Generalized Hebbian Algorithm for Dimensionality Reduction in Natural Language Processing

by

Genevieve Gorrell

Department of Computer and Information Science
Linköpings universitet
SE-581 83 Linköping, Sweden

Linköping 2006
Abstract

The current surge of interest in search and comparison tasks in natural language processing has brought with it a focus on vector space approaches and vector space dimensionality reduction techniques. Presenting data as points in hyperspace provides opportunities to use a variety of well-developed tools pertinent to this representation. Dimensionality reduction allows data to be compressed and generalised. Eigen decomposition and related algorithms are one category of approaches to dimensionality reduction, providing a principled way to reduce data dimensionality that has time and again shown itself capable of enabling access to powerful generalisations in the data. Issues with the approach, however, include computational complexity and limitations on the size of dataset that can reasonably be processed in this way. Large datasets are a persistent feature of natural language processing tasks.

This thesis focuses on two main questions. Firstly, in what ways can eigen decomposition and related techniques be extended to larger datasets? Secondly, this having been achieved, of what value is the resulting approach to information retrieval and to statistical language modelling at the n-gram level? The applicability of eigen decomposition is shown to be extendable through the use of an extant algorithm; the Generalized Hebbian Algorithm (GHA), and the novel extension of this algorithm to paired data; the Asymmetric Generalized Hebbian Algorithm (AGHA). Several original extensions to the these algorithms are also presented, improving their applicability in various domains. The applicability of GHA to Latent Semantic Analysis-style tasks is investigated. Finally, AGHA is used to investigate the value of singular value decomposition, an eigen decomposition variant, to n-gram language modelling. A sizeable perplexity reduction is demonstrated.
Parts of this doctoral thesis appear in other publications:


Also by this author:


Thanks to:

Arne Jönsson, my supervisor, who has been a great supporter and an outstanding role model for me; Robin Cooper, second supervisor, for keeping me sane, on which all else depended; Joakim Nivre, third supervisor, for stepping in with additional support later on; Manny Rayner, Ian Lewin and Brandyn Webb, major professional and intellectual influences; Robert Andersson, the world’s best systems administrator; everyone in GSLT, NLPLAB, KEDRI and Lingvistik (GU) for providing such rich working environments; and finally, my beloved family, who mean more to me than anything else.
Contents

1 Introduction ........................................... 9
   1.1 Eigen Decomposition ............................. 12
   1.2 Applications of Eigen Decomposition in NLP .... 17
   1.3 Generalized Hebbian Algorithm .................. 18
   1.4 Research Issues .................................. 19

2 Matrix Decomposition Techniques and Applications 23
   2.1 The Vector Space Model .......................... 24
   2.2 Eigen Decomposition ............................. 25
   2.3 Singular Value Decomposition .................... 27
   2.4 Latent Semantic Analysis ........................ 29
   2.5 Summary ......................................... 38
3 The Generalized Hebbian Algorithm 39
   3.1 Hebbian Learning for Incremental Eigen Decomposition 41
   3.2 GHA and Incremental Approaches to SVD 44
   3.3 GHA Convergence 46
   3.4 Summary 49

4 Algorithmic Variations 51
   4.1 GHA for Latent Semantic Analysis 52
      4.1.1 Inclusion of Global Normalisation 53
      4.1.2 Epoch Size and Implications for Application 54
   4.2 Random Indexing 55
   4.3 GHA and Singular Value Decomposition 57
   4.4 Sparse Implementation 62
   4.5 Convergence 64
      4.5.1 Staged Training 65
      4.5.2 Convergence Criteria 65
      4.5.3 Learning Rates 68
      4.5.4 Asymmetric GHA Convergence Evaluation 69
   4.6 Summary 74
5 GHA for Information Retrieval  
5.1 GHA for LSA  
5.1.1 Method  
5.1.2 Results  
5.2 LSA and Large Training Sets  
5.2.1 Memory Usage  
5.2.2 Time  
5.2.3 Scaling LSA  
5.3 Summary  

6 SVD and Language Modelling  
6.1 Modelling Letter and Word Co-Occurrence  
6.1.1 Word Bigram Task  
6.1.2 Letter Bigram Task  
6.2 SVD-Based Language Modelling  
6.2.1 Large Corpus N-Gram Language Modelling using Sparse AGHA  
6.2.2 Small Corpus N-Gram Language Modelling using LAS2  
6.3 Interpolating SVDLM with Standard N-gram  
6.3.1 Method
<table>
<thead>
<tr>
<th>6.3.2</th>
<th>Small Bigram Corpus</th>
<th>104</th>
</tr>
</thead>
<tbody>
<tr>
<td>6.3.3</td>
<td>Medium-Sized Trigram Corpus</td>
<td>105</td>
</tr>
<tr>
<td>6.3.4</td>
<td>Improving Tractability</td>
<td>106</td>
</tr>
<tr>
<td>6.3.5</td>
<td>Large Trigram Corpus</td>
<td>111</td>
</tr>
<tr>
<td>6.4</td>
<td>Summary</td>
<td>115</td>
</tr>
</tbody>
</table>

7 Conclusion

<table>
<thead>
<tr>
<th>7.1</th>
<th>Detailed Overview</th>
<th>119</th>
</tr>
</thead>
<tbody>
<tr>
<td>7.2</td>
<td>Summary of Contributions</td>
<td>121</td>
</tr>
<tr>
<td>7.3</td>
<td>Discussion</td>
<td>122</td>
</tr>
<tr>
<td>7.3.1</td>
<td>Eigen Decomposition A Panacea?</td>
<td>122</td>
</tr>
<tr>
<td>7.3.2</td>
<td>Generalized Hebbian Algorithm Overhyped or Underrated?</td>
<td>124</td>
</tr>
<tr>
<td>7.3.3</td>
<td>Wider Perspectives</td>
<td>126</td>
</tr>
<tr>
<td>7.4</td>
<td>Future Work</td>
<td>130</td>
</tr>
</tbody>
</table>
Chapter 1

Introduction

In computational linguistics, as in many artificial intelligence-related fields, the concept of intelligent behaviour is central. Human-level natural language processing requires human-like intelligence, imbued as it is with our very human existence. Furthermore, language processing is highly complex, and so we might hope, broadly speaking, that a more “intelligent” system, however you define it, would be better able to handle language processing tasks. Machine intelligence has been gauged and defined in a number of ways. The Turing Test (50) approached intelligence as the ability to pass for human. Other definitions are based on the adaptivity of the system. Adaptivity is a powerful concept of intelligence given that arguably the point of a plastic central nervous system is to enable adaptation within an organism’s lifetime (34). Work such as (32) develops such notions into a formal measure. But what intelligence can be said to be embodied in a simple static system? What definition might we apply to the intelligence of, for example, a system that does not adapt at all, but is nonetheless able to handle a predetermined set of circumstances in useful ways? Does such a system embody any intelligence at all, as the term is commonly used? Surely it can be said to have more or less intelligent behaviours, if not intelligence per se? Since the systems used in natural language processing today for the most part fall into
this category, some measure of their power might be useful.

In order to produce appropriate behaviours in response to input, a system needs first of all to be able to distinguish between different kinds of input in task-relevant ways. This may be simply achieved. For example, a touch-tone telephone system needs only distinguish between a small number of simple and distinct tones in response to a question. This step may also be more challenging. For example, if you make a command via a spoken natural language interface, the system needs to be able to distinguish between different commands. It may not need to distinguish between different speakers, or the same speaker in different moods. These skills are within the capabilities of most human listeners, but are irrelevant to our system. So the task of distinguishing between commands is a complex modelling task involving identifying the features relevant to determining the users’ wishes within the domain of the task being performed. Even getting as far as identifying the words the user most likely spoke is a non-trivial task, requiring many layers of abstraction and reasoning, and so a sophisticated model is required. Having formed a model by which to identify relevant input, the ability to generate appropriate responses follows, and the requirements at this stage depend on the nature of the system.

Human developmental studies have demonstrated the great significance of modelling within the mind, both on a high and a low level. Even very young babies show more interest in stimuli that challenge their world model, suggesting that right from the start, learning is a process of tuning the world model. For example, the distinction between animate and inanimate appears very quickly in the child’s model of the world. Babies will look for longer at inanimate objects moving of their own volition in the manner of complex objects, and conversely, animate objects such as other people following Newton’s Laws as simple objects do (54). The world model provides a way to sift through the input and give attention to phenomena that most require it. On a low level, adaptation describes a nerve’s ceasing to fire in response to unchanging input. You stare at the same thing for long enough and the image begins to disappear. You stare at a particular colour, and your optical field compensates such that when you
look away your visual field is marked by an absence of that colour. Direction and speed of motion are also compensated for. You are able to tune out constant noises. Adaptation is a very simple form of modelling, but other, more complex varieties also appear at a low level in the brain (25).

Approaches to creating an appropriately powerful model in an artificial system fall loosely into two categories. The model can be designed by a human, possibly from human intuitive perceptions of the structure of the input or complex human-generated theories such as linguistic theories of grammar, or the model can be acquired automatically from data. Advantages to the former include that humans are very powerful creators of models, and could we only encode our models well, they would surely surpass any automatically acquired model available today. Advantages to the latter include that they are a lot less work to create (for the humans) and potentially less prone to error, since they can be set up to make fewer assumptions, and choose a solution based on optimising the result. Computer chess can illustrate the difference well, with programs based on searching all possible paths to a depth beyond human abilities competing well, but by no means always prevailing, against human players using more advanced strategies but less raw computational power. Creating a system capable of acquiring a sufficiently powerful model automatically from the data requires that the basic learning framework is sufficiently sophisticated. For example, a system that learns semantic concepts through word co-occurrence patterns is never going to produce a theory of grammar, since it has no access to word order information. The input is insufficiently rich. A system that models the world based on the assumption that observations comprise the additive sum of relevant factors (for example, if it is sunny, I am happy. If it is Sunday, I am happy. Therefore if it is a sunny Sunday, I am very happy) will fail to accurately model circumstances where cause and effect have a more involved relationship.

An adequate format by which to encode input is a critical first step. In this work I focus on vector space models. In the vector space model, a feature set describing the data is indefinitely extensible, allowing for very rich input. Each datum comprises a point in a hyperspace in
which there is one dimension for each feature. Similarity between the
data points can then be thought of in terms of the distance between
the points in space, and so the framework is ideal for problems in
which similarity relationships between data need to be determined.
The range of problems which can be characterised in these terms is
very large indeed, and there are many ways in which different kinds
of similarity can be targeted. Within the vector space representation
there are a variety of ways in which the variation in the position
of these points can then be processed and the relevant information
sifted from the irrelevant and brought to the fore. I focus on one
in particular; eigen decomposition, the properties of which will be
discussed in more detail in the next section.

1.1 Eigen Decomposition

Eigen decomposition is a much-used technique within natural lan-
guage processing as well as many other fields. You wish to model a
complex dataset. The relations between the features are not clear to
you. You focus on including as much information as possible. Your
hyperspace therefore has a high dimensionality. As it turns out, two
of the features depend entirely on each other (for example, it is night,
and it is not day), and therefore one of your dimensions is superfluous,
because if you know the value of one of the features, you know the
value of the other. Within the plane formed by these two dimensions,
points lie in a straight line. Some other features have more complex
interdependencies. The value of a feature follows with little variation
from the combined values of several other features (for example, tem-
perature might relate to number of daylight hours in the day, amount
of cloud cover and time of day). Lines, planes and hyperplanes are
formed by the data within subspaces of the vector space. Collapsing
these dependencies into superfeatures can be thought of in terms of
rotating the data, such that each dimension captures as much of the
variance in the data as possible. This is what eigen decomposition
does for us.
1.1 Eigen Decomposition

In addition, we can take a further step. By discarding the dimensions with the least variance we can further reduce the dimensionality of the data. This time, the reduction will produce an imperfect approximation of the data, but the approximation will be the best possible approximation of the data for that number of dimensions. The approximation might be valuable as a compression of the data. It might also be valuable as a generalisation of the data, in the case that the details are overfitting/noise.

The most important superfeatures in a dataset say something significant and important about that data. Between them they cover much of the variance of the dataset. It can be interesting in itself to learn the single most important thing about a dataset. For example, as we learn later in using a related technique to learn word bigrams, given one and only one feature, the single most important thing you can say about word bigrams in the English language is what words precede “the”. With this one feature, you explain as much as you possibly can about English bigrams using only one feature. Each datum, which previously contained values for each of the features in your original hyperspace, now contains values positioning it in the new space. Its values relate it to the new superfeatures rather than the original feature set we started out with. An unseen datum should be able to be approximated well in terms of a projection on each dimension in the new space. In other words, assuming an appropriate training corpus, unseen data can be described as a quantity of each of the superfeatures.

Eigenfaces provide an appealing visual illustration of the general idea of eigen decomposition. Eigen decomposition is widely used in computer vision, and one task to which it has been usefully applied is face recognition. Eigen decomposition can be applied to corpora of images of faces such that superfeatures can be extracted. Figures 1.1 (36) and 1.2 (unknown source) show some examples of these “eigenfaces” found on the web. (The differences in these eigenfaces are attributable to the training data on which they were prepared.) Note that the first eigenface in 1.1 is indeed a very plausible basic male face, that might provide a reasonable starting point for describing many male faces. Later eigenfaces become increasingly unhuman, as
Figure 1.1: Eigenfaces
Figure 1.2: More Eigenfaces

Figure 1.3: Eigenface Convergence
Figure 1.4: More Eigenface Convergence

they only make sense in the context of being combined with other eigenfaces to produce an additive effect. Figures 1.3 (27) and 1.4 (51) show images converging on a target as more and more eigenfaces are included. (In these particular examples however the target image formed part of the training set.)

Note at this point that eigen decomposition only removes linear dependency in the original feature set. A linear dependency between a feature and one or more others is a dependency in which the value of a feature takes the form of a weighted sum of the other features. For example, if there is a good film showing then Jane is more likely to go to the cinema. If John is going to the cinema, then Jane is more likely to go. Therefore if there is a good film showing and John is going to the cinema, then Jane is even more likely to go. A non-linear dependency might occur, for example, if Jane prefers to see good films alone so she can concentrate. So if there is a good film showing, then she is more likely to go to the cinema. If John is going to the cinema, then she is more likely to go. If, however, there is a good film showing and John is going to the cinema, then Jane is less likely to go. Examples of this in language abound, and so therefore the applicability of eigen decomposition often depends on the extent to which the data can be approximated using an assumption of lin-
1.2 Applications of Eigen Decomposition in NLP

Dimensionality reduction techniques such as eigen decomposition are of great relevance within the field of natural language processing. A persistent problem within language processing is the over-specificity of language given the task, and the sparsity of data. Corpus-based techniques depend on a sufficiency of examples in order to model human language use, but the very nature of 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. You will always see something new at run-time in a task of any complexity.

Furthermore, language can be too rich for the task. The number of underlying semantic concepts necessary to model the target domain is often far smaller than the number of ways in which these concepts might be described, which makes it difficult to, in a search problem for example, establish that two documents, practically speaking, are discussing the same thing. Any approach to automatic natural language processing will encounter these problems on several levels.

Consider the task of locating relevant documents given a search string. Problems arise in that there are often several ways of referring to the same concept. How do we know, for example, that cat and feline are the same thing? There are plenty of documents relevant to felines that feature the word feline not once. This is the kind of problem that Latent Semantic Analysis aims to solve, and in doing so, provides natural language processing with its best-known application of eigen decomposition, or more specifically, singular value decomposition. (Singular value decomposition is a near relative of eigen
decomposition. It allows paired data to be processed, such as, in this case, document/word pairings. Documents containing the word \textit{elevator}, for example, do not typically contain the word \textit{lift}, even though documents about lifts are semantically relevant to searches about elevators. The feature vector for documents about elevators therefore contain no value in the dimension for the word \textit{lift}. However, in describing the variance in the words in a set of documents, there is much redundancy between documents containing \textit{elevator} and documents containing \textit{lift}. Both co-occur with many similar documents. So when eigen decomposition is performed on the dataset, the two are automatically combined into a superfeature, and the differences remaining between them are captured instead in another superfeature, which is probably being reused to explain a number of other phenomena too. For example, one feature might cover several aspects of the differences between UK and US English. Later eigenvectors capture the details of the differences between them, such that given the complete set of eigenvectors, the two words once again become exclusive. However, by discarding some of the less important features we can stop that from happening.

1.3 Generalized Hebbian Algorithm

Having decided to pursue eigen decomposition, a further challenge awaits. Calculating the eigen decomposition is no trivial feat, and the best of the algorithms available are nonetheless computationally demanding. Current research about and using the technique in natural language processing often focuses on adapting the data to the constraints of the algorithm, and adapting the algorithm to the constraints of the data (49) (9). This work is no exception. The Generalized Hebbian Algorithm (GHA) is an algorithm that grew from a different paradigm to the bulk of the work on eigen decomposition, though not an unfamiliar one to many computational linguists and artificial intelligence researchers. Originating as an artificial neural network learning algorithm, it brings with it many of the advantages of that form of learning, such as a certain type of incrementality.
Learning updates are cheap and localised and input is assumed to be a stream of independent observations. Learning behaviour also has some interesting properties. It is, in short, very different from other more standard approaches to calculating eigen decompositions, and is therefore appropriate in a different range of circumstances.

1.4 Research Issues

This work aims to investigate the applicability of eigen decomposition within natural language processing (NLP), and to extend it, both in NLP and in computer science in general. Large corpora have traditionally been problematic for eigen decomposition. Standard algorithms place limitations on the size of dataset that can be processed. In this work the Generalized Hebbian Algorithm is considered as an alternative, potentially allowing larger datasets to be processed. This thesis presents original, published extensions to GHA, via which the algorithm is made more appropriate to relevant tasks within and beyond natural language processing. The algorithm is adapted to paired datasets (singular value decomposition), which is required for the language modelling task as well as many others in computer science in general, and is adapted to sparse data, which is vital for its efficiency in the natural language domain and beyond. Other original algorithmic and implementational variations are also presented and discussed.

Eigen decomposition has already proved valuable in some areas of NLP and many beyond it. In this work, a further area is considered. This thesis presents original work in using eigen decomposition for n-gram language modelling for the first time. It will be shown that eigen decomposition can be used to improve single-order n-gram language models and potentially to improve backoff n-gram models. The approach is demonstrated on training corpora of various sizes.

The questions central to this thesis are as follows:
• What is the utility of eigen decomposition and related techniques to natural language processing? More specifically:

Can eigen decomposition and related techniques be used to improve language models at the n-gram level?

What are the implications of this result and other applications of eigen decomposition in natural language processing for its overall utility in this domain?

• What is the value of the Generalized Hebbian Algorithm and its variants in performing eigen decomposition and related techniques in the natural language processing domain? More specifically:

What is the value of the Generalized Hebbian Algorithm in performing Latent Semantic Analysis?

Can the Generalized Hebbian Algorithm be used to perform singular value decomposition on n-gram data? Is the technique valuable for performing this task.

In what ways can the Generalized Hebbian Algorithm be extended to increase its utility in this domain?

The thesis is primarily technical and implementation-focused. Chapter 2 gives mathematical background on eigen decomposition and its variants. Chapter 3 gives mathematical background on the Generalized Hebbian Algorithm. Chapter 4 describes original extensions to the Generalized Hebbian Algorithm at an algorithmic level. Extensions relevant to Latent Semantic Analysis are presented. A sparse variant is presented which allows computational efficiency to be much improved on the sparse datasets typical to the language processing domain among others. GHA is extended to asymmetric datasets and evaluated. Other developments of the practical applicability of Asymmetric GHA are also discussed. Chapter 5 discusses the application of eigen decomposition and the Generalized Hebbian Algorithm in information retrieval. GHA is presented as a valuable alternative for larger datasets. Further ways in which LSA can be applied to problematically large datasets are discussed. Chapter 6 discusses the
application of singular value decomposition (SVD) and the Generalized Hebbian Algorithm to language modelling at the n-gram level. It is demonstrated that SVD can be used to produce a substantial decrease in perplexity in comparison to a baseline trigram model. Application of the approach to backoff language models is discussed as a focus for future work. AGHA is shown to be a valuable alternative for performing singular value decomposition on the large datasets typical to n-gram language modelling. Other alternatives to performing large-scale singular value decompositions in this context are also discussed. Chapter 7 concludes.
Chapter 2

Matrix Decomposition Techniques and Applications

This chapter aims to give the reader unacquainted with eigen decomposition and related techniques an understanding sufficient to enable them to follow the remainder of the work. A familiarity with basic matrix and vector mathematics is assumed in places. For readers unfamiliar with matrix/vector operations, there are a number of excellent text books available. In Anton and Rorres’ “Elementary Linear Algebra” (2) for example, the reader will find definitions of the following concepts and operations: square matrices, symmetrical matrices, matrix transposition, dot product of vectors, outer product of vectors, the multiplying together of matrices, the multiplying of matrices by vectors and normalisation and orthogonalisation of vectors. The reader satisfied with a more surface understanding should hopefully find that this chapter provides them with an intuitive grasp of the relevant concepts, and so is encouraged to read on. The reader

\(^1\)Wikipedia also provides useful and readily-available material on these topics.
already familiar with eigen decomposition, singular value decomposition and Latent Semantic Analysis is directed to the next chapter, since nothing mentioned here will be new to them.

