Generalized Hebbian Algorithm for Incremental Latent Semantic Analysis

Genevieve Gorrell and Brandyn Webb*

*Department of Information and Computer Science
Linköping University, LINKÖPING 583 81, Sweden
gengo@ida.liu.se
brandyn@sifter.org

Abstract

The Generalized Hebbian Algorithm is shown to be equivalent to Latent Semantic Analysis, and applicable to a range of LSA-style tasks. GHA is a learning algorithm which converges on an approximation of the eigen decomposition of an unseen frequency matrix given observations presented in sequence. Use of GHA allows very large datasets to be processed.

1. Introduction

Latent Semantic Analysis (LSA) is an established method for automatically inferring the contextual similarity of words from a large corpus of text, and has been shown qualitatively (and in some cases quantitatively) to mimic human performance in many of its properties and applications [1]. It has been used to great effect in information retrieval, where its ability to find synonyms is particularly relevant. More recent applications involve incorporation of LSA semantic information into language models [2]. LSA is typically formulated in terms of large matrix operations on the corpus as a whole, and is thus principally a batch algorithm. However, a number of methods do exist for folding in new data after the initial batch process is complete, with varying tradeoffs between exactness and compute time [3]. In this paper, we demonstrate that LSA can be performed in a purely incremental manner from the ground up and with minimal memory requirements via the Generalized Hebbian Algorithm (GHA) [4], making LSA amenable to continuous, on-line learning of non-finite data sets via a fairly simple (and somewhat biologically plausible) algorithm.

2. Latent Semantic Analysis

The principle data matrix representing the input corpus in LSA consists of columns representing documents (or passages, which we will collectively refer to as documents throughout this paper), and rows representing words, with any given cell representing the frequency of a particular word in a particular document. This data matrix is typically highly sparse, and it is this sparseness which defines the difficulty in making useful comparisons from the raw data: Two documents which may be conceptually nearly synonymous may, by chance, have very little overlap in their exact vocabulary, which is to say, the dot product of their respective column vectors in the matrix may be quite small. LSA, roughly, can be seen as a data smoothing algorithm for un-sparsifying this matrix in a meaningful way, such that contextually similar elements (either words or documents) have strong overlap while contextually dissimilar elements do not.

LSA accomplishes this via the singular value decomposition (SVD) of the original word by document data matrix, A, as:

$$A = U\Sigma V^T$$

Where U and V are orthogonal matrices of left and right singular vectors (columns) respectively, and Σ is a diagonal matrix of the corresponding singular values. The U and V matrices can seen as a matched set of orthogonal basis vectors into their corresponding spaces (words and documents, respectively), while the singular values specify the effective magnitude of each such basis-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.

This least-squared-error reduction in rank of the original matrix is how LSA smoothes the data, from an initial rank that may be as high as the number of documents (easily tens of thousands) to a more manageable rank, typically empirically selected in the range of 100 to 300. By forcing the reconstruction of the entire data matrix through the bottle neck of a small number of representative vectors (U and V), it is hoped that these vectors represent ”meaningful” generalizations of the data as a whole, and ergo that the reconstructed matrix captures the contextual essence of the elements while dismissing the finer distinctions (including mere sampling noise).

3. An Incremental Approach–The Generalized Hebbian Algorithm

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 eigen vectors of $A * A^T$ and $A^T * A$, respectively, and the singular values, Σ, are the square-roots (with possible sign correction) of the corresponding eigenvalues. Further, by definition $V = A^T * U * \Sigma^{-1}$, so it suffices, for instance, to find just the eigenvectors and eigenvalues of $A * A^T$, from which the rest is readily inferred.

Note that $A * A^T = \sum_j (A_j * A_j^T)$, where A_j is the j'th column vector in A (i.e., in our case the vector of (log) word counts for the j'th document). Thus $A * A^T$ is essentially a co-occurrence matrix of modified word counts, and U is the set of eigenvectors of this matrix.

