilde{A} \\ \mathbf{0} \end{bmatrix} = \mathcal{U}B\mathcal{V}^T. \tag{3.7}$$

It is worth to note that $\mathcal{U}\mathcal{U}^T\tilde{A}$ is the projection of $\tilde{A}$ into the subspace of $\mathcal{U}$. If we compute the SVD of B such that

$$B = XYZ^T,$$

where X, Y and Z are all $m \times m$ matrices, then (3.7) can be rewritten as

$$\begin{aligned}
\mathcal{U}\mathcal{U}^T \begin{bmatrix} \tilde{A} \\ \mathbf{0} \end{bmatrix} &= \mathcal{U}XYZ^T\mathcal{V}^T && (3.8) \\
&= (\mathcal{U}X)Y(\mathcal{V}Z)^T. && (3.9)
\end{aligned}$$

3.2.1. Case 1: $m < n$ and $k = m$. This is the case where A is a short-and-fat matrix and $\mathcal{A}$ is the full-rank approximation to A. In such a scenario, $\mathcal{U}$ would be an $m \times m$ unitary matrix and $\mathcal{U}\mathcal{U}^T = I$. Now, (3.9) can be rewritten as

$$\begin{bmatrix} \tilde{A} \\ \mathbf{0} \end{bmatrix} = (\mathcal{U}X)Y(\mathcal{V}Z)^T. \tag{3.10}$$

Using (3.2), $\begin{bmatrix} \tilde{A} \\ \mathbf{0} \end{bmatrix}$ can also be written as

$$\begin{bmatrix} \tilde{A} \\ \mathbf{0} \end{bmatrix} = \begin{bmatrix} \tilde{U} & \mathbf{0} \\ \mathbf{0} & P \end{bmatrix} \begin{bmatrix} \tilde{S} & \mathbf{0} \\ \mathbf{0} & \mathbf{0} \end{bmatrix} \begin{bmatrix} \tilde{V} & Q \end{bmatrix}^T, \tag{3.11}$$

where P and Q are some $q \times q$ and $n \times q$ matrices respectively. By comparing the RHS of (3.10) and (3.11), we get

$$\begin{aligned} \begin{bmatrix} \tilde{U} & \mathbf{0} \\ \mathbf{0} & P \end{bmatrix} &= (\mathcal{U}X), \\ \begin{bmatrix} \tilde{S} & \mathbf{0} \\ \mathbf{0} & \mathbf{0} \end{bmatrix} &= Y, \\ \begin{bmatrix} \tilde{V} & Q \end{bmatrix}^T &= (\mathcal{V}Z)^T. \end{aligned}$$

Since $\mathcal{U}$ and X both are unitary matrices, the product will also be a unitary matrix. In fact, the product here turns out to be a block diagonal matrix. Since B is a matrix of rank $(m - q)$, Y will have $(m - q)$ non-zero singular values along the diagonal, which we retain in $\tilde{S}$. $\mathcal{V}$ is a column-wise orthonormal matrix and Z is a unitary matrix and hence the product $\mathcal{V}Z$ will be a column-wise orthonormal matrix. $\tilde{V}$ retains only the leftmost $(m - q)$ columns of the product.

Finally, we can extract the SVD of $\tilde{A}$ as follows:

$$\begin{aligned} \tilde{U} &= \mathcal{U}(1 : m - q, :) * X(:, 1 : m - q) \\ \tilde{S} &= Y(1 : m - q, 1 : m - q) \\ \tilde{V} &= \mathcal{V} * Z(:, 1 : m - q). \end{aligned} \tag{3.12}$$

It is clear that the rank of $\tilde{A}$ is $m - q$ is reflected in the extraction above.

3.2.2. Case 2. In general, matrices could be neither short-and-fat nor need full-rank approximation. In fact, it is quite common in practice to see matrices that are square or tall. Also, the benefits in storing the SVD happens only if $k < \min(m, n)$. We discuss SVD-RD, without loss of generality, for such a case now where k is chosen sufficiently lesser than m and n such that $k-$ rank approximation is “good enough” for the application. The correctness of proposed algorithm does not depend on the choice of k.

Let the matrix $\mathcal{A}$ be a low-rank approximation to A. Then, $\mathcal{U}$ and $\mathcal{V}$ would contain only the leftmost k singular vectors and $\mathcal{S}$ would contain only the topmost k singular values.

$\mathcal{U}\mathcal{U}^T\tilde{A}$ is in fact the projection of $\tilde{A}$ in to the $\mathcal{U}$ subspace and $\mathcal{U}\mathcal{U}^T \neq I$. Approximating the projection of $\tilde{A}$ as $\tilde{A}$ in (3.9), we have

$$\begin{bmatrix} \tilde{A} \\ \mathbf{0} \end{bmatrix} \approx (\mathcal{U}X)Y(\mathcal{V}Z)^T. \tag{3.13}$$

We can also write $\begin{bmatrix} \tilde{A} \\ \mathbf{0} \end{bmatrix}$ as

$$\begin{bmatrix} \tilde{A} \\ \mathbf{0} \end{bmatrix} = \begin{bmatrix} \tilde{U} \\ \mathbf{0} \end{bmatrix} \tilde{S}\tilde{V}^T. \tag{3.14}$$

Comparing the RHS of (3.14) and (3.13), we have

$$\begin{aligned} \tilde{S} &\approx Y, \\ \tilde{V}^T &\approx (\mathcal{V}Z)^T, \\ \begin{bmatrix} \tilde{U} \\ \mathbf{0} \end{bmatrix} &\approx (\mathcal{U}X). \end{aligned}$$

This gives us a way to approximate $\tilde{U}$, $\tilde{S}$ and $\tilde{V}$(the matrices that form the decomposition of $\tilde{A}$), using the decomposition of $\mathcal{A}$.

3.3. Algorithm SVD-RD to perform row downdate. The outline of the SVD downdate algorithm is presented in Algorithm 1. The procedure SVD-RD takes three sets of parameters: the SVD of $\mathcal{A}$, k - the rank approximation that is desired, and q - the number of rows to be downdated. It returns the SVD of the downdated matrix. `svd( )` is a call to a function which computes SVD.

