Unsupervised Linguistic Inference

By Michael T. Lamar

B.A., Washington University, 1999
B.S., Washington University, 1999
M.S., Washington University, 1999
M.S., Johns Hopkins University, 2004
Sc.M., Brown University, 2006

A Dissertation Submitted in Partial Fulfillment of the

Requirements for the Degree of Doctor of Philosophy in the

Department of Applied Mathematics at Brown University

Providence, Rhode Island

May 2010

This dissertation by Michael T. Lamar is accepted in its present form by the Department of Applied Mathematics as satisfying the dissertation requirement for the degree of Doctor of Philosophy.

Date_____ _____

Dr. Lucien Bienenstock, Advisor

Recommended to the Graduate Council

Date_____ _____

Dr. Stuart Geman, Reader

Date_____ _____

Dr. Eugene Charniak, Reader

Approved by the Graduate Council

Date_____ _____

Dr. Sheila Bond, Dean of the Graduate School

Personal Information

Michael Taylor Lamar
e-mail: michaeltlamar@gmail.com
Born: 11/25/1976 in Owensboro, KY USA

Education

9/2004 – 5/2010 **Brown University**
PhD, ScM: Applied Mathematics
Dissertation: *Unsupervised Linguistic Inference*

1/2001 – 5/2004 **Johns Hopkins University**
MS: Applied and Computational Mathematics

5/1999 – 9/2000 **Washington University in St. Louis**
MS: Mechanical Engineering
Thesis: *The Formation of Lobed Holes in Drilling*

9/1995 – 5/1999 **Washington University in St. Louis**
BA with majors: Economics, Mathematics
BS with major / minor: Physics / History

Work Experience

10/2000 – 8/2004 **Physics Modeling & Applications** (PMA)
Johns Hopkins University Applied Physics Lab (JHU/APL)
Research in electromagnetic propagation and scattering, quantum computation, and computer vision.

Summer 1997, 1998 **Center for Talented Youth**
Johns Hopkins University
I taught mathematics in an intensive residential summer program for gifted middle school and early high school students.

Publications

Lamar, Maron, Bienenstock. "SVD-and-Clustering for Unsupervised POS Tagging." Proceedings of the Association of Computational Linguistics. July 2010 (in press)

Awadallah, Lamar, Kuttler. "An Accelerated Boundary Integral Equation Scheme for Propagation over the Ocean Surface." *Radio Sciene.* 37(5), 1075 October 2002

Bayly, Lamar, Calvert. "Low-Frequency Regenerative Vibration and the Formation of Lobed Holes in Drilling." *Journal of Manufacturing Science Eng*. May 2002 Volume 124, Issue 2

Research Interests

Computational Linguistics, Machine Learning, Probabilistic Modeling, Computational Neuroscience

Additional Information

As a teaching assistant for eight semesters at Brown, I taught: honors differential equations I & II, introduction to probability and statistics, operations research: probabilistic models, and advanced statistics

Dedicated to my wonderful
wife, Onalee. Thank you for
your continual love and
support as we have gone
through this process together.

Acknowledgments

The work described in this thesis has been the result of an amazingly enjoyable collaboration among myself, Dr. Elie Bienenstock of Brown University, and Yariv Maron of Bar Ilan University in Israel. Their contributions have been essential and consistent, and none of these results would have been possible without their efforts.

I would like to thank Dr. Mark Johnson of Macquarie University in Australia for several valuable insights he provided into unsupervised part-of-speech tagging and for providing us with data from his implementation of a Hidden Markov Model for tagging. We understand the problem better because of his contributions. I would also like to thank Jonathan Cannon and Hoy Loper for their contribution to the early work of this project before they began their own graduate careers.

Table of Contents

List of Tables

List of Figures

Notation and remarks

Notation:

Note: When a variable is used for more than one meaning, both are noted by 1) and 2), and its use is always clear in context.

N_{types}: the number of word types in the corpus (see remark 1)

N_{tokens}: the number of word tokens in the corpus (see remark 2)

k: number of clusters or labels in an unsupervised tagger (see remark 3)

t: iteration index in any iterative algorithm

n: *k*-means step index

i: index running over clusters/labels from 1 to k

j: index running over word types from 1 to N_{types}

m: index running over word tokens from 1 to N_{tokens}

D: set of descriptor vectors

A: collection of word-type-to-cluster assignment sets (i.e. a set of sets)

C: 1) set of cluster centroids
2) clustering produced by model

D_t: descriptor vectors indexed by iteration

A_t: assignments indexed by iteration

C_t: cluster centroids indexed by iteration

$A_{t,n}$: assignments indexed by iteration and *k*-means step

$C_{t,n}$: centroids indexed by iteration and *k*-means step

d_j: the descriptor vector of word type j, can be an element of D or D_t

a_i: one of the sets comprising either A, A_t, or $A_{t,n}$; holds word types assigned to set i

c_i: the centroid of cluster i; stored in either C, C_t, or $C_{t,n}$

X_m: HMM visible random variable taking values in the set of word types

Y_m: HMM hidden random variable taking values in the set of labels

Ξ: Any collection of data points to be clustered

Ψ: A clustering of that data

H: entropy

I: mutual information

G: gold standard "clustering" from the annotated labels

B: 1) another generic alternate clustering
2) the matrix of corpus bigram counts

p: probability mass function

M: any matrix to be factored using singular value decomposition (superscript * indicates the reduced rank approximation)

U: matrix of left singular vectors (a subscript indicates the decomposed matrix)

V: matrix of right singular vectors (a subscript indicates the decomposed matrix)

S: matrix of singular values (a subscript indicates the decomposed matrix; superscript * indicates the reduced rank approximation)

L: matrix of left descriptor vectors (one superscript * indicates the reduced rank approximation, and superscript ** indicates the approximation is made without the multiplication by right singular vectors)

R: matrix of right descriptor vectors (one superscript * indicates the reduced rank approximation, and superscript ** indicates the approximation is made without the multiplication by right singular vectors)

~: this marking over a matrix indicates rows and/or columns of zeros have been removed

unit( ): function that normalizes each row of a matrix to unit length

r: reduced rank of a matrix

ρ: original rank of a matrix

M^*: approximating matrix

S^*: truncated matrix of singular values

W: a word type

α: coefficient of Hebbian attraction

β: coefficient of orthogonalization

μ: mean of multiple descriptor vectors

ζ: frequency adjustment factor

#: a corpus specific counting function

F: an objective function to be optimized in the Hebbian algorithm

v: choice of descriptor vector for each word token

(τ, ω): a sample observed bigram

VI: Variation of information

NVI: normalized VI

OTO: the one-to-one mapping criterion taking labels to tags (see remark 4)

MTO: the many-to-one mapping criterion taking labels to tags (see remark 4)

PTB45: the tagset of the Penn Tree Bank

PTB17: Smith and Eisner (2005) reduced tagset

Remarks:

1: We use “word type” to refer to the unique spelling forms observed in the corpus ignoring capitalization. As examples, dog and dogs are two different types, distinct punctuation marks are different types, and every distinct cardinal number is a different type.

2: A word token refers to a particular occurrence of a word type in the corpus. For example, a single sentence may have two tokens of the type “my” as in, “My dog ate my homework.”

3: We use “tag” to refer to the hand annotated part of speech tag and “label” to refer to an unsupervised model's inferred tag. In a clustering-based approach, the label is equivalent to the cluster number.

4: One can create a map from a subset of the labels to a subset of the tags. The one-to-one map (Johnson 2007) does not mean a bijection. It simply constrains the map by requiring at most one label to be mapped to any tag. The many-to-one map has no such constraint so that any number of labels can be mapped to the same tag.

5: Notation in the final half of chapter 4 regarding context free grammars is not intended to be consistent with the remainder of the dissertation. This choice was made so that the grammars could be described in the most straightforward and conventional way possible.

Chapter 1: Introduction

In this dissertation, we explore unsupervised linguistic inference with a strong emphasis on the part-of-speech tagging problem. We also explore unsupervised learning of simple context-free grammars, by extending a new modeling approach that we use with great success for part-of-speech tagging. Many natural-language processing problems have been well-studied using supervised methods, which require the training data to be supplemented with additional information, annotated by a language expert. Such approaches often perform quite well, but they suffer from the usual drawback that they require annotated data, which is often time-consuming or expensive to produce, and which may not be available in many natural languages of potential interest. Unsupervised algorithms are also arguably more appropriate for gaining insight into human language acquisition, a process which is dominated by mere exposure to examples rather than to any kind of supplementary information (although extra-linguistic input no doubt plays an important role in language acquisition). Exploring the bounds of unsupervised inference allows us to better understand what the brain may be able to learn from raw data consisting exclusively of examples from a natural language.

While unsupervised grammar induction of natural languages is the ultimate prize, we focus in this work on what may be described as either a sub-problem or a pre-problem – part-of speech-tagging. Whereas fully unsupervised grammar induction requires the simultaneous inference of both the syntactic categories and the rules and regularities that govern their use, part-of speech-tagging requires only categorization; hence, it is a sub-

problem. On the other hand, several established unsupervised grammar-induction algorithms such as the Constituent Context Model (Klein and Manning 2002) and the Dependency Model with Valence (Klein and Manning 2004) take POS tags as inputs to the algorithm. From this perspective, part-of-speech tagging is actually a pre-problem, as successful unsupervised grammar induction depends on successful unsupervised tagging as an input.

Unsupervised POS tagging has been well studied over the past two decades. Recent (and not so recent) emphasis has been placed on the established generative approach, exemplified chiefly by Hidden Markov Modeling. Many other algorithms have been developed, often more engineered and without the same firm mathematical foundation. Such approaches to modeling may use several steps, each of which is designed to account for different natural-language properties that can make tagging difficult; examples of such properties are the prevalence of rare words and the tendency for many word types to be used as more than one POS. Some approaches attempt to make use of morphology to improve the tagging rather than relying solely on distributional information. (Note for the non-linguistic reader: in computational linguistics, the word "distributional" is, to a good approximation, synonymous with "contextual".) The variety of approaches to unsupervised tagging over the previous twenty years is broad, but, as we demonstrate, the performance of these approaches leaves ample room for improvement.