Oja and Karhunen [5] demonstrated an incremental solution to finding the first eigenvector from data arriving in this form, and Sanger [4] later generalized this to finding the first N eigenvectors with the Generalized Hebbian Algorithm (GHA). The essence of those algorithms is a simple Hebbian learning rule as follows:

$$U_n(t+1) = U_n + \Lambda * (U_n^T * A_j) * A_j$$

Where U_n is the n'th column of U (i.e., the n'th word-space singular vector). The only modification to this 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 [4] is:

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

Where c_{ij} is an individual element in the word-space singular vector, t is the time step, x_j is the input vector and y_i is the activation (that is to say, the dot product of the input vector with the ith feature vector). To summarise, the formula updates the current feature vector by adding to it the input vector multiplied by the activation minus the projection of the input vector on all the feature vectors so far *including the current feature vector*, multiplied by the activation. Including the current feature vector in the projection subtraction step has the effect of keeping the feature vectors normalised. Note that Sanger includes an explicit learning rate, γ. In this work, we vary the formula slightly by not including the current feature vector in the projection subtraction step. In the absence of the autonormalisation influence, the feature 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. For this reason, no explicit learning rate is included in our implementation. Weng et al. [6] demonstrate the efficacy of this approach.

In terms of an actual algorithm, this amounts to storing a set of N word-space feature vectors (which develop over time into the left-side singular vectors of LSA) and updating them with the above delta computed from each incoming document as it is presented. I.e., the full data matrix itself (A) need never be held in memory all at once, and in fact the only persistent storage requirement is the N developing singular vectors themselves.

LSA often includes an entropy-normalisation step [3], in which word frequencies of the original data matrix are modified to reflect their usefulness as distinguishing features. The word count is modified by setting the cell value as follows[1]:

$$c_{ij} = 1 + \sum_j \frac{p_{ij} log(p_{ij})}{log(n)}$$

Where $p_{ij} = \frac{tf_{ij}}{gf_i}$ and n is the number of documents in the collection. tf is the term frequency, ie. the original cell count, and gf is the global frequency, ie. the total count for that word across all documents.

By modifying the word count in this way, words that are of little value in distinguishing between documents, for example, words such as "the", that are very frequent, are down-weighted. Observe that the calculation of the entropy depends on the total document count and on the total number of a given word across all the documents, as well as the individual cell count. For an incremental method, this means that it must be calculated over the documents seen so far, and that word and document counts must be accumulated on an ongoing basis. A little algebra produces:

[1] A known error in [3] has been corrected here.

$$c_{ij} = 1 + \frac{\sum_j tf_{ij} log(tf_{ij}) - gf_i log(gf_i)}{gf_i log(n)}$$

This arrangement has the convenient property of isolating the summation over a quantity that can be accumulated, ie $tf_{ij} log(tf_{ij})$, whereas the previous arrangement would have required the term frequencies to be stored for an accurate calculation to be made.

This figure however becomes less useful over very large numbers of training items, such as one might use with an incremental algorithm. Consider that it is the nature of language that most words are extremely infrequent. As the number of seen items tends to infinity, the weighting of words that occur with midrange frequencies will tend to zero, and words that occur virtually never will come to dominate. This is not useful in a method based on words co-occurring in similar documents. In fact, it is not the very infrequent words that are important but the mid-frequency words that are good differentiators. For this reason, the concept of an "epoch size" has been introduced as follows:

$$c_{ij} = \frac{(\frac{1}{gf_i}\sum_j tf_{ij} log(tf_{ij})) - log(gf_i) + log(n) - log(N_{epoch})}{log(N_{epoch})}$$

