Large Scale Latent Semantic Analysis

Andrew M. Olney

Institute for Intelligent Systems

&

Department of Psychology

University of Memphis

Dr. Andrew McGregor Olney

Institute for Intelligent Systems &

Department of Psychology

University of Memphis

365 Innovation Drive, Suite 303

Memphis, TN 38152

e-mail: aolney@memphis.edu

Abstract

Latent semantic analysis (LSA) is a statistical technique for representing word meaning that has been widely used for making semantic similarity judgments between words, sentences, and documents. In order to perform an LSA analysis, an LSA space is created in a two stage procedure, involving the construction of a word frequency matrix and the dimensionality reduction of that matrix through singular value decomposition (SVD). This paper presents LANSE, a SVD algorithm specifically designed for LSA, which allows extremely large matrices to be processed using off the shelf computer hardware.

Large Scale Latent Semantic Analysis

Introduction

Latent Semantic Analysis (LSA) is a statistical technique for representing world knowledge (Deerwester, Dumais, Furnas, Landauer, & Harshman, 1990; Landauer, Foltz, & Laham, 1998). Since its discovery, LSA has been heavily used in both the psychological and computational linguistics communities. In psychological research, LSA has been shown to approximate vocabulary acquisition in children, grade essays, match students with optimal texts for learning, predict text coherence, make human-like text similarity judgments, take subject matter multiple choice tests with human performance, mirror lexical priming, and understand student input during tutorial dialogue, amongst many others (Graesser, VanLehn, Rose, Jordan, & Harter, 2001; Landauer et al., 1998; Landauer, McNamara, Dennis, & Kintsch, 2007; Landauer & Dumais, 1997; Rehder et al., 1998; Wolfe et al., 1998; Foltz, Kintsch, & Landauer, 1998). In computational linguistics, LSA has been used for text segmentation, speech recognition, entailment detection, summarization, and information retrieval, again, amongst many others (Coccaro & Jurafsky, 1998; Dumais, 1991; Foltz et al., 1998; Bellegarda, 2000; Deng & Khudanpur, 2003; Olney & Cai, 2005b, 2005a; Olney, 2007a; Deerwester et al., 1990). The duality of use across these communities underlines the multiple viewpoints surrounding LSA. On the one hand, LSA can be seen as a valuable tool for imbuing computers with some notion of semantic relatedness, and on the other, LSA can be seen as a computational model of cognition with wide ranging implications for cognitive theory (Landauer et al., 2007).

The fact that LSA enjoys wide use in many communities is a testament to the elegance of its model and the simplicity of its use. Conceptually, LSA maps words into points in a space. Similar words tend to be nearby in this space, while unrelated words are more distant. Since each point in this space can be represented as a vector, representations for documents can be created by summing the vector representations of their constituent words. The vector addition property has two important consequences. First, any size collection of words can be compared to any other size collection in the same way that two individual words can be compared to each other. Secondly, the representation of any collection of words has the same dimensionality as a single word in the collection: both are a vector of fixed size.

Using this conceptual description as a background, we now describe the process of LSA space creation in more detail. At a high level, creating LSA spaces involves two steps, construction of a term-document matrix and the singular value decomposition of that matrix. A term-document matrix is created by counting term (or word) frequencies across a collection of documents. In the matrix, the value at row i column j is the number of times term i appeared in document j. Weighting schemes can further be applied to this matrix to improve task performance (Dumais, 1991). Several observations can be made about the term-document matrix for natural languages such as English. First, the matrix will necessarily be quite sparse, since not all words occur in all documents. Thus for any given column of the matrix corresponding to a document, many of its entries will be zero. Moreover, the matrix is likely to be rectangular in shape, since there is no constraint that the number of words equals the number of documents.

The second step of LSA is singular value decomposition (SVD). Singular value decomposition (SVD) is a fundamental technique in linear algebra. SVD is also an unsupervised method of dimensionality reduction that is optimal in the least squares sense. To see why, consider the definition of SVD:

$$A = U\Sigma V^T \tag{1}$$

where U and V are orthonormal matrices and $\Sigma = diag(\sigma_1, \ldots, \sigma_n)$ and

$\sigma_1 \geq \ldots \geq \sigma_n \geq 0$. The σ_i are the singular values of the matrix A.

A theorem by Eckart and Young (1936) establishes the dimensionality reduction property of SVD. The theorem states that a rank k approximation of the original rank n matrix may be created by setting singular values $k+1 \leq q \leq n$ to zero. Moreover, the theorem states that this reduced rank matrix A_k has minimal distance to A in terms of the Frobenius norm:

$$\|A\|_F = \sqrt{\sum_{i=1}^{n} \sigma_i^2} \tag{2}$$

Thus, the theorem states:

$$\|A - A_k\|_F = \sqrt{\sigma_{k+1}^2, \ldots, \sigma_n^2} \tag{3}$$

In other words, by choosing a smaller number of dimensions, the resulting matrix A_k is an optimal approximation of the original matrix A in the least squares sense. For this reason, SVD can be a useful tool for dimensionality reduction and noise elimination: since the dimensions retained account for most of the variance in the matrix, the eliminated dimensions can be considered noise.