2.1 The Vector Space Model

As mentioned in the introduction, the vector space model is a powerful approach to describing and interacting with a dataset. Data takes the form of feature value sets in vector form. Each feature takes a dimension in the vector space in which the data positions itself. The theory is that the relationships between the positions of the datapoints in the space tells us something about their similarity. A dataset takes the form of a set of vectors, which can be presented as a matrix, and all the power of the matrix format becomes available to us. Here, for example, is a dataset:

"The man walked the dog"
"The man took the dog to the park"
"The dog went to the park"

These data can be used to prepare a set of vectors, each of which describes a passage in the form of a wordbag. The wordbag, in which the vector describes the counts of each word appearing in the document in a fashion that does not preserve word order, is popular in a variety of approaches to natural language processing, particularly information retrieval. The wordbag vectors are presented as a matrix as follows:
Data in this form can be interacted with in a variety of ways. Similarity can be measured using the distance between the points, or using the cosine of the angle between them, as measured using dot product. Transformations can be performed on the dataset in order to enhance certain aspects. For example, the dataset could be multiplied by another matrix such as to skew and/or rotate it. Non-linear transformations can also be introduced. The matrix presented above can be multiplied by its own transpose to produce a matrix that describes the occurrence of a word with every other word across the dataset. Such a matrix would be square and symmetrical.

2.2 Eigen Decomposition

Eigen decomposition allows us to rotate our dataset in a manner that collapses linear dependencies into a single dimension or component, and provides a measure of the importance of the components within the dataset. On a conceptual level, it models the dataset in terms of additive influences: this observation is produced by so much of this influence, so much of that influence etc. It aims to tell us what the influences are that most efficiently describe this dataset. The benefits to such a representation are many. Such influences/components/dimensions may well tell us something valid and important about the nature of the data. Such a representation is also
efficient, and provides a principled way to perform dimensionality re-
duction on the data, such as to compress and generalise it.

Let us introduce a formal definition. In the following, \( M \) is the matrix
we are performing eigen decomposition on, \( v \) is an eigenvector of \( M \)
and \( \lambda \) is the eigenvalue corresponding to \( v \).

\[
\lambda v = Mv
\]

(2.1)

As mentioned earlier, an eigenvector of a transform/matrix is one
which is unchanged by it in direction. The eigenvector may be scaled,
and the scaling factor is the eigenvalue. The above simply says that
\( v \) multiplied by \( \lambda \) equals \( M \), our original matrix, multiplied by \( v \):
multiplying \( v \) by \( M \) has the same effect as scaling it by \( \lambda \).

For any square symmetrical matrix there will be a set of such eigenvectors. There will be no more eigenvectors than the matrix has rows
(or columns), though there may be fewer (recall that linear dependen-
cies are collapsed into single components thus reducing dimensionality). The number of eigenvectors is called the rank of the matrix.
The eigenvectors are normalised and orthogonal to each other, and
effectively define a new space in terms of the old one. The resulting
matrix of eigenvectors can be used to rotate the data into the new
space. Formally,

\[
MV^T = M' 
\]

(2.2)

where \( V \) is the column matrix of eigenvectors, \( M \) is the original data
matrix and \( M' \) is the rotated and possibly approximated data matrix.
This step is included in a worked example of Latent Semantic Analysis later in the chapter, that may provide additional clarification. Relationships between the datapoints are preserved, but in the
new space, each axis captures as much of the variance in the dataset as possible. Superfluous dimensions fall away. Dimensions contributing least to the variance of the data can be discarded to produce a least mean squared error approximation to the original dataset. $k$ is commonly used to refer to the number of dimensions remaining, and is used in this way throughout this work.

### 2.3 Singular Value Decomposition

Eigen decomposition is specific to symmetrical data. An example of symmetrical data is word co-occurrence. Word $a$ appears with word $b$ exactly as often as word $b$ occurs with word $a$. The set of wordbags given as an example at the beginning of this chapter is an example of asymmetric data. The matrix pairs documents with their wordbags. Each cell count gives us the number of appearances of a particular word in a particular document. If we were to multiply such a matrix by its own transpose what we would get would be a symmetrical dataset describing the occurrence of every word with every other word across the entire document set. Were we to multiply the transpose of the matrix by the matrix we would get a symmetrical dataset describing the number of shared words between each document and every other. A symmetrical dataset is always described by a square, diagonally symmetrical matrix. Eigen decomposition can be extended to allow a similar transform to be performed on paired data, i.e. rectangular matrices. The process is called singular value decomposition, and can be formalised as follows:

$$M = U\Sigma V^T$$  \hspace{1cm} (2.3)

In the above, $M$ is our rectangular matrix, $U$ is the column matrix of left singular vectors, and parallels the eigenvectors, $V^T$ is the transpose of the column matrix of right singular vectors, which again parallels the eigenvectors, and $\Sigma$ is a diagonal matrix containing the
singular values (which take the place of eigenvalues) in the appropriate order. Where eigen decomposition creates a new space within the original space, singular value decomposition creates a pair of new spaces, one in “left vector space” (for example, word space, in our wordbag example) and one in “right vector space” (for example, document space). The two spaces are paired in a very tangible sense. They create a forum in which it is valid to compare datapoints of either type directly with each other (for example word vectors with document vectors)\(^{(3)}\).

The relationship between eigen decomposition and singular value decomposition is not a complicated one. We define:

\[
A = MM^T \quad \quad (2.4)
\]

\[
B = M^TM \quad \quad (2.5)
\]

That is to say, we form two square symmetrical matrices from our original rectangular data matrix, each correlating an aspect of the dataset (i.e. words or documents, in our example) with itself. Then:

\[
A = U\Lambda_u U^T \quad \quad (2.6)
\]

\[
B = V\Lambda_v V^T \quad \quad (2.7)
\]

\[
\Lambda_u = \Lambda_v = \Sigma^2 \quad \quad (2.8)
\]

Or in other words, the eigenvectors of the matrix \(A\) are the left singular vectors of the matrix \(M\). The eigenvectors of the matrix \(B\) are the right singular vectors of the matrix \(M\). The eigenvalues of the matrix \(A\) are the eigenvalues of the matrix \(B\) and are the squares of the singular values of \(M\).
2.4 Latent Semantic Analysis

As the single best known usage of singular value decomposition (SVD) in natural language processing, Latent Semantic Analysis provides an excellent practical illustration of its application. In addition, LSA makes an appearance later in this work, so an acquaintance with the principle will prove beneficial. Search tasks, in which documents relevant to a search string are to be retrieved, run into difficulties due to the prevalence of word synonyms in natural language. When the user searches on, to cite an earlier example, “elevator” they would also like documents containing the word “lift” to be returned, although documents rarely use both terms. An automatic approach to finding these synonyms is potentially of benefit in information retrieval as well as a variety of other tasks. Latent Semantic Analysis (14) approaches this problem with the aid of singular value decomposition.

The approach aims to utilise the fact that whilst, for example, “lift” and “elevator” might not appear together, they will each appear with a similar set of words (for example, “floor”). This information can be tapped through SVD and principled dimensionality reduction. If there is some superfeature that captures the basic “liftiness” of the document, that is later refined such as to specify whether “lift” or “elevator” is used, then by isolating the principles and discarding the refinements we might be able to access this information. Dimensionality reduction maps data to a continuous space, such that we can now compare words that previously we couldn’t. The remainder of this chapter provides a worked example of performing Latent Semantic Analysis on a small document set. A brief summary follows.

Returning to our earlier example, we have the following document set:

“The man walked the dog”
“The man took the dog to the park”
“The dog went to the park”
This document set is transformed into a matrix of wordbags like so:

<table>
<thead>
<tr>
<th></th>
<th>Passage 1</th>
<th>Passage 2</th>
<th>Passage 3</th>
</tr>
</thead>
<tbody>
<tr>
<td>the</td>
<td>2</td>
<td>3</td>
<td>2</td>
</tr>
<tr>
<td>man</td>
<td>1</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>walked</td>
<td>1</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>dog</td>
<td>1</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>took</td>
<td>0</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>to</td>
<td>0</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>park</td>
<td>0</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>went</td>
<td>0</td>
<td>0</td>
<td>1</td>
</tr>
</tbody>
</table>

To be very clear, the vectors this matrix embodies are the following paired data (documents and wordbags):

\[
\begin{pmatrix}
1 & 0 & 0 \\
0 & 1 & 0 \\
0 & 0 & 1
\end{pmatrix}
\begin{pmatrix}
2 & 1 & 1 & 1 & 0 & 0 & 0 & 0 \\
2 & 0 & 0 & 1 & 0 & 1 & 1 & 1 \\
3 & 1 & 0 & 1 & 1 & 1 & 1 & 0
\end{pmatrix}
\]

If left (document) data matrix $D$ is the row matrix formed from the document vectors,

\[
D = \begin{pmatrix}
1 & 0 & 0 \\
0 & 1 & 0 \\
0 & 0 & 1
\end{pmatrix}
\]

(2.9)

and right (wordbag) data matrix $W$ is the row matrix formed from the wordbag vectors,

\[
W = \begin{pmatrix}
2 & 1 & 1 & 1 & 0 & 0 & 0 & 0 \\
3 & 1 & 0 & 1 & 1 & 1 & 1 & 0 \\
2 & 0 & 0 & 1 & 0 & 1 & 1 & 1
\end{pmatrix}
\]

(2.10)

then,
2.4 Latent Semantic Analysis

\[ W^T D = M \] (2.11)

where \( M \) is our data matrix shown above. We decompose this matrix using SVD, to obtain two sets of normalised singular vectors and a set of singular values. This operation can be performed by any of a variety of readily available mathematics packages. Algorithms for performing singular value decomposition are discussed in more detail later on in this work. Recall,

\[ M = U \Sigma V^T \] (2.12)

Then,

\[ U^T = \begin{pmatrix} 0.46 & 0.77 & -0.45 \\ -0.73 & -0.04 & 0.68 \\ -0.51 & -0.64 & -0.58 \end{pmatrix} \] (2.13)

\[ \Sigma = \begin{pmatrix} 5.03 & 0 & 0 \\ 0 & 1.57 & 0 \\ 0 & 0 & 1.09 \end{pmatrix} \] (2.14)

\[ V^T = \begin{pmatrix} -0.82 & -0.24 & -0.09 & -0.34 & -0.14 & -0.25 & -0.25 & -0.10 \\ 0.10 & 0.47 & 0.49 & 0.06 & -0.02 & -0.43 & -0.43 & -0.40 \\ 0.01 & 0.22 & -0.41 & -0.31 & 0.63 & 0.10 & 0.10 & -0.53 \end{pmatrix} \] (2.15)

Transposes are presented here for the convenience of presenting row matrices. So we have two different factorisations of \( M \): our original
data and our singular value decomposition. The difference between our original data and the singular value decomposition is that our singular value decomposition comprises orthogonal vectors, thereby removing redundancy. In the worst case there will be as many singular vector pairs as there were data pairs, but usually there is some linear redundancy, and therefore there are fewer singular vector pairs. A corpus of three documents can produce no more than three singular triplets (pairs of vectors with associated singular value). However, a more realistic corpus might produce numbers of singular triplets in quadruple figures for a vocabulary/document set size of hundreds of thousands. Dimensionality reduction is then performed by discarding all but the top few hundred singular triplets. The precise number of singular triplets retained is chosen on an ad hoc basis. Around two to three hundred is often found to be optimal for LSA. This stage will be simulated here by discarding the last singular triplet to produce a two-dimensional semantic space, which has the advantage of being readily visualisable. Here are the remaining singular triplets:

\[
U'T = \begin{pmatrix}
0.46 & 0.77 & -0.45 \\
-0.73 & -0.04 & 0.68
\end{pmatrix}
\] (2.16)

\[
\Sigma' = \begin{pmatrix}
5.03 & 0 \\
0 & 1.57
\end{pmatrix}
\] (2.17)

\[
V'T = \begin{pmatrix}
-0.82 & -0.24 & -0.09 & -0.34 & -0.14 & -0.25 & -0.25 & -0.10 \\
0.10 & 0.47 & 0.49 & 0.06 & -0.02 & -0.43 & -0.43 & -0.40
\end{pmatrix}
\] (2.18)

Figure 2.1 depicts the documents represented as points in semantic space. Documents in reduced space are column vectors of \(D'\):

\[
D' = DU'
\] (2.19)
Figure 2.1: *Three Example Documents Depicted in a Two-Dimensional Semantic Space*
(In this case, the above is rather trivial since $D$ happens to be the identity matrix. $DU'$ therefore equals $U'$. By multiplying the document vectors by the reduced left singular vector set (or the wordbag vectors by the right singular vector set) we can move them into the new space. Figure 2.1 illustrates this. We can then compare the semantic similarity of the documents using for example their dot products with each other in this new space, which will typically constitute an improvement.

A typical use of LSA is returning the best match among a set of documents given a string such as a user query. This can be illustrated in the context of the above example. Suppose the query is "the dog walked". This string is used to form a wordbag vector in the same manner as the documents were. It becomes a "pseudodocument". The pseudodocument would therefore be,

$$P = \begin{pmatrix} 1 & 0 & 1 & 1 & 0 & 0 & 0 & 0 \end{pmatrix}$$

(2.20)

We can move this pseudodocument into semantic space by multiplying it by the matrix $V'$ as shown:

$$PV' = P'$$

(2.21)

This produces the two-dimensional semantic-space vector,

$$P' = \begin{pmatrix} -1.25 \\ 0.65 \end{pmatrix}$$

(2.22)

In this form, the pseudodocument can be compared to any other document in the same space, see figure 2.2.
Figure 2.2: *Pseudodocument with Training Documents*
Additionally, a number of techniques are available that allow the data to be preprocessed in such a way as to further increase the effectiveness of the technique. Words contribute to varying extents to the semantic profile of a passage. For example, the word “the” has little impact on the meaning of passages in which it appears. A word which distributes itself evenly among the documents in a collection is of little value in distinguishing between them. LSA can therefore be made more effective by reducing the impact of such words on word count matrix and increasing the impact of less evenly distributed words. Dumais (16) outlines several methods of achieving this. The one used in this thesis is the most sophisticated and effective of these. It will now be presented. The reader interested in learning about the others is directed to the original source.

$c_{ij}$ is the cell at column $i$, row $j$ of the corpus matrix. The entropy normalisation step most commonly used, and used throughout this thesis, involves modifying this value as follows,

\[
p_{ij} = \frac{t_f}{g_f}
\]

\[
g_w = 1 + \sum_j p_{ij} \frac{\log(p_{ij})}{\log(n)}
\]

\[
c_{ij} = g_w (c_{ij} + 1)
\]

where $g_w$ is the “global weighting” of the word at $i$, $n$ is the number of documents in the collection, $t_f$ is the term frequency, i.e. the original cell count, and $g_f$ is the global frequency, i.e. the total count for that word across all documents.

Let us look at the effect of this step on our example dataset. Here is our original matrix:
### 2.4 Latent Semantic Analysis

<table>
<thead>
<tr>
<th></th>
<th>Passage 1</th>
<th>Passage 2</th>
<th>Passage 3</th>
</tr>
</thead>
<tbody>
<tr>
<td>the</td>
<td>2</td>
<td>3</td>
<td>2</td>
</tr>
<tr>
<td>man</td>
<td>1</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>walked</td>
<td>1</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>dog</td>
<td>1</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>took</td>
<td>0</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>to</td>
<td>0</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>park</td>
<td>0</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>went</td>
<td>0</td>
<td>0</td>
<td>1</td>
</tr>
</tbody>
</table>

and here is the same matrix following the preprocessing step described above:

<table>
<thead>
<tr>
<th></th>
<th>Passage 1</th>
<th>Passage 2</th>
<th>Passage 3</th>
</tr>
</thead>
<tbody>
<tr>
<td>the</td>
<td>0.019</td>
<td>0.024</td>
<td>0.019</td>
</tr>
<tr>
<td>man</td>
<td>0.255</td>
<td>0.255</td>
<td>0.0</td>
</tr>
<tr>
<td>walked</td>
<td>0.693</td>
<td>0.0</td>
<td>0.0</td>
</tr>
<tr>
<td>dog</td>
<td>0.0</td>
<td>0.0</td>
<td>0.0</td>
</tr>
<tr>
<td>took</td>
<td>0.0</td>
<td>0.693</td>
<td>0.0</td>
</tr>
<tr>
<td>to</td>
<td>0.0</td>
<td>0.255</td>
<td>0.255</td>
</tr>
<tr>
<td>park</td>
<td>0.0</td>
<td>0.255</td>
<td>0.255</td>
</tr>
<tr>
<td>went</td>
<td>0.0</td>
<td>0.0</td>
<td>0.693</td>
</tr>
</tbody>
</table>

Values for “the” are much reduced: this word appears fairly indiscriminately across all the documents. “Dog” disappears completely, being perfectly uniform in its occurrence. “Took” and “went” remain high, being good discriminators. “To” and “man” find themselves somewhere in between. It is easy to see that we can carry out the LSA technique equally well on this second matrix, and that we might expect superior results.

Latent Semantic Analysis has been applied in an impressive diversity of domains (53) (18) (41), although it is best-known in the information retrieval context. Impressive results have also been demonstrated in using LSA to incorporate long-span semantic dependencies in language modelling (55) (3) (12). Language modelling, including LSA in this context, is further discussed later in this work.
2.5 Summary

This chapter has provided the reader with background necessary to place the thesis in context with regards to matrix decomposition techniques. The value of eigen and singular value decomposition has been discussed in terms of their allowing data to be smoothed, simplified and compressed in a principled fashion. Latent Semantic Analysis has been presented as a well-known application of singular value decomposition within natural language processing. Work within the LSA domain will follow later in the thesis. The next chapter introduces the Generalized Hebbian Algorithm, which is the basis of the approach on which this thesis focuses.
Chapter 3

The Generalized Hebbian Algorithm

The previous chapter introduced eigen decomposition and singular value decomposition. These techniques have been widely applied within information science and computational linguistics. However, their applicability varies according to the constraints introduced by specific problems.

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 (19) and Jacobi-like approaches, which can be very efficient. They are inappropriate to sparse matrices however, because they work by rotating the matrix, which has the effect of desparsifying it, inflating it in size. Techniques such as those described in Berry’s 1992 article (6) are more appropriate in the natural language domain. A Lanczos-based algorithm from Berry
et al's SVDPACK (4) is used later on in this work.

Optimisation work is of particular importance because the decomposition techniques are expensive, and there are strong constraints on the size of matrices that can be processed in this way. This is of particular relevance within natural language processing, where corpora are often very large, and the success of many data-driven techniques depends on the use of a large corpus.

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 the most 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 over time.

There are many areas of application in which efficient incrementality is of importance. Since it is computationally expensive to calculate a matrix decomposition, it may not be feasible to recalculate when new data becomes available. Effective incrementality would remove the ceiling on matrix size that current techniques impose. The data need not be processed all at once. Systems that learn in real time need to be able to update data structures quickly. Various ways of updating an eigen or singular value decomposition given new data items have been proposed. This chapter presents the Generalized Hebbian Algorithm and contrasts it with other approaches currently available.
3.1 Hebbian Learning for Incremental Eigen Decomposition

The Generalised Hebbian Algorithm was first presented by Oja and Karhunen in 1985 (38), who demonstrated that Hebbian learning could be used to derive the first eigenvector of a dataset given serially-presented observations (vectors). Sanger (46) later extended their architecture to allow further eigenvectors to be discovered within the same basic framework.

Figure 3.1 depicts first the simple Hebbian learning rule, in which inputs are multiplied by connection weights and summed to generate
an output, and second, the update step which leads to the system’s
learning the strongest eigenvector. The figure shows how data is re-
ceived in the form of activations to the input nodes. The activations
are altered according to the strength of the weighting on the con-
nection between them and the output node. The activation at the
output node is then fed back in the form of updates to the weights.
The weights, which can be considered a vector of numbers, converge
on the strongest eigenvector.

Equation 3.1 describes the algorithm by which Hebbian learning can
be made to discover the strongest eigenvector, and is simply another
way of stating the procedure described by figure 3.1.

\[ u(t + 1) = u + \lambda (u^T \cdot a)a \]  \hspace{1cm} (3.1)

In the above, \( u \) is the eigenvector, \( a \) is the input vector (data ob-
servation) and \( \lambda \) is the learning rate (not to be confused with the
\( \lambda \) used in the previous chapter to represent the eigenvalue). \( t + 1 \)
describes the fact that \( u \) is updated to take on a new value in the
next timestep. Intuitively, the eigenvector is updated with the input
vector scaled proportionally to the extent to which it already resem-
bles it, as established by the dot product operation. In this way, the
strongest direction in the input comes to dominate.