One consistent theme in the field is the difficulty in evaluating performance. Many different criteria have been used over the years, and three are common in the recent

literature. The obvious reason that evaluation of fully unsupervised POS tagging is difficult is that, in contrast to supervised approaches which assign a "true" part of speech to the words being tagged, unsupervised approaches simply group the words into categories that are created by the tagger. It is left to the user to try to relate this collection of categories to a system of part- of-speech tags, such as, for instance, the much-used Penn Treebank (PTB) tagset; this system is then referred to as the "gold standard." If the POS taggers produced categories that were highly homogeneous with respect to the gold-standard POS, evaluation would be a simpler task, because the correspondence would be clear. Unfortunately, the categories produced by even the best tagging algorithms are typically far from homogeneous with respect to the gold standard, which makes evaluation a challenge.

The second less obvious but related reason for this difficulty in evaluation is that, in contrast to a supervised approach, there is nothing driving an unsupervised model toward discovering exactly the same syntactic regularities as are used to create the linguistic notion of part of speech implemented in the gold standard. In other words, the most salient regularities discoverable by the unsupervised tagger are not necessarily the same set of regularities that define the choice of POS. The decision as to what properties should be captured by part-of-speech tags is somewhat arbitrary in itself. For example, one tagset may differentiate between singular and plural nouns, while another may not. Similarly, one tagset may differentiate between present- and past-tense verbs, while ignoring the difference between transitive and intransitive verbs. Clearly, whatever statistical regularities the unsupervised tagging algorithm discovers, one should not

expect that they would match the part-of-speech tags exactly. Evaluation of unsupervised taggers is therefore difficult, because the categories they produce are likely not capable of perfect correspondence to any gold-standard POS system.

The second chapter of this dissertation introduces the tagging problem in some detail and explores the process of its evaluation. We demonstrate that of the three most commonly used criteria in the recent literature, two are extremely unreliable. The third criterion, although it also has some properties that make it less than completely desirable, does perform well according to our examination. It is indeed surprising that all three criteria are commonplace, despite the fact that two turn out to be wholly unsuitable for the problem.

The third chapter focuses on the use of dynamic, i.e., latent, descriptor vectors for tagging. We present a novel algorithm for unsupervised part-of-speech tagging, which clusters a collection of *dynamic descriptor vectors*, allowing the cluster assignments to inform the evolution of these vectors and vice versa. Chapter 3 is the center and most substantial and innovative part of this dissertation, and we now give, in the following six pages, a synopsis of it.

Our algorithm uses reduced-rank singular value decomposition (SVD) to achieve dimensionality reduction for the extraction of latent features. We first implement a simple two-step version of our algorithm, which mirrors an approach proposed by Schütze (1995). We demonstrate that, on a simple 17-tag tagset, the method achieves performance that is significantly better than the performance of Hidden Markov Model (HMM) approaches, which represent the previous state of the art. A succinct description

of this two-step implementation can be found in a paper in press (Lamar et al. 2010).

While this simple implementation is interesting for its connection to the work of Schütze, it does not take advantage of the full generality of our novel algorithm. Indeed, on the more finely grained (and more frequently used) 45-tag tagset, it does not *significantly* improve on the performance of HMM's. However, using the fully general implementation of our algorithm, which will be referred to as ISVD, we demonstrate a significant, and indeed quite substantial, improvement over the HMM's on *both* tagsets. While there is a sense in which our approach to this problem is over-constrained, we argue that such over-constraining places our model in a position to be successful, whereas, in contrast, the over-parameterization of the HMMs puts them in a position to struggle.

The ISVD implementation of our algorithm is similar to *k*-means clustering, which is an EM-style algorithm. In *k*-means, a vector to be clustered is assigned to the cluster of the nearest centroid, in a deterministic *Assignment* step that parallels the *Expectation* step of EM. Then, these assignments are used to compute a new set of centroids, in a *Model-Update* step that mirrors the *Maximization* step of EM. ISVD is more complex, because the vectors being clustered are dynamic, i.e., they are themselves treated as *latent variables* rather than fixed observations; yet the process is still of the EM style. Figure 1.1 contrasts a single iteration of *k*-means clustering with a single iteration of the ISVD algorithm, by reducing each to a simple sketch.

In *k*-means, the descriptor vectors, D, are fixed and do not appear as a latent variable in the sketch. In ISVD, these vectors, D_t, change each iteration, based on the

previous set of assignments, A_{t-1}. The updated descriptors then combine with the previous assignments to update the cluster centroids, C_t. These steps can be thought of as the *Maximization* portion of the algorithm. Then, with new centroids and descriptors, the assignments, A_t, are recomputed, paralleling the *Expectation* step of EM.

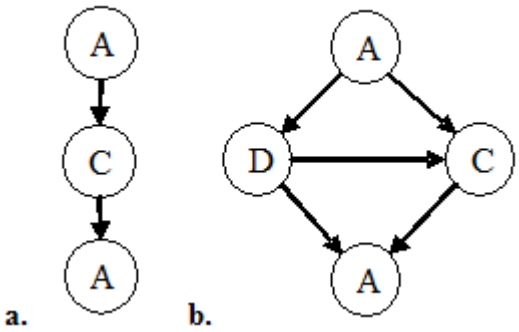

Figure 1.1: a.) Sketch of a single iteration of the k-means algorithm. A represents the set of cluster assignments, and C represents the collection of cluster centroids. b.) Similar sketch of one iteration of the ISVD algorithm. Here, the set of descriptor vectors, D, are changed by the assignments, and along with the assignments, they update the centroids. Then the updated centroids combine with the updated descriptors to produce new assignments.

Our dynamic-descriptor-vector approach to unsupervised part-of-speech tagging (UPOST) can be more easily understood by contrasting three alternatives to tagging in an optimization framework: classical *k*-means clustering, our two-step approach (similar to Schütze 1995), and our full ISVD algorithm.

Non-disambiguating UPOST in "classical" k-means-clustering

We first define set of descriptor vectors, D, with one vector per word type. That is, $D = \{d_j, j = 1,..., N_{types}\}$. The process of creating these vectors is described in detail in chapter 3 as the initialization of the iterative algorithms, and it is based on the empirical distribution of *word type* contexts. In other words, word type *j*'s descriptor vector comes from how often all other word types are observed as an immediate neighbors of *j*. The exact form of these vectors is not important to this introduction, but we emphasize the essential point that these vectors must remain fixed to allow the use of the classical *k*-means clustering algorithm. (In the two later approaches by contrast, the descriptor vectors change.)

Next, we define the collection of cluster assignment sets, $A= \{a_i, i = 1,\ldots,k\}$, so that a_i is the set of word types assigned to cluster i. Finally, we define the model parameters, $C = \{c_i, i = 1,...k\}$, with c_i storing the centroid of cluster *i*.

The optimization problem which *k*-means attempts to solve is to find the A that minimizes the total sum of squared distances between descriptor vectors and their assigned clusters.

$$A = \underset{\tilde{A}=\{\tilde{a}_i\}}{\operatorname{argmin}} \sum_{i=1}^{k} \sum_{d_j \in \tilde{a}_i} \left\| d_j - c_i \right\|^2$$

So, given the corpus, we define descriptor vectors, and we seek an optimal set of assignments which cluster these vectors as tightly as possible. Although this is an NP-hard problem, the classic *k*-means algorithm, which typically converges in a few dozen iterations with our corpus, often gives good results.

To maintain consistent notation with the next approaches, we write the assignments and centroids indexed as $A_{1,n}$ and $C_{1,n}$. Two subscripts are needed because in these later approaches we use two nested iterative processes. The second subscript, n, indicates the n^{th} *k*-means step. (We use *step* with this index and *iteration* with the first index to reduce confusion.) The subscript 1 indicates that this is the first (and only) iteration in which *k*-means steps are applied. (In the later approaches we iterate these *k*-means steps on changing vectors.)

The first half of each *k*-means step updates the latent variables which comprise $A_{1,n}$ from the previous set of centroids, $C_{1,n-1}$, by assigning word type *j* to the cluster *i* whose centroid c_i is closest to d_j. (To simplify notation, no iteration or step indices are used for the elements of $A_{1,n}$ and $C_{1,n}$, only element number subscripts are used.

$$j \in a_m \quad \Leftrightarrow \quad m = \underset{i}{argmin} \left\| d_j - c_i \right\|^2$$

The second half of a *k*-means step updates the centroids, $C_{1,n}$, from this newly created collection of assignments, $A_{1,n-1}$, by computing the means. (The notation, $|\cdot|$, indicates the cardinality of a set.)

$$c_i = \frac{1}{|a_i|} \sum_{d_j \in a_i} d_j$$

Iterative "k-means algorithm"
initialize $C_{1,0}$
alternate, until convergence, between
Assignment step (analogous to E step of EM)
update $A_{1,n} = A_{1,n}(C_{1,n-1})$
Model-Update step (analogous to M step of EM)
update $C_{1,n} = C_{1,n}(D, A_{1,n})$

The *k*-means algorithm may be viewed as EM for estimating a mixture of Gaussians, provided the variances are fixed rather than estimated, and provided they are fixed to a sufficiently small value that, for all practical purposes, the probability of a set of assignments given a set of centroids is degenerate, each word type *j* being assigned with probability 1 to the centroid closest to its descriptor. The E step of EM then becomes the assignment step of *k*-means, and the M step becomes the model-update step.

Non-disambiguating UPOST in "Two-Step" k-means-clustering

The goal now consists of two subgoals, and the method proceeds in two corresponding "iterations." Conveniently, Goal 1 and Iteration 1 are identical to the goal and method in the "classical" *k*-means setup described above.

To describe goal 2, we first define a new set of descriptor vectors D_2 based on the converged assignments, $A_{1,\infty}$, obtained from iteration 1. The new descriptor d_j for word-type *j* is derived from the *label* context distribution of j (see preliminary remark 3 for the meaning of labels). Again, the exact details of how these vectors are constructed is given in chapter 3, but the key detail is that component *m* of word type *j*'s vector is based on how often *j* is neighbored in the corpus by any word from cluster *m* as defined by the previous assignments, $A_{1,\infty}$.

Using this new set of descriptors D_2, Goal 2 is defined in much the same way as

Goal 1: find a new set of assignments, A_2, and centroids, C_2, that minimizes the sum of squared distances between descriptors and their assigned clusters' centroids.

For Iteration 2 we apply the same *k*-means algorithm as above, only using D_2, $C_{2,n}$, and $A_{2,n}$.

Non-disambiguating UPOST in "SELF-INFORMED" k-means clustering

In this truly iterative framework, the model is parameterized in terms of assignments, A_t, rather than centroids. Each iteration, from the new set of assignments, A_t, we define descriptor vectors, D_t, in much the same way as was done in the second iteration above. Again, the details are included in chapter 3, but the important detail is that these vectors are based on the *label* context just as above. As the assigned labels change each iteration, the descriptor vectors change as well. Finally, we define the set of centroids, C_t, as the means of all the newly formed descriptor vectors assigned to each cluster. Notice that the descriptors are determined from the assignments (by a complicated function not specified in the introduction), and the centroids are determined from the assignments and the descriptors making them just elaborate functions of the assignments as well.