Equation 1 provides the definition of SVD but says nothing about how to calculate it. Indeed, calculation of the SVD is the most complex and challenging stage of creating an LSA space. Although a great deal of research has established multiple methods for calculating SVD (Bai, Demmel, Dongarra, Ruhe, & Van Der Vorst, 2000), LSA research to date has focused on a single method, the Lanczos algorithm with selective reorthogonalization (LANSO) (Martin & Berry, 2007). For reasons discussed in detail below, traditional SVD algorithms like LANSO, despite their speed, require large amounts of random access memory proportional to the size of the space being created. The size limitation has restricted the kinds of LSA spaces that have been made to date. For example, bigram spaces potentially contain N^2 rows, where N is the number of word types in the original corpus. Such large spaces require either a computer with a very large quantity of random access memory or an alternative algorithm without such a size limitation. In the remainder of this article, we outline an alternative algorithm called the Lanczos algorithm for semantic spaces (LANSE). Our algorithm is specifically designed for large-scale LSA spaces, and has previously been used in spaces with millions of bigram terms (Olney, 2007b, 2009) as well as traditional spaces from large collections like Wikipedia (Willits, D'Mello, Duran, & Olney, 2007).

The Lanczos Algorithm

In this section we outline the Lanczos algorithm, which is the basis for both LANSO and LANSE. Before describing the individual steps of the algorithm, it is worthwhile to step back and reconsider the goal of the algorithm, which is the SVD of the input matrix shown in Equation 1. It is straightforward to show a strong correspondence between the SVD in Equation 1 and a related eigendecomposition given in Equation 7.

$$A = U\Sigma V^T \tag{4}$$

$$AA^T = U\Sigma V^T(U\Sigma V^T)^T \tag{5}$$

$$AA^T = U\Sigma V^T V\Sigma U^T \tag{6}$$

$$AA^T = U\Sigma^2 U^T \tag{7}$$

These equations reveal two relationships between the SVD of A and the eigendecomposition of AA^T. First, the singular values of A are the square roots of the eigenvalues of AA^T, and second, the left singular vectors U of A are the eigenvectors of AA^T. Since LSA is typically concerned with only the left singular vectors U (the term vectors) and not the right singular vectors V (the document vectors) no further work is necessary. However, a similar result for V could be obtained either by backsolving from the obtained value of U or by beginning with A^TA rather than AA^T. In this way the SVD of a matrix can be found by transforming the SVD problem into an eigendecomposition of a square matrix.

Equation 7 provides an approach to calculating the SVD, indeed calculating the *complete* SVD of a matrix. But what if, as in LSA, one wishes to extract only the largest singular values from a matrix? This is possible through an approach known as triangularization. Triangularization transforms the matrix AA^T into another matrix T such that the complete eigendecomposition of T is only the partial eigendecomposition of AA^T. T is a tridiagonal matrix of the form:

$$T = \begin{bmatrix} \alpha_1 & \beta_1 & & \\ \beta_1 & \ddots & \ddots & \\ & \ddots & \ddots & \beta_{n-1} \\ & & \beta_{n-1} & \alpha_n \end{bmatrix} \tag{8}$$

The Lanczos algorithm will tridiagonalize a matrix. Applying the Lanczos algorithm to AA^T yields:

$$AA^T = Q_1 T Q_1^T \tag{9}$$

where T is the tridiagonal matrix given in Equation 8 and Q_1 is an orthogonal matrix related to T and AA^T through the Lanczos recursion (Stewart, 2001):

$$AA^T q_j = \beta_{j-1} q_{j-1} + \alpha_j q_j + \beta_j q_{j+1} \tag{10}$$

where $\alpha_j = q_j^T AA^T q_j$ and $\beta_j = \|AA^T q_j - \alpha_j q_j - \beta_{j-1} q_{j-1}\|$.

As defined in Equation 10, each q_{j+1} is calculated from the previous q_j and q_{j-1}; likewise each α_j and β_j is calculated in sequence from α_1 and β_1. Thus each iteration of the Lanczos algorithm to AA^T will grow the set of q_j by one vector and the set of α_j and β_j by one value, i.e. matrices Q_1 and T increase by one on each iteration.

Returning to Equations 7 and 9, it is now possible to partially solve the eigendecomposition of AA^T by solving the eigendecomposition of T. To see that this is the case, consider the eigendecomposition of T:

$$T = Q_2 \Lambda Q_2^T \tag{11}$$

where Λ are the eigenvalues of T and Q_2 are the eigenvectors. Then:

$$AA^T = Q_1(Q_2 \Lambda Q_2^T)Q_1^T \tag{12}$$

$$AA^T = Q_1 Q_2 \Lambda (Q_1 Q_2)^T \tag{13}$$

The eigenvalues and eigenvectors of T are preserved because each Q is orthogonal and therefore a unitary transformation. Therefore, from Equation 7 we have that $\Lambda = \Sigma^2$, so the singular values of A are the square roots of the eigenvalues of AA^T. Likewise $Q_1 Q_2$ are the eigenvectors of AA^T and the left singular vectors of A.

The problem with orthogonality

The Lanczos method as presented has a single significant weakness: loss of orthogonality. Although the equations presented above are sound, they require exact arithmetic to function properly. Unfortunately, computer hardware has finite precision. This means that instead of being able to represent a number like $.\bar{6}$ fully, the same number might only be represented to say 15 decimal places, a rounding error. In practice, this means that Equation 10 will become unstable and the columns of Q_1 will lose orthogonality, which, as mentioned above, is vital for maintaining the proper relationship between the eigendecomposition of T and the eigendecomposition of AA^T. Interestingly, formal error analyses of the finite-precision Lanczos algorithm show that loss of orthogonality happens just as an eigenvalue begins to converge (Paige, 1971; Parlett, 1998).

The traditional strategy for dealing with loss of orthogonality is to enforce it explicitly through reorthogonalization. Intuitively, the way to enforce orthogonality is to keep track of all the previous q_j of Q_1, such that a new q_{j+1} can be compared to them. If q_{j+1} is not orthogonal to the previous q_j, it can be orthogonalized against them using a method like the Gram-Schmidt process. Stewart (2001) presents an excellent overview of the extensive literature on reorthogonalization strategies.