To relate the above to the formalisations introduced in the previous
chapter, our data observations, which might for example take the
form of wordbag vectors (this time not paired with document vec-
tors, since we are using eigen decomposition and therefore require
symmetrical data) are the vectors \( a \). Together they form the column
matrix \( A \). Our eigenvectors \( u \), produced by the Generalized Hebbian
Algorithm, are therefore eigenvectors of the following matrix:

\[ M = AA^T \]  \hspace{1cm} (3.2)
3.1 Hebbian Learning for Incremental Eigen Decomposition

This foundation is extended by Sanger to discover multiple eigenvectors. The only modification to equation 3.1 required to uncover further eigenvectors is that the update needs to be made orthogonal to previous eigenvectors: since the basic procedure finds the strongest of the eigenvectors, in order to prevent that from happening and find later eigenvectors, the previous eigenvectors are removed from the training update in order to take them out of the picture. The current eigenvector is also included in the orthogonalisation.

\[ u_n(t + 1) = u_n + \lambda (u_n^T \cdot a) (a - \sum_{i \leq n} (u_i^T \cdot a) u_i) \]  

(3.3)

Here, \( u_n \) is the \( n \)th eigenvector. This is equivalent to Sanger’s final formulation, in the original notation (46),

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

(3.4)

where \( c_{ij} \) is an individual element in the \( i \)th eigenvector, \( t \) 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 \( i \)th eigenvector). \( \gamma \) is the learning rate.

To summarise from an implementation perspective, 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, \( \gamma \). A potential variation, utilised in this work, involves excluding the current eigenvector from the projection subtraction step. In the absence of the autonormalisation influence, the eigenvector is allowed to grow long.
Chapter 3 The Generalized Hebbian Algorithm

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, such that the data reinforces it, and since further learning has less impact once the vector has become long. Weng et al. (52) demonstrate the efficacy of this approach.

In terms of an actual algorithm, this amounts to storing a set of $N$ word-space eigenvectors and updating them with the above delta computed from each incoming document as it is presented. This means that the full data matrix need never be held in memory all at once, and in fact the only persistent storage requirement is the $N$ developing singular vectors themselves.

3.2 GHA and Incremental Approaches to SVD

GHA calculates the eigen decomposition of a matrix based on single observations presented serially. It allows eigenvectors to be learnt using no more memory than is required to store the eigenvectors 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. (Complexity analysis is presented later in this work in the context of applying specific implementations of GHA to LSA-style tasks.) The algorithm produces eigenvectors starting with the most significant, since it is the greater eigenvectors that converge most quickly, which means that useful data immediately begins to become available. 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 eigen 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 and time is not a priority.
3.2 GHA and Incremental Approaches to SVD

A key reason for using GHA to produce eigen decompositions is therefore its incremental nature, both for purposes of adaptivity and because it makes the approach amenable to very large data dimensionalities. Natural language research has generated work in the area of incrementality in singular value decomposition, because natural language processing is a key example of a field of research in which large corpora are used, and standard approaches to matrix decomposition are pushed to their limits. As discussed in the previous chapter, SVD and eigen decomposition are closely related and in some contexts even interchangeable, so although, strictly speaking, GHA is a method for performing eigen decomposition, SVD and eigen decomposition in this section are treated interchangeably.

Extant incremental approaches to singular value decomposition typically fall into three categories. The first essentially involves adding the new data to the dataset previously decomposed and then recomputing the decomposition. To call such an approach incremental is therefore somewhat of a misnomer, though depending on the context, some aspects of the process might be considered incremental. For example, Ozawa et al (39) take this approach in the context of Principal Component Analysis for face recognition. Principal Component Analysis (PCA) is a near relative of SVD. Berry et al (5) also discuss recomputing as an option in the case where a database of documents for LSA is extended.

In the second category of approaches to incrementality we find approximations. The decomposition of a dataset, having been increased with new data, can be approximated without recomputing completely. “Folding in”, as described by Berry et al (5) is an example of this approach. It works on the assumption that new data is typical of the data on which the original decomposition was performed. Pseudodocuments are formed in the manner described in the previous chapter, and these are then treated as part of the original document set. As larger quantities of data are added and the assumption of representativity starts to break down, the accuracy of the approximation decreases. However, it can be a useful option in the case that a new batch of data needs to be efficiently represented in the context of an existing semantic space. The approach is not unlike that of
creating a model based on a training set and then using it to process an unseen test set. The principle is well-known, but it is clear that the model is not updated with the “test set”.

In the third category, an existing decomposition is updated with new data such that the resulting decomposition is a perfect result for the dataset. O’Brien (37) presents an example of this, as does Brand (7). Brand describes an approach to SVD updating in the context of which missing or noisy data is also discussed. These approaches are appropriate in the case that a new batch of data needs to be added to an existing decomposition offline. The step is more expensive than folding in (though cheaper than recomputing) and as such is applicable in different circumstances. Brand (7) also provides a review of earlier work in SVD incrementality.

GHA differs from each of these categories in some key ways. The above approaches are incremental inasmuch as they provide ways to add new data to an existing decomposition. None of them are designed to accommodate the situation in which data is streamed. The update operations are typically expensive. All of them assume an existing decomposition into which the new data will be added. GHA is different in that its incrementality is far more intrinsic. It assumes no existing decomposition (though might potentially benefit from being seeded with an existing decomposition). It converges on the strongest eigenvectors first, thereby producing useful information quickly. As a learning algorithm, however, it does need to be used appropriately: unlike other approaches which at any stage have the perfect decomposition for the data they have seen so far, GHA needs to be allowed to converge. Reruns through smaller datasets will most likely be required.

### 3.3 GHA Convergence

The usability of the GHA algorithm is in no small way connected to the way in which the vectors being trained approach the target
direction, that is to say, the eigenvector. If the aim is to have a complete decomposition in which every GHA-trained vector differs minimally in direction from the actual eigenvector, and eigenvalues are appropriate, then the time taken to achieve this end, and the reliability with which a tolerable accuracy level is reached, is critical.

Although convergence of GHA is proven (46), previous authors have noted the absence of large-scale evaluation of convergence behaviour in the literature despite widespread interest in the algorithm (15). Some attempt has been made here to remedy this on a practical level with a plot of convergence against number of training steps required using a decomposition done using the better-known LAS2 algorithm (6) as reference. A subsection of the 20 Newsgroups corpus (11), specifically the atheism section, was preprocessed into a sparse trigram matrix with a dimensionality of around 15,000 by 90,000. Words formed columns and two-word histories, rows. Matrix cells contained trigram counts. This matrix was then decomposed using LAS2. The resulting left-side vector set was then used to plot dot product with GHA eigenvector as they converged. This plot is presented in Figure 3.2. The implicit learning rate described earlier in the chapter was used here. The impact of this design choice on the data presented here needs to be considered. Other approaches to learning rates are discussed in the next chapter, in the context of asymmetric convergence. The convergence criterion used was based on the distance between the end of the vector being trained, normalised, compared with the vector in its earlier position, in this case 50,000 training steps previously (and so will inevitably decrease as the vector grows long, and new data has less impact on direction). The graph shows dot product of the GHA eigenvector currently being trained with the LAS2 target, and so when the GHA vector reaches convergence the graph shows a jump as we move on to the next vector. The dot product we are aiming at is 1; the two vectors should point in the same direction. The graph shows convergence of eleven eigenvectors (the first three being difficult to make out because they converged almost immediately). Around $2.5 \times 10^7$ training presentations were required to achieve this many eigenvectors. As can be seen from the graph, varying degrees of precision were achieved, suggesting that the convergence criterion might yet be made more effective. Furthermore
Figure 3.2: Dot Product of GHA Eigenvector with Reference Set Against Number of Training Steps

A tendency to level off, in some cases well before a high precision is achieved, suggests that the implicit learning rate approach leaves something to be desired. The implicit learning rate used here is contrasted with Sanger’s original explicit learning rate in a comparison of convergence behaviour on asymmetric data later in this work, where convergence is discussed in more detail in the context of evaluating an original algorithm, the Asymmetric Generalized Hebbian Algorithm. Other ways of improving convergence through the choice of an appropriate convergence criterion, and issues around selecting such a criterion are also discussed in the context of asymmetric data.
3.4 Summary

This chapter introduced the Generalized Hebbian Algorithm, and explained how it can be used to learn the eigen decomposition of a matrix based on single observations presented serially. Advantages to such an approach have been discussed, both in terms of allowing matrix sizes too large for conventional approaches to be decomposed, and in terms of implementing learning behaviour such as adaptation to new input patterns. The next chapter discusses work in the area of adapting GHA to varying contexts, with natural language applications in mind.
Chapter 4

Algorithmic Variations

This section describes a number of algorithmic developments to the Generalized Hebbian Algorithm. It begins by discussing developments of the basic GHA formulation. The technique is applied to Latent Semantic Analysis, and is modified to accommodate the pre-processing steps commonly included in LSA implementations. Random Indexing (28) is introduced here as a supplement to GHA providing a means of fixing and reducing vector length.

An extension of GHA to paired data (singular value decomposition) is then presented. Sparse variants of the algorithms are described. Sparse variants are contrasted with the Random Indexing approach introduced in the context of GHA. Approaches to setting learning rates and determining convergence of the algorithms are discussed.1

1In this section, the work on including LSA entropy normalisation in GHA is joint work done with Brandyn Webb. The basic design of the Asymmetric Generalized Hebbian Algorithm is Brandyn's; the derivation is my own. The sparse version of GHA is joint work with Brandyn.
4.1 GHA for Latent Semantic Analysis

Latent Semantic Analysis has been used to great effect in the field of information retrieval and beyond. Limitations on corpus size are however a documented problem (49). Since only the first few hundred eigenvectors are required in LSA, GHA is a potential candidate for an alternative algorithm. GHA provides an alternative with a low memory footprint, but takes progressively longer to produce each eigenvector, ultimately meaning that time is an issue. Since eigenvectors are produced in order starting with the greatest, however, requiring only a small number of eigenvectors mitigates this. GHA is quick to converge on the greatest of the eigenvectors. Additionally, GHA is of interest from the point of view of creation of a learning system that can develop a potentially very large LSA-style semantic model over a period of time from continuous streamed input. The learning behaviour of such a system would be of interest, and furthermore, there is potential for interesting performance features in a very large mature LSA-style model. The work presented in this section is previously published (24).

At a glance, GHA may not be an obvious candidate for LSA, since LSA is traditionally performed using singular value decomposition of paired data, to produce two sets of singular vectors. One set can be used to rotate wordbag vectors into the shared space created by the SVD process, and the other, document vectors. Since a document vector can be just as well represented as a wordbag vector, however, this is a little redundant. In fact, the primary task of LSA is to establish word interrelationships, and this is a task to which eigen decomposition is very well suited. In practical terms, using eigen decomposition for LSA simply involves using a word correlation matrix prepared over a set of training documents, which is square and symmetrical, to create an eigen decomposition, then using this decomposition to rotate the test set of documents presented as wordbags into the reduced dimensionality space, where they can be compared. The test set may be the same as the training set: this would in fact reflect standard LSA procedure. If \( M \) is our supposed standard LSA document by wordbag training matrix, we perform eigen decomposition
on the matrix $A = MM^T$, where $A$ describes word correlations, to produce the column matrix of eigenvectors $U$. $U$ is reduced in dimensionality by discarding later columns to produce $U'$. Test documents in the form of wordbags are presented as the row matrix $D$. $U'$ is used to reduce the dimensionality of $D$ as follows:

\[ D' = DU' \]  \hspace{1cm} (4.1)

Row document vectors in $D'$ can then be compared to each other.

### 4.1.1 Inclusion of Global Normalisation

LSA often includes an entropy-normalisation step (16), discussed in the previous chapter, in which word frequencies of the original data matrix are modified to reflect their usefulness as distinguishing features. Since this step has significant benefits, and has indeed become a part of the standard, no suggestion for an approach to performing LSA can be complete without its inclusion. This preprocessing requires that the entire corpus be available up-front, such that probabilities etc. can be calculated across the entire corpus, and therefore does not fit well with GHA, one of the main selling points of which is its incrementality. As outlined in the previous chapter, the word count is modified by setting the cell value $c_{ij}$ as follows\(^2\):

\[
    p_{ij} = \frac{tf_{ij}}{gf_i} \hspace{1cm} (4.2)
\]

\[
    gw_{ij} = 1 + \sum_j p_{ij} \frac{log(p_{ij})}{log(n)} \hspace{1cm} (4.3)
\]

\[
    c_{ij} = gw_{ij}(c_{ij} + 1) \hspace{1cm} (4.4)
\]

\(^2\)A known minor error in (16) has been corrected here.
where \( n \) is the number of documents in the collection. \( tf \) is the term frequency, i.e. the original cell count, and \( gf \) is the global frequency, i.e. 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\(^3\):

\[
gw_{ij} = 1 + \frac{\sum_j tf_{ij} \log(tf_{ij}) - gf_i \log(gf_i)}{gf_i \log(n)}
\]  

(4.5)

This arrangement has the convenient property of isolating the summation over a quantity that can be accumulated, i.e. \( tf_{ij} \log(tf_{ij}) \), whereas the previous arrangement would have required the individual term frequencies to be stored separately for an accurate calculation to be made. This is problematic where the number of such frequencies tends to infinity and the storage requirement increases as the tractability of the calculation decreases.

4.1.2 Epoch Size and Implications for Application

The entropy-normalised cell count 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

\(^3\)A minor error in (24) has been corrected here.
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:

$$gw_{ij} = 1 + \left( \frac{\frac{1}{2} \sum_i tf_j log(tf_{ij}) - log(gf_i) + log(n) - log(N_{epoch})}{log(N_{epoch})} \right)$$

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. Chapter 5 discusses the effect of increasing corpus size on term weighting effectiveness in more detail in the context of LSA for very large corpora.

Note that the impact of the introduction of the term weighting step on convergence has not been discussed here. An important next step in this work is the investigation and evaluation of the convergence properties of this variation of GHA on a large corpus. A limited investigation of the convergence behaviour of this variant of GHA follows in the next chapter.

4.2 Random Indexing

A shortcoming of GHA as an approach to LSA-style language processing is that the eigenvectors need to be extended in dimensionality each time a novel word appears. Whilst not in itself problematic, since dimensionality can be readily increased with an initialisation value as vocabulary increases, and a little training data rapidly moves

4A further minor error in (24) has been corrected here.
the value into the appropriate range, there are implementational advantages to knowing that dimensionality will not exceed a certain bound. For example, an upper limit may need to be included for tractability purposes. A modified approach is to assign each newly-encountered word a random, highly-sparse vector of fixed length, and to proceed as before accepting that the word vectors are now merely almost orthogonal to each other instead of truly orthogonal. This approach, Random Indexing (28), has been used as a dimensionality reduction technique competitive to eigen decomposition in its own right. Whilst lacking some of eigen decomposition’s technical advantages, its superior efficiency makes it more appropriate in some circumstances. Here, it is used as an approach to fixing dimensionality and reducing it for the sake of speed gains in conjunction with eigen decomposition.

Assigning a random vector to a word is easily done where word vectors are normalised and orthogonal. Using Random Indexing with arbitrary input vectors requires an extra step but is still relatively easy to arrange. For example, Random Indexing can be used in conjunction with LSA including the entropy normalisation step described in the previous section, in which data comprise bags of weighted words, as follows. The Random Indexing term-weighted vector for a particular document is constructed by first computing the entropy-normalised log count of each word separately, and then multiplying by that word’s random indexing vector, then summing over all words in the document. In matrix notation, this can be described using a multiplication of the input vector with the row matrix of Random Indexing vectors, one per word. That is, if \( R \) is the matrix of random vectors by words, our random-vector encoded datum \( a' \) is defined by:

\[
a' = aR
\]

Preliminary investigation suggests that Random Indexing is very effective in reducing dimensionalities in the region of tens of thousands and higher to as low as several hundred with tolerable impact on accuracy, though the reader is directed to the Random Indexing literature
for a thorough evaluation (28). Furthermore, Random Indexing has no apparent impact on the convergence of GHA. The two work well together. Note that at low dimensionalities (maybe hundreds), RI is not of benefit since RI dimensionalities would need to be higher to get good performance. The space of random vectors is too small at low dimensionalities.

As alluded to earlier in the section, dimensionality increase at run time is in itself not a problem. Indeed an advantage to GHA is that new vocabulary is quickly incorporated into an existing decomposition as its direction is determined based on similarity to existing features. Where an appropriate model of the data is already in existence, new data is simply positioned within its framework rather than used to any great extent to inform it. This is loosely parallel to a human learning a new word primarily in terms of the linguistic constructs they are already familiar with; part of speech etc. However, limiting the total dimensionality has tractability advantages. In the next section a sparse variant of GHA is presented which provides another approach to improving tractability where data is sparse. Random Indexing would be a preferable approach where data is not sparse.

4.3 GHA and Singular Value Decomposition

Extending GHA to paired data is desirable in that it allows us to apply the algorithm in contexts that we would otherwise be unable to. An example would be applying singular value decomposition to word n-grams. Modelling word n-grams is the core of tasks such as n-gram language modelling. Word n-grams can be thought of as word/history pairs, where a history is the $n - 1$ word sequence preceding the current word. Word n-grams can of course be modelled using standard singular value decomposition techniques, and this will be done later on in the work. This section discusses the extension of GHA to paired data in order to increase its applicability and to give us another tool at our disposal in the n-gram language modelling
work that follows later on in this thesis. The work presented in this
section is previously published (22).

Let us begin by recapping the Generalized Hebbian Algorithm, ex-
plained more thoroughly in the previous chapter.

\[
\mathbf{u}_n(t + 1) = \mathbf{u}_n + \lambda (\mathbf{u}_n^T \cdot \mathbf{a}) (\mathbf{a} - \sum_{i<n} (\mathbf{u}_i^T \cdot \mathbf{a}) \mathbf{u}_i) \quad (4.8)
\]

In the above, \( \mathbf{u}_n \) is the \( n \)th eigenvector, \( \mathbf{a} \) is the input vector (data
observation) and \( \lambda \) is the learning rate. The formula updates the
current eigenvector by adding to it the input vector multiplied by the
activation (dot product of \( \mathbf{a} \) with \( \mathbf{u}_n \)) minus the projection of the
input vector on all the eigenvectors so far. In vector form, expanding
\( y \) out and using the implicit learning rate as previously discussed:

\[
\Delta \mathbf{u}_n = \mathbf{u}_n \cdot \mathbf{a} (\mathbf{a} - \sum_{i<n} (\mathbf{a} \cdot \mathbf{u}_i) \mathbf{u}_i) \quad (4.9)
\]

Delta notation is used to describe the update here, for further read-
ability. The equation describes the quantity by which \( \mathbf{u} \) is modified.
The subtracted element is responsible for removing from the training
update any projection on previous singular vectors, thereby ensuring
orthogonality. 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:

\[
\Delta \mathbf{u} = \mathbf{u} \cdot \mathbf{a} (\mathbf{a}) \quad (4.10)
\]

Let us begin with a simplification of 4.10:
\[
\Delta u = \frac{1}{n} u A (A) \quad (4.11)
\]

Here, \( A \) is the entire data matrix. \( n \) is the number of training items. The simplification is valid in the case that \( u \) 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 \( M \), i.e. paired data, would therefore be achievable using standard GHA as follows:

\[
\Delta u = \frac{1}{n} u M M^T (M M^T) \quad (4.12)
\]

\[
\Delta v = \frac{1}{n} v M^T M (M^T M) \quad (4.13)
\]

In the above, \( u \) and \( v \) are left and right singular vectors. However, to be able to feed the algorithm with rows of the matrices \( M M^T \) and \( M^T M \), 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. However,

\[
\sigma u = v M^T = \sum_x (v \cdot b_x) a_x \quad (4.14)
\]

\[
\sigma v = u M = \sum_x (u \cdot a_x) b_x \quad (4.15)
\]
Here, $\sigma$ is the singular value pertaining to $u$ and $v$. The above is valid in the case that left and right singular vectors $u$ and $v$ have settled (which will become more accurate over time) and that data vector pairs $a$ and $b$ outer-product and sum to $M$, a state of affairs easily arranged in the case that the assumed matrix comprises the additive sum of the observation stream.