Algorithm 1: SVD-RD Row Downdate

```
 1: procedure SVD-RD FOR ROW DOWNDATE
 2:     Input: $\mathcal{U}$, $\mathcal{S}$, $\mathcal{V}$, $k$ and $q$
            ▷ $k$ : Rank approximation given
            ▷ $q$ : Number of rows to be downdated
            ▷ $\mathcal{S}$ : diagonal matrix containing top $k$ singular values of $\mathcal{A}$
            ▷ $\mathcal{U}$, $\mathcal{V}$ : corresponding left and right singular vectors of $\mathcal{A}$
 3:     $D = \mathcal{U}(m-q+1 : m, 1 : k)$
 4:     $C = D * \mathcal{S}$
 5:     $B = \mathcal{S} - D^T * C$
 6:     $[X, Y, Z] = \texttt{svd}(B)$
 7:     $\tilde{U} = \mathcal{U}(1 : m-q, 1 : k) * X(:, 1 : k)$
 8:     $\tilde{S} = Y(1 : k, 1 : k)$
 9:     $\tilde{V} = \mathcal{V} * Z(:, 1 : k)$
10:     return $\tilde{U}$, $\tilde{S}$, $\tilde{V}$
11: end procedure
```

$\tilde{U}, \tilde{S}$ and $\tilde{V}$ are of sizes $(m-q) \times k$, $k \times k$ and $n \times k$, respectively. These matrices can be passed on as $\mathcal{U}, \mathcal{S}$ and $\mathcal{V}$ for the next call of SVD-RD downdate.

If we have a short-and-fat matrix and if we desire full rank approximation, Algorithm 1 should be passed the value m for the parameter k. In the general case, the desired rank of approximation should be passed as the value for k.

3.3.1. Refinement for Case 1. For case 1, the steps outlined in Algorithm 1 can be refined slightly. Since q rows are downdated on a full-rank short-and-fat matrix of size $m \times n$, the resultant matrix will also be a full-rank (with rank $m-q$) short-and-fat matrix (of size $(m-q)\times n$). One can notice that $\tilde{S}$ would have $m-q$ singular values along the diagonal and the other entries would be 0. The last q columns of $\tilde{U}$ and $\tilde{V}$ will not have any contribution. One possibility is to perform the computation as suggested in Algorithm 1 and throw away the entries. A small refinement could be to compute only the necessary singular vectors to beign with. (3.12) extracts only the necessary singular vectors. Steps 7-9 of Algorithm 1 can be replaced with the calculations shown in (3.12) easily without affecting the correctness of the method. This refinement would result in SVD to have sizes $(m-q)\times(m-q)$, $(m-q)\times(m-q)$ and $n \times (m-q)$ respectively. We will assume from here on that this refinement is performed for Case 1.

3.3.2. Discussion on Case 2. Case 2 is the general case and relaxes the constraints that Case 1 poses on the relative sizes of m and n as well as on the rank of approximation used. This comes at the cost of the following two disadvantages:

1. The algorithm does not compute the exact decomposition of $\tilde{A}$ because of the projection of $\tilde{A}$ into the $\mathcal{U}$ subspace. It only approximates the SVD of $\tilde{A}$.
2. The $\tilde{U}$ returned by the algorithm will not be column-wise orthonormal because $\tilde{U}$ is only the top $(m-q)$ rows of $\mathcal{U}X$ and the bottom q rows of $\mathcal{U}X$ need not be 0s. Therefore, we do not have $\tilde{U}^T\tilde{U} = I$ anymore.

3.3.3. Streaming nature of SVD-RD. Computation of B using 3.4 requires remembering the original entries of $\mathcal{A}$. If we use Equation 3.5, calculation of B requires $\mathcal{U}$ and $\mathcal{S}$ only. Also, the steps that follow the computation of B require only the SVD of $\mathcal{A}$ (namely $\mathcal{U}, \mathcal{S}$ and $\mathcal{V}$) to be remembered and entries of $\mathcal{A}$ can be "forgotten". Because of this reason, we call SVD-RD "streaming". The entries of $\mathcal{A}$ can stream through our algorithm without having to store those entires indefinitely.

3.4. Complexity of SVD-RD. To analyze Algorithm 1, we need the sizes of the different matrices. The sizes of the matrices involved are tabulated in Table 2.

Table 2: Matrices used in Algorithm 1 and their dimensions.

Matrix	Dimensions	
	Case 1	Case 2
$\mathcal{U}, \mathcal{S}, \mathcal{V}$	$m\times m, m\times m, m\times n$	$m\times k, k\times k, k\times n$
D	$q\times m$	$q\times k$
C	$q\times m$	$q\times k$
B	$m\times m$	$k\times k$
$\tilde{U}, \tilde{S}, \tilde{V}$	$(m-q)\times(m-q), (m-q)\times(m-q), (m-q)\times n$	$(m-q)\times k, k\times k, k\times n$

3.4.1. Runtime analysis for Case 1. The proposed algorithm involves basic matrix operations and requires one SVD computation of an $m \times m$ matrix. The computations of C and B in lines 4 and 5 take $O(qm)$ and $O(qm^2)$ steps respectively. Computing SVD of the $m \times m$ matrix B in line 6 takes $O(m^3)$ steps. With the refinement suggested earlier, steps in Lines 7 and 9 take $O((m-q)^2m)$ and $O(nm(m-q))$ time, respectively. The overall time complexity of the proposed algorithm is $O(m(m+q)^2+mn(m-q))$. In terms of space, SVD-RD would require $O((m+n+q)m)$ matrix entries to be stored.

In practice, a small number of rows is downdated at a time and hence $q = O(1)$. Therefore, the complexity of SVD-RD is $O((m+n)m^2)$. Table 3 compares the runtime complexity of the proposed algorithm with Brand's algorithm and with recomputation.

Table 3: Comparison of time complexity of a q-row downdate operation for Case 1.

Downdate Size	Recomputation	Brand's Method	SVD-RD
q-row downdate	$O((m-q)^2 n)$	$O((m+n+q)(m+q)^2)$	$O(m(m+q)^2 + mn(m-q))$
$q = O(1)$	$O(m^2 n)$	$O((m+n)m^2)$	$O((m+n)m^2)$

In general, for q-row downdate, SVD-RD is asymptotically better than Brand's method. However, when $q = O(1)$, the analysis seems to indicate that both methods have the same rate of growth. Later in section 4.1 we show that even though both the methods have the same complexity, our method is much faster than Brand's method for both the cases.