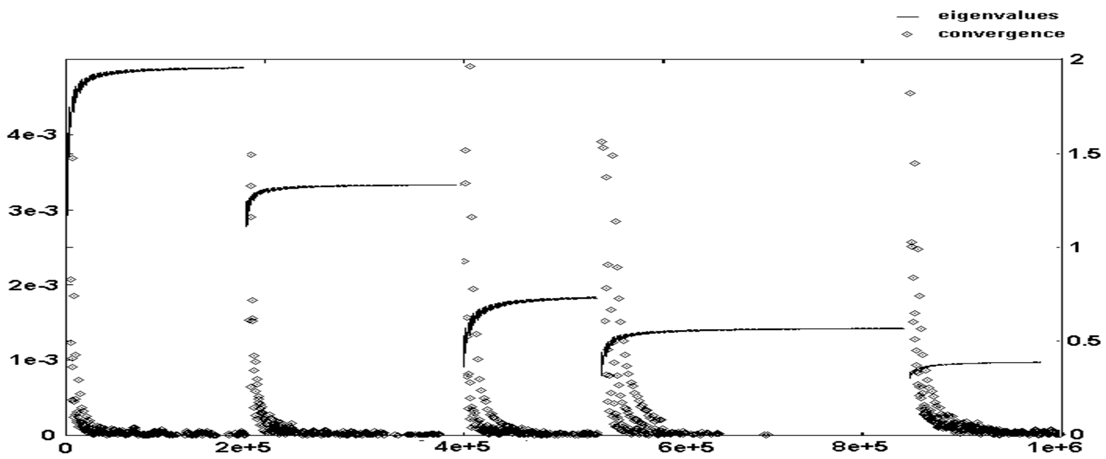
This is equivalent to setting the lower bound on frequency to one occurrence per epoch, and serves the purpose of fixing the weighting values for certain word frequencies even as the number of data items continues to increase.

4. Results

The 20 Newsgroups dataset [7] was used to demonstrate the equivalence of GHA to standard LSA. A test corpus was prepared using data from two newsgroup subsections (atheism and hockey). GHA was then used to decompose the dataset. The data were presented one document at a time, in the form of sparse vectors containing raw word counts. The same dataset was also presented to Matlab, in order to obtain a set of reference vectors. The dataset selected is quite small, in order that it would be feasible to calculate the eigen decomposition using a batch method such as Matlab uses. It should be noted however that one of the main advantages to the GHA algorithm is that extremely large datasets can be processed, and that since GHA is a learning algorithm, it converges gradually on the right answer rather than having at each point a perfect decomposition of the data it has seen so far. For this reason, the data, for the purposes of comparison with Matlab, was passed by the algorithm many times to converge on the eigen decomposition of a small dataset.

The dataset was passed to Matlab in the form of a 1998 (documents) by 29568 (unique words) matrix. Passing this data to GHA in the form of document vectors 29568 long is equivalent to eigen decomposing the square of the dataset matrix, 29568 by 29568. In order to obtain Matlab's decomposition of the data matrix, squaring and eigen decomposing it would be a valid approach. We chose however to take the singular value decomposition of the matrix, and discard the right (document space) vector set. To obtain a singular value set from an eigenvalue set, the singular values must be squared. In the case of this algorithm, where the values are calculated on a per-item basis, the values also needed to be divided by the number of training items in an epoch, in this case, 1998.

Figure 1: *Convergence criterion and eigenvalue against number of data items.*



A comparison of the results for the first ten eigenvalues is shown in table 1. “Error” is defined to be one minus the dot product of the eigenvector with the Matlab eigenvector. “LSA Value” is the eigenvalue as calculated using Matlab. Figure 1 shows the algorithm’s convergence on the correct vector directions. The x-axis shows the number of data items presented. On the left, the y-axis shows the error in vector direction and on the right, the eigenvalue. Observe that as the convergence criterion (the amount by which the direction of the vector changes over, in this case, 500 data presentations) settles to zero, the eigenvalue approaches and settles on its final value. After one million data items have been presented, four vectors have converged and a fifth is close to converging.