Goal: Find a set of assignments, A, (and from it the corresponding fully determined D and C), so that A minimizes the sum of squared distances between the descriptors and their corresponding cluster centroids.

$$\mathrm{A} = \underset{\tilde{\mathrm{A}}=\{\tilde{a}_i\}}{\operatorname{argmin}} \sum_{i=1}^{k} \sum_{d_j \in \tilde{a}_i} \left\| d_j - c_i \right\|^2$$

Note that:

(a) the optimization problem looks identical to the classical *k*-means framework;

(b) the problem differs because now the descriptors also depend on the assignments, not just the centroids. It remains NP hard.

Our iterative algorithm yields a desirable solution which achieves a local optima much like classical *k*-means for the original optimization problem. Each iteration resembles one step of the *k*-means algorithm, but with one important addition.

We first build new descriptor vectors, D_t, based on the previous iterations assignments, A_{t-1}. (Again, the details of this construction are omitted here.) From these updated descriptor vectors and the previous assignments, we update the centroids, C_t, by averaging in the obvious way. From D_t and C_t, we can update the assignments to produce A_t by assigning the newly created descriptors to the nearest newly created centroid.

ISVD algorithm

initialize A_0
alternate, until convergence, between
  Model-Update step (analogous to M step)
    update: $D_t = D_t(A_{t-1})$
    update: $C_t = C_t(A_{t-1} , D_t(A_{t-1})) = C_t (A_{t-1})$
  Assignment step (analogous to E step)
    update: A_t

We conclude this discussion of the optimization framework with an important remark. Just like the classical *k*-means algorithm, we cannot claim that ISVD achieves its optimization goal. The algorithm does, in practice, tend to produce a steady decrease in the objective function, but not with the provable certainty of *k*-means that comes from its

connection to EM with a mixture of Gaussians. However, the extremely strong performance of this algorithm, when evaluated for POS tagging against classical gold-standards, is certainly an indication that ISVD does an excellent job of achieving the goal.

As we conclude this third chapter, we promote the idea of targeted supervision in a post-processing step, which we refer to as prototype tagging. In contrast to traditional supervised approaches that require extensive annotation before training can proceed, prototype tagging first produces a fully unsupervised categorization of the data. After the data has been categorized, we extract from each of these categories a single word for the language expert to hand tag. This targeted supervision comes at minimal effort compared to what is needed for fully supervised taggers. The details of our algorithm make it perfectly suited for prototype tagging because it can produce a very large number of syntactic categories with limited increase in computational burden. HMM approaches, by contrast, would struggle mightily to perform well either in runtime or in accuracy when the number of hidden states becomes very large.

The fourth chapter begins by developing an approach to part of speech tagging that is, at least on the surface, rather different. By relying on elements and mechanisms with loose but clear connections to neural implementation, we produce a simple but surprisingly effective tagger. We find this approach to unsupervised modeling quite appealing because it helps in understanding how natural language may be processed by the brain. Given the remarkable efficiency with which the brain is capable of learning natural language and the nearly flawless performance it achieves in natural language

processing tasks, we also hope that, by better understanding how the brain works, we will be able to design more efficient and accurate models.

Like the more statistical approach of the third chapter, this biologically inspired model uses dynamic, self-organizing descriptor vectors; yet the forces at work on these vectors are extremely simple. We show that such a simple model can out-perform the much more complicated HMM's although the improvement is not as dramatic on the fine grained tagset as the more complex approach of the third chapter. We connect our model to the Generalized Hebbian Algorithm for eigen-decomposition by relating its biological mechanisms to the terms of a gradient-descent matrix-factorization algorithm. This connection establishes a loose mathematical framework for understanding the basic behavior of the algorithm.

The success of the biologically inspired model leads us to considering how the approach might be extended to grammar induction. Because fully unsupervised grammar induction of natural language is such a daunting undertaking, we restrict ourselves to investigating only relatively simple context-free grammars (CFGs). We demonstrate that this extended model can successfully learn, in a completely unsupervised way, a series of progressively more challenging (but still relatively simple) CFGs. We conclude by discussing some limitations of the approach that must be overcome before the CFG learner can be extended to natural language data.

The two new approaches to unsupervised POS tagging developed in this thesis are valuable for two different reasons: one more theoretical, the other more applied. The biologically motivated model demonstrates that attention to neural mechanisms can

produce a simple model that competes with the much more complex HMMs. It gives some insight into how linguistic structure can be represented by a self-organizing process that closely relates to mechanisms present in the brain. Our ISVD algorithm, a clustering approach in which the observations are turned into latent variables, demonstrates a quantum leap in performance of unsupervised POS tagging over the most successful HMMs – representing the previous state of the art. This increased performance is seen on both a coarse and a fine grained tagset. Not only is this algorithm more accurate, it can be implemented in a way that is orders of magnitude less expensive computationally. These approaches share two common properties. Both rely on self-organization of geometric descriptor vectors, and neither attempts to disambiguate tokens of the same type; instead, they assign a unique POS label to each word type. *Together*, these models represent a significant advancement in the study of unsupervised POS tagging.

Chapter 2: Evaluation of Part-of-Speech Tagging

Introduction to POS Tagging

In this chapter, we begin by introducing the problem of part-of-speech (POS) tagging and exploring how this problem has been addressed, with varying degrees of success, over the past two decades. We pay special attention to so-called unsupervised approaches, which do not require training data that has been hand-annotated with POS tags. These unsupervised methods are of interest for two reasons. First, human language acquisition is arguably much better described as an unsupervised process. Second, not all natural languages have readily available annotated data, and the production of this training data can be costly and time consuming. The most successful unsupervised models to date are Hidden Markov Models (HMMs), so we outline how these HMMs are employed and briefly discuss limitations and difficulties of taking this approach.

We then focus our attention on the inherent difficulty of evaluating the performance of any unsupervised POS tagging model. In contrast to supervised models which produce actual POS tags, unsupervised approaches can only produce generic labels, having never been exposed to any annotation to indicate a "true" POS. There is no obvious or agreed-upon way to compare these induced labels to the annotated tags for the purpose of evaluation. We demonstrate that two of the three most commonly used evaluation criteria are, in fact, seriously flawed and should not be used for algorithms that perform at the current state-of-the-art level. We further demonstrate that the third commonly used criterion has exactly the desired properties that the flawed criteria lack,

and therefore appears to be reliable.

Survey of the POS Problem

At the heart of the POS tagging problem is the goal of learning syntactic categories. Simply put, this problem entails automatic labeling of a large collection of text (i.e., a corpus) in such a way that each word token's label corresponds to its actual POS. The performance of these tagging algorithms is tested using corpora that have been hand-annotated, and the hand annotation is treated as a gold standard for evaluation purposes.Yet not all linguists would agree on the "correct" POS tag for every token in the corpus, or even on the most appropriate collection of tags from which to choose (i.e. the tagset). The POS tagging problem has been studied for the better part of two decades with gradually improving performance. Supervised tagging models perform with accuracies over 97% (Collins 2002; Toutanova et al. 2003) but require training with annotated data. The most strongly supervised models are given a portion of the hand annotated corpus for training, less strongly supervised methods (Goldwater and Griffiths, 2007) may require only a lexicon, which lists all the types along with all their possible POS tags, and the most weakly supervised methods may require as little as single prototype word types for each POS tag (Haghighi and Klein 2006). While supervised POS tagging can perform very well, we focus here on the problem of unsupervised tagging, in which no annotated data is used for tagging (although, of course, this annotated data is needed for evaluation of performance).

One of the early algorithms for unsupervised tagging that is of particular interest

because of its close relationship to our so-called SVD2 model described in the following chapter is that of Schütze (1995). Schütze's approach involved a two-step process. For every word type in the corpus, w_i, he assembles a vector that counts the number of times each of the most common words in the corpus is seen as the left neighbor and right neighbor of w_i. These high-dimensional descriptor vectors can, with the aid of singular value decomposition (discussed in detail in the following chapter), be forced into a much lower dimension in order to extract latent features. These low-dimensional vectors are then clustered, as a first approximation to syntactic categories. In one version of his algorithm, Schütze then assembles new high-dimensional descriptor vectors for each type; these new descriptors count how often the inferred categories (rather than particular word types) are seen as immediate left and right neighbors. These vectors can then be treated again with SVD and clustering in order to extract a label for each type. Unfortunately, Schütze did not use the same corpus, tagset, or the now accepted evaluation criteria; also, because many of the details of his method are left unexplained, it is not possible to directly compare his work with more recent models. We demonstrate in the following chapter that the ideas behind Schütze's early work, modified into our SVD2 algorithm, can be used to achieve unmatched performance on a particular English tagset, so his early work was indeed ahead of its time.

Clarke (2000) describes a somewhat ad hoc iterative algorithm for this problem. From an initial clustering of the types, he computes a context distribution for each cluster. He then uses this distribution to estimate the context distribution for a word in the clustering in a way that is less sensitive to the particular co-occurrence statistics of the

corpus. Using K-L divergence as a pseudo-distance, he then updates the clustering. This process is iterated until the most common and unambiguous word types are clustered. He then treats the remaining ambiguous words by assuming that their distribution is a linear combination of the distribution of the clusters of the non-ambiguous words, and uses an expectation-maximization procedure to estimate the degree to which the type "belongs" to each cluster. Finally, he treats rare words using an external Bayesian prior to help ensure that their distributions are more likely to match up with nouns or adjectives than with closed-classes like pronouns. Like Schütze, his choice of corpus and evaluation criteria prevent direct comparison with his approach, and his description leaves out enough details that it is again not possible to reproduce his approach exactly.

Clark (2003) describes the use of morphological information to supplement the usual context distributional information for several unsupervised POS tagging models. While morphology does have the potential to improve performance, we focus on distributional information only, acknowledging that a well chosen strategy for inclusion of morphology may be appropriate at a future time – although published results to date indicate that incorporating morphology leads to only marginal improvement in performance (Headden et al. 2008).