Inserting 4.14 and 4.15 into 4.12 and 4.13 allows them to be reduced as follows:

$$\Delta u = \frac{\sigma}{n} v M^T M M^T \quad (4.16)$$
$$\Delta v = \frac{\sigma}{n} u M M^T M \quad (4.17)$$

$$\Delta u = \frac{\sigma^2}{n} u M M^T \quad (4.18)$$
$$\Delta v = \frac{\sigma^2}{n} v M^T M \quad (4.19)$$

$$\Delta u = \frac{\sigma^3}{n} v M^T \quad (4.20)$$
$$\Delta v = \frac{\sigma^3}{n} u M \quad (4.21)$$

In the next step, the division by $n$ is cancelled with the summation over $x$ in equations 4.14 and 4.15:

$$\Delta u = \sigma^3 (v \cdot b) a \quad (4.22)$$
$$\Delta v = \sigma^3 (u \cdot a) b \quad (4.23)$$
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.

\[
\Delta u_i = v_i \cdot b(a - \sum_{j<i} (a \cdot u_j) u_j) \quad (4.24)
\]

\[
\Delta v_i = u_i \cdot a(b - \sum_{j<i} (b \cdot v_j) v_j) \quad (4.25)
\]

\[
\Delta u_i = u_i \cdot a(a - \sum_{j<i} (a \cdot u_j) u_j) \quad (4.26)
\]

In equations 4.11 and 4.14/4.15 we introduced approximations that become more accurate as the direction of the singular vectors settle. These approximations will therefore not interfere with the accuracy of the final result, though they might interfere with the rate of convergence. The constant \(\sigma^3\) has been dropped in 4.24 and 4.25. Its relevance is purely with respect to the calculation of the singular value. Recall that in Weng et al (52) the eigenvalue is calculable as the average magnitude of the training update \(\Delta u\). In our formulation, according to 4.22 and 4.23, the singular value would be \(\Delta u\) divided by \(\sigma^3\). Dropping the \(\sigma^3\) in 4.24 and 4.25 achieves that implicitly. The singular value is once more the average length of the training update.
4.4 Sparse Implementation

Where vectors are sparse, for example, perfectly orthogonal single word vectors, and dimensionality is high, for example, large vocabulary natural language tasks, there are enormous advantages to be gained by exploiting sparse algorithms, in which vectors take the form of index/non-zero value pairs rather than one value per dimension. Unmodified, GHA is not amenable to this approach because calculation of eigenvectors past the first one involves the removal of the projection of the previous eigenvectors (not sparse) from the input vector (sparse) after which it is no longer sparse. However, an alternate approach is, rather than removing the projections of previous eigenvectors \( \mathbf{u}_{<n} \) from input vector \( \mathbf{a}_n \), to remove them directly, or effectively, from the current eigenvector \( \mathbf{u}_n \).

We define the vector \( \mathbf{d} \) to be the product of training vector \( \mathbf{a} \) with the matrix of eigenvectors \( U \), that is to say, \( \mathbf{d} \) is a vector describing the extent to which \( \mathbf{a} \) projects on each eigenvector. This is a sparse operation, and produces the vector that in the original non-sparse GHA would be multiplied by the eigenvectors to produce a vector set each of which is subtracted from the training vector in order to orthogonalise it to previous eigenvectors.

\[
\mathbf{d} = U\mathbf{a}
\]  

(4.27)

We define helper vector \( \mathbf{h} \) and pseudo-eigenvector \( \mathbf{u}' \), the vector currently being trained, not necessarily orthogonal to previous eigenvectors, as follows,

\[
\mathbf{h} = U\mathbf{u}'
\]  

(4.28)

\[
\mathbf{u} = \mathbf{u}' - U^T\mathbf{h}
\]  

(4.29)

where \( \mathbf{u} \) is the current eigenvector. The above equations simply say
that pseudovector $\mathbf{u}'$ has the projection $\mathbf{h}$ on previous eigenvectors, therefore to obtain $\mathbf{u}$, the current eigenvector, this non-orthogonal part must be subtracted from $\mathbf{u}'$. The scalar quantity $d$, the activation, in Sanger’s terminology, is calculated in the original GHA as the dot product of the training datum with the current eigenvector. This can be calculated, avoiding the use of $\mathbf{u}$, a non-sparse vector, as follows:

\[ d = \frac{\mathbf{u}' \cdot \mathbf{a} - \mathbf{h} \cdot \mathbf{d}}{\sqrt{|\mathbf{u}'|^2 - |\mathbf{h}|^2}} \quad (4.30) \]

This is a much less expensive way to calculate $d$ where data is sparse. Having trained vector $\mathbf{u}'$ as follows,

\[ \Delta \mathbf{u}' = \mathbf{d} \mathbf{a} \quad (4.31) \]

all that remains is to calculate $\mathbf{u}$ with the aid of helper vector $\mathbf{h}$. Fortunately this vector can be updated incrementally, removing the remaining non-sparseness in the algorithm.

\[ \Delta \mathbf{h} = \mathbf{d} \mathbf{d} \quad (4.32) \]

Extended to paired data, this becomes,

\[ \mathbf{d}_u = \mathbf{U} \mathbf{a} \quad (4.33) \]
\[ \mathbf{d}_h = \mathbf{V} \mathbf{b} \quad (4.34) \]
\[ \mathbf{h}_u = \mathbf{U} \mathbf{u}' \quad (4.35) \]
\[ \mathbf{h}_h = \mathbf{V} \mathbf{v}' \quad (4.36) \]
\[ \mathbf{u} = \mathbf{u}' - U^T \mathbf{h}_a \]  
(4.37)  
\[ \mathbf{v} = \mathbf{v}' - V^T \mathbf{h}_b \]  
(4.38)  
\[ d_a = \frac{\mathbf{u}' \cdot \mathbf{a} - \mathbf{h}_d \cdot d_a}{\sqrt{\|\mathbf{u}'\|^2 - \|\mathbf{h}_a\|^2}} \]  
(4.39)  
\[ d_b = \frac{\mathbf{v}' \cdot \mathbf{b} - \mathbf{h}_b \cdot d_b}{\sqrt{\|\mathbf{v}'\|^2 - \|\mathbf{h}_b\|^2}} \]  
(4.40)  
\[ \Delta \mathbf{u}' = d_b \mathbf{a} \]  
(4.41)  
\[ \Delta \mathbf{v}' = d_a \mathbf{b} \]  
(4.42)  
\[ \Delta \mathbf{h}_a = d_b \mathbf{d}_a \]  
(4.43)  
\[ \Delta \mathbf{h}_b = d_a \mathbf{d}_b \]  
(4.44)

Where sparse data is presented in sparse format, the above sparse formulation produces great benefits in terms of calculation complexity. The cost of the training step using the original formulation where data is not sparse is linear with data dimensionality. Using the sparse formulation, the cost of the training step becomes linear with the number of non-zero elements in the vector, much reducing the cost of increasing data dimensionality. Complexity is quantified in the next chapter.

### 4.5 Convergence

As discussed previously in the context of the Generalized Hebbian Algorithm, the practicality of GHA and its variants depends on the accuracy of the final result and the efficacy and reliability with which that result is approached. This section discusses a number of implementation issues that need to be resolved with regard to convergence. The section is based on my practical experience with implementing GHA and its variants, and does not present published work.
4.5 Convergence

4.5.1 Staged Training

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. Observe that the second singular vector pair is trained on the datum minus its projection onto the first singular vector pair. This is 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 converged, or close to converged, before beginning to train the next.

4.5.2 Convergence Criteria

The challenge in designing an effective convergence criterion for the GHA and its variants lies in finding a correlate to convergence that
does not depend on knowing the actual position of the target singular vectors. Such criteria can be evaluated on test sets for which we have an independent decomposition available, produced by a known and trusted implementation such as SVDLIBC (1). The chosen criterion can then be applied “in earnest”, in the case that we are using GHA to produce a decomposition we actually need.

Finding a correlate to convergence proves surprisingly challenging. Here are some ideas:

- The length of the feature vector in the case that the implicit learning rate is used
- The extent to which the vector has moved over a certain number of training steps
  Movement can be gauged as the dot product between the current feature vector and the feature vector in its earlier position
  Movement can be gauged as the distance between the end of the (normalised) feature vector and the feature vector in its earlier position
- The extent to which the training updates differ from the feature vector

Each of these ideas will be discussed in more detail in the remainder of this section.

**Length-Based Convergence Criterion**

Note that this approach is specific to the case in which the training vector is not orthogonalised to the current feature vector prior to being used to train it, such that the feature vector grows long over time. Using length of the feature vector to determine convergence has repeatedly shown itself to be the most effective option, and has been
used previously by other researchers (52). Disadvantages, as shown earlier, include that it has a tendency to not quite make it to the target vector, and instead settle in an inaccurate direction that further data has a hard time influencing since the vector has grown so long and stable. Advantages include its stability and predictable progress toward convergence (in terms of close-to-linear rate of approaching the criterion).

**Movement-based Convergence Criterion**

Where the vector is allowed to grow long, the movement-based convergence criterion in practical terms is little more than a variant on the length-based convergence criterion. The greatest influence on the extent to which the feature vector moves over a certain number of training iterations is the length of the vector, since this determines how much of an impact each training datum has on the overall length. Note that determining movement through the distance between the ends of the normalised feature vectors over time would be preferable to using dot product were this approach to be taken, since dot product is not a good measure of very small differences in vector direction.

In the case that autonormalisation is used, that is to say, the training datum is orthogonalised to the current feature vector prior to being used to train it, such that the vector remains (approximately) normalised, and an explicit fixed learning rate is used, movement does not appear to correlate with convergence, but instead remains more or less constant, suggesting that the vector approaches the target then jumps around in the vicinity rather than noticeably settling.

**Similarity of Training Vectors to Feature Vectors**

As a measure of convergence, the similarity of the training data to the feature vectors is not entirely without promise. The theory is that the target direction for the feature vector is that which captures
as much of the training data as one vector possibly can, and so a
good direction for it is one in which the dot products of the training
vectors with the feature vector are low. Note that the calculation
of the dot product of the training vector with the feature vector is
a critical part of the Generalized Hebbian Algorithm anyway, which
would make it efficient to use. One way of measuring this might be
as follows, where $\theta$ is the quantity we hope will correlate well with
convergence, $\gamma$ is a heuristically-chosen rate of extinction (100 seems
about right) and $d$ is the dot product of the training vector with the
current feature vector:

$$\theta = \frac{\theta(\gamma - 1) + d}{\gamma}$$

This quantity reveals itself to broadly correlate with convergence.
However, it is unpredictable close to convergence, and its absolute
value depends on the data and on the eigen/singular value of the cur-
rent feature vector, making it difficult to use. (In fact, this quantity
should converge on the eigen/singular value.) Further develop-
ment might allow this measure to be used, but in its current form it has
not proved particularly successful.

### 4.5.3 Learning Rates

A closely related problem to that of selecting a convergence criterion
is selecting a learning rate. Note that Section 4.5.4 presents con-
vergence curves for two different approaches to learning rate selec-
tion. The first is the implicit learning rate, in which the vector is allowed
to grow long, such that later data has less impact on the direction
of the feature vector. As the graphs show, this approach is stable
but has the disadvantage that sometimes the vectors settle poorly,
and then become so stable that it is difficult to move them closer to
the target. The advantage to the approach however is that it is very
stable. The vector will not tend to jump away from the target having
4.5 Convergence

reached it.

The second learning rate shown in the convergence curves is Sanger’s original explicit learning rate. This approach is shown to reliably and quickly produce converged vectors, but lacks stability having reached its target. Indeed, overly high learning rates can lead to the vector jumping between competing directions (other strong eigen/singular vectors). Combating this problem with a low learning rate compromises speed of convergence, and often still fails to eliminate instability.

Ideally, a learning rate based on the convergence criterion, that reduces as the vector approaches the target, would be developed. Such a learning rate would optimise speed of convergence initially, and stability later on. An attempt has been made to develop such a learning rate based on the convergence criterion discussed earlier of average dot product of the training datum with the feature vector. This attempt has not met with success. The implicit (length-based) learning rate proves remarkably difficult to beat. This remains however an area in which future work might fruitfully be done to improve the performance of the Generalized Hebbian Algorithm and its variants.

4.5.4 Asymmetric GHA Convergence Evaluation

A corpus of 20,000 bigrams was used to produce convergence curves for AGHA. The dataset comprised bigrams from a set of 100 documents from the Newsgroup corpus (11). The data were presented as pairs of unit-length word vectors. The dimensionality of the implied matrix was a little over 4000 by 4000. The variant of AGHA predominantly used in this work uses the implicit learning rate. This algorithm is contrasted here with the explicit learning rate as presented in Sanger’s original GHA formulation. The learning rate used was 0.1 throughout. Figures 4.1, 4.2 and 4.3 present left singular vectors 1, 2 and 3 respectively. Figures 4.4, 4.5 and 4.6 present right singular vectors 1, 2 and 3. Dot-product with a reference vector set produced using SVDLIBC (1) is plotted against number of training
The convergence graphs show that AGHA, both with implicit and explicit learning rates, have little difficulty in locating the first singular vector pair. On this pair, the implicit learning rate produces a curve that most rapidly approaches convergence and is most stable. The explicit learning rate curve jumps around. Experimentation showed that higher learning rates produced more rapid convergence but at the expense of later stability. On the second singular vector pair, AGHA with explicit learning rate is the only variant to achieve reasonable convergence in the time allowed. Further time would allow the vectors to achieve better convergence. On the third, again, AGHA with explicit learning rate is the only variant that really converged, with the AGHA with implicit learning rate in this case coming closer. Again, the implicit learning rate leads to a greater stability. Recall that later singular vector pairs are trained on the input minus earlier singular vectors, so pairs following a poorly converged pair are at a disadvantage. In general it seems that both implicit and explicit learning rates have their advantages in terms of stability and
Figure 4.2: *Second Left Singular Vector Convergence*

Figure 4.3: *Third Left Singular Vector Convergence*
Figure 4.4: First Right Singular Vector Convergence

Figure 4.5: Second Right Singular Vector Convergence
rapidity of convergence. The implicit learning rate has the big disadvantage that having had the opportunity to grow long, it becomes overly stable and resistant to change. However, it is at least stable and predictable. Some hybrid of the two approaches might lead to the most optimal result. The results presented here are representative of experimentation with other corpora and might be expected to apply across a range of circumstances, though there is always the possibility that different datasets might produce different behaviour. The results presented here are intended to illustrate behaviour rather than performance quality. For the most optimal results, using the approach described in this chapter of training each singular vector pair for a while before starting to train the next, and allowing them to train for longer would be recommended.
4.6 Summary

This section has covered a number of variations on the basic GHA theme, many of which will be used later in the work. To summarise briefly:

- GHA has been extended with an original adaptation that makes it possible to perform Latent Semantic Analysis on a dataset including the entropy normalisation step without any preprocessing of the corpus (24). This is relevant because an advantage to GHA is that it can be used to decompose very large datasets "incrementally", such that the entire dataset need not be available from the start. For such an advantage to be meaningful in the case of LSA, the most popular of the term weighting schemes needs to be accounted for.

- Random Indexing has been presented as a potentially advantageous addition to the algorithm from the point of view of tractability.

- An original asymmetric variant to GHA has been presented (22), the Asymmetric Generalized Hebbian Algorithm, which allows paired data to be handled, effectively performing singular value decomposition.

- Original sparse variants of GHA and AGHA have been presented, offering a much more efficient way to apply GHA and AGHA to sparse datasets such as are ubiquitous in language processing.

- Practical aspects of convergence and learning rate selection are discussed. Evaluation has been performed such as to inform the choice of learning rate and convergence strategy favoured in the remainder of the work.

Relevant work by others includes Sanger's Double Generalized Hebbian Algorithm and Orthogonal Asymmetric Encoder (47). These
two algorithms are similar to AGHA and are presented as an attractive solution in the case that an unknown matrix is to be decomposed using paired inputs and outputs to that matrix, where output is the matrix multiplied by the input. Inputs should be orthogonal and uniformly distributed, such that the row matrix of inputs multiplied by its transpose is the identity matrix, or at least a scaled identity matrix. This makes the DGHA and the OAE applicable in a subset of the cases in which AGHA can be used; AGHA will produce the singular value decomposition of a matrix based on arbitrary paired input.

Berry’s LAS2 (4) algorithm will continue to be used as a control and a contrast to the GHA-based algorithms in the remainder of this work. The implementation used is the open-source SVDLIBC (1). Matlab’s SVD function is also used in places.

Other algorithms of interest include the Kernel Hebbian Algorithm (29), in which eigen decomposition is performed on data in a Reproducing Kernel Hilbert Space (RKHS). In other words, the data undergoes a kernel mapping that allows non-linear structure to be extracted in the data. As language undoubtedly contains non-linearity, this work seems particularly relevant, and application and extension within natural language processing could lead to interesting results in the future.

The next two chapters discuss GHA and AGHA in the context of two natural language processing tasks; information retrieval and stochastic language modelling.
Chapter 5

GHA for Information Retrieval

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 (14). 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 (3). LSA is typically formulated in terms of large matrix operations on the corpus as a whole, and is thus principally a batch algorithm although methods offering varying degrees of incrementality are available, as discussed in the previous chapter. This chapter falls into two parts. In the first, GHA is demonstrated in the context of performing LSA on a small dataset. The convergence of LSA-variant GHA is shown. The second part of the chapter focuses on GHA’s extensibility to large datasets and discusses the scalability of LSA in general. Complexity analysis and memory consumption of generic GHA as compared to a standard approach (LAS2) are discussed.
5.1 GHA for LSA

LSA is traditionally performed using singular value decomposition, where GHA performs eigen decomposition. However, as discussed in the previous chapter, singular value decomposition and eigen decomposition are intimately related, and it is easy to map from one to the other. In fact, LSA is a classic example of a task that involves relating symmetrical data to each other: word $a$'s occurrence with word $b$ is exactly the same as word $b$'s occurrence with $a$. In choosing the tool appropriate for the job, the task might be better thought of in these terms. This section simply illustrates the use of GHA to perform LSA on a small dataset. The use of a small dataset allows the behaviour of the technique to be qualitatively examined.

5.1.1 Method

The 20 Newsgroups dataset 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, including the entropy normalisation modification, 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 entropy normalisation preprocessing step was carried out on the data prior to presenting it to Matlab.) 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. Here, however, the singular value decomposition of the matrix was taken, and the right (document space) vector set discarded. 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.

5.1.2 Results

A comparison of the results for the first ten eigenvalues is shown in table 5.1. “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 5.1 shows the algorithm’s convergence on the correct vector directions. The x-axis shows the number of data items presented. Two graphs are overlaid here for comparison purposes; absolute eigenvalue and directional error, again defined to be one minus the dot product of the eigenvector with the Matlab eigenvector. The scale on the left pertains to the error in vector direction, and is plotted using points, and the scale on the right pertains to the eigenvalue, plotted using lines. 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. Whilst stringency of convergence criterion would need to be controlled for, the suggestion is that including entropy normalisation in GHA does not impact unfavourably on convergence.

The vector set produced by GHA differs from standard LSA in that, being an eigen decomposition of the wordbag vectors, no documentspace vector set is produced. Traditionally, LSA makes no distinction between training and test sets. The entire document set is included in the singular value decomposition, and it is these documents that the intention is to compare to each other. This is not possible with this approach, only the word space being produced (or indeed only the document space, but the word space is more useful), and indeed, 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 have a reduced representation of every training doc-
Figure 5.1: *Convergence of GHA with Entropy Normalisation Included*
5.1 GHA for LSA

Table 5.1: Comparison of GHA and LSA.

<table>
<thead>
<tr>
<th>Number</th>
<th>Error</th>
<th>GHA Value</th>
<th>LSA Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>1.2874603E-5</td>
<td>1.957</td>
<td>1.972</td>
</tr>
<tr>
<td>1</td>
<td>3.6120415E-5</td>
<td>1.333</td>
<td>1.339</td>
</tr>
<tr>
<td>2</td>
<td>1.2278575E-5</td>
<td>0.734</td>
<td>0.757</td>
</tr>
<tr>
<td>3</td>
<td>1.928803E-4</td>
<td>0.568</td>
<td>0.575</td>
</tr>
<tr>
<td>4</td>
<td>1.916654E-4</td>
<td>0.397</td>
<td>0.445</td>
</tr>
<tr>
<td>5</td>
<td>8.904934E-5</td>
<td>0.315</td>
<td>0.381</td>
</tr>
<tr>
<td>6</td>
<td>2.5985625E-5</td>
<td>0.403</td>
<td>0.316</td>
</tr>
<tr>
<td>7</td>
<td>3.234148E-4</td>
<td>0.279</td>
<td>0.284</td>
</tr>
<tr>
<td>8</td>
<td>2.4973465E-4</td>
<td>0.248</td>
<td>0.267</td>
</tr>
<tr>
<td>9</td>
<td>1.5366077E-4</td>
<td>0.254</td>
<td>0.245</td>
</tr>
</tbody>
</table>

ument would consume a lot of space, and for most tasks would be unnecessary. To perform LSA-style tasks with eigen decomposition a document set appropriate to the task is selected and used to form a document space in the manner formalised in Chapter 2; by forming a column matrix from the wordbag vectors and multiplying by the left singular vector set in the manner of any new pseudodocument. 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 run time. This approach is more analogous to the separate training and test sets characteristic of many machine learning approaches. Where the “test set” is novel to the decomposition, the approach might be considered inaccurate, (see “folding in” (16)) but in fact, if the “training set” is sufficiently representative of the test set, then even if the test set is novel, the approximation is hard to fault.