While reorthogonalization strategies produce numerically correct results, they have two drawbacks. The first drawback is the time it takes to reorthogonalize against previous vectors. The most naive and wasteful reorthogonalization strategy is to reorthogonalize at every iteration, also known as full reorthogonalization. However, because loss of orthogonality happens just as an eigenvalue begins to converge, it is not necessary to reorthogonalize at every step. Time optimal strategies, including LANSO (Parlett, 1998), attempt to predict or estimate when reorthogonalization is necessary and therefore avoid reorthogonalization when it is unwarranted, resulting in significant speed increases over full reorthogonalization.

The second drawback to reorthogonalization, however, is space. If each of the previous q_j has to be kept in memory for reorthogonalization, the total set can reach very large sizes. Consider that each q_j is the size of AA^T. Then the size of q_j is the same as the number of terms in the LSA space. As the number of terms grow, and as the number of dimensions in the LSA space grows, so does the size of Q_1. Ultimately, Q_1 overwhelms the amount of available memory. Assuming available memory of 4 gigabytes, 8 byte (double-precision) floating point numbers, and an LSA space of 300 dimensions, the number of terms is limited to approximately 1.7 million. Note that this is an extremely generous estimate because it does not include all of the other items which must be held in memory to compute the SVD such as A, T, Q_1, and Q_2.

The space problem for reorthogonalization is a problem for LANSO, the SVD algorithm commonly used by the LSA community (Martin & Berry, 2007) via the BellCore LSI tools (Schütze, 1995; Landauer & Dumais, 1997; Landauer et al., 1998; Coccaro & Jurafsky, 1998; Foltz et al., 1998; Bellegarda, 2000; Deng & Khudanpur, 2003; Olney & Cai, 2005b, 2005a). Indeed any reorthogonalization scheme suffers from requiring large amounts of random access memory. Large amounts of random access memory are needed to store all of the orthogonal vectors and to so ensure that new vectors are kept orthogonal to this existing set.

Alternatives to reorthogonalization

The alternative to reorthogonalization has become largely identified with the approach of Cullum and Willoughby (1985/2002). Cullum and Willoughby (1985/2002) present an extensive treatment of Lanczos algorithms for eigendecomposition, focusing in particular on alternatives to the reorthogonalization problem described above. The essence of the Cullum and Willoughby approach is to allow the vectors to lose orthogonality and deal with the slow convergence and degenerate eigenvalues that result. In their analysis, Cullum and Willoughby find that loss of orthogonality tends to create multiple and spurious eigenvalues. Thus the degenerate eigenvalues cause slow convergence because rather than finding the correct and distinct eigenvalues, the Lanczos recursion tends to find the same eigenvalues repeatedly and worse, eigenvalues that do not exist.

Cullum and Willoughby outline a number of measures to combat this problem, a full treatment of which is beyond the scope of this article. Overall, the complexity of the Cullum and Willoughby approach stems from its generalizability to many different problems, including finding some, or all, of the largest, middle, or smallest eigenvalues. Clearly this is quite different from the needs of the LSA community, where often only the 300 largest eigenvalues are required. We briefly describe two significant differences between the Cullum and Willoughby approach and LANSE. We will see that Cullum and Willoughby's arguments for these in the general case do not apply to the LSA case, motivating our alternatives.

The first significant feature of the Cullum and Willoughby approach is that they advocate using a matrix B of the form

$$B = \begin{bmatrix} 0 & A \\ A^T & 0 \end{bmatrix} \tag{14}$$

rather than AA^T or A^TA. If A is m by n, then B is $(m+n)$ by $(m+n)$ and so is symmetric. The advantage to using B over the AA^T matrix is that the eigenvalues are no longer the square roots of Λ, which could be a problem if the eigenvalues were very close together. Using B instead of AA^T supports the generality that Cullum and Willoughby want to provide for applications exclusively seeking the largest, smallest, or intermediate eigenvalues.

The second significant feature of Cullum and Willoughby's approach is their technique for identifying spurious eigenvalues. Their method compares the eigenvalues of T with eigenvalues from the matrix formed by removing the first row and column of T, $\hat{T}$. The test checks three conditions. First, if an eigenvalue occurs more than once in T, it is a real eigenvalue that has converged multiple times. Secondly, if the eigenvalue occurs in both T and $\hat{T}$, then the eigenvalue is spurious. Finally, and most interestingly, if an eigenvalue of T is not present in $\hat{T}$, then it is a true eigenvalue that will converge in time.

Cullum and Willoughby integrate this test for "good" eigenvalues with the eigenvalue finding procedure itself, which uses a bisection approach.

By combining these two features, Cullum and Willoughby are able to overcome the disadvantages of no reorthogonalization and capitalize on its advantages. Reorthogonalization approaches (using B) store $k * (m + n)$ vectors in memory, where k is equal to the number of iterations/dimensions. However, the Cullum and Willoughby approach without reorthogonalization stores $3 * (m + n)$ vectors in memory, This reduction in storage is possible because only two vectors are required to calculate the next vector using Lanczos recursion if orthogonality is not required. Reduction in storage has excellent consequences for large-scale LSA. In LSA, the number of dimensions k is often 300. Therefore the Cullum and Willoughby method allows roughly 100 times more documents to be processed in the same amount of memory. For example, the Touchstone Applied Sciences Associates corpus (TASA) (Landauer et al., 1998), a commonly used corpus in LSA research, contains approximately 9 million trigrams, bigrams, and unigrams. Using the calculation above, a reorthogonalization approach would require about 20 gigabytes of RAM, whereas the Cullum and Willoughby method would require about 206 megabytes of RAM. In other words, the Cullum and Willoughby method can run supercomputer-size problems on an ordinary desktop computer.