A more principled approach to POS tagging involves the use of Hidden Markov Models (HMM's) for generative modeling and much of the early work had this focus (Jelinek 1985; Church 1988; Derose 1988; DeMarcken 1990; Cutting et al. 1992; Kupiec 1992; Charniak et al. 1993; Murakami et al. 1993; Weischedel et al. 1993; Schütze and Singer, 1994; Lin et al., 1994; Elworthy, 1994; Merialdo, 1994). More recently, the

technique was presented by Johnson 2007 and Gao and Johnson 2008 using the *Wall Street Journal* corpus of the Penn Treebank and the standard evaluation criteria. In an HMM, the sequence of unknown POS tags are represented by a sequence of random variables described as a first-order Markov chain whose state transitions are governed by unknown (but stationary) transition probabilities – see figure 2.1. Each underlying state can emit any word type according to its constant (but also unknown) emission probability distribution. The HMM is a generative model, meaning that, once the parameters are learned, it can then be used to produce a sequence of word types and associated POS hidden states governed by these learned distributions; generated sequences should resemble the data on which the model was trained.

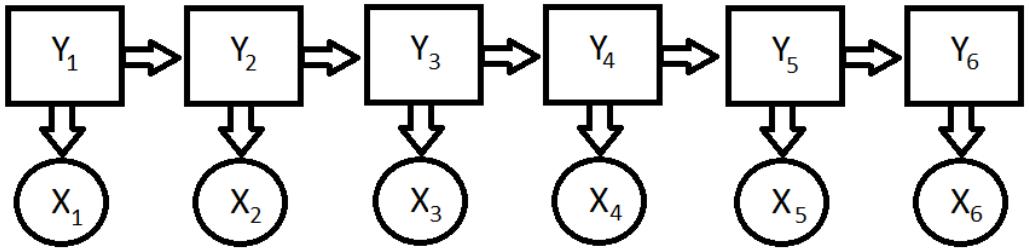

Figure 2.1: The sequence of Y's represents the hidden random variables that take on values of POS states. The sequence of X's represents the random variables that take on values of observed word types. The horizontal arrows indicate that the transition from one Y to the next is governed by a first-order Markov chain with unknown transition probabilities. The vertical arrows indicate that the word types are emitted, independently, based on an unknown distribution conditioned on the underlying POS state.

There are several well studied strategies for estimating the emission and transition probabilities and predicting the hidden states in an HMM, and Gao and Johnson explore three for the POS tagging problem. They first consider Expectation Maximization (EM),

which produces a point estimate for the unknown parameters that locally maximizes an expected value of the likelihood function. With this set of parameters, it is possible to find the most likely sequence of hidden states or to sample from the conditional distribution of states given the observed data.

Gao and Johnson also explore a Bayesian framework, which treats the unknown parameters as random variables and incorporates a Dirichlet prior to control for a particular sparsity property of natural language. They use both the Variational Bayesian EM method of Beal (2003) as well as the Gibbs sampler of Geman and Geman (1984) to carry out the inference.

The Bayesian approach helps to address the sparsity problem: unconstrained HMM's tend to assign several different states (and therefore POS tags) to each word type, while in natural languages such as English most types can be used as only one or two parts of speech. Gao and Johnson choose an appropriate Dirichlet prior to encourage sparseness in the emission probability parameters, causing a state to prefer to emit only a small number of types. While this prior helps, Graca et al. (2009) note that this approach does not directly address the sparseness issue of natural-language data. The prior on the emission probabilities tries to limit the number of types that can be emitted by any one tag; this is appropriate for closed-class tags (e.g. articles, conjunctions, prepositions, etc) that have only a few word types, but it is not appropriate for open-class tags (e.g. nouns, verbs, adjectives, etc.) which have many tens of thousands of types. Said differently (referring to the notation of figure 2.2), it encourages zeros in the emission probabilities, $P(X|Y)$, which only indirectly leads to zeros in $P(Y|X)$, and these zeros are typically

inadequately placed.

Graca et al. (2009) use an HMM with a modification of the posterior regularization framework to address the desired sparsity more directly. Their method represents the best performance of any approach (HMM or otherwise) to date on the unsupervised POS task.

Three Evaluation Criteria: VI

In contrast to supervised methods, which label the corpus with the tags given by the annotated data, unsupervised methods can only produce generic labels for each category of word. This generic labeling leads to an inherent difficulty in evaluation that is not present in supervised taggers. Several methods of evaluation have been used in the literature over the history of the problem, but three criteria have become conventional in recent years.

The first criterion, variation of information (VI) (Meilă. 2003), attempts to compare the inferred tags (which we call labels) with the gold standard tags (which we refer to simply as tags) without creating a map between the two. In order to define VI, it is convenient to first define what we mean by a clustering as well as the entropy of a clustering (or equivalently a partition). Consider a collection of data, $\Xi = \{\xi_1, \xi_2, \xi_3, \ldots \xi_m\}$. If the following three conditions hold, then $\Psi = \{\Psi_1, \Psi_2, \Psi_3, \ldots, \Psi_k\}$ is a clustering of Ξ.

1.) $\Psi_i \subset \Xi \quad \forall i \in \{1,2,\ldots,k\}$

2.) $\Psi_g \cap \Psi_h = \emptyset \quad \forall g \neq h$

3.) $\bigcup_{i=1}^{k} \Psi_i = \Xi$

We call the subsets of Ψ its clusters, so these rules require that every data point in Ξ be in one and only one cluster.

For a clustering, Ψ, we define its entropy, $H(\Psi)$, in an analogous way to the information-theoretic entropy of a discrete probability distribution. We construct an empirical probability mass function based on the relative size of each cluster of Ψ.

$$p_i^{\Psi} = |\Psi_i| / \sum_{q=1}^{k} |\Psi_q| \qquad \forall i = 1,2,\ldots,k$$

(We use the notation, $|\cdot|$, to mean the total number of elements in a set. If used with a clustering [*e.g.* $|\Psi|$]*, it counts the total number of clusters. If used with an individual cluster* [*e.g.* Ψ_3]*, it counts the number of data points in the cluster.)*

Using this probability mass function, we can define the entropy of the clustering just as we would the entropy of a discrete random variable governed by the same distribution.

$$H(\Psi) = -\sum_{i=1}^{|\Psi|} p_i^{\Psi} \log_2 p_i^{\Psi}$$

In like manner, we define the mutual information, I, between two clusterings, G and C, of the same N data points. We require a similarly defined joint probability mass function which counts the number of data points that are in a particular cluster of G and a particular cluster of C. This joint distribution is used alongside the marginals defined as above:

$$I(G;C) = \sum_{i=1}^{|G|} \sum_{k=1}^{|C|} p_{ik}^{G,C} \log_2 \left(\frac{p_{ik}^{G,C}}{p_i^G \, p_k^C} \right) \qquad \text{with} \quad p_{ik}^{G,C} = |G_i \cap C_k| / N$$

Similarly, conditional entropies can be defined using similarly computed conditional

distributions. Conditional entropies are related to entropy and mutual information by the identity:

$$H(G|C) = H(G) - I(G;C)$$

Finally, we define the variation of information, VI, between two clusterings, *G* and *C,* as:

$$VI(G;C) = H(G|C) + H(C|G) = H(G) + H(C) - 2\mathrm{I}(G;C)$$

It should be noted that VI is a true distance between clusterings (i.e. a metric), and this property is not hard to show. Conditional entropy is non-negative and is zero if and only if the two clusterings are identical (up to permuted cluster indices) so the same is true of VI. Symmetry of VI is equally trivial when regarded as the sum of symmetric conditional entropies. The final requirement of a metric is that it satisfy the triangle inequality. Although not quite trivial, this property can be shown using the well known properties of information theoretic entropy (see Cover and Thomas 2006 for more detail):

$$\begin{aligned} VI(G;B) + VI(C;B) - VI(C;G) &= H(B|G) + H(C|B) - H(C|G) + \\ &\quad H(B|C) + H(G|B) - H(G|C) \end{aligned} \tag{0}$$

$$\begin{aligned} &\geq H(B|G) + H(C|B,G) - H(C|G) + \\ &\quad H(B|C) + H(G|B,C) - H(G|C) \end{aligned} \tag{1}$$

$$\begin{aligned} &= H(C,B|G) - H(C|G) + \\ &\quad H(G,B|C) - H(G|C) \end{aligned} \tag{2}$$

$$\geq 0 \tag{3}$$

We write (0) directly from the definition of VI applied to three possible clusterings: C, G, and B. The first inequality, (1), follows because adding more conditioning in the two terms that have changed from (0) can never increase entropy. The equality in (2) follows from twice using the factoring property of a conditional entropy which allows two

conditional entropies to be combined into one. The final inequality, (3), follows because in each line of (2), the joint conditional entropy can be no smaller than the marginal conditional entropy.

Although this metric property is valuable for intuition, the primary advantage of VI over the remaining two criteria is that it does not require the construction of a map between the two clusterings (i.e. gold tags and inferred labels) in order to compare them. Philosophically then, VI is somewhat more in keeping with unsupervised tagging than the map-based criteria.

Reichart and Rappoport (2009) suggest a slight modification of VI when it is used in unsupervised POS tagging to compare a model's inferred labels, *C*, to the gold standard tags, *G*. Because the VI distance can vary greatly in magnitude with the size of the corpus and the particular choice of tagset, they suggest normalizing VI (which they then call NVI) by dividing it by the entropy of the gold standard.

$$NVI(G;C) = \frac{H(G) + H(C) - 2\mathrm{I}(G;C)}{H(G)}$$

Although NVI is no longer a true distance, we agree with their assertion that this drawback is more than offset by the benefit that comes with placing all VI scores on the same scale regardless of the corpus and tagset, hence we use NVI throughout this chapter. It should be noted that since VI is a distance, the more similar two clusterings are, the lower their VI distance and corresponding NVI . This property is in direct contrast to the next two criteria which are accuracy scores (meaning that better agreement raises the score).

Three Evaluation Criteria: MTO

The other two most commonly used criteria require the construction of a map between the set of labels and the gold tags. This map then takes the inferred label of each token to an inferred tag which can be compared directly with the gold tag. The tag accuracy score is just given by the fraction of tokens whose inferred tag matches its gold tag. The difference between the remaining two criteria is in the construction of the map.

Under the first of these map-based criteria, the map can freely assign labels to tags so as to maximize the tag accuracy. Since it is acceptable to map several labels to a single tag, this procedure produces the so-called many-to-one map or just MTO. The globally optimal MTO map can be constructed simply by finding the optimal tag for each individual label independently of all other labels. This optimum choice of tag for an individual label is the most frequently observed tag of its constituent tokens. Table 2.1 gives an example contrasting the MTO map to the map used in the third criterion, OTO.