To clarify, the method can be outlined as follows. The test document set is formed into a set of wordbag “pseudodocuments”, forming rows of \( D \). The eigenvector set \( U \) is multiplied by this matrix to produce a reduced dimensionality representation \( D' \). This representation is as it would have been had singular value decomposition been used and the original document vectors used to create the reduced dimensionality representation.
\[ D' = U D^T \] (5.1)

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. Although mathematically identical to standard LSA, the reader might benefit from a more realistic illustration than was provided in Chapter 2 and a practical demonstration of the use of eigen decomposition to perform LSA. Ten documents were chosen, five from \textit{atheism} and five from \textit{hockey}. Tables 5.2, 5.3 and 5.4 constitute a many-to-many comparison of these documents, via the dot product. \textit{Atheism} items are denoted with a preceding “a” in the document name, and \textit{hockey} documents with a preceding “h”. The reader should find herself able to locate the given documents in the corpus using the name, should she wish to do so. Tables 5.5, 5.6 and 5.7 give the comparison matrix produced using unprocessed documents, that is to say, vectors of raw word counts, in order to illustrate the impact of LSA.

<table>
<thead>
<tr>
<th></th>
<th>a:53366</th>
<th>a:53367</th>
<th>a:51247</th>
<th>a:51248</th>
<th>a:51249</th>
</tr>
</thead>
<tbody>
<tr>
<td>a:53366</td>
<td>1.00</td>
<td>0.86</td>
<td>0.86</td>
<td>0.83</td>
<td>0.91</td>
</tr>
<tr>
<td>a:53367</td>
<td>0.86</td>
<td>1.00</td>
<td>0.82</td>
<td>0.77</td>
<td>0.76</td>
</tr>
<tr>
<td>a:51247</td>
<td>0.86</td>
<td>0.82</td>
<td>1.00</td>
<td>0.74</td>
<td>0.80</td>
</tr>
<tr>
<td>a:51248</td>
<td>0.83</td>
<td>0.77</td>
<td>0.74</td>
<td>1.00</td>
<td>0.80</td>
</tr>
<tr>
<td>a:51249</td>
<td>0.91</td>
<td>0.76</td>
<td>0.79</td>
<td>0.80</td>
<td>1.00</td>
</tr>
</tbody>
</table>

Observe that in general, the numbers in tables 5.2 and 5.3, are larger than those in 5.4. This shows a predictable tendency for \textit{atheism} documents to be more like other \textit{atheism} documents and for \textit{hockey} documents to be more like other \textit{hockey} documents. Observe also that the clustering effect is less strong in tables 5.5, 5.6 and 5.7. 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
Table 5.3: Many-to-many document comparison with LSA.

<table>
<thead>
<tr>
<th></th>
<th>h:54776</th>
<th>h:54777</th>
<th>h:54778</th>
<th>h:54779</th>
<th>h:54780</th>
</tr>
</thead>
<tbody>
<tr>
<td>h:54776</td>
<td>1.00</td>
<td>0.42</td>
<td>0.88</td>
<td>0.84</td>
<td>0.92</td>
</tr>
<tr>
<td>h:54777</td>
<td>0.42</td>
<td>1.00</td>
<td>0.45</td>
<td>0.39</td>
<td>0.40</td>
</tr>
<tr>
<td>h:54778</td>
<td>0.88</td>
<td>0.45</td>
<td>1.00</td>
<td>0.85</td>
<td>0.88</td>
</tr>
<tr>
<td>h:54779</td>
<td>0.84</td>
<td>0.39</td>
<td>0.85</td>
<td>1.00</td>
<td>0.90</td>
</tr>
<tr>
<td>h:54780</td>
<td>0.92</td>
<td>0.40</td>
<td>0.88</td>
<td>0.90</td>
<td>1.00</td>
</tr>
</tbody>
</table>

Table 5.4: Many-to-many document comparison with LSA.

<table>
<thead>
<tr>
<th></th>
<th>h:54776</th>
<th>h:54777</th>
<th>h:54778</th>
<th>h:54779</th>
<th>h:54780</th>
</tr>
</thead>
<tbody>
<tr>
<td>a:53366</td>
<td>0.78</td>
<td>0.22</td>
<td>0.63</td>
<td>0.52</td>
<td>0.70</td>
</tr>
<tr>
<td>a:53367</td>
<td>0.63</td>
<td>0.15</td>
<td>0.44</td>
<td>0.38</td>
<td>0.55</td>
</tr>
<tr>
<td>a:51247</td>
<td>0.66</td>
<td>0.18</td>
<td>0.44</td>
<td>0.40</td>
<td>0.58</td>
</tr>
<tr>
<td>a:51248</td>
<td>0.78</td>
<td>0.23</td>
<td>0.63</td>
<td>0.61</td>
<td>0.77</td>
</tr>
<tr>
<td>a:51249</td>
<td>0.78</td>
<td>0.23</td>
<td>0.64</td>
<td>0.57</td>
<td>0.73</td>
</tr>
</tbody>
</table>

word counts select “h:54776” as the most similar document. This is a document from the hockey section, and as 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.

5.2 LSA and Large Training Sets

Latent Semantic Analysis runs into some difficulties with larger datasets. Published LSA results usually do not attempt singular value decomposition on datasets larger than around 80,000 by 80,000 dimensions, that is to say, vocabularies of around 80,000 and document sets comprising a similar number of documents (49). The rea-
Table 5.5: Many-to-many document comparison without LSA.

<table>
<thead>
<tr>
<th>a:53366</th>
<th>a:53367</th>
<th>a:51247</th>
<th>a:51248</th>
<th>a:51249</th>
</tr>
</thead>
<tbody>
<tr>
<td>1.00</td>
<td>0.52</td>
<td>0.54</td>
<td>0.33</td>
<td>0.53</td>
</tr>
<tr>
<td>0.52</td>
<td>1.00</td>
<td>0.54</td>
<td>0.40</td>
<td>0.47</td>
</tr>
<tr>
<td>0.54</td>
<td>0.54</td>
<td>1.00</td>
<td>0.35</td>
<td>0.57</td>
</tr>
<tr>
<td>0.33</td>
<td>0.40</td>
<td>0.35</td>
<td>1.00</td>
<td>0.35</td>
</tr>
<tr>
<td>0.53</td>
<td>0.47</td>
<td>0.57</td>
<td>0.35</td>
<td>1.00</td>
</tr>
</tbody>
</table>

Table 5.6: Many-to-many document comparison without LSA.

<table>
<thead>
<tr>
<th>h:54776</th>
<th>h:54777</th>
<th>h:54778</th>
<th>h:54779</th>
<th>h:54780</th>
</tr>
</thead>
<tbody>
<tr>
<td>1.00</td>
<td>0.24</td>
<td>0.62</td>
<td>0.57</td>
<td>0.63</td>
</tr>
<tr>
<td>0.24</td>
<td>1.00</td>
<td>0.20</td>
<td>0.18</td>
<td>0.20</td>
</tr>
<tr>
<td>0.62</td>
<td>0.20</td>
<td>1.00</td>
<td>0.55</td>
<td>0.57</td>
</tr>
<tr>
<td>0.57</td>
<td>0.18</td>
<td>0.55</td>
<td>1.00</td>
<td>0.71</td>
</tr>
<tr>
<td>0.63</td>
<td>0.20</td>
<td>0.57</td>
<td>0.71</td>
<td>1.00</td>
</tr>
</tbody>
</table>

son for this is that the algorithms typically used to perform singular value decomposition do not scale well to larger datasets. This section compares Berry's LAS2 (4) algorithm for performing singular value decomposition to GHA in order to provide information in deciding which algorithm is most appropriate to a given LSA task. LAS2 has been chosen because it is a popular algorithm optimised for sparse datasets and capable of handling large amounts of data. (The better-known Matlab is more restrictive in this respect.) LAS2 is a Lanczos-based approach. Lanczos-based algorithms derive from the fact that a random vector repeatedly multiplied by a matrix will converge on the first eigenvector of that matrix (power method). Specifically, the open-source implementation SVDLIBC (1) has been used here.

5.2.1 Memory Usage

Leading algorithms for deriving the eigen/singular value decomposition of a matrix typically run into difficulties with RAM, placing an
5.2 LSA and Large Training Sets

<table>
<thead>
<tr>
<th></th>
<th>h:54776</th>
<th>h:54777</th>
<th>h:54778</th>
<th>h:54779</th>
<th>h:54780</th>
</tr>
</thead>
<tbody>
<tr>
<td>a:53366</td>
<td>0.57</td>
<td>0.18</td>
<td>0.44</td>
<td>0.46</td>
<td>0.54</td>
</tr>
<tr>
<td>a:53367</td>
<td>0.51</td>
<td>0.20</td>
<td>0.38</td>
<td>0.40</td>
<td>0.45</td>
</tr>
<tr>
<td>a:51247</td>
<td>0.60</td>
<td>0.22</td>
<td>0.51</td>
<td>0.46</td>
<td>0.61</td>
</tr>
<tr>
<td>a:51248</td>
<td>0.42</td>
<td>0.19</td>
<td>0.27</td>
<td>0.29</td>
<td>0.29</td>
</tr>
<tr>
<td>a:51249</td>
<td>0.61</td>
<td>0.18</td>
<td>0.50</td>
<td>0.47</td>
<td>0.57</td>
</tr>
</tbody>
</table>

upper limit on the amount of data that can be processed. LAS2 (4), as a rough guide, needs 100 megabytes of RAM to process a matrix of dimensionality around 50,000 by 50,000. More specifically, the memory required is \((10 + D + q)N + (4 + q)q\) (presumably bytes), where \(N\) is the width plus the height of the data matrix, \(D\) is the number of singular vectors to return and \(q\) is \(N\) or 600, whichever is lowest (31). Space is also required for the data matrix itself. For a small \(D\), such as is typical in LSA-style tasks, this approximates 600\(N\) bytes plus space for the matrix. In contrast, the GHA algorithm requires \(4(N + 1)D\) bytes, which is much less, and scales better. Space is not required for the data matrix.

5.2.2 Time

The computational complexity of the LAS2 algorithm is \(O(3Dz)\) (31) where \(z\) is the number of non-zero elements and \(D\) is the number of dimensions returned. This means that as the number of non-zero elements in the matrix increases, time taken to perform the decomposition increases at three times that rate. An algorithm optimised for sparse matrices is highly desirable in a natural language context. The computational complexity of GHA is more complicated to present. For the original GHA algorithm, the training step itself has a linear complexity with the dimensionality of the (symmetric) data \(w\) \(O(w)\). Likewise for the asymmetrical formulation \(O(N)\). For the sparse versions, the training step is linear with the number of non-zero elements in the training data \(O(z)\) or where this number does
not remain constant, the average. However, this is only half the story. The number of training steps required to reach convergence depends on the nature of the data, and convergence may not be predictable and consistent depending on data and initial conditions. The best way to get a clear feel for the time it takes to decompose a dataset using the algorithm is to study convergence curves.

5.2.3 Scaling LSA

It is an ongoing research problem to improve the scalability of LSA. Various solutions have been proposed. Random Indexing (28) provides a far more scalable approach to dimensionality reduction than singular value decomposition. Indeed, it can be used as a tool for improving scalability in conjunction with singular value decomposition. Preliminary investigation has shown the approach to be of great utility in fixing the dimensionality of the training vectors to GHA and AGHA and thereby producing performance improvements on large corpora with negligible impact on accuracy. Sparse formulations however reduce the necessity for dimensionality reduction of
training data. Other approaches are based on splitting the data into
groups, and then performing singular value decomposition on each
group (49). This approach has the desirable side-effect of improving
the effectiveness of the entropy-based row normalisation step typically
used in conjunction with LSA. Entropy of word occurrence becomes
less effective as a measure of semantic significance as the number of
observations approaches infinity (24). Were larger datasets able to
be decomposed using singular value decomposition, a solution would
need to be found to this problem. The row normalisation step would
possibly need to be improved in order for the method to remain effec-
tive. Figure 5.2 presents plots of hypothetical idealised word occur-
rence entropy curves, in which words with probabilities of 0.1 clump
themselves in every tenth, thirtieth and one hundredth document re-
spectively. The curves illustrate that despite the extremely regular
pattern of the hypothetical words’ occurrence, entropy is still drop-
ning at 100,000 repetitions (clumps). The elevated entropies at lower
document counts are for whatever reason more practically useful in
the LSA task.

In the case that a training set contains many more documents than
unique words, there might be some utility from a tractability point of
view in squaring the dataset to create a word co-occurrence matrix,
and performing eigen decomposition on this instead, if standard algo-
rithms are to be preferred. A dataset containing a million documents
and having a vocabulary of 100,000 might be squared to produce a
100,000 by 100,000 word co-occurrence matrix which can then be de-
composed more easily than the original matrix. Documents required
for the task can then be mapped into reduced-dimensionality space
using the eigenvector set as described earlier in the chapter. The as-
sumption here is that the sparsity and dimensionality of such a data
matrix would decrease more slowly than the size of the document
set would increase. Less significant words could also be excluded to
further improve tractability.

Whilst hardly an approach to scaling LSA, Probabilistic LSA (26))
is increasingly being regarded as an attractive alternative. Based on
probability theory, it applies in many of the same situations. It is not
clearly more scalable than LSA, but it does have different properties,
and so future work might make it potentially preferable in terms of tractability as well as from a theoretical point of view. The problem of establishing semantic relationships between words based on a smaller number of underlying concepts fits very well into a probabilistic framework. The reduced dimensionality representation produced in SVD can only loosely be thought of in terms of probabilities, a consideration that becomes relevant later in this work, as SVD is applied to n-gram language modelling.

5.3 Summary

This chapter began with an illustration of LSA-variant GHA on a typical LSA-style task, before moving on to consider GHA, as contrasted with LAS2, in terms of computational complexity and memory consumption. Approaches to increasing the scalability of LSA have also been discussed, since these may in the future have an impact on the requirements of the task. Previous attempts to apply GHA to LSA include Delichere and Memmi’s work (15). This work does not focus on scalability however, using a very small corpus and a stop list to further reduce dimensionality. The fact that the LSA task requires only the top few hundred singular vector pairs is critical for the viability of GHA in this context. The fact that later vector pairs take longer to converge upon is a great disadvantage to the algorithm in contexts where more complete decompositions are required. Memory issues will be eased by the progress of technology. However, scalable memory consumption will always be an advantage, and the practical value of an algorithm with minimal memory consumption can be great. Future work will include an exploration of the performance of LSA as the corpus size is increased beyond that traditionally used.
Chapter 6

SVD and Language Modelling

Many tasks within natural language processing are made easier, or indeed, possible, through being able to refer to an appropriate model of the language. Decoding a speech stream into words, for example, without the aid of a language model, would be extremely difficult because there are too many ways in which phonemes can be combined, and the chances of the correct interpretation being uniquely distinguishable from the others are small. Interference in the input, such as speaker variation and background noise, makes it difficult to identify the words that were spoken. Furthermore, the range of possibilities is so large that it would take a long time for a system to work through them all. Searching the entire space of phoneme combinations however is not necessary because only a limited number are meaningful within the language. We can more or less rule out the others, leaving us with a much reduced number.

Creating a model of allowable sequences is a difficult task. Natural languages are complex, and so designing a good language model is challenging. Approaches fall loosely into two camps; data driven
and theory-driven. The data driven approach typically uses a simple structure, modelling the target language as a series of $n$-grams or $n$ word sequences, and assigning probabilities to words given the preceding $n$-1 based on large numbers of observations from the language (33) (8). Theory-driven approaches use a more structured human-designed representation of the language, often in the form of a context-free grammar, and may or may not use probabilities derived from a corpus (33).

Smoothing is an ongoing area of research within $n$-gram language modelling. There is a huge, indeed, infinite, variety of allowable utterances in a natural language. So a given corpus, no matter how huge, cannot hope to represent all the things that a person might say, let alone provide representative statistics. For this reason, many things a person might say to a speech recognition system will never have been encountered in the corpus. Unseen words and word sequences will always come up. Naively assigning probabilities to $n$-grams based on proportion of observations of a word given a particular $n$-1 word history will lead to a language model’s allocating zero probability to sequences that were never seen in the corpus, but this is disastrous for performance. Smoothing, or discounting, describes the process of allocating some probability space to the unseen combinations. For example, a trivial smoothing technique would be to allocate a fixed, small probability to each possible $n$ word combination. There are various ways of determining an appropriate fixed probability. Other, more advanced techniques involve defining back-off schemes to lower order $n$-grams. Chen and Goodman provide an excellent review of the state of the art, as well as adding a variant of their own (10). Ultimately, however, it is extremely difficult to distinguish between plausible sequences that happen not to have come up in the training corpus, implausible sequences, and indeed, implausible sequences that did come up in the corpus, for whatever reason.

Class $n$-gram language modelling (8) is akin to smoothing in that the approach leverages probabilities from words grouped in some way with the target word to again overcome the data sparsity problem. This chapter describes a proposed technique that can loosely be considered smoothing, based on singular value decomposition. It
smooths in the sense that SVD allows the probability distribution to be desparsified meaningfully, by mapping into a continuous reduced-dimensionality space. The approach might also be considered a form of class n-gram language modelling, in that SVD is used to produce a set of superfeatures over which the probabilities are then calculated. The superfeatures might be thought of as word classes.

N-gram language modelling dominates the state of the art currently, but it is arguably close to its limit in terms of performance. It is easier to see how the theory-driven approach, in the form of, for example, context-free language models, might be improved and come to be superior (45). Comparison of the two approaches suggests that the relative merits of each approach depend on the task being performed (30). There is also much potential to combine the two approaches, to leverage the strengths of each (23) (20) (21). Grammar-based language modelling is somewhat of a younger science. Not producing the better large-scale results, it has attracted less attention. However, work such as Rayner et al’s (42) begins to explore the qualities of a good grammar-based language model. Grammar-based language models can also be amenable to circumstances where flexible coverage is required (44) (43).

The chapter falls into three parts. In the first, the behaviour of singular value decomposition in modelling word n-grams is qualitatively examined. The second part introduces the SVD-based language model, or SVDLM, in which an n-gram language model is decomposed using SVD and reduced in dimensionality. The impact of dimensionality reduction on the performance of the language model is investigated. The third part interpolates the SVDLM with a standard single order n-gram language model, that is to say, one comprising only one order of n-gram, with no back-off to lower orders included. Performance is investigated.
6.1 Modelling Letter and Word Co-Occurrence

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 are paired. 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. Words appearing after word a are very different to words appearing before it. 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 a great extent to a rule system which governs the occurrence of the individual instances that form its surface manifestation. 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 data 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 generalised system that applies even to instances not encountered at training time.

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 a letter-level model of the language would be appropriate. Section 6.1.1 discusses results on the word bigram task and section 6.1.2 discusses results on the letter bigram task. All results were calculated using the non-sparse Asymmetric Generalised Hebbian Algorithm, although other algorithms would have served the purpose equally well.
6.1 Modelling Letter and Word Co-Occurrence

6.1.1 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. As discussed earlier, length forms a reasonable heuristic for deciding if vectors are settled enough to begin training the next pair. 2000 was chosen ad hoc based on observation of the behaviour of the algorithm during training.

In order to qualitatively examine the behaviour of singular value decomposition on word bigram data, the two strongest singular vector pairs are presented here, that is to say, the two singular vector pairs with the highest singular values. Note that these vector pairs form only a tiny fraction of the complete description of bigram behaviour, being just two of thousands. They are however interesting, in that they are the greatest. The vectors are presented in the form of a list of the words whose vectors the singular vectors most strongly resemble. Dot products with these word vectors are also given. In this way, we can think of the singular vectors in terms of the words they are most similar to. Table 6.1 shows the words with highest dot products with 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 6.1: Top words in 1st singular vector pair

<table>
<thead>
<tr>
<th>Vector 1, Eigenvalue 0.00938</th>
</tr>
</thead>
<tbody>
<tr>
<td>of 0.5125468 the 0.8755944</td>
</tr>
<tr>
<td>in 0.49723375 her 0.28781646</td>
</tr>
<tr>
<td>and 0.39370865 a 0.23318098</td>
</tr>
<tr>
<td>to 0.2748083 his 0.14336193</td>
</tr>
<tr>
<td>on 0.21759394 she 0.1128443</td>
</tr>
<tr>
<td>at 0.17932475 it 0.06529821</td>
</tr>
<tr>
<td>for 0.16905183 he 0.063333265</td>
</tr>
<tr>
<td>with 0.16042696 you 0.058997907</td>
</tr>
<tr>
<td>from 0.13463423 their 0.05517004</td>
</tr>
</tbody>
</table>

Table 6.2 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 6.2: Top words in 2nd singular vector pair

<table>
<thead>
<tr>
<th>Vector 2, Eigenvalue 0.00427</th>
</tr>
</thead>
<tbody>
<tr>
<td>she 0.6633538 was 0.58067155</td>
</tr>
<tr>
<td>he 0.38005337 had 0.50169927</td>
</tr>
<tr>
<td>it 0.30803354 could 0.2315106</td>
</tr>
<tr>
<td>and 0.18058427 would 0.17589279</td>
</tr>
</tbody>
</table>

6.1.2 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 6.1\(^1\) 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 vector referring to the first letter in the letter bigram is on the left and the vector referring to the second is on the right. The first two singular vector pairs are dominated by letter frequency effects, and are therefore of less interest, 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 \(a\) following letter \(b\) is whether letter \(a\) is a vowel. If letter \(a\) is a vowel, letter \(b\) 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

\(^1\)Thanks to Brandyn Webb for producing the letter bigram decomposition and figure.
about them.