However, one remaining problem with the Cullum and Willoughby approach is that it is quite slow in practice. This slowdown occurs because although only two vectors are required to calculate the next vector, all the previous vectors Q_1 are needed to obtain the eigenvectors of AA^T, as in Equation 13. Slowdown occurs because Q_1 needs to either be stored on disk or to be regenerated on demand. Although disk operations are orders of magnitude slower than processor operations, the better of these two options depends on how long it takes to regenerate Q_1. Under this analysis, the Cullum and Willoughby approach trades space for time: less random access memory is required, but the result takes longer to obtain than reorthogonalization approaches.

There are several observations that can be made with respect to the Cullum and Willoughby approach and its suitability for LSA. The first is that use of the B matrix for enhanced precision is unnecessary for LSA, as has been previously demonstrated by (Berry, 1992). Discarding the B matrix in favor of the AA^T matrix has several advantages. First, the matrix itself takes less space: $m + n$ vectors become m vectors. The size reduction affects both the vectors in memory as well as the vectors saved to disk. Secondly, AA^T yields eigenvalues in half the number of iterations as B, with corresponding computing time speedups of half the time or better than B (Berry, 1992).

A second observation of Cullum and Willoughby's approach is that its generality is not required for LSA. Rather than needing singular values and vectors ranging from the largest, to the smallest, and to those in between, LSA is concerned only with the largest singular values, usually the 300 largest. Requiring only the 300 largest singular values is an enormous simplification in several respects. First, since the Lanczos algorithm finds the largest values first, one can simply start it from the beginning. Secondly, when one tests for convergence of eigenvalues, one can simply find them all using a robust algorithm like QR (Demmel, 1997; Trefethen & Bau, 1997). Thus the more sophisticated approach of Cullum and Willoughby is unnecessary.

Recent empirical work further lends support that these observations are well grounded. The summary result is that the tolerances for LSA are much lower than Cullum and Willoughby require because the distribution of singular values in LSA spaces follow Zipf's law (Ding, 2005). Using Ding's published models for standard document collections including Aeronautics from Cranfield Institute of Technology (CRAN), Communications of ACM (CACM), National Library of Medicine (MED), and Institute of Scientific Information (CISI), the 300th singular value differs from the 299th by only the second or third decimal place. The goodness of fit and similar parameters of these models led Ding to conjecture that singular values across all document collections obey a similar Zipf-law. This result suggests that using AA^T instead of B is justified because the singular values of LSA spaces are well-separated. Moreover, this result suggests that a simple alternative test for convergence can be used in favor of Cullum and Willoughby's test for spurious values during bisection.

A quite simple test, which Cullum and Willoughby attribute to Kats and Vorst (1976, 1977), is based on the interlacing theorem, which states that between any two eigenvalues of a matrix T_j is an eigenvalue of the matrix T_{j-1}. This suggests a simple test of checking whether (to some specified tolerance) an eigenvalue is in both T_j and T_{j-1}. If an eigenvalue is in both, it has converged. Cullum and Willoughby reject this test because a fixed tolerance is not a solution for SVD in the general case. For LSA however, as stated above, we know from Ding's work that a tolerance of a few decimal places will be adequate for the 300th singular value. A slightly more conservative and convenient tolerance is to require single (4 byte) precision, which roughly corresponds to 7 decimal places.

We synthesize the previous discussion into the following Lanczos algorithm for semantic spaces (LANSE), which is particularly appropriate for large-scale LSA spaces. LANSE proceeds as follows. Using the Lanczos method without reorthogonalization on AA^T, periodically check convergence of the eigenvalues of T using the Kats and Vorst test. Equation 10 has several possible algorithmic implementations, but we use the standard algorithm (2,7) (Paige, 1972), which is concisely given by (Bai et al., 2000). Once the desired number of eigenvalues has been found, stop the Lanczos algorithm and find the corresponding eigenvectors of T using inverse iteration. Then multiply these eigenvectors by the vectors Q_1 that were written to disk during the Lanczos algorithm, or regenerate Q_1 by repeating the tridiagonalization of T. The resulting matrix is U, the left singular vectors of A. The square roots of the eigenvalues are Σ, the singular values of A. The precise LANSE algorithm is specified below.

Algorithm 1 Calculate $AA^T = Q_1Q_2\Lambda(Q_1Q_2)^T$

Compute the tridiagonalization of AA^T *as follows:* (Equation 10)

Start with $r = v$, *a random starting vector*

$\beta_0 = \|r\|_2$

for $j = 1, 2, \ldots$ until *convergence* **do**

$q_j = r/\beta_{j-1}$

$r = Av_j$

$r = r - q_{j-1}\beta j - 1$

$\alpha_j = q_j^T r$

$r = r - q_j\alpha_j$

$\beta_j = \|r\|_2$

Compute approximate eigenvalues Λ *of* T_j *using QR.* (Equation 11)

Test for convergence of Λ *using the van Kats - van der Vorst test.*

end for

Compute approximate eigenvectors Q_2 *using inverse iteration.* (Equation 11)

Compute $\frac{Q_1Q_2}{\|Q_1Q_2\|} = U$ *, the left singular vectors.* (Equation 12)

Compute $\sqrt{\Lambda} = \Sigma$ *, the singular values.* (Equation 7)

Evaluation

To demonstrate the practical application of LANSE, we apply it to the standard Touchstone Applied Sciences Associates corpus (TASA) (Landauer et al., 1998). Earlier versions of LANSE have created semantic spaces for extremely large collections (Olney, 2007b, 2009; Willits et al., 2007), however using TASA allows us to compare with a traditional method that uses reorthogonalization. The TASA corpus contains random samples of text read during K-13 study, approximately 11 million words in all. After removing some punctuation, we created a term document matrix from the TASA corpus using log entropy weighting (Dumais, 1991) . The matrix had 129,477 rows corresponding to unique words, 38,962 columns corresponding to paragraph sized documents, and 5,829,247 non-zero entries. This matrix was then input to both LANSE and the ARPACK[1] SVD algorithm (Lehoucq, Sorensen, & Yang, 1998). ARPACK has good performance relative to other methods on comparable problems (Bergamaschi & Putti, 2002), has been widely used for SVD applications, and is distributed in MATLAB. Both LANSE and ARPACK algorithms were run until 300 singular values converged.