Three Evaluation Criteria: OTO

Under the second map-based criterion, the map is constrained so that at most one label can correspond to any gold tag. Although there exists a globally optimal (not necessarily unique) map under this constraint, it is computationally expensive to search for it exhaustively. (With N tags and N labels, there are $N!$ possible maps.) Instead, it is common to use a greedy search, in which the tag/label pairing with the largest number of correctly paired tokens is assigned first; then, from the remaining tags and labels, the next best pairing is found, and so on until all the tags (or labels) are exhausted. (Any unused

labels are mapped to a null tag and cannot contribute to the score.) The complexity of this greedy search grows only like N^3 and can find the global optimum if there is a sufficiently strong correspondence between the tags and the labels. For a particular clustering produced by a tagger, the hope is that the accuracy score produced by the greedy search will reliably reflect the unknown accuracy under the globally optimal map (although we will see later that the greedy search is problematic in practice). We refer to this greedy one-to-one mapping criteria as OTO. (Again, table 2.1 gives a simple example of greedy OTO.)

Data	Label	Gold Tag	MTO Tag	OTO Tag
a	1	N	N	N
b	1	N	N	N
c	1	N	N	N
d	1	V	N	N
e	1	V	N	N
f	2	N	N	V
g	2	N	N	V
h	2	V	N	V

Table 2.1: A simple illustration of the difference between MTO and OTO. Data from both labels 1 & 2 are assigned an MTO tag of N because there are more N's than V's in each label. However, the OTO tag for label 2 is V, since label 1 acquires the N tag, having one more N than label 2. The MTO accuracy is 0.625 since five of the eight tags match. The OTO accuracy is four out of eight or 0.5.

Global and Local Performance Measures

An evaluation criterion can be useful if it captures our intuitive understanding of performance in one of two ways. First, if it can compare any two models and tell which is better (in agreement with intuition), we would say that it is a globally reliable criterion.

However, it is also common to use a criterion to compare two closely related models to determine which is better – a local performance measure. Local evaluation is important in its own right even if no global measure is known. For instance, if the two models being compared belong to the same family, and differ only in the value of some parameter, local evaluation is then equivalent to the study of a performance gradient. A criterion that behaves well (i.e. in agreement with intuition) locally, even if it fails as a global performance measure, can still be used to provide important theoretical insight and may be useful for the study of convergence of training algorithms and for parameter adjustment.

We first begin investigating the evaluation criteria in order to better understand how reliably VI measures global performance. Johnson (2007) points out that VI tends to favor a small number of clusters. To emphasize this observation, we point out that the NVI score is always one when comparing the gold tags to a trivial "clustering" (denoted by C_0) that gives all tokens the same label.

$$NVI(G;C_0)=\frac{H(G)+H(C_0)-2\mathrm{I}(G;C_0)}{H(G)}=\frac{H(G)+0-2*0}{H(G)}=1$$

It stands to reason then that that the NVI score of a good tagger should lie between zero (the score assigned to a "perfect" clustering) and one, and indeed should be substantially smaller than 1. Surprisingly, leading unsupervised POS taggers typically report NVI scores of approximately one (and often slightly larger than one), calling into question the value of NVI as a global performance measure, and pointing to the need for a careful study of all three criteria as both global and local measures.

Global performance

The fact that the trivial clustering described above scores better than many taggers already indicates a potential problem with NVI. We use a more carefully constructed, but still trivial, clustering as our baseline for studying the global performance of these evaluation criteria. This "most frequent word" (MFW) baseline was first described by Clark (2003). For a set of k labels, the MFW clustering assigns label 1 to all tokens of the most frequent word type in the corpus. It assigns label 2 to the second most frequent word, and so on for each of the $k - 1$ most frequent words. It then assigns the tokens of all the remaining (less frequent) types the label k which we call the *catch-all* label. This baseline has the important property that it captures the Zipfian nature of type distribution (i.e. types obey a heavy tailed power-law) in natural language, but because it only depends on type frequencies, it cannot capture any of the context information available to the taggers. If a criterion reports that a collection of taggers fail to score higher than the MFW baseline, then the criterion is seriously flawed as a global performance measure unless, of course, all the taggers themselves are seriously flawed.

In order to test thee three criteria (VI, MTO, and OTO), we compare the performance of several models against the MFW baseline. We use the full Wall Street Journal corpus of the Penn Treebank (see Marcus et al. 1993) which consists of 1,173,766 tokens, resulting, with all words lower-cased, in 43,766 word types. These tokens have been hand labeled using forty-five POS tags (the PTB45 tagset). A second coarser tagset with only seventeen tags (PTB17) is constructed by combining some similar PTB45 tags into a single tag, following the work of Smith and Eisner (2005). The two tagsets are shown in table 2.2 in their entirety, and we use each as we explore these criteria.

PTB45 Tag	Part of speech	Example
$	dollar	*$, US$, HK$*
``	opening quotation mark	*", '*
'	closing quotation mark	*", '*
(	opening parenthesis	*([{*
)	closing parenthesis	*)] }*
,	comma	*,*
--	dash	*--*
.	sentence terminator	*. ! ?*
:	colon or ellipsis	*: ; ...*
CC	conjunction, coordinating	*and, both, but*
CD	numeral, cardinal	*10, ten*
DT	determiner	*the, this, these*
EX	existential there	*there*
FW	foreign word	*habeas corpus*
IN	preposition or conjunction, subordinating	*in, below, for*
JJ	adjective or numeral, ordinal	*happy, first*
JJR	adjective, comparative	*happier*
JJS	adjective, superlative	*happiest*
LS	list item marker	*First, Second, Third*
MD	modal auxiliary	*can, should, ought*
NN	noun, common, singular or mass	*citizen*
NNP	noun, proper, singular	*Obama, Bush*
NNPS	noun, proper, plural	*Americans*
NNS	noun, common, plural	*citizens*
PDT	pre-determiner	*many, half*
POS	genitive marker	*'s*
PRP	pronoun, personal	*him, herself*
PRP$	pronoun, possessive	*my, mine*
RB	adverb	*happily*
RBR	adverb, comparative	*further*
RBS	adverb, superlative	*furthest*
RP	particle	up (*as in*: *give* up)
SYM	symbol	*% & @*
TO	"to" as preposition or infinitive marker	*to*
UH	interjection	*Wow*
VB	verb, base form	to *eat*
VBD	verb, past tense	*ate*
VBG	verb, present participle or gerund	*eating*
VBN	verb, past participle	*eaten*
VBP	verb, present tense, not 3rd person singular	*terminate*
VBZ	verb, present tense, 3rd person singular	*terminates*
WDT	WH-determiner	*what, which*
WP	WH-pronoun	*who, whom*
WP$	WH-pronoun, possessive	*whose*
WRB	Wh-adverb	*whence, where, why*

PTB17 Tag	Part of speech	Example
ADJ	adjective	*red*
ADV	adverb	*quickly*
CONJ	conjunction	*but*
DET	determiner	*the*
ENDPUNC	end mark punctuation	*!*
INPUNC	interior punctuation	*,*
LPUNC	left punction marker	*[*
N	noun	dog
POS	possessive marker	dog*'s*
PREP	preposition	*in*
PRT	particle	give *up*
RPUNC	right punction marker	*]*
TO	to marker	*to* jump
V	Verb: past, present, or future	*eat, ate*
VBG	verb: gerund or present participle	*eating*
VBN	verb: past participle	*eaten*
W	"wh" word	*who, which, where*

Table 2.2: (Top) The full Penn Treebank tagset contains 45 tags.

(Bottom) The reduced tagset of Smith and Eisner 2005 has 17 tags.

Criterion	Model	PTB17	PTB45
MTO	MFW	0.621	0.546
	SVD2	**0.74**	0.658
	HMM-EM	0.647	0.621
	HMM-VB	0.637	0.605
	HMM-GS	0.674	**0.660**
OTO	MFW	**0.564**	0.419
	SVD2	0.523	0.466
	HMM-EM	0.431	0.405
	HMM-VB	0.514	0.461
	HMM-GS	0.466	**0.500**
NVI	MFW	**0.711**	**0.709**
	SVD2	0.928	0.888
	HMM-EM	1.198	1.031
	HMM-VB	1.067	0.99
	HMM-GS	1.074	0.931

*Table 2.3: MFW-baseline levels and average model performances under the three criteria. All HMM figures are from Gao and Johnson (2008), and we include only their best result for each. A score is **bold** when it is superior to all other methods and is underlined when it is inferior to all others.*

We chose several models for comparison including the three HMM taggers of Gao and Johnson (2008), which use expectation-maximization (EM), variational Bayesian EM (VB), and the Gibbs sampler (GS). We also include the two-step SVD tagger, SVD2, described in the subsequent chapter to ensure that our conclusions are robust to these two very different types of models. We report the scores in table 2.3.

We begin our table and this discussion with MTO because it reliably ranks all methods above the corresponding MFW baseline for both tagsets. This behavior is exactly as expected, although two methods only narrowly defeat their baseline for the

PTB17 tagset. In other words, this evaluation criterion does match our intuition about global performance by ranking (presumably) informative models above the uninformed baseline.

This behavior is in stark contrast to NVI, which is shown last. Observe that the NVI score of the baseline under both tagsets is *significantly* better (i.e. lower) than *any* of the NVI scores of the models. In fact, it gives the models scores that range from 25% to 70% higher (i.e. worse) than the baseline, telling a radically different story about global performance than MTO. We believe that the models are, in fact, informative, as MTO would indicate, so we are left to conclude that NVI is grossly unreliable as a global performance measure for unsupervised taggers performing at current state-of-the-art levels. It is reasonable to expect that if the state of the art improves so that these taggers can produce NVI scores significantly better than the MFW baseline, NVI may then become useful as an informative criterion, but until then it is globally unreliable.

The OTO criterion is somewhat ambiguous. While the models consistently score worse than the baseline under PTB17, most of the models perform better, albeit only slightly, under PTB45. As with NVI, we expect that as unsupervised POS taggers improve, their OTO scores could grow to the point that OTO would become unambiguous and therefore useful for evaluation; however, it appears that, at present, this criterion is unreliable. We further speculate that reliable performance of OTO may require a smaller advancement in the state of the art than would be needed to make NVI reliable, because of the smaller difference between the baseline and the taggers.