These results are interesting in illustrating that singular value decomposition can produce intuitive generalisations about bigram data, both at the word and letter level, and provide us with a high-level understanding of the behaviour of the method in this context. We now go on to investigate the practical value of applying singular value decomposition to linguistic n-grams.

6.2 SVD-Based Language Modelling

This section presents an approach to n-gram language modelling based on singular value decomposition. The approach described here fills in some missing probabilities intelligently; the theory being that many of the zero probabilities in an n-gram language model are a result of overfitting in the training data. However, a simple smoothing constant is also included to ensure that absolutely no zero probabilities remain. This section presents work in developing this approach. Again, the LAS2 algorithm, as implemented in SVDLIBC (1), is contrasted with the GHA-based algorithm.

Constraints presented in earlier chapters regarding complexity of singular value decomposition algorithms on LSA-style tasks apply equally well in the context of n-gram language model smoothing. The large dimensionalities of language data are if anything more of a problem in the n-gram task. Data takes the form of word/history pairs presented as vectors, meaning that the dimensionality of the word space is the vocabulary of the dataset and the dimensionality of the history space is much higher, assuming n greater than 2. Large language modelling tasks may have vocabularies around 100,000, and two-word histories numbering around 500,000 to 1,000,000.

The next section presents data acquired from running sparse AGHA on a trigram corpus of 1 million words. Section 6.2.2 uses two smaller corpora.
6.2 SVD-Based Language Modelling

6.2.1 Large Corpus N-Gram Language Modelling using Sparse AGHA

Sparse AGHA was used to decompose a large trigram dataset with the intention of exploring the value of singular value decomposition in smoothing a large n-gram language model, and the utility of sparse AGHA in performing the task. This section describes the work. First, the method will be outlined, and then the results.

Method

The corpus used for this part of the work comprised five newsgroups from the newsgroups corpus (11). The documents were prepared by stripping off the headers before turning them into trigrams in the form of word/history index pairs. Two-word histories were allocated an index from the space of all histories that came up. (They were not allocated indices from the space of all possible two-word histories, which would have been an extremely large space.) Trigrams did not cross sentence boundaries, and sentence boundaries were not included in the training data. Sentence-final punctuation was used to detect sentence boundaries; full stop, question mark and exclamation mark. All other punctuation was stripped off aside from the inverted comma, and words were lower-cased. This rough and ready preprocessing no doubt made a few errors. The nature of the corpus is such that some non-linguistic data such as people's signatures will have raised perplexity somewhat. However, the point of the work is not to produce low perplexities in absolute terms but to demonstrate perplexity reductions, so a very high quality corpus was not considered a priority. A total of 1,283,016 trigrams were thus produced, of which 1,154,715 remained after a test corpus had been selected. The test corpus was formed from evenly distributed observations from throughout the corpus.

Trigram probabilities were calculated as the cell value of the matrix reconstructed from varying numbers of singular vector pairs (varying
\( k \) was stripped off and a small smoothing constant was added. The smoothing constant needs to be very small because cell values constructed with tens or hundreds of singular values when the rank of the matrix is more likely to be many thousand are very small, and a higher smoothing value would swamp the data. Likewise the smoothing constant used here would be inadequate for a language model constructed from the complete data matrix.

The necessity of removing negative cell values could have been avoided by using non-negative eigen decomposition, but it was felt that the assumption that linguistic n-gram phenomena can be usefully accounted for as an additive combination of non-negative factors would not be appropriate. The question remains whether n-gram phenomena can be accounted for as an additive combination of factors both positive and negative, but this seems less of a stretch.

Perplexity is used here as a measure of the quality of a language model. Perplexity provides a measure of the extent to which our model makes a test set predictable to us. A low perplexity signifies that the model is appropriate to the test set, and is therefore desirable.

**Results and Discussion**

A total of 180 singular triplets were produced over a period of 46 hours on a Sun Fire v440 using a 1MHz UltraSPARC-IIIi CPU, 8GB of RAM and running SunOS 5.9. \( k \) is traditionally used to describe the number of singular vector pairs used. The greatest of the singular vector pairs (those with the highest singular values) are assumed to be retained. Language models were produced for all \( k \)s up to 180. Smoothing constants were chosen heuristically. Several were tried in order to find the most effective. As expected, appropriate smoothing constants are small, since such small \( k \)s mean that most of the data is discarded. Figure 6.2 shows perplexity against \( k \) for three smoothing constants. The smoothing constants shown here were chosen because they demonstrate a likely optimum, since both lower and higher
6.2 SVD-Based Language Modelling

Figure 6.2: Perplexity Against k for a 1 Million Word Trigram Corpus

smoothing constants appear to result in poorer perplexities over the range shown. Optimal smoothing constant varies with k, introducing a further complication.

At k=180, the perplexity curves are continuing to drop. Furthermore, a baseline figure was prepared using the complete dataset (equivalent to k equals the rank of the matrix; all singular triplets are included) and a heuristically-derived smoothing constant (to create as close to fair conditions as possible for a comparison) and a perplexity of 373 was arrived at (smoothing constant of 5 x 10^-3), which is far lower than the approximately 1000 reached at k=180. This suggests that the perplexity curve continues to drop as k increases, and since perplexity is decreasing only slowly at k=180, the implication is that an optimal k would be much higher than 180.

The hope would be that at the optimal k, the result would be better than the baseline. In other words, were the graph continued such that k approaches the rank of the matrix, the resulting perplexity curve would be U-shaped. This experiment however has failed to
show this. After several days of running the algorithm, results show that many more singular vector pairs are required. Since later vector pairs take longer to produce, it was felt that it was not feasible to produce sufficient singular vector pairs for the task, on a dataset of this size, using sparse AGHA in its current form. An attempt to perform the same task using SVDLIBC failed to produce any data whatsoever due to memory constraints.

6.2.2 Small Corpus N-Gram Language Modelling using LAS2

Experiments with a corpus of 1 million trigrams failed to demonstrate that the SVD-based language model can improve on the baseline trigram constant-smoothed language model. This section therefore uses smaller corpora, such as can be processed by SVDLIBC without problematic RAM requirements, to produce more complete decompositions.

Method

Two corpora were prepared. The first comprised trigrams in the Atheism section of the 20 Newsgroups corpus (11) and contained around 233,000 training examples. The second comprised bigrams in a sample of 100 atheism documents. The LAS2 algorithm was used to produce 2000 singular triplets on the larger of the two and 600 singular triplets on the second. Perplexity curves for each decomposition were then calculated for all ks up to the total available and with a variety of smoothing constants.
Figure 6.3: Perplexity Against k for 230,000 Word Trigram Corpus

Figure 6.4: Perplexity Against k for 20,000 Word Bigram Corpus
Results and Discussion

Results again show perplexities failing to fall below the baseline with the best of the smoothing constants. For the larger corpus, the baseline is 397, produced using a smoothing constant of 0.0001, and for the smaller, 464 using a smoothing constant of 0.005. (The smaller corpus presumably produces a higher perplexity because it is a bigram corpus.)

6.3 Interpolating SVDLM with Standard N-gram

Examination of the qualitative behaviour of SVD on toy bigram sets lead to the following observation: whilst $k$s lower than the rank of the matrix result in meaningful fill-in, they also lead to removal of values for less significant bigrams. In other words, lowering $k$ both creates and deletes information. Some method of allowing the added information in the reduced dimensionality language model to be utilised without incurring the cost of the information loss would potentially be advantageous. For this reason, two methods are proposed by which the SVD-based language model can be combined with a single-order n-gram language model such as to retain the advantages of each.

6.3.1 Method

The SVD-based language models referred to in the previous section were interpolated with standard n-grams and performance evaluated. Linear interpolation involves creating a language model with probabilities derived as a weighted sum of the language models being combined. In our case,
\[ p_{\text{add}}(w_i | w_{i-n+1}^{i-1}) = \frac{c(w_{i-n+1}^{i-1}) + \lambda \text{svd}(w_{i-n+1}^{i-1})}{c(w_{i-n+1}^{i-1}) + \lambda \text{svd}(w_{i-n+1}^{i-1})} \]  

(6.1)

in which \text{svd}(x) signifies the svd-smoothed count for the word sequence \( x \) and \( c(x) \) signifies the raw count for the word sequence \( x \). Subscripts and superscripts are used to indicate the start and finish of the word sequence in question. \( \lambda \) is a weighting constant. As in the previous section, a simple smoothing scheme was used, in which a constant is added to all the probabilities. Again, this smoothing constant was chosen heuristically. Probabilities therefore become,

\[ p_{\text{add}}(w_i | w_{i-n+1}^{i-1}) = \frac{c(w_{i-n+1}^{i-1}) + \lambda \text{svd}(w_{i-n+1}^{i-1}) + \delta}{c(w_{i-n+1}^{i-1}) + \lambda \text{svd}(w_{i-n+1}^{i-1}) + \delta |V|} \]  

(6.2)

where \( \delta \) is the smoothing constant and \( V \) is the vocabulary. In other words, \( \delta \) is added to every count, and the denominator is therefore increased by \( \delta |V| \) so that the probabilities sum to 1. Language models were constructed for varying \( k \)s and combined in varying balances with the appropriate n-gram, that is to say, if the SVDLM is a bigram language model, then it was interpolated with the corresponding bigram SLM. If the SVDLM is a trigram language model, then it was interpolated with a trigram SLM.

Additionally, a different method of combining the two language models was tested. This method involved weighting the two language models and then using the highest probability in each case. Practically speaking, this equates to using the SVDLM to “fill in blanks” in the SLM, but not alter existing probabilities. This approach was designed following the observation that the useful information in the SVDLM is often the blank cell values it populates.

\[ p_{\text{max}}(w_i | w_{i-n+1}^{i-1}) = \frac{\max(c(w_{i-n+1}^{i-1}), \lambda \text{svd}(w_{i-n+1}^{i-1})) + \delta}{\sum_{w_i} \max(c(w_{i-n+1}^{i-1}), \lambda \text{svd}(w_{i-n+1}^{i-1})) + \delta |V|} \]  

(6.3)
\[ \max(x, y) \] signifies the greater of the quantities \( x \) and \( y \). Note that calculating the denominator involves performing the \( \max \) function for every single word following the current \( n - 1 \) word history and summing the total. This is a rather expensive operation compared to calculation of the denominator in linear interpolation. For convenience, this method will be referred to as the “max” method. Linear interpolation will be referred to as the “add” method.

### 6.3.2 Small Bigram Corpus

This section presents an investigation of the language model combination methods described above on the 20,000 bigram corpus. Results on the small corpus are quick to obtain, and provide a guide for investigations on larger corpora. In the following tables, “Wg” is the weight assigned to the SVDLM, as a way of varying the balance between the SVDLM and the n-gram LM. “SC” is smoothing constant, “N-Gram Perp” is the perplexity of the n-gram language model alone, and provides the baseline, and “Comb Perp” is the perplexity of the combined language model. The SVDLM was created using a singular value decomposition produced using SVDLIBC.

Table 6.3: “Max” method for varying ks on the 20,000 bigram corpus.

<table>
<thead>
<tr>
<th>( k )</th>
<th>Wg</th>
<th>SC</th>
<th>N-Gram Perp</th>
<th>Comb Perp</th>
</tr>
</thead>
<tbody>
<tr>
<td>100</td>
<td>0.5</td>
<td>0.005</td>
<td>4.647e+02</td>
<td>4.539e+02</td>
</tr>
<tr>
<td>50</td>
<td>0.5</td>
<td>0.005</td>
<td>4.647e+02</td>
<td>4.149e+02</td>
</tr>
<tr>
<td>25</td>
<td>0.5</td>
<td>0.005</td>
<td>4.647e+02</td>
<td>3.946e+02</td>
</tr>
<tr>
<td>10</td>
<td>0.5</td>
<td>0.005</td>
<td>4.647e+02</td>
<td>3.787e+02</td>
</tr>
<tr>
<td>5</td>
<td>0.5</td>
<td>0.005</td>
<td>4.647e+02</td>
<td>3.863e+02</td>
</tr>
</tbody>
</table>

Tables 6.3, 6.4, 6.5, 6.7 and 6.6 present results on the 20,000 bigram corpus. Results are generally positive. Perplexity reductions in the region of 20% appear to be obtainable by combining an SVDLM with a regular n-gram LM. Linear interpolation appears to be superior to the “max” method. Interestingly, the optimal \( k \) is remarkably low. This is convenient from the point of view of the computation required.
Table 6.4: “Max” method for varying SVDLM weights on the 20,000 bigram corpus.

<table>
<thead>
<tr>
<th>k</th>
<th>Wg</th>
<th>SC</th>
<th>N-Gram Perp</th>
<th>Comb Perp</th>
</tr>
</thead>
<tbody>
<tr>
<td>10</td>
<td>1</td>
<td>0.005</td>
<td>4.647e+02</td>
<td>3.782e+02</td>
</tr>
<tr>
<td>10</td>
<td>0.7</td>
<td>0.005</td>
<td>4.647e+02</td>
<td>3.770e+02</td>
</tr>
<tr>
<td>10</td>
<td>0.5</td>
<td>0.005</td>
<td>4.647e+02</td>
<td>3.787e+02</td>
</tr>
</tbody>
</table>

Table 6.5: “Max” method for varying smoothing constants on the 20,000 bigram corpus.

<table>
<thead>
<tr>
<th>k</th>
<th>Wg</th>
<th>SC</th>
<th>N-Gram Perp</th>
<th>Comb Perp</th>
</tr>
</thead>
<tbody>
<tr>
<td>50</td>
<td>1</td>
<td>0.05</td>
<td>6.349e+02</td>
<td>6.153e+02</td>
</tr>
<tr>
<td>50</td>
<td>1</td>
<td>0.005</td>
<td>4.647e+02</td>
<td>4.149e+02</td>
</tr>
<tr>
<td>50</td>
<td>1</td>
<td>0.0005</td>
<td>6.150e+02</td>
<td>4.534e+02</td>
</tr>
</tbody>
</table>

to calculate a sufficiency of singular vectors. The approximate preferred smoothing constants established in the previous section remain appropriate here. Equal weighting to each language model seems to be optimal. The approach used here of tuning one parameter whilst the others remain constant does have the potential to miss superior configurations where parameters interact. We saw previously that $k$ interacts with the smoothing constant. However, $k$ spans small ranges here, and it seemed unlikely that there would be any other interactions, so the practicality of the approach won out over the risk.

6.3.3 Medium-Sized Trigram Corpus

Results in the previous section indicate that low $k$s are optimal, and that linear interpolation is the preferred method of combining the language models. They also suggest that an equal balance between SVDLM and n-gram LM is to be preferred. Therefore, investigations on the medium-sized corpus have been streamlined based on these indicators. Varying $k$ allows us to see whether a different $k$ is optimal
Table 6.6: “Add” method for varying ks on the 20,000 bigram corpus.

<table>
<thead>
<tr>
<th>k</th>
<th>Wg</th>
<th>SC</th>
<th>N-Gram Perp</th>
<th>Comb Perp</th>
</tr>
</thead>
<tbody>
<tr>
<td>25</td>
<td>1</td>
<td>0.005</td>
<td>4.647e+02</td>
<td>3.892e+02</td>
</tr>
<tr>
<td>10</td>
<td>1</td>
<td>0.005</td>
<td>4.647e+02</td>
<td>3.695e+02</td>
</tr>
<tr>
<td>5</td>
<td>1</td>
<td>0.005</td>
<td>4.647e+02</td>
<td>3.788e+02</td>
</tr>
</tbody>
</table>

Table 6.7: “Add” method for varying SVDM weights on the 20,000 bigram corpus.

<table>
<thead>
<tr>
<th>k</th>
<th>Wg</th>
<th>SC</th>
<th>N-Gram Perp</th>
<th>Comb Perp</th>
</tr>
</thead>
<tbody>
<tr>
<td>10</td>
<td>1</td>
<td>0.005</td>
<td>4.647e+02</td>
<td>3.695e+02</td>
</tr>
<tr>
<td>10</td>
<td>0.7</td>
<td>0.005</td>
<td>4.647e+02</td>
<td>3.706e+02</td>
</tr>
</tbody>
</table>

on a larger dataset. Following from observations in the previous section, the smoothing constant used is that which was established to be optimal for this corpus earlier. Again, LAS2 was used to produce the singular value decomposition. A very positive 30% perplexity improvement on the baseline is demonstrated, see table 6.8.

Table 6.8: “Add” method for varying ks on the 200,000 trigram corpus.

<table>
<thead>
<tr>
<th>k</th>
<th>Wg</th>
<th>SC</th>
<th>N-Gram Perp</th>
<th>Comb Perp</th>
</tr>
</thead>
<tbody>
<tr>
<td>100</td>
<td>1</td>
<td>0.0005</td>
<td>4.057e+02</td>
<td>3.196e+02</td>
</tr>
<tr>
<td>50</td>
<td>1</td>
<td>0.0005</td>
<td>4.057e+02</td>
<td>3.009e+02</td>
</tr>
<tr>
<td>25</td>
<td>1</td>
<td>0.0005</td>
<td>4.057e+02</td>
<td>2.835e+02</td>
</tr>
<tr>
<td>10</td>
<td>1</td>
<td>0.0005</td>
<td>4.057e+02</td>
<td>2.890e+02</td>
</tr>
</tbody>
</table>

6.3.4 Improving Tractability

Scaling the method to a full-sized corpus requires some modification. Consider that even the 20,000 bigram SVDM constructed from 10 singular triplets shown to be optimal for that corpus earlier contains over a million non-zero values, and that the implied matrix of the 1 million trigram corpus is over ten million times the size of the implied
matrix of the 20,000 bigram corpus. We can anticipate trouble.

In the results presented above, a small positive cutoff value on cell value (SVD-smoothed n-gram count) is used to eliminate cell values that might better be considered inaccurate zeros. Recall that “probabilities” are created from SVD reduced dimensionality cell values by discarding the negative cell values and normalising the rows. In extending our definition of negative slightly above zero we reduce the size of the language model by excluding a large number of probably meaningless small values. An obvious approach to improving tractability is increasing this value. Note that in the results presented above, the cutoff used is \( 1 \times 10^{-6} \). This data is repeated in table 6.10 as the result of the smallest cutoff. As previously, results on the smallest corpus, which are quick to obtain, were used to guide experimentation on the larger corpus.

Table 6.9: Performance of “Add” Language Models on the 20,000 Bigram Corpus for Varying Minimum SVDLM Cell Values

<table>
<thead>
<tr>
<th>k</th>
<th>Wg</th>
<th>SC</th>
<th>Cutoff</th>
<th>Cell Val</th>
<th>N-Gram Perp</th>
<th>Comb Perp</th>
</tr>
</thead>
<tbody>
<tr>
<td>10</td>
<td>1</td>
<td>0.005</td>
<td>0.00001</td>
<td>1096819</td>
<td>4.647e+02</td>
<td>3.695e+02</td>
</tr>
<tr>
<td>10</td>
<td>1</td>
<td>0.005</td>
<td>0.0001</td>
<td>830625</td>
<td>4.647e+02</td>
<td>3.695e+02</td>
</tr>
<tr>
<td>10</td>
<td>1</td>
<td>0.005</td>
<td>0.01</td>
<td>461039</td>
<td>4.647e+02</td>
<td>3.692e+02</td>
</tr>
<tr>
<td>10</td>
<td>1</td>
<td>0.005</td>
<td>0.05</td>
<td>139575</td>
<td>4.647e+02</td>
<td>3.713e+02</td>
</tr>
<tr>
<td>10</td>
<td>1</td>
<td>0.005</td>
<td>0.1</td>
<td>37872</td>
<td>4.647e+02</td>
<td>3.879e+02</td>
</tr>
</tbody>
</table>

Table 6.10: Performance of “Add” Language Models on the 200,000 Trigram Corpus for Varying Minimum SVDLM Cell Values

<table>
<thead>
<tr>
<th>k</th>
<th>Wg</th>
<th>SC</th>
<th>Cutoff</th>
<th>Size</th>
<th>N-Gram Perp</th>
<th>Comb Perp</th>
</tr>
</thead>
<tbody>
<tr>
<td>25</td>
<td>1</td>
<td>0.0005</td>
<td>0.000001</td>
<td>697MB</td>
<td>4.057e+02</td>
<td>2.835e+02</td>
</tr>
<tr>
<td>25</td>
<td>1</td>
<td>0.0005</td>
<td>0.001</td>
<td>88MB</td>
<td>4.057e+02</td>
<td>2.855e+02</td>
</tr>
<tr>
<td>25</td>
<td>1</td>
<td>0.0005</td>
<td>0.01</td>
<td>16MB</td>
<td>4.057e+02</td>
<td>3.044e+02</td>
</tr>
</tbody>
</table>

The “cell values” column in table 6.9 gives the number of non-zero probabilities remaining in the SVDLM with various minimum probabilities. Note however that the SVDLMs used here lack probabilities
for word histories that do not appear in the test corpus. Experimentation was sped up by tailoring the language model to the test corpus in this way. Results of course were unaffected. Actual numbers of non-zero cell values would be much higher, and these numbers should only be taken as a guide to the reduction of the size of the SVDLMs rather than as being of absolute value. In table 6.10 the size of the SVDLM is given instead. These SVDLMs also lack probabilities for histories that do not appear in the test corpus, and furthermore, were saved to disk as text, which is suboptimal. In short, they too are of value only inasmuch as they indicate a size reduction.