Our first analysis considers the accuracy of LANSE, using ARPACK as a gold standard. Singular values for both LANSE and ARPACK can be compared directly, using correlation. The correlation between the respective singular values was strong to seven decimal places, $r = 0.99999998586079$, slightly lower than expected. Upon further inspection, we realized that singular value 294 of LANSE had not converged, i.e. it was the sole missing eigenvalue relative to ARPACK. By inserting a similar hole in ARPACK's singular values (which correctly realigns singular values 295-300), the correlation is stronger to nine decimal places, $r = 0.99999999917588$. The implications of a such a small missing singular value are relatively minor, according to Equation 2. By the same argument, the high correlation between the singular values of LANSE and ARPACK indicate that LANSE found the correct singular values to an extremely high degree of precision.

Our second analysis considers the left singular vectors of LANSE and ARPACK. There are multiple ways of considering this evaluation. One approach would be to randomly select a large set of word pairs, say a thousand, compute the cosines between them in each space, and compare the cosines. However it is possible to obtain a more global, mathematically sound result. Because the SVD of a matrix has a unique solution, the left singular vectors from a LANSE SVD and an ARPACK SVD of the same matrix should be equivalent (excluding sign) in column space. Therefore the dot product between the corresponding column vectors of the spaces should be 1, just as the LSA cosine between a word and itself is 1. However, this dot product comparison methodology is based on exact arithmetic. Therefore we calculate two sets of dot products. The first set contains a dot product between each ARPACK vector and itself. This represents the upper bound of agreement that could be expected between ARPACK and LANSE in finite precision. The second set contains the dot products between the nth vector of ARPACK and the nth vector of LANSE. By comparing the two sets of dot products one can obtain a measure of similarity across all possible combinations of terms in the respective spaces. The maximum difference between these two sets was .0000057, and the average difference was .000000031. The relative closeness of the left singular vectors indicates that LANSE has also found the left singular vectors with high precision.

Aside from correctness of results, it is worthwhile to consider the run-time characteristics of LANSE as opposed to other algorithms. As with any algorithm, the two most important run-time factors are speed and memory consumption. As previously discussed, most reorthogonalization approaches attempt to maximize speed. LANSE minimizes memory consumption in order to make large scale LSA possible. Thus in some respects, this is a non-informative comparison because LANSE allows large scale LSA spaces that simply would not be possible with typical reorthogonalization approaches. However, the comparison does serve to illustrate the how LANSE performs. The following comparisons took place on an Intel Duo 1.66Mhz computer with a 7200rpm hard disk.

When calculating the TASA space, ARPACK took approximately 1.7 hours to complete, and used approximately 1,183 MB of RAM. The memory consumption of ARPACK falls within its designed storage bounds given by $(m+n)k+k^2$ and $2(m+n)k+k^2$ (Lehoucq et al., 1998), 774 MB and 1,509 MB. However, since ARPACK was called through the MATLAB clone Octave, a closer analysis was made to uncover any potential inefficiencies incurred by Octave. The profiling software Valgrind[2] and a manual inspection of the Octave source revealed that approximately 210 MB of RAM were used in Octave rather than ARPACK, bringing ARPACK's true memory consumption down to 1,002 MB.

Contrastingly, LANSE took approximately 10.1 hours to complete using approximately 444 MB of RAM, approximately half the RAM. The difference in RAM consumption between the two approaches is explainable as the absence of Q_1 in memory required to reorthogonalize during tridiagonalization. Because the ARPACK implementation is using B to represent the matrix instead of AA^T, the theoretical estimation of Q_1's size is 771 MB. Since LANSE used only about 3 MB to store the last three vectors of Q_1, LANSE is much more memory efficient than the ARPACK implementation.

This memory comparison between ARPACK and LANSE is favorable, but it does not fully indicate the differences at scale. Recall that while reorthogonalization approaches require 300 such vectors for 300 dimensions, LANSE requires only 3, a scaling factor of 100. Note that these differences are for storage of the Q_1 vectors alone. However, for small collections, storage of A, T, and Q_2 dominate, but Q_1 quickly dominates for larger collections. Consider the 300 dimension LSA space mentioned previously made from the 9 million trigrams, bigrams, and unigrams in the TASA corpus. An reorthogonalization approach would require 20 gigabytes of 8 byte values just to store Q_1. Contrastingly, the minimal storage for A would be around 70 megabytes with negligible storage required for T and Q_2. The original matrix A is so small relative to Q_1 because A is extremely sparse: only a fraction of all words occur in each document. As the size of the corpus grows, this sparsity is largely maintained. Q_1, on the other hand, is dense, i.e. completely filled with non-zero values. As the size of the corpus grows, Q_1 grows proportionally. In this case, that proportional growth is 100 times greater for reorthogonalization approaches than for LANSE.

The speed comparison between ARPACK and LANSE is less favorable. In the case of TASA, LANSE takes 6 times as long to complete. However, the story is more nuanced than this. The two most time consuming operations for LANSE are tridiagonalizing and calculating Q_1Q_2. Tridiagonalizing TASA takes only 25 minutes, and the eigenvalues converge readily, as shown in Figure 1.

