Generalized Hebbian Algorithm for Incremental Singular Value Decomposition in Natural Language Processing

Genevieve Gorrell
Department of Computer and Information Science
Linköping University
581 83 LINKÖPING
Sweden
gengo@ida.liu.se

Abstract

An algorithm based on the Generalized Hebbian Algorithm is described that allows the singular value decomposition of a dataset to be learned based on single observation pairs presented serially. The algorithm has minimal memory requirements, and is therefore interesting in the natural language domain, where very large datasets are often used, and datasets quickly become intractable. The technique is demonstrated on the task of learning word and letter bigram pairs from text.

1 Introduction

Dimensionality reduction techniques are of great relevance within the field of natural language processing. A persistent problem within language processing is the over-specificity of language, and the sparsity of data. Corpus-based techniques depend on a sufficiency of examples in order to model human language use, but the Zipfian nature of frequency behaviour in language means that this approach has diminishing returns with corpus size. In short, there are a large number of ways to say the same thing, and no matter how large your corpus is, you will never cover all the things that might reasonably be said. Language is often too rich for the task being performed; for example it can be difficult to establish that two documents are discussing the same topic. Likewise no matter how much data your system has seen during training, it will invariably see something new at run-time in a domain of any complexity. Any approach to automatic natural language processing will encounter this problem on several levels, creating a need for techniques which compensate for this.

Imagine we have a set of data stored as a matrix. Techniques based on eigen decomposition allow such a matrix to be transformed into a set of orthogonal vectors, each with an associated "strength", or eigenvalue. This transformation allows the data contained in the matrix to be compressed; by discarding the less significant vectors (dimensions) the matrix can be approximated with fewer numbers. This is what is meant by dimensionality reduction. The technique is guaranteed to return the closest (least squared error) approximation possible for a given number of numbers (Golub and Reinsch, 1970). In certain domains, however, the technique has even greater significance. It is effectively forcing the data through a bottleneck; requiring it to describe itself using an impoverished construct set. This can allow the critical underlying features to reveal themselves. In language, for example, these features might be semantic constructs. It can also improve the data, in the case that the detail is noise, or richness not relevant to the task.

Singular value decomposition (SVD) is a near relative of eigen decomposition, appropriate to domains where input is asymmetrical. The best known application of singular value decomposition within natural language processing is Latent Semantic Analysis (Deerwester et al., 1990). Latent Semantic Analysis (LSA) allows passages of text to be compared to each other in a reduced-dimensionality semantic space, based on the words they contain.

The technique has been successfully applied to information retrieval, where the overspecificity of language is particularly problematic; text searches often miss relevant documents where different vocabulary has been chosen in the search terms to that used in the document (for example, the user searches on "eigen decomposition" and fails to retrieve documents on factor analysis). LSA has also been applied in language modelling (Bellegarda, 2000), where it has been used to incorporate long-span semantic dependencies.

Much research has been done on optimising eigen decomposition algorithms, and the extent to which they can be optimised depends on the area of application. Most natural language problems involve sparse matrices, since there are many words in a natural language and the great majority do not appear in, for example, any one document. Domains in which matrices are less sparse lend themselves to such techniques as Golub-Kahan-Reinsch (Golub and Reinsch, 1970) and Jacobi-like approaches. Techniques such as those described in (Berry, 1992) are more appropriate in the natural language domain.

Optimisation is an important way to increase the applicability of eigen and singular value decomposition. Designing algorithms that accommodate different requirements is another. For example, another drawback to Jacobi-like approaches is that they calculate all the singular triplets (singular vector pairs with associated values) simultaneously, which may not be practical in a situation where only the top few are required. Consider also that the methods mentioned so far assume that the entire matrix is available from the start. There are many situations in which data may continue to become available.

(Berry et al., 1995) describe a number of techniques for including new data in an existing decomposition. Their techniques apply to a situation in which SVD has been performed on a collection of data, then new data becomes available. However, these techniques are either expensive, or else they are approximations which degrade in quality over time. They are useful in the context of updating an existing batch decomposition with a second batch of data, but are less applicable in the case where data are presented serially, for example, in the context of a learning system. Furthermore, there are limits to the size of matrix that can feasibly be processed using batch decomposition techniques. This is especially relevant within natural language processing, where very large corpora are common. Random Indexing (Kanerva et al., 2000) provides a less principled, though very simple and efficient, alternative to SVD for dimensionality reduction over large corpora.