Another important point that deserves a little more than a mere mention is that of the complexity of recomputation. Table 3 suggests that recomputation is actually better than both the incremental methods! In practice, this is not the case. We will see later in section 4.1 that this theoretical result could be far away from the practical results - a testament to the aphorism, "There's many a slip between the cup and the lip." Recomputation is indeed orders of magnitude slower than both the incremental methods when $m \ll n$ and is marginally slower if $m < n$ but m and n are comparable.

3.4.2. Runtime analysis for Case 2. When downdate is tried on a low rank approximation, we expect $k < \min(m, n)$. The first impact of using low rank approximation is on the size of $\mathcal{U}, \mathcal{S}$ and $\mathcal{V}$. This has a cascading effect on the runtime of the algorithm. The computations of C and B in line 4 and 5 will take $O(qk)$ and $O(qk^2)$ steps respectively. Computing SVD of the $k \times k$ matrix B in line 6 will take $O(k^3)$ time. Matrix multiplications in line 7 and 9 will take $O((m-q)k^2)$ and $O(nk^2)$, respectively. The overall time complexity is $O((m+n+k+q)k^2)$. In terms of space, SVD-RD would require $O((m+n+k+q)k)$ matrix entries to be stored. For $k = O(1)$ and $q = O(1)$, both time and space complexity are $O(m+n)$. Table 4 compares the runtime of SVD-RD, Brand's method and recomputation for downdating q rows of a matrix given its k-rank SVD.

Table 4: Comparison of time complexity of a q-row downdate operation for Case 2.

Downdate Size	Recomputation	Brand's Method	SVD-RD
q-row downdate	$O((m-q)^2 n)$	$O((m+n+q+k)(k+q)^2)$	$O((m+n+q+k)k^2)$
$q = O(1)$	$O(m^2 n)$	$O((m+n+k)k^2)$	$O((m+n+k)k^2)$

3.5. Interesting properties of SVD-RD. Some of the interesting properties of SVD-RD are listed below:

- The algorithm is approximation-free for Case 1.
- Computing SVD of the $(m-q) \times n$ matrix $\tilde{A}$ is replaced by computing that of a much smaller $k \times k$ matrix B.
- The technique is fairly simple algorithmically. These operations can also be easily parallelized.

- If $q > 1$, all the operations that we perform are matrix-matrix operations, allowing us to use BLASv3. For $q = 1$, some operations are matrix-vector operations and SVD-RD will be forced to use the inferior BLASv2 routines. (More details are in the experimental section 4).
- SVD-RD works as a streaming algorithm.
- SVD-RD can easily be used to downdate rows from any position. We would need to multiply the matrix with an appropriate permutation matrix to bring the corresponding rows to the bottom and then use the proposed algorithm.
- The technique can be easily adopted for column downdate operations.

4. Experiments and results. We first describe the hardware platform and the software framework that we used to perform our experiments. We then briefly describe the dataset that was used in the experiments. We later get into the detailed analysis of performance results.

We used a 6-core 2.66 GHz Xeon platform with 12MB shared L3 cache and 256K private L2 cache for our experiments. The system had 12GB RAM and was running Debian Linux 6.0. For the software routines, we used the ATLAS framework [21] which provides a portable and automatically tuned linear algebra library. We used ATLAS 3.8.4 to tune the libraries for performance on the 6-core machine. We used the LAPACK and BLAS function calls for the numerical routines and matrix operations. Most of the matrix operations use BLAS v3 function calls. Since ATLAS was tuned for the 6-core machine, all the BLAS function calls used in our implementation were performing matrix operations in parallel using 6 cores. We used LAPACKE 3.4.1 [1] as the C interface. We implemented SVD-RD and Brand's downdate method using C++ and compiled the code using g++ 4.4.5 and used the ATLAS libraries for both the methods. For SVD recomputation, we used the `dgesvd` routine from LAPACKE. The programs were all compiled with option `-O3` turned on.

The experimental results for the two cases are presented in separate sections.

4.1. Analysis for Case 1. We created random matrices (entries of random matrix are in the range 0-200) of dimension $m \times n$ where $m = 40, \cdots, 100$ in the steps of 10 and $n = 20\text{K}, \cdots, 120\text{K}$ in the steps of 20K for evaluating the performance of the algorithm[2]. The performance of proposed algorithm is compared with SVD recomputation and the Brand's algorithm.

We analyze the effect of three different parameters on performance: number of rows (m), number of columns (n) and number of rows downdated (q). We used these three parameters because all of them have a direct influence on the runtime. We show the behavior of our algorithm with respect to all these parameters. For each matrix size, we considered all the values of q listed in column 3 of Table 5.

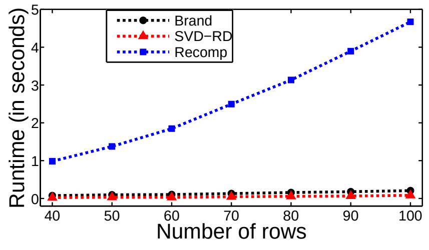
Table 5: Input dataset and parameters used in experimentation for Case 1.

Number of Rows (m)	Number of Columns (n)	Number of Rows Downdated (q)
$40, 50, \cdots, 100$	20K, 40K, $\cdots$, 120K	$q = 1, 2, \cdots, 16$

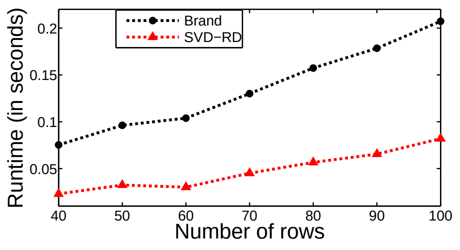
4.1.1. Runtime against number of rows and columns for single-row downdate. In Figures 1 and 2, we compare the runtime of SVD-RD taken for single

[2] We use the notation K written as an upper case regular typeset letter to mean "kilo" or thousand.

row downdate with the runtimes of recomputation and Brand's algorithm, for fixed n and fixed m respectively. In recomputation, we compute the SVD of the $(m-q) \times n$ matrix from scratch without using the previous decomposition. In Figure 1a and 2a, it is evident that both Brand's algorithm and SVD-RD are much better than SVD recomputation. In Figures 1b and 2b, we only plot the runtime of Brand's algorithm and SVD-RD so that the difference in performance can be seen clearly. It is clear that as the number of rows increases SVD-RD performs better than Brand's algorithm. It is also interesting to observe that both SVD-RD and Brand's algorithm show linear increase.