The second time intensive step, calculating Q_1Q_2 takes approximately 9.5 hours to compute. The principle reason is that the Q_1 vectors were stored on disk during tridiagonalization – this is where the memory savings of LANSE over reorthogonalization approaches is realized. However, since disk operations are roughly a million times slower than CPU operations, reading Q_1 back off disk is a time-consuming process. However, it is possible to do so in a single pass, i.e. by reading each vector of Q_1 only once. Alternatively, it is possible to regenerate each vector of Q_1 when needed by repeating the tridiagonalization process. Whether regenerating vectors is more time-efficient than reading them from disk depends on the speed of the processor being used. When computing the same TASA space on the machine mentioned previously, regeneration took 11.2 hours to complete, almost 2 hours longer than the disk condition. However, when implemented on an i7 processor at 1.6 GHZ, the same computation completed in 8 hours, a 1.5 hour improvement. Therefore the preferred method of computing Q_1Q_2 is situation dependent, and LANSE provides both options.

In summary, the comparison of LANSE with ARPACK on the TASA corpus is favorable. LANSE achieves comparable accuracy to ARPACK for both singular values and singular vectors. LANSE also performs its computations with about half the memory consumption of ARPACK, and this difference in memory consumption would be even greater for larger spaces. Finally, LANSE executes about six times more slowly than ARPACK, but this is by design, since LANSE trades speed for low memory consumption.

Conclusion

Latent semantic analysis has been widely used in multiple research communities to investigate many types of phenomena, ranging from vocabulary acquisition to information retrieval (Landauer & Dumais, 1997; Dumais, 1991). However, the questions that can be asked are inherently limited by the computing resources available to researchers. We have presented LANSE as a solution to this problem. LANSE computes the singular value decomposition of LSA spaces in roughly 1/100th of the memory required by traditional reorthogonalization approaches like LANSO (Parlett, 1998). LANSE achieves this goal by combining previous work on the Lanczos algorithm without reorthogonalization (Cullum & Willoughby, 1985/2002; Kats & Vorst, 1976, 1977; Paige, 1972) together with empirical work based on LSA (Berry, 1992; Ding, 2005). By attempting to solve SVD for LSA, rather than SVD in general, LANSE offers a relatively simple means of implementing large scale latent semantic analysis.

The supercomputing community has been exploring algorithms for computing large matrix decompositions over the past several decades, focusing on parallel schemes using shared memory and distributed memory (Berry, 1992; Berry, Mezher, Philippe, & Sameh, 2006; Berry & Martin, 2006; Maschhoff & Sorensen, 1996). In shared memory schemes, multiple processors have read/write access to a large pool of common memory, e.g. 64GB. Alternatively, in distributed memory schemes, the problem is broken up into many similar parts, distributed over a networked cluster of machines, and then reassembled when the parts have been solved. In both cases, the algorithms require large amounts of memory for reorthogonalization but offer a greater speed of computation.

Although not currently parallelized, LANSE is highly complimentary to these mainstream approaches. The two most time consuming operations for LANSE are tridiagonalizing and calculating $Q_1 Q_2$, both of which are essentially vector matrix multiplications that are highly parallelizable on multi-core desktops (Bai et al., 2000;

Berry et al., 2006). Likewise, the lower overhead of LANSE, by not requiring reorthogonalization, could in principle streamline existing parallel algorithms that operate over networks of workstations (Berry & Martin, 2006).

With respect to current LSA research, the LANSE algorithm has both practical and theoretical implications. Practically, LANSE allows researchers without access to supercomputers to create supercomputer-sized LSA spaces. Even those with access to large shared memory supercomputers will be able to create larger LSA spaces than was previously possible. Theoretically, LANSE opens a door to new potential applications of LSA in psychological and semantic research. Previously, limits on input matrix size have restricted the kinds of phenomena that could be investigated. By mitigating these size restrictions, LANSE allows new questions to be asked, such as the effect of n-grams, syntactic dependencies, and sub-sentence documents on LSA spaces. By allowing larger questions to be asked, LANSE increases the potential scope of LSA research. [3]

References

Bai, Z., Demmel, J., Dongarra, J., Ruhe, A., & Van Der Vorst, H. (Eds.). (2000). *Templates for the solution of algebraic eigenvalue problems.* Philadelphia: Society for Industrial and Applied Mathematics.

Bellegarda, J. (2000). Exploiting latent semantic information in statistical language modeling. *Proceedings of the IEEE*, *88*(8), 1279-1296.

Bergamaschi, L., & Putti, M. (2002, October). Numerical comparison of iterative eigensolvers for large sparse symmetric positive definite matrices. *Computer Methods in Applied Mechanics and Engineering*, *191*(45), 5233–5247. Available from http://www.sciencedirect.com/science/article/B6V29-470M6HS-3/2/f970c176e37d02920ae5fda2ff32d9cb

Berry, M. W. (1992). Large scale singular value computations. *International Journal of Supercomputer Applications*, *6*(1), 13-49.

Berry, M. W., & Martin, D. (2006). Principal component analysis for information retrieval. In *Handbook of parallel computing and statistics.* Chapman and Hall/CRC. Available from http://dx.doi.org/10.1201/9781420028683.ch13

Berry, M. W., Mezher, D., Philippe, B., & Sameh, A. (2006). Parallel algorithms for the singular value decomposition. In *Handbook of parallel computing and statistics.* Chapman and Hall/CRC. Available from http://dx.doi.org/10.1201/9781420028683.ch4

Coccaro, N., & Jurafsky, D. (1998, 30th November-4th December). Towards better integration of semantic predictors in statistical language modeling. In *Proceedings of the international conference on spoken language processing* (p. 2403-2406). Piscataway, NJ: IEEE. Available from citeseer.ist.psu.edu/article/coccaro98towards.html

Cullum, J. K., & Willoughby, R. A. (2002). *Lanczos algorithms for large symmetric* *eigenvalue computations, volume 1: Theory*. Philadelphia: Society for Industrial and Applied Mathematics. (Original work published 1985)

Deerwester, S. C., Dumais, S. T., Furnas, G. W., Landauer, T. K., & Harshman, R. A. (1990). Indexing by latent semantic analysis. *Journal of the American Society of Information Science*, *41*(6), 391-407. Available from `citeseer.ist.psu.edu/deerwester90indexing.html`

Demmel, J. W. (1997). *Applied numerical linear algebra*. Philadelphia: Society for Industrial and Applied Mathematics.