This paper describes an approach to singular value decomposition based on the Generalized Hebbian Algorithm (Sanger, 1989). GHA calculates the eigen decomposition of a matrix based on single observations presented serially. The algorithm presented here differs in that where GHA produces the eigen decomposition of symmetrical data, our algorithm produces the singular value decomposition of asymmetrical data. It allows singular vectors to be learned from paired inputs presented serially using no more memory than is required to store the singular vector pairs themselves. It is therefore relevant in situations where the size of the dataset makes conventional batch approaches infeasible. It is also of interest in the context of adaptivity, since it has the potential to adapt to changing input. The learning update operation is very cheap computationally. Assuming a stable vector length, each update operation takes exactly as long as each previous one; there is no increase with corpus size to the speed of the update. Matrix dimensions may increase during processing. The algorithm produces singular vector pairs one at a time, starting with the most significant, which means that useful data becomes available quickly; many standard techniques produce the entire decomposition simultaneously. Since it is a learning technique, however, it differs from what would normally be considered an incremental technique, in that the algorithm converges on the singular value decomposition of the dataset, rather than at any one point having the best solution possible for the data it has seen so far. The method is potentially most appropriate in situations where the dataset is very large or unbounded: smaller, bounded datasets may be more efficiently processed by other methods. Furthermore, our

approach is limited to cases where the final matrix is expressible as the linear sum of outer products of the data vectors. Note in particular that Latent Semantic Analysis, as usually implemented, is not an example of this, because LSA takes the log of the final sums in each cell (Dumais, 1990). LSA, however, does not depend on singular value decomposition; Gorrell and Webb (Gorrell and Webb, 2005) discuss using eigen decomposition to perform LSA, and demonstrate LSA using the Generalized Hebbian Algorithm in its unmodified form. Sanger (Sanger, 1993) presents similar work, and future work will involve more detailed comparison of this approach to his.

The next section describes the algorithm. Section 3 describes implementation in practical terms. Section 4 illustrates, using word n-gram and letter n-gram tasks as examples and section 5 concludes.

2 The Algorithm

This section introduces the Generalized Hebbian Algorithm, and shows how the technique can be adapted to the rectangular matrix form of singular value decomposition. Eigen decomposition requires as input a square diagonally-symmetrical matrix, that is to say, one in which the cell value at row x, column y is the same as that at row y, column x. The kind of data described by such a matrix is the correlation between data in a particular space with other data in the same space. For example, we might wish to describe how often a particular word appears with a particular other word. The data therefore are symmetrical relations between items in the same space; word a appears with word b exactly as often as word b appears with word a. In singular value decomposition, rectangular input matrices are handled. Ordered word bigrams are an example of this; imagine a matrix in which rows correspond to the first word in a bigram, and columns to the second. The number of times that word b appears after word a is by no means the same as the number of times that word a appears after word b. Rows and columns are different spaces; rows are the space of first words in the bigrams, and columns are the space of second words.

The singular value decomposition of a rectangular data matrix, A, can be presented as;

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

where U and V are matrices of orthogonal left and right singular vectors (columns) respectively, and Σ is a diagonal matrix of the corresponding singular values. The U and V matrices can be seen as a matched set of orthogonal basis vectors in their corresponding spaces, while the singular values specify the effective magnitude of each vector pair. By convention, these matrices are sorted such that the diagonal of Σ is monotonically decreasing, and it is a property of SVD that preserving only the first (largest) N of these (and hence also only the first N columns of U and V) provides a least-squared error, rank-N approximation to the original matrix A.

Singular Value Decomposition is intimately related to eigenvalue decomposition in that the singular vectors, U and V, of the data matrix, A, are simply the eigenvectors of $A * A^T$ and $A^T * A$, respectively, and the singular values, Σ, are the square-roots of the corresponding eigenvalues.