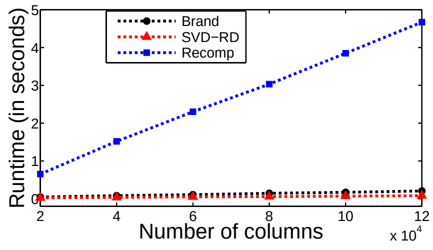


(a) Runtime comparison of Recomputation, Brand's Method and SVD-RD

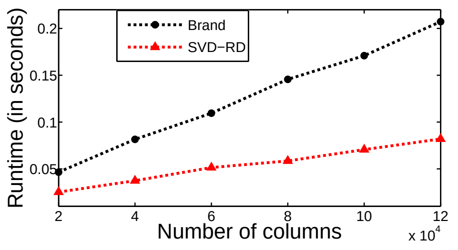


(b) Runtime comparison of Brand's Method and SVD-RD

Fig. 1: Runtime vs. number of rows for $q = 1$ and $n = 120$K



(a) Runtime Comparison of Recomputation, Brand's Method and SVD-RD

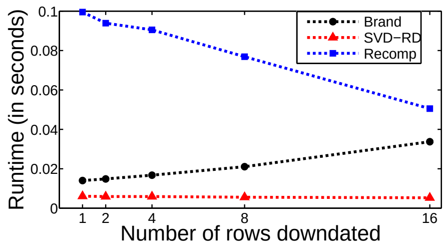


(b) Runtime Comparison of Brand's Method and SVD-RD

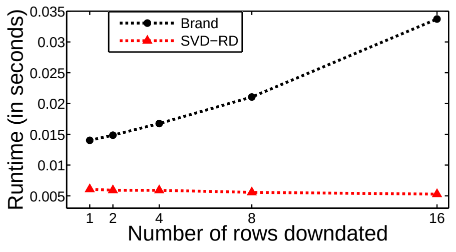
Fig. 2: Runtime vs. number of columns for $q = 1$ and $m = 100$

Overall, we observed similar trends against m and n for other values of q mentioned in Table 5. The cubical growth of runtime against m is not seen in Figure 1b because of the relatively large value that n has compared to m.

4.1.2. Runtime against number of rows downdated. In Figure 3, we show the effect of q on the runtime of the three methods. We fixed the matrix size to be 40×20K and downdated 1 to 16 rows at a time. The runtime of recomputation reduces because of reduction in the size of matrix after removal of rows. The figure however demonstrates an interesting property that SVD-RD has. Even though the complexity analysis suggests an increase in runtime with respect to q when m and n are fixed, we observe in Figure 3a that the runtime of SVD-RD reduces slightly. In contrast, Brand's method requires more runtime if more rows have to be removed.



(a) Runtime Comparison of Recomputation, Brand's Method and SVD-RD



(b) Runtime Comparison of Brand's Method and SVD-RD

Fig. 3: Runtime vs. number of rows downdated for $m = 40$ and $n = 20$K

4.1.3. Explanation of the runtime behavior. The complexity analysis of both SVD-RD and Brand's method suggests that they have the same rate of growth. The complexity of SVD recomputation is $O(m^2 n)$ as shown in Table 3. The runtime behavior however shows that SVD-RD is faster than the Brand's method as well as SVD recomputation. At the first glance this seems anomalous given the complexity analysis in Table 3. We now explain this anomaly in detail.

Overall, there are two reasons why SVD-RD is faster than Brand's method. First, the number of basic arithmetic operations done in our method is lesser than those of Brand's. Second, the total number of memory references to various matrices are lesser in number. To validate these points, we show the statistics collected during these runs using a kernel level performance monitoring tool called `perf`. `perf` is a profiler for Linux 2.6+ based systems which can provide detailed statistics about a program that is running by monitoring both hardware counters and software counters. Specifically, we watched counters for `cache-references`, `cache-misses` and `instructions`. We used the 40×120K matrix for this and downdated one row ($q = 1$). The data is summarized in Table 6 in the fifth and sixth columns. We see that the number of cache references is down by 21.96% and the number of instructions executed is down by 2.33%. Cache misses in SVD-RD is lesser than in Brand's method[3]. Besides these factors, we also observed a 31.7% reduction in the miss rate of loads at the last level cache (LLC) and a 21.6% reduction in the miss rate of stores at the LLC.

Table 6: Performance measurements for single-row downdate Using hardware counters profiled by `perf` using a matrix of size $m = 40, n = 120$K.

Performance Counters	SVD-RD Algorithm	SVD Recomputation	Brand's Method	Reduction in SVD-RD w.r.to	
				Recomputation	Brand's Method
Instructions	3626351810	4874398707	3710553110	34.4%	2.33%
Cache-misses	9537246	58545784	10711221		
Cache-references	13992797	81041308	17065154	479%	21.96%

It can also be observed that SVD-RD cuts down the number of instructions by a third and has 4.79 times less cache references than recomputing the SVD from scratch.

[3]We do not provide percentage difference in cache miss rates because of the difference in the number of memory accesses. We present this metric for the sake of completion.

Furthermore, recomputation can be seen to have > 5X cache misses than the other methods. Cache misses incur a heavy penalty - usually an order of magnitude increase in the latency to access data. These three aspects also indeed explain the anomaly behind the significantly higher runtime of recomputation despite what the complexity analysis shows.

In summary, SVD-RD needs fewer instructions to be executed, makes lesser memory accesses, and uses caches better, resulting in an effective speedup over Brand's method and an enormous speedup compared to recomputation even for one row downdate. These effects would be even more pronounced for larger values of q. In the next section, we provide more detailed results of speedup obtained using our method.