Deng, Y., & Khudanpur, S. (2003). Latent semantic information in maximum entropy language models for conversational speech recognition. In *Proceedings of Human Language Technology Conference of the North American Chapter of the Association for Computational Linguistics* (p. 56-63). Philadelphia: Association for Computational Linguistics.

Ding, C. H. Q. (2005). A probabilistic model for latent semantic indexing: Research articles. *Journal of the American Society for Information Science and Technology*, *56*(6), 597–608.

Dumais, S. (1991). Improving the retrieval of information from external sources. *Behavior Research Methods, Instruments and Computers*, *23*(2), 229-236.

Eckart, C., & Young, G. (1936). The approximation of one matrix by another of lower rank. *Psychometrika*, *1*(3), 211-218.

Foltz, P. W., Kintsch, W., & Landauer, T. K. (1998). The measurement of textual coherence with latent semantic analysis. *Discourse Processes*, *25*(2&3), 285-308.

Graesser, A. C., VanLehn, K., Rose, C., Jordan, P., & Harter, D. (2001). Intelligent tutoring systems with conversational dialogue. *AI Magazine*, *22*, 39-51.

Kats, J. M. van, & Vorst, H. A. van der. (1976). *Numerical results of the Paige-style Lanczos method for the computation of extreme eigenvalues of large sparse matrices*

(Vol. 7; Academisch Computer Centrum No. TR-3). University of Utrecht.

Kats, J. M. van, & Vorst, H. A. van der. (1977). *Automatic monitoring of Lanczos schemes for symmetric or skew-symmetric generalized eigenvalue problems* (Vol. 7; Academisch Computer Centrum No. TR-7). University of Utrecht.

Landauer, T. K., & Dumais, S. T. (1997). A solution to plato's problem: The latent semantic analysis theory of the acquisition, induction, and representation of knowledge. *Psychological Review*, *104*, 211-240.

Landauer, T. K., Foltz, P. W., & Laham, D. (1998). Introduction to latent semantic analysis. *Discourse Processes*, *25*(2&3), 259-284.

Landauer, T. K., McNamara, D. S., Dennis, S., & Kintsch, W. (Eds.). (2007). *Handbook of latent semantic analysis.* Mahwah, New Jersey: Lawrence Erlbaum.

Lehoucq, R., Sorensen, D., & Yang, C. (1998). *ARPACK users' guide: solution of large-scale eigenvalue problems with implicitly restarted Arnoldi methods.* Philadelphia: Society for Industrial & Applied Mathematics.

Martin, D. I., & Berry, M. W. (2007). Mathematical foundations behind latent semantic analysis. In T. K. Landauer, D. S. McNamara, S. Dennis, & W. Kintsch (Eds.), *Handbook of latent semantic analysis* (pp. 35–55). Mahwah, New Jersey: Lawrence Earlbaum.

Maschhoff, K. J., & Sorensen, D. C. (1996, October). *PARPACK: An efficient portable large scale eigenvalue package for distributed memory parallel architectures* (Tech. Rep. No. CRPC-TR96659). Houston, TX: Rice University.

Olney, A. M. (2007a). Dialogue generation for robotic portraits. In *Proceedings of the International Joint Conference on Artificial Intelligence 5th workshop on Knowledge and Reasoning in Practical Dialogue Systems* (p. 15-21). Hyderabad, India.

Olney, A. M. (2007b). Latent semantic grammar induction: Context, projectivity, and prior distributions. In *Proceedings of the Second Workshop on TextGraphs:*

Graph-Based Algorithms for Natural Language Processing (pp. 45–52). Rochester, NY, USA: Association for Computational Linguistics. Available from http://www.aclweb.org/anthology/W/W07/W07-0207

Olney, A. M. (2009). Generalizing latent semantic analysis. In *IEEE International Conference on Semantic Computing* (p. 40-46). Los Alamitos, CA, USA: IEEE Computer Society.

Olney, A. M., & Cai, Z. (2005a). An orthonormal basis for entailment. In *Proceedings of the Eighteenth International Florida Artificial Intelligence Research Society Conference* (p. 554-559). Menlo Park, CA: AAAI Press.

Olney, A. M., & Cai, Z. (2005b). An orthonormal basis for topic segmentation in tutorial dialogue. In *Proceedings of the Human Language Technology Conference and Conference on Empirical Methods in Natural Language Processing* (p. 971-978). Philadelphia: Association for Computational Linguistics.

Paige, C. C. (1971). *The computation of eigenvalues and eigenvectors of very large sparse matrices.* Unpublished doctoral dissertation, University of London.

Paige, C. C. (1972). Computational variants of the Lanczos method for the eigenproblem. *IMA Journal of Applied Mathematics*, *10*(3), 373-381.

Parlett, B. (1998). *The symmetric eigenvalue problem.* Philadelphia: Society for Industrial Mathematics.

Rehder, B., Schreiner, M., Laham, D., Wolfe, M., Landauer, T., & Kintsch, W. (1998). Using Latent Semantic Analysis to assess knowledge: Some technical considerations. *Discourse Processes*, *25*, 337-354.