Local performance

Despite the fact that OTO and NVI do not perform consistently well as global evaluation criteria, it is possible that they may still be quite useful as measures of local performance when comparing reasonably similar taggers. This similarity restriction prevents comparison with the MFW baseline, because this baseline is simply too different from the kind of labeling produced by a serious model. Furthermore, it is unclear that any of the models we use are similar enough to one another to assess the criteria as local performance measures. Therefore, we developed a process for generating, from any single model, a "continuum" of very similar models. As we shall see, there is an obvious intuitive sense in which this process should be incrementally crippling the model's ability to tag effectively, so we expect that the NVI, OTO, and MTO scores should steadily degrade throughout this process, if indeed the criteria are reliable as local performance measures.

We call the process of slowly crippling a model *decimation*, and it can be applied to the labeling produced by any model. We begin with the original labeling which gives us our first model in the continuum. To get the next model, we find the least frequent word type in the corpus and replace the labels assigned to each of its tokens with a new *amnesia* label leaving all other labels unaltered. (Actually, the several least frequent types only have one token to change.) For each successive new model, we take the labeling of the previous model in the continuum and change the labels of the tokens of the next rarest word to the amnesia label. With over forty thousand types in the corpus to be decimated, this process produces a sequence of over forty thousand labelings, which locally differ only in the tokens of a single type. At one extreme, we have the original

labeling of the model. At the other extreme, when all types have been decimated, all tokens have the amnesia label.

We use the term *amnesia* to describe the decimation label because, in effect, the tagger simply forgets how it intends to label tokens of the decimated types. Although it uses the decimated types for learning and intends to label their tokens, when the time comes for evaluation, the decimated model simply forgets the intended labels and reports the amnesia label instead. This decimation strategy is somewhat reminiscent of the MFW baseline that was used for global evaluation. When a model is severely decimated, it may only attempt to label the tokens of a few most frequent word types, while all tokens of all other types are given the amnesia label (similar to the *catch-all*). Unlike the MFW, however, the tokens the most frequent types preserve the labels that were originally assigned by the model. In contrast to the MFW baseline, it need not be the case that all tokens of an un-decimated type share a common label. It is therefore possible for the decimated model that forgets all but the k most frequent types to score better or worse than the k MFW baseline.

We apply the decimation process to both our SVD2 tagger and the HMM-VB tagger, using, for the latter, data provided by Mark Johnson. As with the global performance investigation, we use the full WSJ corpus with both PTB45 and PTB17 tagsets. Figure 2.2 shows the scores of each criterion on the collection of decimated labelings that come from the HMM-VB model and PTB17. Each plot from left to right represents progressively more severe decimation. The x-axis is on a log scale and records the number of word types that are spared. The MFW baseline score (17 most frequent

words) and the score of the full, un-decimated model are indicated for comparison.

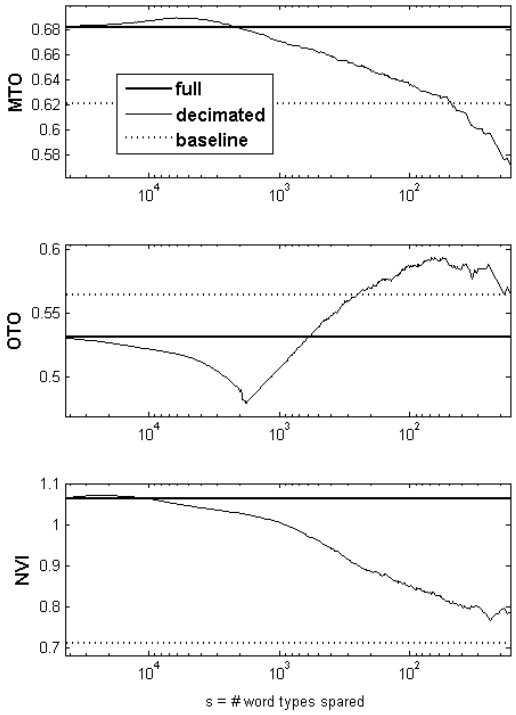

Figure 2.2: Performance of the decimated HMM-VB model on PTB17 according to all three criteria. Solid horizontal lines show the performance of the full model with no decimation. Dotted horizontal lines show the MFW baseline.

Again, we begin our analysis with MTO because it behaves almost exactly as expected. After a very minor increase in MTO score (which occurs over the range of the rarest words, which the tagger is less likely to label correctly), progressively more severe decimation results in a steady decrease in MTO score, which is no surprise. This behavior indicates that MTO's evaluation matches our intuition, which predicts that

crippling the model by decimation should hamper its ability to perform (at least over the range of word types that the model is likely to be tagging well).

Remarkably, NVI exhibits the exact opposite behavior! After a brief period of minuscule worsening (increase) of the NVI score, the NVI score begins to monotonically decrease, indicating steady *improvement* in tagging performance. Again, according to NVI, decimating the model actually improves it. This behavior is indeed counter-intuitive, and, given the contradictory MTO results, indicates a fundamental flaw with NVI.

The OTO scoring is, once again, ambiguous. We observe the expected steady decrease in performance as the number of spared types decreases from more than 40,0000 to about 2,000. Then, unexpectedly, the OTO score increases quickly as yet more types are decimated, so that when fewer than about 800 hundred types are spared, the performance exceeds that of the full model. This dramatic change in the gradient behavior is unsettling, and again indicates a deficiency in the evaluation criterion.

The story remains the same when the decimation process is applied to our SVD2 model. Figure 2.3 shows the scores for each criteria on this very different model, this time using PTB45.

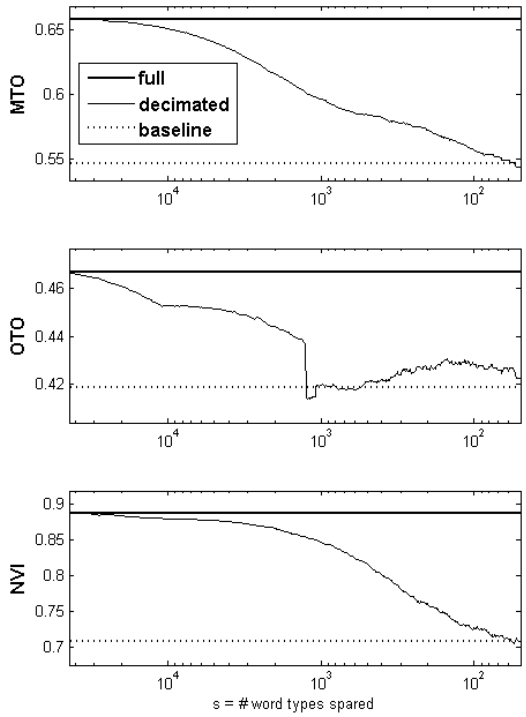

Figure 2.3: Performance of the decimated SVD2 model on PTB45 according to all three criteria.

The story does not change much for either MTO or NVI even when using the different tagset and this very different model. We now see no small early increase in MTO with decimation (presumably because SVD2 is handling the infrequent words better than the HMM-VB), but rather the steady decrease is consistent throughout. Furthermore, throughout the entire decimation process, NVI wrongly indicates that decimation causes improvement, while MTO correctly indicates the opposite. The OTO behavior is, once again, disturbingly strange. Although we do not observe the sharp change in gradient observed with HMM-VB, we now see a discontinuous drop of over 2% in the OTO score at around 1200 spared types, followed by a smaller discontinuous

jump at just over 1000 spared types. In this example, although OTO for the most part indicates that decimation does not improve the performance, it remains unreliable, as two very similar labelings (on either side of the discontinuity) produce very different OTO scores. Between the two figures, we see that OTO can (and frequently does) show discontinuous behavior in both the scores and the gradients.

The discontinuous behavior in the scores is a manifestation of the greedy search procedure used in OTO. It is useful to consider a simple example illustrated in table 2.4 to see how this discontinuity can arise. Suppose our corpus consists of only 33 tokens (21 nouns and 12 verbs), and our tagger generates one of two labels for each token. Suppose it assigns label one to 11 nouns and 3 verbs while label two is assigned to 10 nouns and 9 verbs. The greedy OTO then maps label one to the tag noun, because the 11 matches represent the largest possible single contribution to the score. Label two is then assigned to the tag verb, which gives a total accuracy score of 20 tokens tagged correctly out of 33. Suppose next that we decimate two nouns from label one. Now, the OTO map assigns label two to the noun tag, because its 10 nouns make the most possible matches. Label one is then mapped to verb, and we ignore the two "amnesic" nouns which contribute nothing to the score. The OTO score is now only 13 out of 33. We see that a small change in the labels results in a big change in the greedy OTO score.

	Before Decimation			***After Decimation***		
	Noun	Verb	OTO Tag	Noun	Verb	OTO Tag
Label one:	**11**	3	N	9	**3**	V
Label two:	10	**9**	V	**10**	9	N

*Table 2.4: A simple illustration of the greedy OTO map before and after a small decimation. The entries that contribute to the OTO score are in **bold**. The "before" score is 20 while the "after" is only 13, illustrating the potential for discontinuities.*

If we do not use the greedy search, notice that the optimal OTO map continues to map label one to noun and label two to verb, for a total score of 18 out of 33. Under the optimal OTO map, the drop in score comes from exactly the two amnesic words. In fact, it is easy to see that the optimal OTO score can never change in a discontinuous way, because the score cannot change by more than the number of tokens that change. If one token switches labels and the score increases (or decreases) by more than one, then the map used for the higher score could be applied to the other labeling with a drop in score of only one. Unfortunately, the optimal OTO map is computationally difficult to find, and we are left to rely on the greedy search.

Figures for the SVD2 on PTB17 and HMM-VB on PTB45 are included for completeness (figure 2.4), but the results are completely consistent. MTO always shows a smooth decrease as the model is impaired, while the NVI score steadily improves. The OTO score often shows cusps and discontinuities, and fails to tell a consistent story for either model and either tagset. While we explore the map-based criteria in more detail in the next section, the conclusion is already apparent. Of the three criteria commonly used with unsupervised POS taggers, MTO is the only one that can be reliably used for either global or local evaluation.