Table 1: *Comparison of GHA and LSA.*

Number	Error	GHA Value	LSA Value
0	1.2874603E-5	1.957	1.972
1	3.6120415E-5	1.333	1.339
2	1.2278557E-5	0.734	0.757
3	1.9288063E-4	0.568	0.575
4	1.9168854E-4	0.397	0.445
5	8.904934E-5	0.315	0.381
6	2.5987625E-5	0.403	0.316
7	3.234148E-4	0.279	0.284
8	2.4974346E-4	0.248	0.267
9	1.5366077E-4	0.254	0.245

The vector set produced by GHA differs from standard LSA in that, being an eigen decomposition, no document-space vector set is produced. One way to describe how standard LSA works at this point would be to say that the singular value decomposition is reduced in dimensionality by discarding lower value vector pairs and then used to recreate the original (now smoothed) data matrix, such that a query vector in word space can then be multiplied by the matrix to produce a vector in document space. This vector effectively comprises the overlap (in the form of the dot product) of the query with each of the documents. This approach is not appropriate to the incremental algorithm, where the intention is that the number of documents that the algorithm has seen would be very large. To perform LSA-style tasks with the incremental algorithm a document set appropriate to the task is selected and used to form a document space. For example, the eigenvector set might be formed over a large corpus of domain-general passages, but then used to run queries on domain-specific datasets chosen at runtime.

The method can be outlined as follows. The document set is formed into a document by word frequency matrix. The eigenvector set is multiplied by this matrix to “complete” the singular vector set, that is to say, produce a singular vector set in document space. This completed singular value decomposition can then be used to reassemble the document by word frequency matrix, such as it would be described using the given number of singular vectors. This matrix can be used to compare the new representations of the documents to each other. Queries are also passed through the matrix to produce “pseudodocuments”, which are then compared to the other documents using the dot product.

This method has been performed here using a subset of the training documents and the set of 100 eigenvectors produced by Matlab for the sake of illustration. It should be noted that the document set should be preprocessed using the most recent weighting data from the algorithm, in the case that the weighting steps are being used. Ten documents were chosen, five from atheism and five from hockey. Table 2 constitutes a many-to-many comparison of these documents, via the dot product. Atheism items are denoted with a preceding “a” in the document name, and hockey documents with a preceding “h”. The reader should find themself able to locate the given documents in the corpus using the name, should they wish to do so. Table 3 gives the comparison matrix produced using unprocessed documents, that is to say, vectors of raw word counts, in order to illustrate the impact of LSA.

Observe that in general, the numbers in the top left and bottom right sections of the matrix are larger than those in other sections. This shows a predictable tendency for atheism documents to be more like other atheism documents and for hockey documents to be more like other hockey documents. Observe also that the clustering effect is less strong in table 3: LSA has increased the similarity between documents within domains. Treating the first document, “a:53366”, as a query, we can use the matrix of dot products to find the document most similar to it. The raw word counts select “h:54776” as the most similar document. This is a document from the hockey section, and as

Table 2: *Many-to-many document comparison with LSA.*

	a:53366	a:53367	a:51247	a:51248	a:51249	h:54776	h:54777	h:54778	h:54779	h:54780
a:53366	1.00	0.86	0.86	0.83	0.91	0.78	0.22	0.63	0.52	0.70
a:53367	0.86	1.00	0.82	0.77	0.76	0.63	0.15	0.44	0.38	0.55
a:51247	0.86	0.82	1.00	0.74	0.80	0.66	0.18	0.44	0.40	0.58
a:51248	0.83	0.77	0.74	1.00	0.80	0.78	0.23	0.68	0.61	0.77
a:51249	0.91	0.76	0.79	0.80	1.00	0.78	0.23	0.64	0.57	0.73
h:54776	0.78	0.63	0.66	0.78	0.78	1.00	0.42	0.88	0.84	0.92
h:54777	0.22	0.15	0.18	0.23	0.23	0.42	1.00	0.45	0.39	0.40
h:54778	0.63	0.44	0.44	0.68	0.64	0.88	0.45	1.00	0.85	0.88
h:54779	0.52	0.38	0.40	0.61	0.57	0.84	0.39	0.85	1.00	0.90
h:54780	0.70	0.55	0.58	0.77	0.73	0.92	0.40	0.88	0.90	1.00