2.1 Generalised Hebbian Algorithm

Oja and Karhunen (Oja and Karhunen, 1985) demonstrated an incremental solution to finding the first eigenvector from data arriving in the form of serial data items presented as vectors, and Sanger (Sanger, 1989) later generalized this to finding the first N eigenvectors with the Generalized Hebbian Algorithm. The algorithm converges on the exact eigen decomposition of the data with a probability of one. The essence of these algorithms is a simple Hebbian learning rule:

$$U_n(t+1) = U_n(t) + \lambda * (U_n^T * A_j) * A_j \tag{2}$$

U_n is the n'th column of U (i.e., the n'th eigenvector, see equation 1), λ is the learning rate and A_j is the j'th column of training matrix A. t is the timestep. The only modification to this required in order to extend it to multiple eigenvectors is that each U_n needs to shadow any lower-ranked $U_m(m > n)$ by removing its projection from the input A_j in order to assure both orthogonality and an ordered ranking of

the resulting eigenvectors. Sanger's final formulation (Sanger, 1989) is:

$$c_{ij}(t+1) = c_{ij}(t) + \gamma(t)(y_i(t)x_j(t) - y_i(t)\sum_{k\leq i} c_{kj}(t)y_k(t)) \quad (3)$$

In the above, c_{ij} is an individual element in the current eigenvector, x_j is the input vector and y_i is the activation (that is to say, $c_i.x_j$, the dot product of the input vector with the ith eigenvector). γ is the learning rate.

To summarise, the formula updates the current eigenvector by adding to it the input vector multiplied by the activation minus the projection of the input vector on all the eigenvectors so far *including the current eigenvector*, multiplied by the activation. Including the current eigenvector in the projection subtraction step has the effect of keeping the eigenvectors normalised. Note that Sanger includes an explicit learning rate, γ. The formula can be varied slightly by not including the current eigenvector in the projection subtraction step. In the absence of the autonormalisation influence, the vector is allowed to grow long. This has the effect of introducing an implicit learning rate, since the vector only begins to grow long when it settles in the right direction, and since further learning has less impact once the vector has become long. Weng et al. (Weng et al., 2003) demonstrate the efficacy of this approach. So, in vector form, assuming C to be the eigenvector currently being trained, expanding y out and using the implicit learning rate;

$$\triangle c_i = c_i.x(x - \sum_{j<i}(x.c_j)c_j) \quad (4)$$

Delta notation is used to describe the update here, for further readability. The subtracted element is responsible for removing from the training update any projection on previous singular vectors, thereby ensuring orthgonality. Let us assume for the moment that we are calculating only the first eigenvector. The training update, that is, the vector to be added to the eigenvector, can then be more simply described as follows, making the next steps more readable;

$$\triangle c = c.x(x) \quad (5)$$

2.2 Extension to Paired Data

Let us begin with a simplification of 5:

$$\triangle c = \frac{1}{n} cX(X) \quad (6)$$

Here, the upper case X is the entire data matrix. n is the number of training items. The simplification is valid in the case that c is stabilised; a simplification that in our case will become more valid with time. Extension to paired data initially appears to present a problem. As mentioned earlier, the singular vectors of a rectangular matrix are the eigenvectors of the matrix multiplied by its transpose, and the eigenvectors of the transpose of the matrix multiplied by itself. Running GHA on a non-square non-symmetrical matrix M, ie. paired data, would therefore be achievable using standard GHA as follows:

$$\triangle c^a = \frac{1}{n} c^a MM^T(MM^T) \quad (7)$$

$$\triangle c^b = \frac{1}{n} c^b M^T M(M^T M) \quad (8)$$

In the above, c^a and c^b are left and right singular vectors. However, to be able to feed the algorithm with rows of the matrices MM^T and M^TM, we would need to have the entire training corpus available simultaneously, and square it, which we hoped to avoid. This makes it impossible to use GHA for singular value decomposition of serially-presented paired input in this way without some further transformation. Equation 1, however, gives:

$$\sigma c^a = c^b M^T = \sum_x (c^b.b_x)a_x \quad (9)$$

$$\sigma c^b = c^a M = \sum_x (c^a.a_x)b_x \quad (10)$$

Here, σ is the singular value and a and b are left and right data vectors. The above is valid in the case that left and right singular vectors c^a and c^b have settled (which will become more accurate over time) and that data vectors a and b outer-product and sum to M.

Inserting 9 and 10 into 7 and 8 allows them to be reduced as follows:

$$\triangle c^a = \frac{\sigma}{n} c^b M^T M M^T \quad (11)$$

$$\triangle c^b = \frac{\sigma}{n} c^a M M^T M \quad (12)$$

$$\triangle c^a = \frac{\sigma^2}{n} c^a M M^T \quad (13)$$

$$\triangle c^b = \frac{\sigma^2}{n} c^b M^T M \quad (14)$$

$$\triangle c^a = \frac{\sigma^3}{n} c^b M^T \quad (15)$$

$$\triangle c^b = \frac{\sigma^3}{n} c^a M \quad (16)$$

$$\triangle c^a = \sigma^3 (c_b.b)a \quad (17)$$

$$\triangle c^b = \sigma^3 (c_a.a)b \quad (18)$$

This element can then be reinserted into GHA. To summarise, where GHA dotted the input with the eigenvector and multiplied the result by the input vector to form the training update (thereby adding the input vector to the eigenvector with a length proportional to the extent to which it reflects the current direction of the eigenvector) our formulation dots the right input vector with the right singular vector and multiplies the left input vector by this quantity before adding it to the left singular vector, and vice versa. In this way, the two sides cross-train each other. Below is the final modification of GHA extended to cover multiple vector pairs. The original GHA is given beneath it for comparison.

$$\triangle c_i^a = c_i^b.b(a - \sum_{j<i} (a.c_j^a)c_j^a) \quad (19)$$

$$\triangle c_i^b = c_i^a.a(b - \sum_{j<i} (b.c_j^b)c_j^b) \quad (20)$$

$$\triangle c_i = c_i.x(x - \sum_{j<i} (x.c_j)c_j) \quad (21)$$

In equations 6 and 9/10 we introduced approximations that become accurate as the direction of the singular vectors settles. These approximations will therefore not interfere with the accuracy of the final result, though they might interfere with the rate of convergence. The constant σ^3 has been dropped in 19 and 20. Its relevance is purely with respect to the calculation of the singular value. Recall that in (Weng et al., 2003) the eigenvalue is calculable as the average magnitude of the training update $\triangle c$. In our formulation, according to 17 and 18, the singular value would be $\triangle c$ divided by σ^3. Dropping the σ^3 in 19 and 20 achieves that implicitly; the singular value is once more the average length of the training update.

The next section discusses practical aspects of implementation. The following section illustrates usage, with English language word and letter bigram data as test domains.

3 Implementation

Within the framework of the algorithm outlined above, there is still room for some implementation decisions to be made. The naive implementation can be summarised as follows: the first datum is used to train the first singular vector pair; the projection of the first singular vector pair onto this datum is subtracted from the datum; the datum is then used to train the second singular vector pair and so on for all the vector pairs; ensuing data items are processed similarly. The main problem with this approach is as follows. At the beginning of the training process, the singular vectors are close to the values they were initialised with, and far away from the values they will settle on. The second singular vector pair is trained on the datum minus its projection onto the first singular vector pair in order to prevent the second singular vector pair from becoming the same as the first. But if the first pair is far away from its eventual direction, then the second has a chance to move in the direction that the first will eventually take on. In fact, all the vectors, such as they can whilst remaining orthogonal to each other, will move in the strongest direction. Then, when the first pair eventually takes on the right direction, the others have difficulty recovering, since they start to receive data that they have very little projection on, meaning that they learn very

slowly. The problem can be addressed by waiting until each singular vector pair is relatively stable before beginning to train the next. By "stable", we mean that the vector is changing little in its direction, such as to suggest it is very close to its target. Measures of stability might include the average variation in position of the endpoint of the (normalised) vector over a number of training iterations, or simply length of the (unnormalised) vector, since a long vector is one that is being reinforced by the training data, such as it would be if it was settled on the dominant feature. Termination criteria might include that a target number of singular vector pairs have been reached, or that the last vector is increasing in length only very slowly.

4 Application

The task of relating linguistic bigrams to each other, as mentioned earlier, is an example of a task appropriate to singular value decomposition, in that the data is paired data, in which each item is in a different space to the other. Consider word bigrams, for example. First word space is in a non-symmetrical relationship to second word space; indeed, the spaces are not even necessarily of the same dimensionality, since there could conceivably be words in the corpus that never appear in the first word slot (they might never appear at the start of a sentence) or in the second word slot (they might never appear at the end.) So a matrix containing word counts, in which each unique first word forms a row and each unique second word forms a column, will not be a square symmetrical matrix; the value at row a, column b, will not be the same as the value at row b column a, except by coincidence.

The significance of performing dimensionality reduction on word bigrams could be thought of as follows. Language clearly adheres to some extent to a rule system less rich than the individual instances that form its surface manifestation. Those rules govern which words might follow which other words; although the rule system is more complex and of a longer range that word bigrams can hope to illustrate, nonetheless the rule system governs the surface form of word bigrams, and we might hope that it would be possible to discern from word bigrams something of the nature of the rules. In performing dimensionality reduction on word bigram data, we force the rules to describe themselves through a more impoverished form than via the collection of instances that form the training corpus. The hope is that the resulting simplified description will be a generalisable system that applies even to instances not encountered at training time.