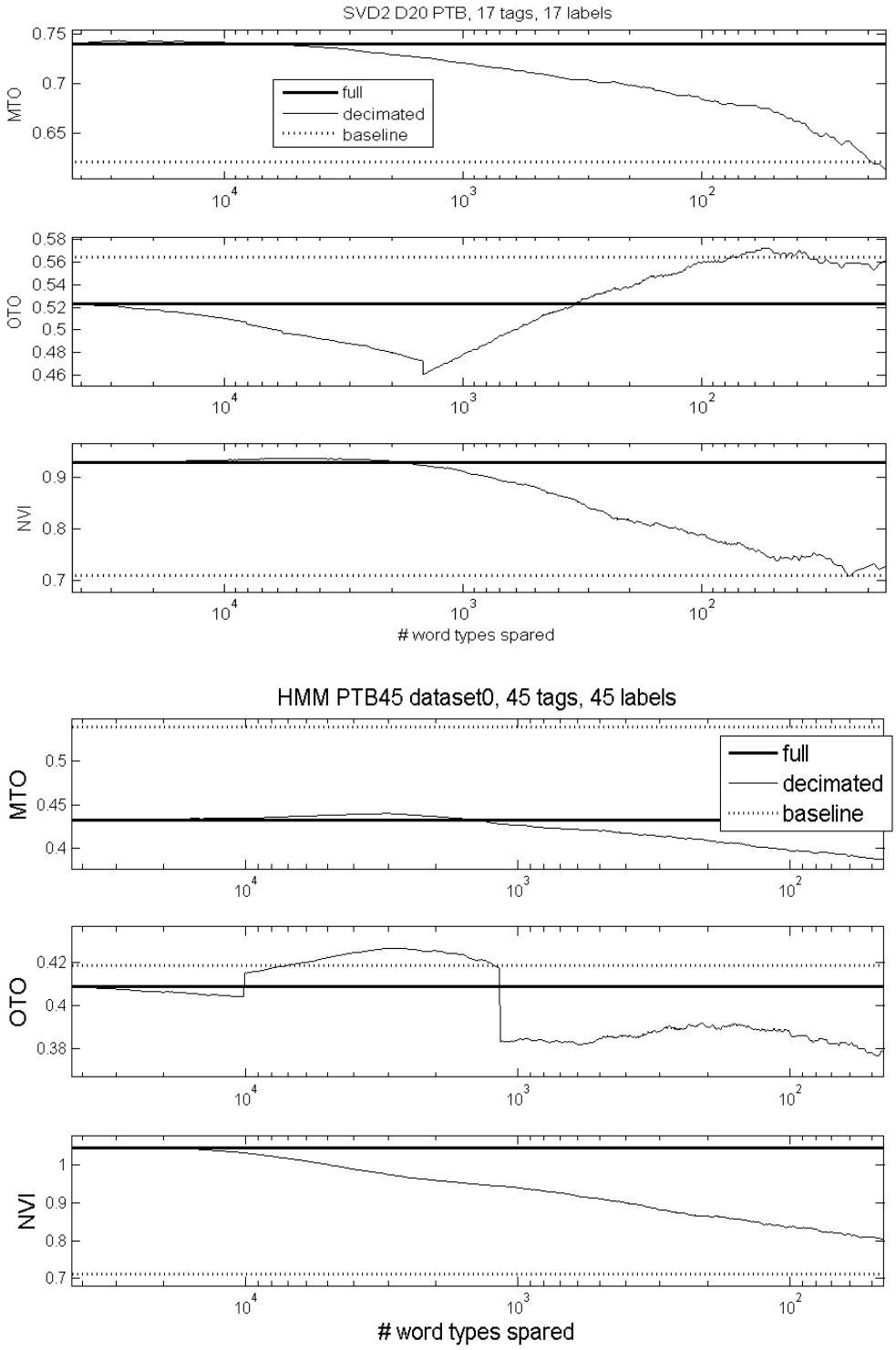

Figure 2.4: Decimation performance of all criteria for SVD2 PTB17 (top) and HMM-VB PTB45 (bottom). Results are consistent with those shown in figures 2.2 and 2.3.

A Closer Look

Although we want our tagger to find exactly the fifty or seventeen POS categories of the tagsets created by the linguists, the unsupervised nature of the task makes this impossible to control. Any unsupervised approach can only hope to extract the statistically most salient features, whether they are POS, semantic themes, or some other feature. As a consequence of this limitation, it is possible for the tagger to make distinctions between words that the linguists classify with just a single tag, causing the tagger to use two or more labels when only one is desired. For example, the tagger might, find transitive and intransitive verbs, or animate and inanimate nouns, neither of which are distinguished with either PTB17 or PTB45. On the other hand, the tagger may fail to make a distinction between the words of two or more distinct tags, combining them instead into a single label. For example, present and past participles, which are given separate tags, might mistakenly be combined into one label, or, worse, the same might happen for nouns and verbs.

While neither type of "mistake" is desirable, one could argue that the first is less critical than the second. While failure to distinguish past and present participles may not seem too severe, failure to distinguish nouns and verbs certainly is. The potential for this kind of catastrophic failure is present anytime the tagger fails to distinguish two tags. On the other hand, if a single tag like noun is split into two or more labels, as long as each label contains only nouns, the mistake is arguably only minor.

In a sense, the OTO criterion punishes each of these two very different kinds of "mistakes" in the same way, while MTO treats them very differently. If all the nouns and all the verbs are mistakenly given an identical label, both OTO and MTO correctly

penalize the score for all the mislabeled verbs (because nouns are more frequent). However, if the nouns are broken into two equal-sized groups with different labels, OTO cannot map both labels to the noun tag, so the OTO score drops by the population of one entire group. MTO has no such constraint, so both groups of nouns contribute to the score. In practice, taggers do regularly make both kinds of errors, but, as we will demonstrate, the tendency to split certain tags is strong, and we argue that MTO is a more appropriate way of addressing this behavior.

When evaluating a tagger, using either MTO or OTO accuracy, it is convenient to construct an object called the confusion matrix. Element *(i,j)* of the confusion matrix counts the number of times any token of tag *j* is assigned label *i* by the model. Usually, the confusion matrix is normalized by dividing each element by the total number of tokens in the corpus. The MTO accuracy is then the sum of the largest element from each row. For OTO, we collect the largest element of the matrix, eliminate its row and column, take the largest element from the remaining matrix, eliminate its row and column, and continue this process until all rows or columns are exhausted. This collection of "largest" elements are added to compute the OTO accuracy. It is useful to investigate how tokens of each label and tag contribute to the score. For each criterion, the non-zero elements of the confusion matrix that do not contribute to the score are the "mistakes."

In figure 2.5, we show the confusion matrices generated using the SVD2 tagger with PTB17 at either end of the decimation process (i.e., all words spared, and only one word spared for each of the 17 labels). A similar confusion matrix for the undecimated

SVD2 using PTB45 is shown in figure 2.6. The gray level of an element of the matrix indicates magnitude. The red star (one per row) indicates the label-tag pairing for the MTO map. The blue circle (at most one per row and at most one per column) indicates the label-tag pairing for the OTO map. The left panel of this figure highlights the difference in the way the MTO and OTO handle the tag splitting behavior.

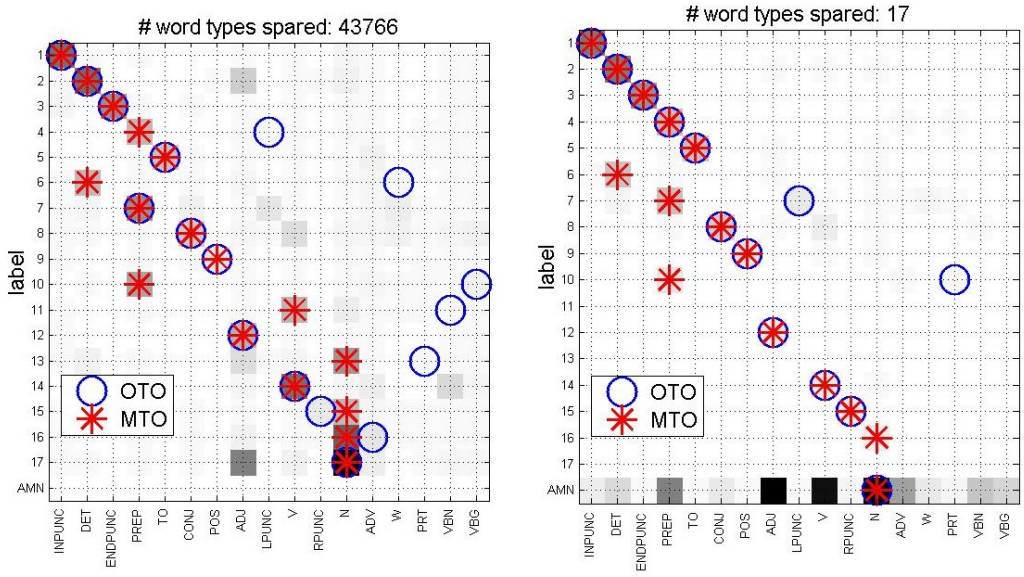

Figure 2.5: Confusion matrix between PTB17 gold-standard and SVD2 clusterings. Grey level indicates proportion of corpus tokens assigned a given label (row index) and a given PTB tag (column index). MTO and OTO maps are indicated, respectively, by red stars and blue circles. Left panel shows the confusion matrix for the full model, where all 43766 word types are spared from decimation. Accordingly, the bottom row, corresponding to the amnesia label, is not populated. Right panel shows a severely decimated model, where all but the 17 most frequent words have dropped into the amnesia label, AMN. Both the MTO and the OTO maps interpret label AMN as PTB17 tag N.

Looking at the noun (N) column of the un-decimated (left) confusion matrix, we see that labels 15 and 16 contain almost exclusively nouns. Labels 13 and 17 are also dominated by nouns, although each contains a significant number of adjectives as well. The red star for each of these four labels in the noun column indicates that MTO maps all

four labels to the noun tag. The mislabeled adjectives in 13 and 17 do not contribute to the score. OTO is not free to make such a choice of mapping: because label 17 contains the largest number of nouns, greedy OTO assigns label 17 the noun label. Labels 15 and 16 (which contain almost exclusively nouns) are then mapped to right punctuation and adverb respectively, giving virtually no contribution to the score. Label 13, which has mostly nouns but also many adjectives, is mapped to the seldom used particle tag, also giving no contribution to the score. While breaking the nouns into four labels is not ideal, and confusing nouns and adjectives is certainly a mistake, OTO maps three of the four labels in question to nonsense, which is not helpful in capturing the true behavior of the model. We argue that by failing to capture this kind of behavior, the criterion is failing to match our intuition about what makes one unsupervised labeling better than another, causing it to be an unfit tool for evaluation.

The right panel of figure 2.5 helps demonstrate why NVI fails so consistently. Recall that VI is the sum of the entropies of the clusterings minus twice their mutual information.

$$VI(G;C) = H(G) + H(C) - 2\mathrm{I}(G;C)$$

The shortcomings of the greedy OTO lie in the imposed "tug-of-war" between labels over particular tags, which can produce very different OTO scores for very similar labelings, leading to discontinuities or sharp gradient changes. The trivial example in Table 2.4 shows how this discontinuity can arise, but it is interesting to see how this process plays out on a non-trivial example. We first explore the decimation that leads to the cusp previously shown in figure 2.2. Later, we explore the discontinuity of figure 2.3.