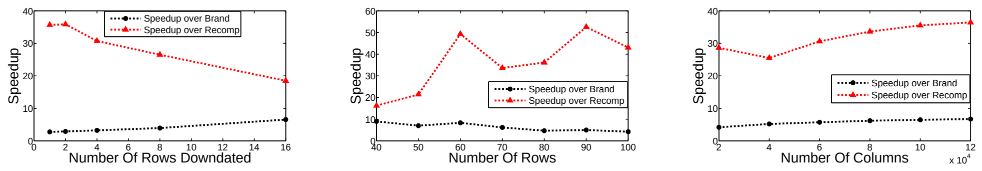
4.1.4. Speedup analysis. In Tables 7 and 8, we show the speedup of proposed algorithm over Brand's algorithm and recomputation of SVD for different matrix sizes. The value in each cell is the speedup for that matrix size for single row downdate and the value is rounded off to one decimal place. The general trend indicates that for a fixed m, speedup increases when n is increased. The speedup over Brand's method (refer to Table 7), for a fixed n, decreases with increase in m but opposite to that speedup over recomputation (refer to Table 8) increases with increase in m.

Table 7: Speedup of SVD-RD over Brand's method for $q = 1$.

	$n = 20$K	$n = 40$K	$n = 60$K	$n = 80$K	$n = 100$K	$n = 120$K
$m = 40$	2.3	2.8	3.0	3.1	3.2	3.3
$m = 50$	2.2	2.7	2.8	2.9	2.9	3.0
$m = 60$	2.3	2.8	3.1	3.4	3.4	3.5
$m = 70$	2.1	2.5	2.7	2.9	2.8	2.9
$m = 80$	2.0	2.4	2.5	2.7	2.6	2.8
$m = 90$	1.9	2.3	2.4	2.7	2.6	2.7
$m = 100$	1.8	2.2	2.1	2.5	2.4	2.5

Table 8: Speedup of SVD-RD over recomputation for $q = 1$.

	$n = 20$K	$n = 40$K	$n = 60$K	$n = 80$K	$n = 100$K	$n = 120$K
$m = 40$	16.4	24.6	33.6	37.8	40.6	42.8
$m = 50$	17.3	29.4	35.7	39.1	39.9	42.4
$m = 60$	21.7	41.9	51.0	56.5	58.5	61.3
$m = 70$	21.5	39.9	45.6	50.7	51.7	55.4
$m = 80$	23.0	40.1	46.1	50.5	53.3	55.3
$m = 90$	23.7	40.0	49.7	54.5	57.7	59.5
$m = 100$	25.9	40.6	44.7	51.8	54.5	57.0



(a) Speedup vs. q for m=50 and n=60K (b) Speedup vs. m for n=80K and q=16 (c) Speedup vs. n for m=70 and q=16

Fig. 4: Speedup of SVD-RD vver Brand's method and recomputation

Influence of q, m and n on the speedup is shown in Figure 4. In Figure 4(a) we show the effect of q on speedup obtained for the 50×60K matrix. We can notice that with increase in q, speedup over recomputation reduces but speedup over Brand's method increases. At $q = 16$, the number of rows removed is not of $O(1)$ anymore: it becomes comparable to m. When q becomes comparable to m it is expected that recomputation would have much lesser work as the SVD of a $(m - q) \times n$ matrix becomes easier to calculate.

In Figure 4(b) we show the effect of m on speedup obtained by fixing $n = 80$K and $q = 16$. With increase in m, the speedup over recomputation increases but the speedup over Brand's method decreases.

Finally, in Figure 4(c) we plot the effect of increase of n on the speedup obtained by fixing $m = 70$ and $q = 16$. With increase in n, there is hardly any effect on the speedup. With respect to both recomputation and Brand's method, SVD-RD achieves a constant speedup over a wide range of columns.

The observations made for $q = 16$ in Figures 4(b) and (c) are consistent with the trends seen in Tables 7 and 8.

For other choices of combinations of m, n and q, the observations made in this section hold true. We do not provide all the results due to space constraints.

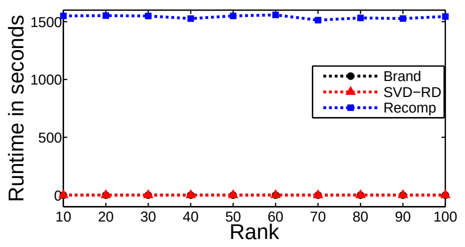
4.1.5. What if $m \to n$?. In the earlier section, we assumed that $m \ll n$. We now discuss a sub-case where m is still less than n but $m \to n$ (for example, if $m = n/4$). Note that $k = m$ also becomes comparable to n.

Referring to Table 3, we see that the runtime of both SVD-RD and Brand's method have a m^3 term and a m^2n term. When $m \ll n$, the m^2n term will dominate the m^3 term. However, when $m \to n$, the m^3 term may start dominating the other. We could expect the speedup obtained by both SVD-RD and Brand's method w.r.to recomputation to reduce significantly. Our experiments with $m = 3K, 4K, ..., 10K$ and $n = 2m, 4m$, $k = m$ and $q = 1$ revealed that the that the cache misses were within 2.5% of each other. The speedup of SVD-RD over Brand's method was a modest 2% and over recomputation was a modest 4%. If 1% of rows were downdated, the speedup w.r.to Brand's method became better and went upto 4%. However, recomputation became faster than SVD-RD by 2%. It should be noted from first row of Table 3 that recomputation has a $(m - q)^2$ factor that reduces with increase in q. Recomputation does lesser work with increasing q if all the other parameters are fixed at the same values. Overall, SVD-RD is not competitive any more.

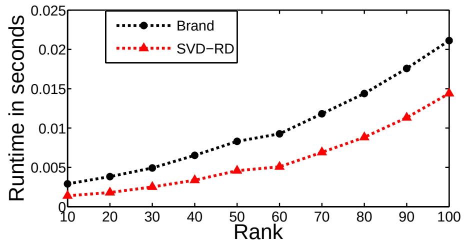
So, if $m \to n$, we propose that a good k-rank approximation be used instead of the full-rank models. Results shown for square matrices in section 4.2 will support this statement.