As can be seen, SVDLMs cope well with relatively high probability cutoffs. For the smaller corpus, the language model can be reduced to 40% of the size of the version with the lowest cutoff without any reduction in performance. Increasing the cutoff to 0.01 has negligible impact on perplexity and allows the language model to be reduced to 13% of its original size. For the 200,000 trigram corpus, the cutoff can be set to 0.001 with a small increase in perplexity. A cutoff of 0.01 leads to a notable increase in perplexity. That the LM with the larger vocabulary should require a lower cutoff follows from the fact that with more words that might follow a given word history, all the probabilities should be a little depressed, and so lower probabilities are significant.

Note that the LM size reductions suggested here are probably somewhat conservative given that the test corpus will contain more of the frequent word histories and fewer of the infrequent ones. This means that the LMs constructed based on the test corpus will tend to lack rows that did not contain many values anyway.

Investigation of the magnitudes of the cell values in the SVDLM gives us an idea of the scale of the data we are able to discard without ill-effect. Figure 6.5 is a histogram of unnormalised cell values. The x-axis is log-scaled. Recall that we were able to discard values below 0.01 with minimal impact on performance. This, as was suggested earlier, is excluding the clear majority of the values. The cell values here are unnormalised. To produce “probabilities”, they each need to be divided by the row magnitude. Figure 6.6 shows row magnitudes
for the 20,000 bigram corpus, and figure 6.7 shows normalised cell values (“probabilities”). Normalised cell values, being the values that are then used in the actual language model, are potentially of more interest. The cutoff, however, is imposed pre-normalisation. The decision to impose the cutoff before normalising was taken to make the process more efficient: fewer values then need to be normalised. This may have impacted on performance somewhat, as it means that it is not necessarily the values that would have been smallest in the eventual language model that are excluded.

Figure 6.8 supports the result produced on the small corpus with a histogram that again shows an optimal cutoff excluding the majority of the information in the language model. (A cutoff of 0.001 had minimal ill-effect on performance.) The trends suggested by the two histograms are toward lowered cell values with larger corpora/greater dimensionalities and cell values that increase as \( k \) increases. The suggestion is that a lower cutoff would be required for a larger corpus. The histogram of cell values on the 1 million word trigram corpus shows much reduced cell values relative to the smaller corpora, and
Figure 6.6: Row Magnitudes on the 20,000 Bigram SVDLM

Figure 6.7: Normalised Cell Magnitudes on the 20,000 Bigram SVDLM
Figure 6.8: *Cell Magnitudes on the 200,000 Trigram SVDLM*

A cutoff in the region of $1 \times 10^{-12}$ seems like it would be appropriate, erring perhaps a little on the cautious side. Note in examining this figure that higher $k$s have been used here since earlier work suggested that a higher optimal $k$ might apply to the largest corpus.

### 6.3.5 Large Trigram Corpus

Results on the 1 million trigram corpus initially seemed disappointing. Perplexity reductions virtually disappeared. Observation of the distribution of probabilities in the SVDLM (Figure 6.10) shows however that the probabilities in the SVDLM of the large corpus are very small indeed, much smaller than the 200,000 trigram corpus SVDLM even bearing in mind the dimensionality increase. To a greater extent, the data in the SVDLM seems like it might be being swamped by the smoothing constant, and indeed, the second line of Table 6.11 supports this. Dropping the smoothing constant increases the discrepancy between the combined model and the baseline. However,
Figure 6.9: Cell Magnitudes on the 1,000,000 Trigram SVDLM

since this result is produced at the expense of absolute performance, it is of little practical use.

The observation that the SVDLM might be being swamped by the other data suggests potential benefits in increasing the weighting on the SVDLM past 1. Weighting the SVDLM more highly than the n-gram is not an option previously considered in much detail since initial observations showed it to be of little benefit on the smaller corpora. However, a very small benefit to a small increase past 1 on the medium-sized corpus does perhaps suggest a trend toward higher optimal weights on the SVDLM as corpus size increases. The SVDLM most likely has a far more uniform probability set for each n-1 word history in comparison to the regular n-gram, and its value ought to lie both in decreasing inappropriately high n-gram probabilities and in raising inappropriately low ones. The most likely situation is that probabilities are zero in the n-gram and some very small value in the SVDLM. In this case, the value of the SVDLM in raising the inappropriately low zero value is largely lost since the smoothing constant does this anyway. However, it does so indiscriminately. The
Table 6.11: Varying the Weight of the SVDLM

<table>
<thead>
<tr>
<th>k</th>
<th>Wg</th>
<th>SC</th>
<th>Cutoff</th>
<th>N-Gram Perp</th>
<th>Comb Perp</th>
</tr>
</thead>
<tbody>
<tr>
<td>25</td>
<td>1</td>
<td>0.5e-4</td>
<td>1.00e-12</td>
<td>3.731e+02</td>
<td>3.730e+02</td>
</tr>
<tr>
<td>25</td>
<td>1</td>
<td>0.5e-7</td>
<td>1.00e-12</td>
<td>1.284e+03</td>
<td>1.184e+03</td>
</tr>
<tr>
<td>25</td>
<td>2</td>
<td>0.5e-4</td>
<td>1.00e-12</td>
<td>3.731e+02</td>
<td>3.729e+02</td>
</tr>
<tr>
<td>25</td>
<td>10</td>
<td>0.5e-4</td>
<td>1.00e-12</td>
<td>3.731e+02</td>
<td>3.721e+02</td>
</tr>
<tr>
<td>25</td>
<td>100</td>
<td>0.5e-4</td>
<td>1.00e-12</td>
<td>3.731e+02</td>
<td>3.443e+02</td>
</tr>
<tr>
<td>25</td>
<td>100,000</td>
<td>0.5e-4</td>
<td>1.00e-12</td>
<td>3.731e+02</td>
<td>2.981e+02</td>
</tr>
<tr>
<td>25</td>
<td>1,000,000</td>
<td>0.5e-4</td>
<td>1.00e-12</td>
<td>3.731e+02</td>
<td>2.422e+02</td>
</tr>
<tr>
<td>25</td>
<td>10,000,000</td>
<td>0.5e-4</td>
<td>1.00e-12</td>
<td>3.731e+02</td>
<td>2.188e+02</td>
</tr>
</tbody>
</table>

The difference between zero probabilities that the SVDLM does not see fit to raise and those that it does becomes overly small. This might be remedied by up-weighting the SVDLM.

Indeed, this variation produces the perplexity improvement of 42% on the baseline more akin to (in fact, superior to) the result that experimentation with smaller corpora led us to expect. The optimal weighting for the SVDLM is an astonishing 1,000,000, made perhaps less perplexing by the observation that the probabilities in question are very tiny indeed. Where the n-gram model has a small number of larger probabilities, the SVDLM has a large number of tiny ones, and whilst its raison d'être is this very creation of probabilities where previously there weren't any, it does mean that it has a level of subtlety that needs to be considered in order to make the best use of it.

Again, there is much utility in raising the small cell value cutoff. Increasing the cutoff to $1 \times 10^{-11}$ approximately halves the language model in size compared to the model created using a cutoff of $1 \times 10^{-12}$ whilst impacting negligibly on performance, see Table 6.12. With regards to $k$, 25 seems in fact to be a little generous, see Table 6.13, in which $k$ varies, and the higher cutoff is used for efficiency. These language models consumed around 3GB saved in a sparse bi-
Figure 6.10: Normalised Cell Magnitudes on the 1,000,000 Trigram SVDLM

inary format, though again, the language models contained only probabilities for word histories contained in the test corpus. Full language models might be in the region of 6GB; although these slimmed-down language models contain only one fifth of the total number of word histories (71,000 of 355,000) they contain all the populous ones. To summarise, the greatest perplexity reduction demonstrated on the 1 million word trigram corpus through interpolation of an SVDLM with a baseline trigram smoothed with a heuristically-chosen constant is from 373 to 210, a reduction of 44%.

Table 6.12: Varying the Weight of the SVDLM with Higher Minimum Cell Value Cutoff

<table>
<thead>
<tr>
<th>(k)</th>
<th>(Wg)</th>
<th>(SC)</th>
<th>(Cutoff)</th>
<th>N-Gram Perp</th>
<th>Comb Perp</th>
</tr>
</thead>
<tbody>
<tr>
<td>25</td>
<td>100,000</td>
<td>0.5e-4</td>
<td>1.00e-11</td>
<td>3.731e+02</td>
<td>2.422e+02</td>
</tr>
<tr>
<td>25</td>
<td>1,000,000</td>
<td>0.5e-4</td>
<td>1.00e-11</td>
<td>3.731e+02</td>
<td>2.192e+02</td>
</tr>
<tr>
<td>25</td>
<td>10,000,000</td>
<td>0.5e-4</td>
<td>1.00e-11</td>
<td>3.731e+02</td>
<td>2.788e+02</td>
</tr>
</tbody>
</table>
Table 6.13: Varying $K$

<table>
<thead>
<tr>
<th>$k$</th>
<th>$W_g$</th>
<th>SC</th>
<th>Cutoff</th>
<th>N-Gram Perp</th>
<th>Comb Perp</th>
</tr>
</thead>
<tbody>
<tr>
<td>10</td>
<td>1,000,000</td>
<td>0.5e-4</td>
<td>1.00e-11</td>
<td>3.731e+02</td>
<td>2.102e+02</td>
</tr>
<tr>
<td>15</td>
<td>1,000,000</td>
<td>0.5e-4</td>
<td>1.00e-11</td>
<td>3.731e+02</td>
<td>2.101e+02</td>
</tr>
<tr>
<td>20</td>
<td>1,000,000</td>
<td>0.5e-4</td>
<td>1.00e-11</td>
<td>3.731e+02</td>
<td>2.174e+02</td>
</tr>
<tr>
<td>25</td>
<td>1,000,000</td>
<td>0.5e-4</td>
<td>1.00e-11</td>
<td>3.731e+02</td>
<td>2.192e+02</td>
</tr>
</tbody>
</table>

6.4 Summary

The value of SVD in n-gram language modelling has been the subject of investigation in this chapter, along with the utility of Asymmetrical GHA, or more specifically, sparse Asymmetrical GHA, in performing SVD in this context. Qualitatively, SVD is shown to produce intuitive and interesting generalisations about the relationships between word and letter bigrams. SVD-based dimensionality reduction was not shown to produce improvements with regards perplexity in n-gram language models formed from small, medium-sized and large (around 1 million trigram) corpora. However, interpolating these dimensionality-reduced n-grams with regular n-grams did produce perplexity reductions of over 40%. The optimal dimensionality of the models to be interpolated is low (around 10 to 15).

Sparse AGHA was used for the largest of the singular value decompositions; that of the 1 million word trigram corpus. The fact that the SVD LMs used for interpolation have optimally a low dimensionality increases the desirability of sparse AGHA in this context, since the approach produces singular vector pairs starting with the largest, and since time taken to produce the vectors is the main disadvantage to the approach. In any case, the approach was shown to cope well with the high dimensionality of the dataset, producing the required number of vectors in around ten minutes. The sparse approach reduces the cost of further increasing dimensionality compared to non-sparse variants.

The comparable performance of SVD-based LMs over a variety of
corpus sizes suggests its applicability in a range of different circumstances. It might be of benefit in large domain-independent language models prepared using a very large amount of training data. It might also be of benefit in improving small-domain stochastic language models such as those used in topic-specific dialogue systems. The data might equally well be incorporated into a probabilistic context-free grammar such as is often used in dialogue systems, possibly allowing a small training corpus to be better leveraged. The performance of the approach under these circumstances remains to be investigated.

The absolute value of the language modelling result depends on its extensibility to back-off language models. Unless an improvement on the state of the art, at least in some circumstances, can be demonstrated, the work is of less value. Initial attempts to interpolate SVDLM as defined here with a back-off model met with little success. The SVDLM makes use of counts of only one order, and therefore has little to offer to a language model built on multiple order counts. It is using less information. Interpolating SVDLM with a back-off model therefore reduced performance in comparison to the back-off model alone, though, promisingly, not by very much. A fair test would involve incorporation of multiple order SVDLMs. Formalisation and demonstration of this “back-off SVDLM” is an ongoing project. A back-off trigram model smoothed with Good Turing discounting produces a perplexity of 141 on the 1 million trigram corpus, which is significantly lower than the best of the results presented here (210). It is this figure that would form the baseline for the back-off SVDLM work.

Similar work includes LSA-based language modelling (55) (3) (12), in which LSA is utilised to include long-span semantic dependencies in language models. The approach is similar in that it uses singular value decomposition to include information from a reduced dimensionality representation. The difference is that in working with longer, unordered spans, the information it accesses is semantic. The approach demonstrated here, using SVD at the n-gram level, accesses instead immediate, often syntactic relationships.

Probabilistic LSA has also been used in language modelling (35) (17).
6.4 Summary

In an approach with parallels to LSA-based language modelling described above, benefits are demonstrated in using PLSA to include long-span semantic dependencies. As discussed in the previous chapter, PLSA shows a number of benefits over traditional LSA in terms of its appropriateness to probability-oriented tasks and its performance. For example, in the work presented here, a potential criticism lies in the validity of treating the reduced dimensionality SVD representation as a probability distribution, given the possible presence of negative values etc. This makes PLSA very interesting with regards to future development of the work.

Distributional Clustering (40) is another related technique. It uses distribution patterns to form word clusters which might then be used in class-based language modelling. In targeting short-span syntactic relations, it is similar to the work described in this chapter, where other techniques mentioned in this section have often aimed to capture long-span semantic dependencies. Implementationally, however, it is quite different. It would be interesting to look more closely at the differences between the two approaches. Aggregate and mixed-order models (48) probabilistically map words to classes in a manner similar to PLSA (indeed, both approaches use expectation maximisation to discover the classes). This allows intermediates between for example unigram and bigram models to be created. This is shown to be of benefit. The sharp drop-off in information content between orders is a place in which valuable information is lost, and the approach leverages this information. The approach again is similar to that described here, in that it uses distributional word groupings to improve language model performance at the n-gram level.

An evaluation of the relative merits of each approach mentioned here might be interesting, as would an investigation into the utility of combining for example long and short span approaches in similar paradigms. Lookup speed has not been formally investigated here, such as to verify the practicality of the SVDLM in a run-time system. The SVD-based language model is much larger than the single order or even back-off models that are standard. Performance gains would need to be sufficient to justify the size increase.
Chapter 7

Conclusion

This chapter concludes the thesis with a discussion of the work presented. We begin with a detailed chapter-by-chapter overview. The contributions the thesis has made are then outlined in a separate section for clarity. A high level discussion follows.

7.1 Detailed Overview

In the introduction we discussed the rationale behind vector space models and dimensionality reduction methods in terms of ability to model the data. We talked about the limitations of methods that model the data based on the assumption that observations are explicable as a weighted sum of factors, given that the real world often doesn’t work this way. As well as clarifying the advantages to such approaches, the existence of alternatives was mentioned. Eigen decomposition was briefly introduced, however, as the focus of this thesis. The Generalized Hebbian Algorithm was introduced, which is an algorithm that grew out of the artificial neural network paradigm, and allows the eigen decomposition of a dataset to be derived using
a minimal memory footprint. This is an advantage of great relevance
given that the single greatest limiting factor of the size of the dataset
that can be decomposed using traditional methods is RAM, and that
within the language processing domain, datasets are large.

Chapter 2 introduced eigen decomposition, and its asymmetric coun-
tepart, singular value decomposition. Latent Semantic Analysis was
introduced in this chapter, being the best-known application of SVD
in natural language processing.

Chapter 3 introduced the Generalized Hebbian Algorithm. GHA as
an approach to incremental eigen decomposition was discussed in the
context of other incremental approaches. The performance of GHA
as a method of performing eigen decomposition was discussed.

Chapter 4 presented work in extending GHA to a range of different
circumstances. Extending GHA to LSA, such that streamed un-
bounded datasets can be processed, involved incorporating a common
weighting scheme into the algorithm. Random Indexing was discussed
as a way of fixing dimensionality in the case that vocabulary increases
during use, and as a way of reducing vector dimensionality to improve
tractability. GHA was then extended to asymmetrical (paired) data
in the form of Asymmetrical GHA (AGHA). Sparse vector optimised
versions of GHA and AGHA were presented. Implementation issues
were discussed.

Chapter 5 discussed the application of GHA to LSA-style tasks. The
performance of weighting methods as dataset size increases was con-
sidered. GHA as an alternative to more standard approaches was
discussed and found to be a desirable option where datasets are large.
Other approaches to making LSA applicable to larger datasets were
also mentioned.

Chapter 6 discussed the application of SVD to language modelling
at the n-gram level, and the utility of AGHA for performing the de-
composition. Constructing n-gram language models from reduced
dimensionality singular value decompositions in itself was not found
to produce good results. However, linear interpolation of SVD-based
n-gram language models with single order n-gram models produced impressive perplexity reductions. The required number of dimensions was remarkably low. Producing results for real-sized datasets required some work on tractability. This was presented. The 1 million trigram dataset used here is admittedly small by modern standards but allowed the extensibility of the approach to high dimensional data to be demonstrated. Sparse AGHA was used to produce the results on the large corpus, and it was shown that the size of the language model can be reduced to a fraction of its original size with minimal impact if any on performance. Application of the approach in the context of back-off language models, vital to its interest and usefulness, was discussed as future work.

7.2 Summary of Contributions

Presented in this work are original algorithms allowing the extension of the Generalized Hebbian Algorithm to paired data (for example, n-gram data) (22), the sparse optimisation of GHA and its paired variant, Asymmetric GHA (previously unpublished), and the inclusion of entropy-based term weighting for Latent Semantic Analysis into the GHA algorithm such as to allow it to be used to perform incremental LSA (24). The first two are generally applicable in various areas of computer science, whilst the third is of course specific to LSA. These algorithms are each evaluated and used to produce results presented in this work.

The significant quantitative results of the work pertain to SVD-based language modelling, and can be summarised as follows. SVD-based smoothing of n-gram language models was not demonstrated to produce any improvement on the baseline single order n-gram language model on corpora of 20,000 bigrams, 200,000 trigrams or 1000,000 trigrams. However, linear interpolation of the SVDLM with the n-gram produced large perplexity reductions on all three corpora. A 20% perplexity reduction was demonstrated on the 20,000 bigram corpus, a 30% perplexity reduction was demonstrated on the 200,000
trigram corpus and a 44% perplexity reduction was demonstrated on the 1000,000 trigram corpus. 10 to 15 of the top singular vector pairs was found to be optimal on the largest corpus, and a weighting of 1000,000 in favour of the SVDLM was found to be necessary.

7.3 Discussion

In this section we recall the research questions raised in the introduction and consider how far we have come toward answering them. We begin with the questions relating to eigen decomposition before moving on to discuss GHA.

7.3.1 Eigen Decomposition—A Panacea?

It might seem at a glance that there are few problems that cannot be solved by eigen decomposition and its relatives, magically producing information where previously there didn’t seem to be any. And how do they do this? By throwing away part of the information! Latent Semantic Analysis has been applied with success to many different problems, including long span stochastic language model dependencies (3). This work has gone on to show benefits in language modelling at the n-gram level. Furthermore, other techniques based on a similar principle of using the surface form of data as an unreliable indicator of a smaller number of underlying and predictive constructs have produced admirable results (35). The limitations? Firstly, such approaches can only be effective in the case that there are a smaller number of concepts that predict the surface behaviour. Secondly, the assumptions about how these concepts interact need to be at least mostly accurate. This is potentially the greatest problem, since even relatively simple phenomena might require several layers of reasoning to model successfully. Thirdly, the data needs to be available on which the predictions can be made, and fourthly, the way in which the model is calculated and the form it takes need to be practical
given the situation.

- What is the utility of eigen decomposition and related techniques to natural language processing? More specifically:

  Can eigen decomposition and related techniques be used to improve language models at the n-gram level?