On a practical level, the outcome has applications in automatic language acquisition. For example, the result might be applicable in language modelling. Use of the learning algorithm presented in this paper is appropriate given the very large dimensions of any realistic corpus of language; The corpus chosen for this demonstration is Margaret Mitchell's "Gone with the Wind", which contains 19,296 unique words (421,373 in total), which fully realized as a correlation matrix with, for example, 4-byte floats would consume 1.5 gigabytes, and which in any case, within natural language processing, would not be considered a particularly large corpus. Results on the word bigram task are presented in the next section.

Letter bigrams provide a useful contrasting illustration in this context; an input dimensionality of 26 allows the result to be more easily visualised. Practical applications might include automatic handwriting recognition, where an estimate of the likelihood of a particular letter following another would be useful information. The fact that there are only twenty-something letters in most western alphabets though makes the usefulness of the incremental approach, and indeed, dimensionality reduction techniques in general, less obvious in this domain. However, extending the space to letter trigrams and even four-grams would change the requirements. Section 4.2 discusses results on a letter bigram task.

4.1 Word Bigram Task

"Gone with the Wind" was presented to the algorithm as word bigrams. Each word was mapped to a vector containing all zeros but for a one in the slot corresponding to the unique word index assigned to that word. This had the effect of making input to the algorithm a normalised vector, and of making word vectors orthogonal to each other. The singular vector pair's reaching a combined Euclidean

magnitude of 2000 was given as the criterion for beginning to train the next vector pair, the reasoning being that since the singular vectors only start to grow long when they settle in the approximate right direction and the data starts to reinforce them, length forms a reasonable heuristic for deciding if they are settled enough to begin training the next vector pair. 2000 was chosen *ad hoc* based on observation of the behaviour of the algorithm during training.

The data presented are the words most representative of the top two singular vectors, that is to say, the directions these singular vectors mostly point in. Table 1 shows the words with highest scores in the top two vector pairs. It says that in this vector pair, the normalised left hand vector projected by 0.513 onto the vector for the word "of" (or in other words, these vectors have a dot product of 0.513.) The normalised right hand vector has a projection of 0.876 onto the word "the" etc. This first table shows a left side dominated by prepositions, with a right side in which "the" is by far the most important word, but which also contains many pronouns. The fact that the first singular vector pair is effectively about "the" (the right hand side points far more in the direction of "the" than any other word) reflects its status as the most common word in the English language. What this result is saying is that were we to be allowed only one feature with which to describe word English bigrams, a feature describing words appearing before "the" and words behaving similarly to "the" would be the best we could choose. Other very common words in English are also prominent in this feature.

Table 1: Top words in 1st singular vector pair

Vector 1, Eigenvalue 0.00938			
of	0.5125468	the	0.8755944
in	0.49723375	her	0.28781646
and	0.39370865	a	0.23318098
to	0.2748983	his	0.14336193
on	0.21759394	she	0.1128443
at	0.17932475	it	0.06529821
for	0.16905183	he	0.063333265
with	0.16042696	you	0.058997907
from	0.13463423	their	0.05517004

Table 2 puts "she", "he" and "it" at the top on the left, and four common verbs on the right, indicating a pronoun-verb pattern as the second most dominant feature in the corpus.

Table 2: Top words in 2nd singular vector pair

Vector 2, Eigenvalue 0.00427			
she	0.6633538	was	0.58067155
he	0.38005337	had	0.50169927
it	0.30800354	could	0.2315106
and	0.18958427	would	0.17589279