4.2. Analysis for Case 2. In this section, we present results for the general case: matrix dimensions have no relative constraints on each other and k is set to low values relative to m and n.

In Figures 5 and 6, we present the runtime behavior of the proposed algorithm, Brand's algorithm and SVD recomputation against k and m respectively. Figure 5 shows runtime against k for a 5K×5K Hilbert matrix and $q = 1$. It is evident from Figure 5a that SVD recomputation is independent of k whereas in Figure 5b we can see that both SVD-RD and Brand's method are cubic with respect to k and that SVD-RD performs better than Brand's method. Figure 6 shows runtime against m of square matrices (i.e. $n = m$) for a fixed $k = 50$ and $q = 1$. In Figure 6a, we can see that recomputation is cubic in m because SVD recomputation of an $(m-1) \times m$ matrix will take $O(m^3)$. It is evident in Figure 6b that both SVD-RD and Brand's method are linear in m but SVD-RD performs better than Brand's method.

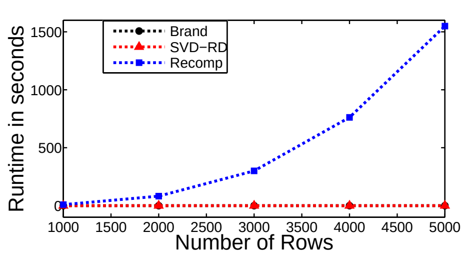


(a) Runtime comparison of recomputation, Brand's method and SVD-RD

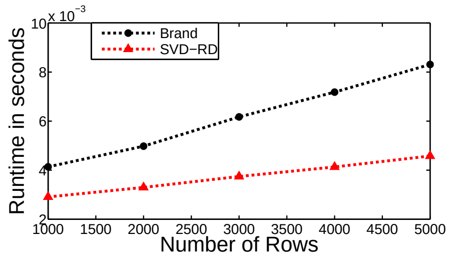


(b) Runtime comparison of Brand's method and SVD-RD

Fig. 5: Runtime vs. rank for $m = n = 5$K and $q = 1$



(a) Runtime comparison of recomputation, Brand's method and SVD-RD



(b) Runtime comparison of Brand's method and SVD-RD

Fig. 6: Runtime vs. size m (with $n = m$) for fixed rank $k = 50$ and $q = 1$.

We also calculated the average error per element incurred due to the proposed algorithm. We calculated per-element error and averaged the absolute values of these errors. For the $m \times n = 5000 \times 5000$ that we used earlier, for $q = 1$ and $k = 10, \cdots, 100$, we observed average error per element in the range $4.71747 \times 10^{-8} - 1.15 \times 10^{-7}$. The average error per element for other matrix sizes were similar too.

4.2.1. Speedup analysis. In Tables 9, 10, 11 and 12, we present all the experimental results for speedup that SVD-RD achieves over Brand's algorithm and SVD recomputation for $q = 1$. Tables 9 and 10 present speedup for square matrices whereas Tables 11 and 12 present speedup for rectangular matrices[4] (with $m > n$). The common trend across the tables is that i) for a fixed matrix size, as k increases, the speedup decreases; and ii) for a fixed k, as m increases, the speedup increases.

Table 9: Speedup of SVD-RD over Brand's method for square matrices and $q = 1$.

	$k = 10$	$k = 20$	$k = 30$	$k = 40$	$k = 50$	$k = 60$	$k = 70$	$k = 80$	$k = 90$	$k = 100$
$m = 1000$	1.7	1.7	1.6	1.5	1.4	1.3	1.3	1.2	1.1	1.1
$m = 2000$	1.8	1.8	1.7	1.6	1.5	1.4	1.4	1.3	1.3	1.2
$m = 3000$	2.0	2.0	1.9	1.8	1.6	1.6	1.5	1.5	1.4	1.3
$m = 4000$	2.1	2.0	2.0	1.9	1.7	1.8	1.6	1.5	1.5	1.4
$m = 5000$	2.1	2.1	2.0	1.9	1.8	1.8	1.7	1.6	1.5	1.5

Table 10: Speedup of SVD-RD over recomputation for square matrices and $q = 1$.

	$k = 10$	$k = 20$	$k = 30$	$k = 40$	$k = 50$	$k = 60$	$k = 70$	$k = 80$	$k = 90$	$k = 100$
$m = 1000$	7877	6666	5498	3969	3039	2469	1828	1420	1118	929
$m = 2000$	66622	55417	43746	33711	24896	21277	15300	11825	9252	7477
$m = 3000$	236140	190503	151683	111736	80218	69807	51102	39429	31176	24605
$m = 4000$	586674	453878	348039	251288	184317	164763	119487	92463	72321	59363
$m = 5000$	1118758	862305	615986	450928	338429	306924	218896	173601	134842	107265

Table 11: Speedup of SVD-RD over Brand's method for rectangular matrices with $n = 100$ and $q = 1$.

	$k = 10$	$k = 20$	$k = 30$	$k = 40$	$k = 50$	$k = 60$	$k = 70$	$k = 80$	$k = 90$	$k = 100$
$m = 100$	1.0	1.0	1.1	1.1	1.1	1.0	1.0	1.0	1.0	1.0
$m = 1000$	1.3	1.2	1.2	1.0	1.0	1.0	1.0	1.0	1.0	1.0
$m = 10000$	1.6	1.6	1.4	1.4	1.3	1.3	1.3	1.3	1.2	1.3
$m = 100000$	4.0	3.0	2.4	2.1	1.9	2.0	1.8	1.8	1.7	1.6
$m = 1000000$	5.3	3.4	2.6	2.2	2.0	2.2	2.0	1.9	1.8	1.7

Table 12: Speedup of SVD-RD over recomputation for rectangular matrices with $n = 100$ and $q = 1$.