Schütze, H. (1995, March 27-31). Distributional part-of-speech tagging. In *Proceedings of the 7th European Association for Computational Linguistics Conference (EACL-95)* (p. 141-149). Philadelphia: Association for Computational Linguistics. Available from http://arxiv.org/abs/cmp-lg/9503009

Stewart, G. (2001). *Matrix Algorithms: Eigensystems*. Philadelphia: Society for Industrial and Applied Mathematics.

Trefethen, L. N., & Bau, D., II. (1997). *Numerical linear algebra*. Philadelphia: Soceity for Industrial and Applied Mathematics.

Willits, J., D'Mello, S., Duran, N., & Olney, A. (2007). Distributional statistics and thematic role relationships. In D. S. McNamara & J. G. Trafton (Eds.), *Proceedings of the 29th annual conference of the Cognitive Science Society* (p. 707-712). Austin, TX.: Cognitive Science Society.

Wolfe, M., Schreiner, M. E., Rehder, B., Laham, D., Foltz, P. W., Kintsch, W., et al. (1998). Learning from text: Matching readers and texts by Latent Semantic Analysis. *Discourse Processes*, *25*, 309-336.

Author Note

The research reported here was supported by the Institute of Education Sciences, U.S. Department of Education, through Grant R305A080594 and by the National Science Foundation, through Grant BCS-0826825, to the University of Memphis. The opinions expressed are those of the authors and do not represent views of the Institute or the U.S. Department of Education or the National Science Foundation.

Footnotes

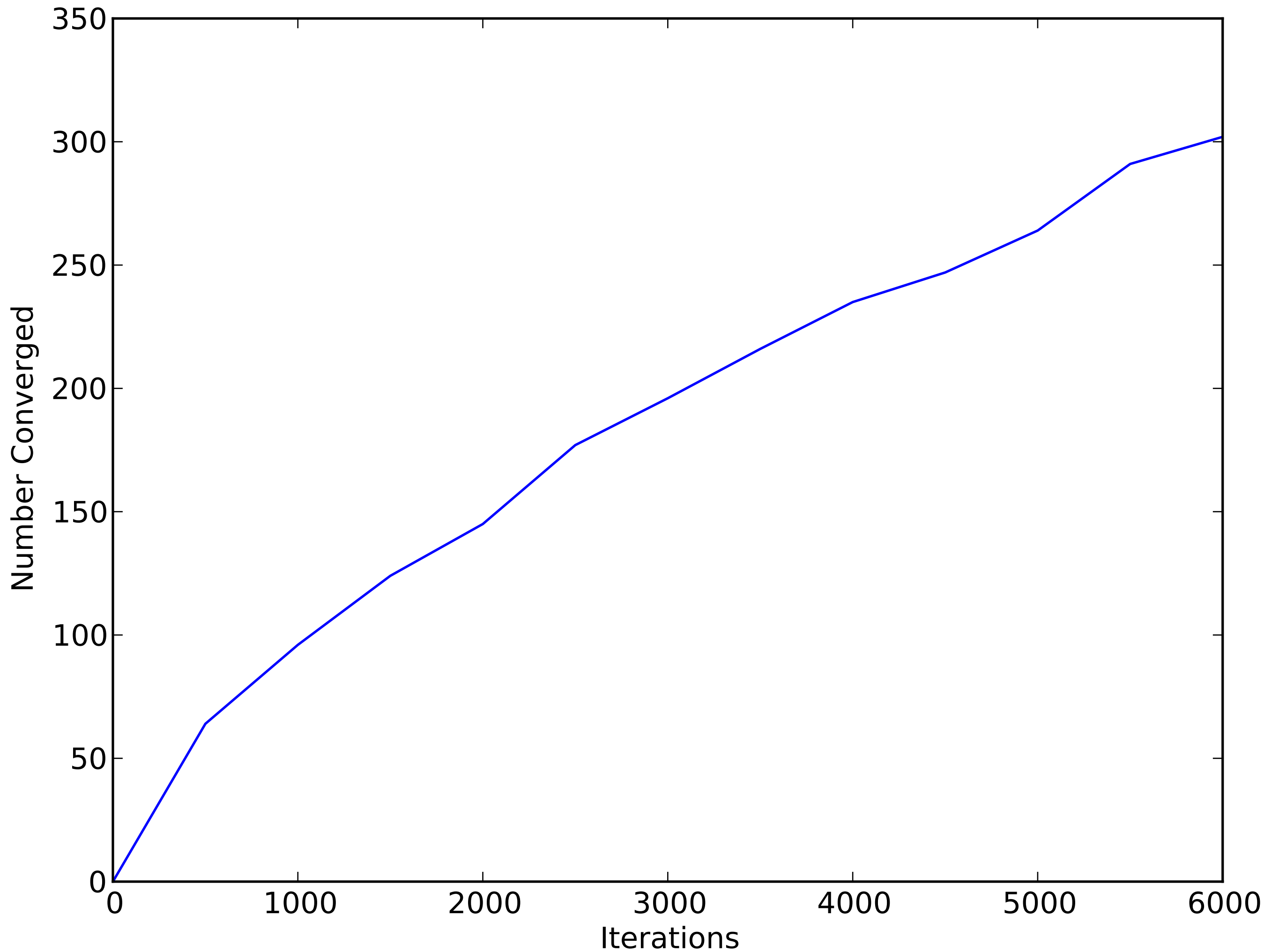
[1]A direct comparison to the Cullum and Willoughby code was impractical because it is incomplete, unmaintained, and written in legacy Fortan. Attempts to contact the authors have been unsuccessful.

[2]http://valgrind.org/

[3]Source code for the LANSE algorithm can be found at http://andrewmolney.name

Figure Captions

Figure 1. Convergence of eigenvalues in LANSE.


Number Converged
Iterations
0
50
100
150
200
250
300
350
0
1000
2000
3000
4000
5000
6000