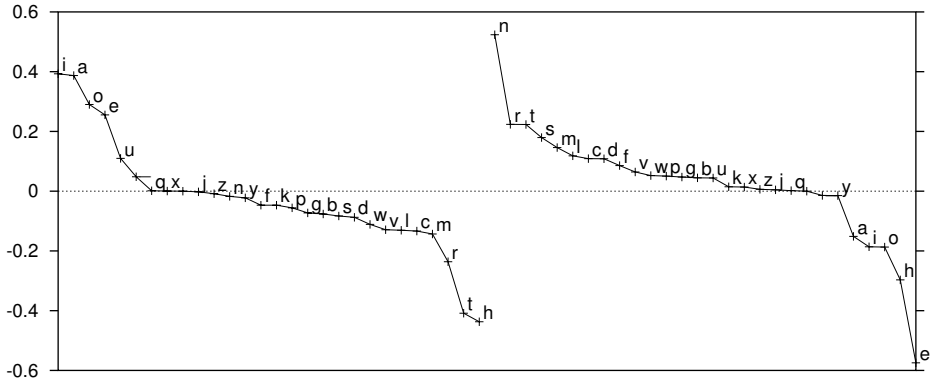
4.2 Letter Bigram Task

Running the algorithm on letter bigrams illustrates different properties. Because there are only 26 letters in the English alphabet, it is meaningful to examine the entire singular vector pair. Figure 1 shows the third singular vector pair derived by running the algorithm on letter bigrams. The y axis gives the projection of the vector for the given letter onto the singular vector. The left singular vector is given on the left, and the right on the right, that is to say, the first letter in the bigram is on the left and the second on the right. The first two singular vector pairs are dominated by letter frequency effects, but the third is interesting because it clearly shows that the method has identified vowels. It means that the third most useful feature for determining the likelihood of letter b following letter a is whether letter a is a vowel. If letter b is a vowel, letter a is less likely to be (vowels dominate the negative end of the right singular vector). (Later features could introduce subcases where a particular vowel is likely to follow another particular vowel, but this result suggests that the most dominant case is that this does not happen.) Interestingly, the letter 'h' also appears at the negative end of the right singular vector, suggesting that 'h' for the most part does not follow a vowel in English. Items near zero ('k', 'z' etc.) are not strongly represented in this singular vector pair; it tells us little about them.

5 Conclusion

An incremental approach to approximating the singular value decomposition of a correlation matrix has been presented. Use

Figure 1: *Third Singular Vector Pair on Letter Bigram Task*



of the incremental approach means that singular value decomposition is an option in situations where data takes the form of single serially-presented observations from an unknown matrix. The method is particularly appropriate in natural language contexts, where datasets are often too large to be processed by traditional methods, and situations where the dataset is unbounded, for example in systems that learn through use. The approach produces preliminary estimations of the top vectors, meaning that information becomes available early in the training process. By avoiding matrix multiplication, data of high dimensionality can be processed. Results of preliminary experiments have been discussed here on the task of modelling word and letter bigrams. Future work will include an evaluation on much larger corpora.

Acknowledgements: The author would like to thank Brandyn Webb for his contribution, and the Graduate School of Language Technology and Vinnova for their financial support.

References

J. Bellegarda. 2000. Exploiting latent semantic information in statistical language modeling. *Proceedings of the IEEE*, 88:8.

Michael W. Berry, Susan T. Dumais, and Gavin W. O'Brien. 1995. Using linear algebra for intelligent information retrieval. *SIAM Review*, 34(4):573–595.

R. W. Berry. 1992. Large-scale sparse singular value computations. *The International Journal of Supercomputer Applications*, 6(1):13–49.

Scott C. Deerwester, Susan T. Dumais, Thomas K. Landauer, George W. Furnas, and Richard A. Harshman. 1990. Indexing by latent semantic analysis. *Journal of the American Society of Information Science*, 41(6):391–407.

S. Dumais. 1990. Enhancing performance in latent semantic indexing. TM-ARH-017527 Technical Report, Bellcore, 1990.

G. H. Golub and C. Reinsch. 1970. Handbook series linear algebra. singular value decomposition and least squares solutions. *Numerical Mathematics*, 14:403–420.

G. Gorrell and B. Webb. 2005. Generalized hebbian algorithm for latent semantic analysis. In *Proceedings of Interspeech 2005*.

P. Kanerva, J. Kristoferson, and A. Holst. 2000. Random indexing of text samples for latent semantic analysis. In *Proceedings of 22nd Annual Conference of the Cognitive Science Society*.

E. Oja and J. Karhunen. 1985. On stochastic approximation of the eigenvectors and eigenvalues of the expectation of a random matrix. *J. Math. Analysis and Application*, 106:69–84.

Terence D. Sanger. 1989. Optimal unsupervised learning in a single-layer linear feedforward neural network. *Neural Networks*, 2:459–473.

Terence D. Sanger. 1993. Two iterative algorithms for computing the singular value decomposition from input/output samples. *NIPS*, 6:144–151.

Juyang Weng, Yilu Zhang, and Wey-Shiuan Hwang. 2003. Candid covariance-free incremental principal component analysis. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 25:8:1034–1040.