	$k = 10$	$k = 20$	$k = 30$	$k = 40$	$k = 50$	$k = 60$	$k = 70$	$k = 80$	$k = 90$	$k = 100$
$m = 100$	11.1	7.6	5.0	3.2	2.2	1.6	1.3	1.0	1.0	0.9
$m = 1000$	29.8	20.4	13.8	8.5	6.5	5.1	3.9	3.0	2.4	2.0
$m = 10000$	151.5	107.2	75.3	46.4	32.1	26.3	20.0	15.0	11.7	9.8
$m = 100000$	917.2	420.2	228.9	140.5	94.4	92.5	62.0	50.9	43.3	33.6
$m = 1000000$	1477.4	556.2	287.1	170.0	113.2	120.9	80.3	65.0	55.7	44.4

The speedup over recomputation is orders of magnitude larger and this is expected given that SVD-RD is asymptotically better than recomputation for downdating k-rank SVD[5]. The issues related to instructions, cache accesses and cache misses are

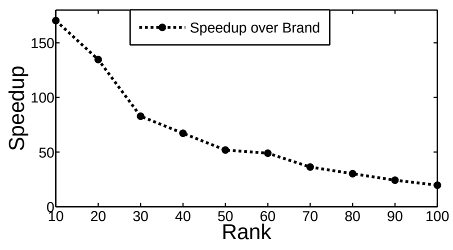
[4] All square matrices used in our experiments reported in this section are Hilbert matrices and all rectangular matrices reported in this section are random matrices with entries in the range 0-200.

[5] We also evaluated a MATLAB version of LMSVD [14] which can compute the dominant k singular values and the corresponding vectors in $O(mnk)$ time. We noticed that LMSVD was good for smaller values of k but started accumulating significant errors (of the range 0.3-0.66) for $k > 30$. Hence we limit our comparison to full recomputation of SVD.

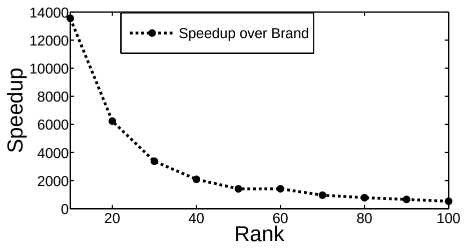
pertinent here too as they were in section 4.1.3. SVD-RD provides better speedup for downdating square matrices than rectangular matrices of comparable size.

For $q = 1$, the speedup over Brand's method is relatively modest and is highest for bigger matrices with lower rank. However, this speedup is promising given that q is still very small. We further studied the runtime for larger values of q. Since we have established that SVD-RD is faster than recomputation even for $q = 1$, we only show speedup of SVD-RD over Brand's algorithm. Figure 7a is for the square matrix of size 5K $\times$5K and $q = 100$ and Figure 7b is for the rectangular matrix of size $10^5 \times 100$ and $q = 1000$. It is evident in the figures that speedup decreases significantly with increase in k but it is worthwhile to note that the minimum speedup in Figures 7a and 7b are 19.6 and 526, respectively. Earlier for the same matrix sizes and $q = 1$, minimum speedup were 1.5 and 1.7, respectively. This large jump in speedup is because Brand's algorithm is cubic in q whereas SVD-RD is (algorithmically) linear in q but remains more or less independent of q in practice. The overall trend is as expected because as $k \to \min(m, n)$ the problem gets closer to using full-rank decomposition and hence the speedup is expected to reduce.

The behavior of speedup with respect to q (for $q = 1,\ 2,\ 4,\ 8,\ 16,\ 32,\ 50$ and 100) for fixed m, n and k is shown in Figure 8. The trends in Figures 8a and 8b are similar to each other. It is evident from the figures that speedup increases quadratically with respect to q.



(a) For $m = n = 5\text{K}$ and $q = 100$



(b) For $m = 100000, n = 100$ and $q = 1000$

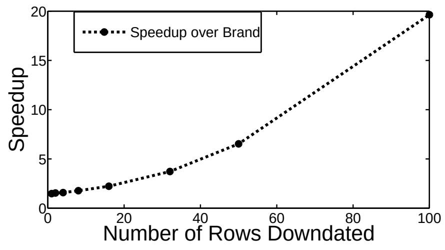
Fig. 7: Speedup of SVD-RD over Brand's method vs. rank (k)

In summary, SVD-RD is a very promising technique for downdating matrices for which low-rank approximation is sufficient. Specifically, if a sequence of downdate requests can be aggregated and if we are allowed to perform a batch downdate, using SVD-RD for the whole batch instead of handling one downdate request at a time would be orders of magnitude faster than the other methods.

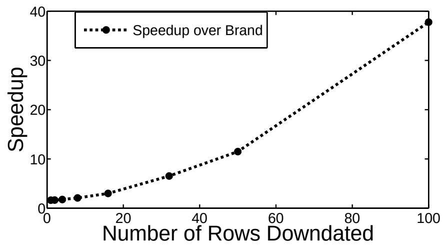
4.3. Bounding the approximation error. We present some preliminary results on the bounds of approximation errors incurred due to downdating using SVD-RD. As we explained earlier, the approximations used by SVD-RD in some calculations can result in errors.

All the notations used here are the same as the ones in section 4.2. Let us define the error matrix E_{SVD-RD} as

$$\begin{aligned} E_{SVD-RD} &= \tilde{A} - \tilde{U}\tilde{S}\tilde{V}^T \\ &= \mathcal{U}(1:m-q,:)\mathcal{S}\mathcal{V}^T - \mathcal{U}(1:m-q,:)B\mathcal{V}^T \end{aligned}$$



(a) For $m = n$ =5K and $k = 100$



(b) For $m = 100000, n = 100$ and $k = 100$

Fig. 8: Speedup of SVD-RD over Brand's Method Vs. Number of Rows Downdated (q)

$$\begin{aligned}
&= \mathcal{U}(1:m-q,:)\mathcal{S}\mathcal{V}^T - \mathcal{U}(1:m-q,:)(I - D^T D)\mathcal{S}\mathcal{V}^T \\
&= \left[\mathcal{U}(1:m-q,:) - \mathcal{U}(1:m-q,:)(I - D^T D)\right]\mathcal{S}\mathcal{V}^T \\
&= \mathcal{U}(1:m-q,:)D^T(D\mathcal{S}\mathcal{V}^T)
\end{aligned}$$