  What are the implications of this result and other applications of eigen decomposition in natural language processing for its overall utility in this domain?

It has been demonstrated that singular value decomposition has the potential to improve language models at the n-gram level, at least as far as perplexity is an appropriate measure with which to determine this. Stochastic (n-gram) language modelling remains the dominant approach to large domain language modelling today, and therefore a usable approach that produces a marked perplexity reduction is of interest. Usability in this case depends on a) methods being available that are capable of performing the required decomposition, and b) the resulting language model being usable, for example, from a tractability point of view.

Tractability issues typically diminish with time, as more powerful computing hardware becomes more readily available, though scalable complexity and memory requirements will always be an advantage. Another thing that is likely to decrease with time is the share of the market that stochastic language modelling holds. Simple n-gram language modelling is ignorant of so many aspects of language structure that it can only be a matter of time before more intelligent approaches take over (45). The approach presented here is just another example of hacking away trying to extract every last ounce of value from an extremely impoverished representation. That said, there is enormous potential for combining optimised n-gram language models with other more knowledge-rich representations, such that the best of each is maintained.
7.3.2 Generalized Hebbian Algorithm—Overhyped or Underrated?

This work has provided an opportunity to work with the Generalized Hebbian Algorithm under a variety of circumstances, and in doing so, develop some intuitions about its reliability, validity and usefulness. Perhaps the single most salient thing to mention from a pragmatic point of view is that, given the practical constraints that applied at the time, the singular value decomposition of the 1 million word trigram corpus, the largest corpus in this work, was produced using the Asymmetric Generalized Hebbian Algorithm because no other means were available by which this might be done. Not to say that it cannot currently be achieved any other way, but that given the plausibly representative conditions under which this work was performed, specifically, reasonable availability of hardware and software, AGHA was the only tool that could do the job. Options for allowing conventional approaches to be used include using hardware with more RAM and using software with lower RAM requirements. For example, there is an anecdotal suggestion that SVDLIBC can be adapted to use disk rather than RAM, much increasing the size of dataset it can be used to decompose at the expense of time. There may also be implementations available that the author is not aware of with lower RAM requirements.

- What is the value of the Generalized Hebbian Algorithm and its variants in performing eigen decomposition and related techniques in the natural language processing domain? More specifically:
  
  What is the value of the Generalized Hebbian Algorithm in performing Latent Semantic Analysis?
  
  Can the Generalized Hebbian Algorithm be used to perform singular value decomposition on n-gram data? Is the technique valuable for performing this task.
  
  In what ways can the Generalized Hebbian Algorithm be extended to increase its utility in this domain?
GHA has been successfully applied to Latent Semantic Analysis and n-gram language modelling in this work, and its utility has been extended in a number of ways, most significantly through the development of an asymmetric version and sparse variants. Areas of concern focus around speed and accuracy. Accuracy issues are tied up in the rate at which the algorithm converges on the eigen or singular vectors, and the means by which convergence is recognised. However, the ways in which decompositions have been used in this work have been tolerant to a certain inaccuracy, and the success of the approach speaks for the success of the basic algorithm. Since AGHA was the only method of performing singular value decomposition on the large corpus readily available, the quality of the decomposition, that is to say, proximity of the AGHA-produced singular vectors to the actual singular vectors, is difficult to verify. Other implementers have used orthogonality of the singular vectors as a measure of the quality of a GHA decomposition. However, this is dubious. A vector can be orthogonal to the others whilst still pointing in any of an infinity of wrong directions.

The performance of the SVD-based language model itself provides some indirect information about the accuracy of the decomposition. Convergence has been studied on the smaller corpora, and curves of perplexity against $k$ produced in the previous chapter. The extent to which the shape of the curve is similar ought to reflect a similar accuracy of decomposition across the corpora. The shape of this curve also guides choice of $k$ in the SVD-based language model chosen for the interpolation work: the optimal $k$ for this scenario ought to be about the same point on the perplexity curve for the pure SVD-based language models, as indeed it turns out to be. This is a very rough and ready guide however.

As discussed earlier, the implementation used here first trains a singular vector, then freezes it and moves on to the next singular vector. This means that inaccuracy in earlier vectors can potentially be covered for by later ones. Singular vectors may be discovered out of order. Vectors may cover parts of the data that earlier or later vectors ought to have covered. Additional vectors may be appended to the end to compensate for parts of the data that were missed. This
may mean that the performance of the SVD-based language model is robust to poor convergence performance in the algorithm. It might be supposed also that optimal $k$ would be a little higher in a poor decomposition. Optimal $k$ is indeed a little higher on the large corpus, decomposed using AGHA, than on the smaller ones, but this can no doubt be at least partly accounted for by the greater complexity of a larger corpus, both in terms of dimensionality and number of observations.

Time taken for GHA and variants to converge is most prohibitive where many eigen/singular vectors are required. The natural language applications discussed in this work are appropriate given this constraint, in the sense that they require only the top few vectors. LSA’s requirement of several hundred might make GHA a less desirable option where time is short, but it is easy to think of a range of situations in which the positive might outweigh the negative with regards to the desirability of GHA in this context. The SVD-based language modelling interpolation work shows great utility to just the top 15 or so singular vector pairs, making GHA rather appealing. The implementation used in this work produced that many singular vector pairs on the large corpus in around ten minutes. There remains the possibility that conventional approaches will surpass GHA as tractability becomes less of an issue. GHA will retain its edge regarding absolute memory footprint, but there will always be arguments for each approach depending on the individual circumstances.

7.3.3 Wider Perspectives

We began the thesis with a discussion of the power a model-based system designed to perform some computational task might be said to embody. Distinct from intelligence, often thought to be the preserve of adaptive systems, the power intrinsic to a model is suggested here to reside in its ability to perceive (that is to say, the richness/adequacy of the information with which it is provided) and the nature and complexity of the dependencies it is able to detect. The vector space model provides us with an “infinitely” extensible
representational framework. Eigen decomposition provides us with a useful, proven way to detect linear dependencies. This is the context in which the work took place.

Recall that eigen decomposition rotates the data such that dimensionality is reduced without any loss of information. This dimensionality, the “rank”, potentially smaller than the original but without any loss of information, might be thought of as the “true” dimensionality of the data; were we to put aside for the moment the fact that nonlinear dependencies are not detectable using this process. However, since depending on the task a smaller dimensionality might be more useful, the implication is that the part of the data that is best discarded is, from the point of view of the task, meaningless noise (for example, that the speaker has a cold or is happy does not affect the semantics of their command).

However, another way of thinking of the concept of noise in the data is to think in terms of what we can model. If we think of a perfect model as one which can accurately predict the next word in a sequence based on what has gone before, then there will be an extent to which the model cannot explain a certain part of the data. That is to say, given an infinity of data, there is still an extent to which future input is unpredictable. For example, even human-level language processing ability fails to predict for the most part what a person will say next. Theoretically, however, (somewhat depending on your world view) given enough information you could predict anything. It is your access to information and your modelling strategy that places the limitations. From this perspective, all “noise” is the fault of a combination of inadequate information and an imperfect modelling strategy.

Given limited access to information and a limited modelling strategy, there is only a certain amount of predictability in the input that is apparent to you, and therefore a certain maximum performance. Eigen decomposition, however, creates a perfect representation of the training input. Only a certain amount of that representation is generalisable. The rest is not. So we go from our earlier “true” dimensionality of the data, that is, an accurate description of the training corpus, to
a dimensionality that describes that part of the data that given our limited ability to model, is the best we can do in terms of creating a representation likely to apply to future unseen data from the same distribution. We discard the “overfitting”. This view of noise, unlike the earlier one, is not task-centric, but model-centric.

The model-centric view of noise is nicely captured in the entropy measure of unpredictability, and fits neatly with Shannon’s information theory (33). Since entropy is a measure of the extent to which the data “surprises” us given our model, a better model is one with a lower entropy. Suppose we have performed eigen decomposition on our data. We then reduce the dimensionality of the data in the new space, one dimension at a time, starting with the least significant. Each time we remove a dimension we compare our new model to a test set of unseen data drawn from the same distribution. As dimensionality decreases, we might expect that entropy would remain constant for a while or even drop, as the information being discarded is detail not applicable to the test data. Then, as generally applicable information starts to be discarded, entropy begins to increase. In finding the lowest number of dimensions we can achieve without compromising entropy, we find the dimensionality of the data, within the constraints imposed by the linearity of the eigen decomposition representation and the information available to the model.

In Chapter 6, this exact experiment was performed using word n-grams. The interesting thing is that entropy (or more specifically, perplexity, which is a simple transformation of entropy) never ceased to drop as the number of dimensions increased toward the maximum. This is interesting, and there are a number of possible explanations. The first is that there is no superfluous information in the dataset. This seems unlikely since any real dataset, with its omissions and its imperfect frequencies, is unlikely to be perfectly representative of the distribution from which it was drawn. The absence of an observation, in this context, is no less superfluous information than the presence of a misleading one, and surely, the absence of relevant observations is the curse of data-driven language modelling if ever there was one. The second explanation is that the superfluous dimensions are not the least numerically significant. For this to fit with our observations,
there would have to be no set of the least significant dimensions, however small, comprising predominantly overfitting. There may be interesting possibilities for detecting and discarding superfluous dimensions based on criteria other than numerical significance. Whilst the technique brings with it the boon of allowing us to treat absence of plausible data as overfitting, it does nothing to help us separate out the noise from the valid low-frequency events. In excluding the noise, we also exclude rare events. There is compelling research showing that excluding exceptions is harmful in language processing tasks (13). In the work presented here, hapax (words occurring only once) were not excluded, despite this being a common approach to improving tractability. A frequency of one is often considered too low for the resulting probability estimate to be of value, but much of the point of the dimensionality reduction technique used here is that it tunes probabilities based on generalisations in the corpus. In improving probabilities based on low frequencies, it might therefore be expected to be of great benefit, and in order for the technique to do this, these words need to be included. In this respect, the approach might be considered to value rare events highly. Nonetheless, the approach is discarding exceptions, and some of the dimensions in the decomposition will be about, or mostly about, these. This might explain why an interpolated language model ultimately achieved the best result: interpolation provided a way to include the exceptions. Daelemans et al. put it strongly: abstraction or editing is never beneficial to generalisation accuracy (13). Exceptions need to be included. The third explanation is that the linearity assumption fails to model the data accurately. Whilst useful, the implication is that there is potentially room for improvement on the results presented in this work through the modelling of nonlinear dependencies. In the context of Latent Semantic Analysis, nonlinear corpus preprocessing methods are already the standard (16). Kernel methods provide another approach to modelling such dependencies, by combining features through nonlinear kernel functions to create new feature dimensions, and raise interesting questions given the discussion so far about the value of increasing the dimensionality of the data.

We conclude this section with some questions for further consideration. There are two examples in this work of very low task-centric
7.4 Future Work

Interesting future work includes the demonstration of LSA on a larger corpus than has yet been used: does the performance of LSA continue to improve as corpus size increases? Within LSA, the development of a sparse version of the variant of GHA that includes the entropy-based row normalisation step would be valuable work. Interest in using GHA for LSA is widespread.

Demonstrating SVD-based language modelling as a valuable addition to the language modelling arsenal requires that it be extended to back-off language models and that an improvement on the state of the art can be demonstrated. This clearly constitutes the next step with the work. There is also room for improvement with regards to tractability. Additionally, there may be possibilities for applying Probabilistic LSA at the n-gram level.

Applying the method to small-domain language models, such as are often used in spoken dialogue systems, raises interesting possibilities. SVD-based language models would have no tractability issues in small domains. However, small domain language models are often grammar-based, though often with corpus-derived probabilities included. How can SVD be used to improve performance of this kind of language model? The approach could help to leverage small training corpora.

The probabilities in the SVD-based language model, as generated by
7.4 Future Work

the approach described here, are only loosely probabilities. They are constructed from a model of n-gram occurrence that assumes surface behaviour is the additive total of a number of influences, and so the cell values in the model are the total arrived at by varying the number of influences. Put that way, it doesn’t seem such a stretch to use them to calculate probabilities, but when you consider that the influences can be negative, concerns may arise. Whilst it is reasonable that a n-gram count might be lowered by an influence, one effect is that sometimes we have negative counts, leading to negative “probabilities”. The problem has been solved here by imposing a floor on n-gram counts. However, there is the opportunity here for developing the approach into something perhaps better founded, and maybe even more effective. Again, PLSA springs to mind.

As discussed in the previous section, it is possible that the overfitting in the singular value decomposition in the language modelling tasks does not comprise the least numerically significant of the dimensions, and so the dimensionality reduction performed here discarded valuable information with the valueless. It is possible that the SVDLM could be significantly improved, and maybe even surpass in performance the interpolated language model that produced the best results here, through the inclusion of some other method for deciding which dimensions to retain than a numerical cutoff on singular value. One approach might be to present an unseen corpus, and choose dimensions to discard based on optimising performance on this set. The approach should be inclusive: we should discard only the dimensions that hurt the result on the unseen test set. Including a valid but rare exception will probably not improve performance on the unseen set, but it won’t damage it either. Excluding overfitting in the form of for example an absence of a plausible observation might be expected to improve performance on an unseen set. A second unseen corpus would then be required to evaluate the resulting language model.

Other interesting directions include the application of kernel methods to the data to see if n-grams and similar can better be modelled through a nonlinear approach.
Bibliography


[34] HR Maturana and FJ Varela. The Tree of Knowledge. 1987.


[42] M Rayner, G Gorrell, BA Hockey, J Dowding, and J Boye. Do
cfg based language models need agreement constraints? In


[45] R Rosenfeld. Two decades of statistical language modeling:
Where do we go from here? Proceedings of the IEEE, 88(8)
2000.

[46] TD Sanger. Optimal unsupervised learning in a single-layer linear
feedforward neural network. Neural Networks, 2:459–473,
1989.

[47] TD Sanger. Two iterative algorithms for computing the singular
value decomposition from input/output samples. NIPS,

[48] L Saul and F Pereira. Aggregate and mixed-order markov models
for statistical language processing. In Proceedings of the Second
Conference on Empirical Methods in Natural Language

[49] C Tang, S Dwarkadas, and Z Xu. On scaling latent semantic
indexing for large peer-to-peer systems. In Proceedings of
the 7th annual international ACM SIGIR conference on
Research and development in information retrieval, 2004.

[50] AM Turing. Computing machinery and intelligence. Mind,

[51] TJ Ulyych. From eigenfaces to eigensections and a touch of

[52] J Weng, Y Zhang, and WS Hwang. Covariance-free incremental
principal component analysis. IEEE Transactions on
Pattern Analysis and Machine Intelligence, 25:8:1034–1040,
2003.


No 281  Christer Bäckström: Computational Complexity
<table>
<thead>
<tr>
<th>No</th>
<th>Name</th>
<th>Title</th>
<th>ISBN</th>
</tr>
</thead>
<tbody>
<tr>
<td>589</td>
<td>Rita Kovordányi</td>
<td>Modeling and Simulating Inhibitory Mechanisms in Mental Image Reinterpretation - Towards Cooperative Human-Computer Creativity, 1999</td>
<td>91-7219-506-1</td>
</tr>
<tr>
<td>592</td>
<td>Mikael Ericsson</td>
<td>Supporting the Use of Design Knowledge - An Assessment of Commenting Agents, 1999</td>
<td>91-7219-532-0</td>
</tr>
<tr>
<td>593</td>
<td>Lars Karlsson</td>
<td>Actions, Interactions and Narratives, 1999</td>
<td>91-7219-534-7</td>
</tr>
<tr>
<td>595</td>
<td>Jörgen Hansson</td>
<td>Value-Driven Multi-Class Overload Management in Real-Time Database Systems, 1999</td>
<td>91-7219-542-8</td>
</tr>
<tr>
<td>597</td>
<td>Vivian Vinarlund</td>
<td>An Economic Perspective on the Analysis of Impacts of Information Technology: From Case Studies in Health-Care towards General Models and Theories, 1999</td>
<td>91-7219-544-4</td>
</tr>
<tr>
<td>598</td>
<td>Johan Jenvald</td>
<td>Methods and Tools in Computer-Supported Taskforce Training, 1999</td>
<td>91-7219-547-9</td>
</tr>
<tr>
<td>607</td>
<td>Magnus Merkel</td>
<td>Understanding and enhancing translation by parallel text processing, 1999</td>
<td>91-7219-614-9</td>
</tr>
<tr>
<td>611</td>
<td>Silvia Coradeschi</td>
<td>Anchoring symbols to sensory data, 1999</td>
<td>91-7219-623-8</td>
</tr>
<tr>
<td>618</td>
<td>Jimmy Tjäder</td>
<td>Systemimplementering i praktiken - En studie av logiker i fyra projekt, 1999</td>
<td>91-7219-657-2</td>
</tr>
<tr>
<td>637</td>
<td>Esa Falkenroth</td>
<td>Database Technology for Control and Simulation, 2000</td>
<td>91-7219-766-8</td>
</tr>
<tr>
<td>660</td>
<td>Erik Larsson</td>
<td>An Integrated System-Level Design for Testability Methodology, 2000</td>
<td>91-7219-890-7</td>
</tr>
<tr>
<td>689</td>
<td>Joakim Gustafsson</td>
<td>Extending Temporal Action Logic, 2001</td>
<td>91-7373-017-3</td>
</tr>
<tr>
<td>720</td>
<td>Carl-Johan Petri</td>
<td>Organizational Information Provision - Managing Mandatory and Discretionary Use of Information Technology, 2001</td>
<td>91-7373-126-9</td>
</tr>
<tr>
<td>724</td>
<td>Paul Scerri</td>
<td>Designing Agents for Systems with Adjustable Autonomy, 2001</td>
<td>91 7373 207 9</td>
</tr>
<tr>
<td>725</td>
<td>Tim Heyer</td>
<td>Semantic Inspection of Software Artifacts: From Theory to Practice, 2001</td>
<td>91 7373 208 7</td>
</tr>
<tr>
<td>726</td>
<td>Pär Carlshamre</td>
<td>A Usability Perspective on Requirements Engineering - From Methodology to Product Development, 2001</td>
<td>91 7373 212 5</td>
</tr>
<tr>
<td>732</td>
<td>Juha Takkinnen</td>
<td>From Information Management to Task Management in Electronic Mail, 2002</td>
<td>91 7373 258 3</td>
</tr>
<tr>
<td>746</td>
<td>Rego Granlund</td>
<td>Monitoring Distributed Teamwork Training, 2002</td>
<td>91-7373-312-1</td>
</tr>
<tr>
<td>757</td>
<td>Henrik André-Jönsson</td>
<td>Indexing Strategies for Time Series Data, 2002</td>
<td>917373-346-6</td>
</tr>
<tr>
<td>747</td>
<td>Anneli Hagdahl</td>
<td>Development of IT-supported Inter-organisational Collaboration - A Case Study in the Swedish Public Sector, 2002</td>
<td>91-7373-314-8</td>
</tr>
<tr>
<td>749</td>
<td>Sofie Pilemalm</td>
<td>Information Technology for Non-Profit Organisations - Extended Participatory Design of an Information System for Trade Union Shop stewards, 2002</td>
<td>91-7373-318-0</td>
</tr>
<tr>
<td>765</td>
<td>Stefan Holmlid</td>
<td>Adapting users: Towards a theory of use quality, 2002</td>
<td>91-7373-397-0</td>
</tr>
<tr>
<td>771</td>
<td>Magnus Morin</td>
<td>Multimedia Representations of Distributed Tactical Operations, 2002</td>
<td>91-7373-421-7</td>
</tr>
<tr>
<td>772</td>
<td>Pawel Pietrzak</td>
<td>A Type-Based Framework for Locating Errors in Constraint Logic Programs, 2002</td>
<td>91-7373-422-5</td>
</tr>
<tr>
<td>758</td>
<td>Erik Berglund</td>
<td>Library Communication Among Programmers Worldwide, 2002</td>
<td>91-7373-349-0</td>
</tr>
<tr>
<td>774</td>
<td>Choong-ho Yi</td>
<td>Modelling Object-Oriented Dynamic Systems Using a Logic-Based Framework, 2002</td>
<td>91-7373-424-1</td>
</tr>
<tr>
<td>779</td>
<td>Mathias Broxvall</td>
<td>A Study in the Computational Complexity of Temporal Reasoning, 2002</td>
<td>91-7373-440-3</td>
</tr>
<tr>
<td>785</td>
<td>Lars Hult</td>
<td>Publikation Informationstjänster. En studie av den Internetbaserade encyklopedins bruksegenskaper, 2003</td>
<td>91-7373-461-6</td>
</tr>
</tbody>
</table>
| 808 | Klas Gäre          | Tre perspektiv på förändringar och förändringar i samband med infor


No 1009 Iakov Nakhimovski: Contributions to the Modeling and Simulation of Mechanical Systems with Detailed Contact Analysis, 2006, ISBN 91-85497-43-X.