Table 3: *Many-to-many document comparison without LSA.*

	a:53366	a:53367	a:51247	a:51248	a:51249	h:54776	h:54777	h:54778	h:54779	h:54780
a:53366	1.00	0.52	0.54	0.33	0.53	0.57	0.18	0.44	0.46	0.54
a:53367	0.52	1.00	0.54	0.40	0.47	0.51	0.20	0.38	0.40	0.45
a:51247	0.54	0.54	1.00	0.35	0.57	0.60	0.22	0.51	0.46	0.61
a:51248	0.33	0.40	0.35	1.00	0.35	0.42	0.19	0.27	0.29	0.29
a:51249	0.53	0.47	0.57	0.35	1.00	0.61	0.18	0.50	0.47	0.57
h:54776	0.57	0.51	0.60	0.42	0.61	1.00	0.24	0.62	0.57	0.63
h:54777	0.18	0.20	0.22	0.19	0.18	0.24	1.00	0.20	0.18	0.20
h:54778	0.44	0.38	0.51	0.27	0.50	0.62	0.20	1.00	0.55	0.57
h:54779	0.46	0.40	0.46	0.29	0.47	0.57	0.18	0.55	1.00	0.71
h:54780	0.54	0.45	0.61	0.29	0.57	0.63	0.20	0.57	0.71	1.00

the dot product of 0.57 suggests, the document is not in fact especially similar to “a:53366”. Using LSA, however, several of the documents are raised in their similarity rating, most notably the atheism documents. The document now selected as being the most similar to “a:53366”, with a dot product of 0.91, is not only an atheism document, but also discusses law, as does the query document.

5. Conclusion

We have demonstrated the use of the Generalized Hebbian Algorithm to perform Latent Semantic Analysis. LSA is an established method for using a corpus of textual passages to infer contextual similarity. It has been applied to a wide range of speech and language-related tasks, including information retrieval and language modelling. The method is based on Singular Value Decomposition, a close relative of eigen decomposition. GHA is an incremental method for deriving the eigen decomposition of an unseen matrix based on serially presented observations. Since it does not require that the entire matrix be held in memory simultaneously, extremely large corpora can be used. Data can be added continuously, making the approach amenable to situations where open-ended corpora are to be used.

6. Acknowledgements

The authors would like to acknowledge Vinnova and the Graduate School of Language Technology, who funded this work.

7. References

[1] S. C. Deerwester, S. T. Dumais, T. K. Landauer, G. W. Furnas, and R. A. Harshman, “Indexing by latent semantic analysis,” *Journal of the American Society of Information Science*, vol. 41, no. 6, pp. 391–407, 1990. [Online]. Available: citeseer.nj.nec.com/deerwester90indexing.html

[2] J. Bellegarda, “Exploiting latent semantic information in statistical language modeling,” *Proceedings of the IEEE*, vol. 88:8, 2000.

[3] S. Dumais, “Enhancing performance in latent semantic indexing,” 1990. [Online]. Available: citeseer.ist.psu.edu/dumais92enhancing.html

[4] T. D. Sanger, “Optimal unsupervised learning in a single-layer linear feedforward neural network,” *Neural Networks*, vol. 2, pp. 459–473, 1989.

[5] E. Oja and J. Karhunen, “On stochastic approximation of the eigenvectors and eigenvalues of the expectation of a random matrix,” *J. Math. Analysis and Application*, vol. 106, pp. 69–84, 1985.

[6] Y. Z. Juyang Weng and W.-S. Hwang, “Candid covariance-free incremental principal component analysis,” *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 25:8, pp. 1034–1040, 2003.

[7] *CMU World Wide Knowledge Base (Web–KB) project*, http://www-2.cs.cmu.edu/ webkb/, as of 6 April 2005.