Let $||E_{SVD-RD||_F}$ be the Frobenius norm of error matrix such that

$$\begin{aligned}
||E_{SVD-RD}||_F &= ||\mathcal{U}(1:m-q,:)D^T(D\mathcal{S}\mathcal{V}^T)||_F \\
&\leq ||\mathcal{U}D^T(D\mathcal{S}\mathcal{V}^T)||_F \\
&= ||D^T D\mathcal{S}\mathcal{V}^T||_F \qquad \because \mathcal{U} \text{ is a column-wise orthonormal matrix} \\
&= ||D^T D\mathcal{S}||_F \qquad \because \mathcal{V} \text{ is a column-wise orthonormal matrix}
\end{aligned}$$

Thus,

$$||E_{SVD-RD}||_F \leq ||D^T DS||_F \tag{4.1}$$

In summary, if SVD-RD is used to downdate a full rank decomposition of an $m \times n$ matrix with $m < n$ (refer to section 3), we incur no approximation error. However, if the downdate is performed on a rank-k approximation of any $m \times n$ matrix, the error incurred is upper bounded by the Frobenius norm of $D^T DS$, as shown above.

5. Conclusions. In this paper, we presented a streaming algorithm called SVD-RD that can perform approximation free downdate on full rank SVD of matrices with $m < n$. We discussed the algorithmic and implementation details of our method and analyzed the source of speedup of our method.

We also showed how to generalize the algorithm to the k-rank approximation problem for general matrices. We observed that the method is an approximation and we derived theoretical upper bounds for errors. We also observed remarkable speedup over SVD recomputation and Brand's method for the low rank case. For applications where speed can be traded-off for accuracy, SVD-RD would be ideal.

In the future, we would like to develop approximation-free algorithms for downdating the k-rank version that can retain the runtime benefits of SVD-RD.

REFERENCES

[1] *The LAPACKE C Interface to LAPACK*, 2013.

[2] M. W. Berry, *Large scale sparse singular value computations*, International Journal of Supercomputer Applications, 6 (1992), pp. 13–49.

[3] R.B. Bradford, *Efficient discovery of new information in large text databases*, in Proceedings of the International Conference on Intelligence and Security Informatics, LNCS '05.

[4] M. Brand, *Fast low-rank modifications of the thin singular value decomposition*, Linear Algebra and its Applications, 415 (2006), pp. 20 – 30. Special Issue on Large Scale Linear and Nonlinear Eigenvalue Problems.

[5] M. Brand, Singular Value Decomposition of Uncertain Data with Missing Values, in Proceedings of the European Conference on Computer Vision, ECCV '02, 2002, pp. 707–720.

[6] R. Cangelosi and A. Goriely, *Component retention in principal component analysis with application to cdna microarray data*, BMC Biology Direct, 2 (2007).

[7] D. Chetverikov and A. Axt, *Approximation-free running svd and its application to motion detection*, Pattern Recogn. Lett., 31 (2010), pp. 891–897.

[8] Y. Gong and X. Liu, *Generic text summarization using relevance measure and latent semantic analysis*, in Proceedings of the 24th Annual International ACM SIGIR Conference on Research and Development in Information Retrieval, SIGIR '01, New York, NY, USA, 2001, ACM, pp. 19–25.

[9] M. Gu and S. C. Eisenstat, *Downdating the Singular Value Decomposition*, SIAM J. Matrix Anal. Appl., 16 (1995), pp. 793–810.

[10] P. Hall, D. Marshall and R. Martin, *Adding and subtracting eigenspaces with eigenvalue decomposition and singular value decomposition*, Image and Vision Computing, 20 (2002).

[11] S.-F. Hsieh, *On Recursive Least Square Filtering Algorithms and Applications*, PhD Thesis, UCLA, 1990.

[12] T. K. Landauer, D. S. Mcnamara, S. Dennis and W. Kintsch, eds., *Handbook of Latent Semantic Analysis*, Lawrence Erlbaum Associates, 2007.

[13] A. Levey and M. Lindenbaum, *Sequential Karhunen-Loeve basis extraction and its application to images*, IEEE Transactions on Image Processing, 9(2000), pp. 1371–1374.

[14] X. Liu, Z. Wen and Y. Zhang, *Limited memory block Krylov subspace optimization For computing dominant singular value decomposition*, SIAM Journal on Scientific Computing, 35 (2013), pp. A1641–A1688.

[15] F. Monay and D. Gatica-Perez, *On image auto-annotation with latent space models*, in Proceedings of the Eleventh ACM International Conference on Multimedia, MULTIMEDIA '03, New York, NY, USA, 2003, ACM, pp. 275–278.

[16] M. Moonen, P. .V. Dooren and J. Vandewalle, *A Systolic Array for SVD Updating*, SIAM Journal on Matrix Analysis and Applications, 14(93), pp. 353–371.
Read More: http://epubs.siam.org/doi/abs/10.1137/0614025

[17] H. Park and S. .v. Huffel, *Two-way bidiagonalization scheme for downdating the singular-value decomposition* , Linear Algebra and its Applications, 222(1995), pp. 23–39.

[18] R. J. Price and A. E. Zukas, *Application of latent semantic indexing to processing of noisy text*, in Proceedings of the International Conference on Intelligence and Security Informatics, LNCS '05.

[19] D. A. Ross, J. Lim, R-S. Lin and M-H. Yang, *Incremental Learning for Robust Visual Tracking*, International Journal of Computer Vision, 77(2008), pp. 124–141.

[20] G. W. Stewart, *On the early history of the singular value decomposition*, SIAM Rev., 35 (1993), pp. 551–566.

[21] R. . Whaley, A. Petitet and J. J. Dongarra, *Automated empirical optimization of software and the atlas project*, PARALLEL COMPUTING, 27 (2000), p. 2001.

[22] D. I. Witter and M. W. Berry, *Downdating the latent semantic indexing model for conceptual information retrieval*, The Computer Journal, 41 (1998), pp. 589–601.

[23] D. I. Witter, *Downdating the latent semantic indexing model for information retrieval*, Master's thesis, University of Tennessee, Knoxville, 1997.

[24] J. Zhang, S. Li, L. Cheng, X. Liao and G. Cheng, *A fast and stable algorithm for downdating the singular value decomposition*, Computers and Mathematics with Applications, 10(2014), pp. 1421-1430.