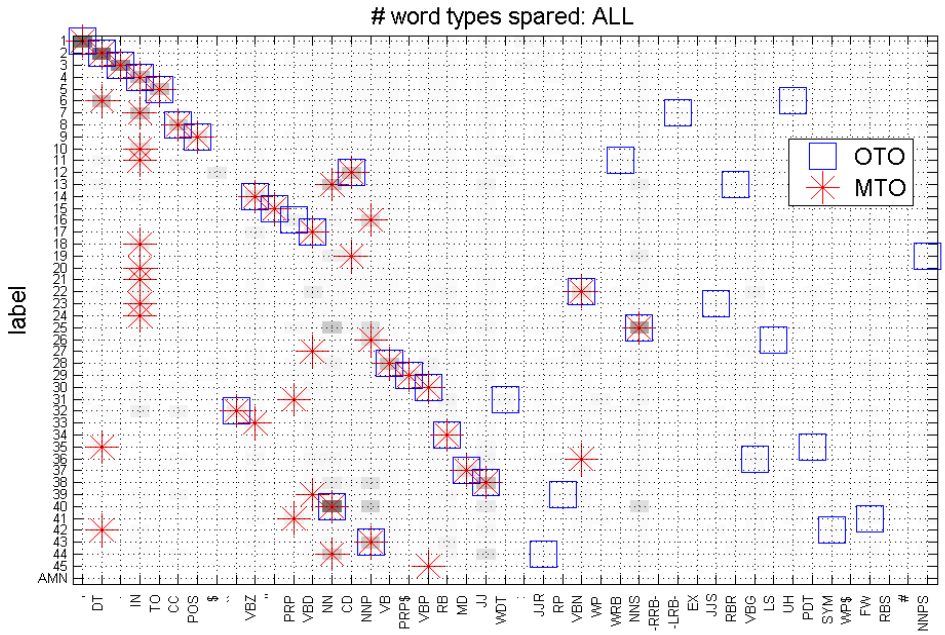

Figure 2.6: Confusion matrix for the undecimated SVD2 tagger using PTB45. Many open-class tags have so few tokens that they do not ever acquire an MTO label, while several closed-class tags are given multiple MTO tags. Many of the OTO tags are assigned to labels in a way that has little or no contribution to the score.

Figurew 2.7 and 2.8 show how a few carefully chosen labels contribute to the overall performance of the OTO and MTO scores as the HMM-VB model (again, previously shown in figure 2.2) is decimated. The total OTO or MTO score at any point in the decimation process is the sum of the points on these curves, along with the curves for all the remaining labels – which are not shown here.

The entropy of the gold standard, G, is fixed. Whether or not a change to clustering, C, will improve VI is determined by a trade-off between the change in the entropy of the clustering and the change in the mutual information that it produces. Comparing the two panels of figure 2.5, we see quite obviously that the entropy decreased from the full model to the severely decimated one. This drop must come with

a corresponding decrease in the mutual information, but the drop in mutual information (which hurts the NVI score) was not nearly sufficient to overcome the large drop in entropy. The net effect was an increase in NVI.

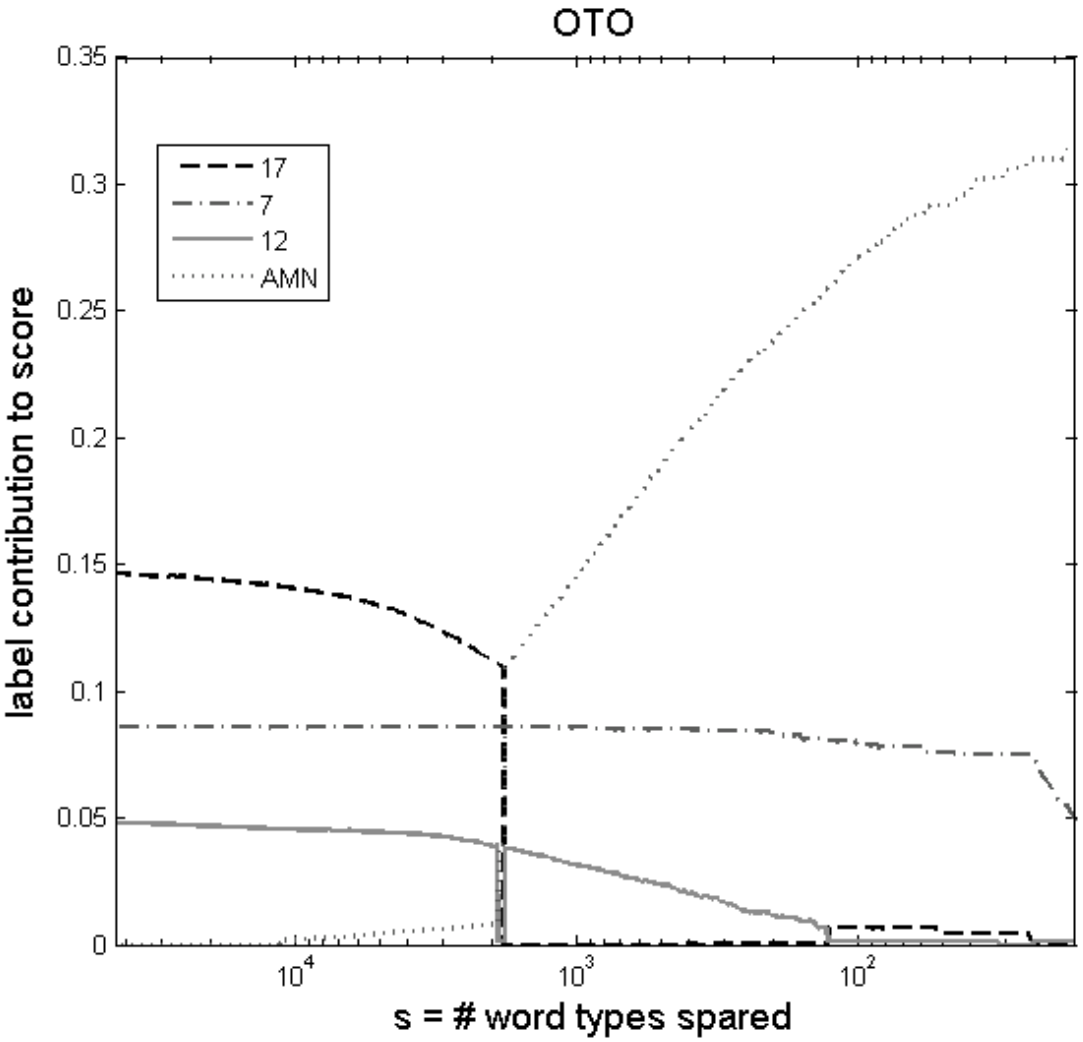

Figure 2.7: The contribution to the OTO score of several labels as the HMM-VB, PTB17 data is decimated. This decimation breakdown is for the same HMM-VB data whose total scores (MTO, OTO, and NVI) are shown in figure 2.2.

Let us begin with the *amnesia* label. When no decimation has taken place (the

leftmost point in each graph), this label contains no tokens, because no words have been "forgotten." As we decimate, this label gets filled with rare words – most often nouns but also other unusual words from other open classes like adjectives and verbs. Because the number of tokens in this label is very small at first (since the forgotten types necessarily have few tokens), the OTO map has no incentive to assign this label to any particular tag and the contribution of the label remains essentially zero at first. Then, even as the population of this label increases, it still cannot receive any of the most significant open-class tags until it acquires enough tokens of any one tag to steal that tag from its original label. This first happens when about thirty thousand of the forty two thousand types have been decimated. At this point (roughly 10^4 spared types), the contribution to the OTO score begins to visibly grow, because verb has now been assigned to this label, and the number of decimated verb tokens is increasing. It should be noted that throughout decimation, there are many more nouns and adjectives with the amnesia label than verbs, but other labels (17 and 12 in particular) are able to hold the noun and adjective tags longer than any label can hold the verbs.

At just under 2000 spared word types, we see several discontinuities in the curves. In fact, there are actually two label swaps that take place in quick succession over the course of only a few additional decimated types. First, label 12, which is originally mapped to adjective, has its tag "stolen" by the amnesia label. Because label 12 contains almost exclusively nouns and adjectives and the more populated label 17 holds the noun tag, label 12 loses its contribution to the OTO score, as it does not receive either of its preferred tags. At this point, there is small discontinuous drop in the total OTO score,

because the amnesia label and label 12 have nearly identical numbers of adjectives, hence the swap does not affect the number of correctly tagged adjectives. However, the verbs of the amnesia label that were formerly tagged correctly are now lost entirely, and there is no corresponding gain from the new tag assigned to label 12. The drop is relatively small, because the number of lost amnesic verbs is small.

This label mapping is short-lived, because just a few more decimated types produce another swap. Now, the number of nouns in the amnesia label has grown larger than the number of nouns in label 17, so *AMN* gives up its adjective tag in favor of the noun tag and its score jumps. Label 17 contains almost exclusively nouns, so, as it loses the noun tag, its contribution to the score plummets to zero. If this were the end of the story, we would see a large drop in the total OTO score equal to the number of adjectives in the amnesia label. This extreme discontinuity does not happen, because this change in the noun tag frees up the adjective tag, and label 12 quickly reclaims it, causing its score contribution to jump back up its previous level. The net effect of these two successive tag swaps is a very small drop in the total OTO score, equal to the loss of the verbs in the amnesia label.

The more important effect of this swap is the resulting cusp in the total score. Prior to the tag swapping, the OTO score steadily decreases as anticipated. This decrease happens in large part because label 17 was losing correctly tagged nouns faster than the amnesia label was gaining any correctly tagged tokens. (Similarly, other labels were losing correctly tagged tokens of other POS tags, but this contribution is smaller.) Once the amnesia label usurps the noun tag, decimation begins to improve performance

because the number of now correctly tagged nouns in the amnesia label grows faster than the number of words of other tags that are lost to the noun-tagged amnesia label. This effect continues until we begin to decimate primarily closed-class words, at about 100 spared types, at which point the slope of the OTO score curve on figure 2.2 reverses again. We do not include in this figure the labels that would be necessary to see this effect, but we do show label 7, which is the most frequent closed-class label. Because the closed-class types tend to have the highest frequencies, they are mostly spared from the decimation process until the end; hence, their contributions to the OTO score vary little.

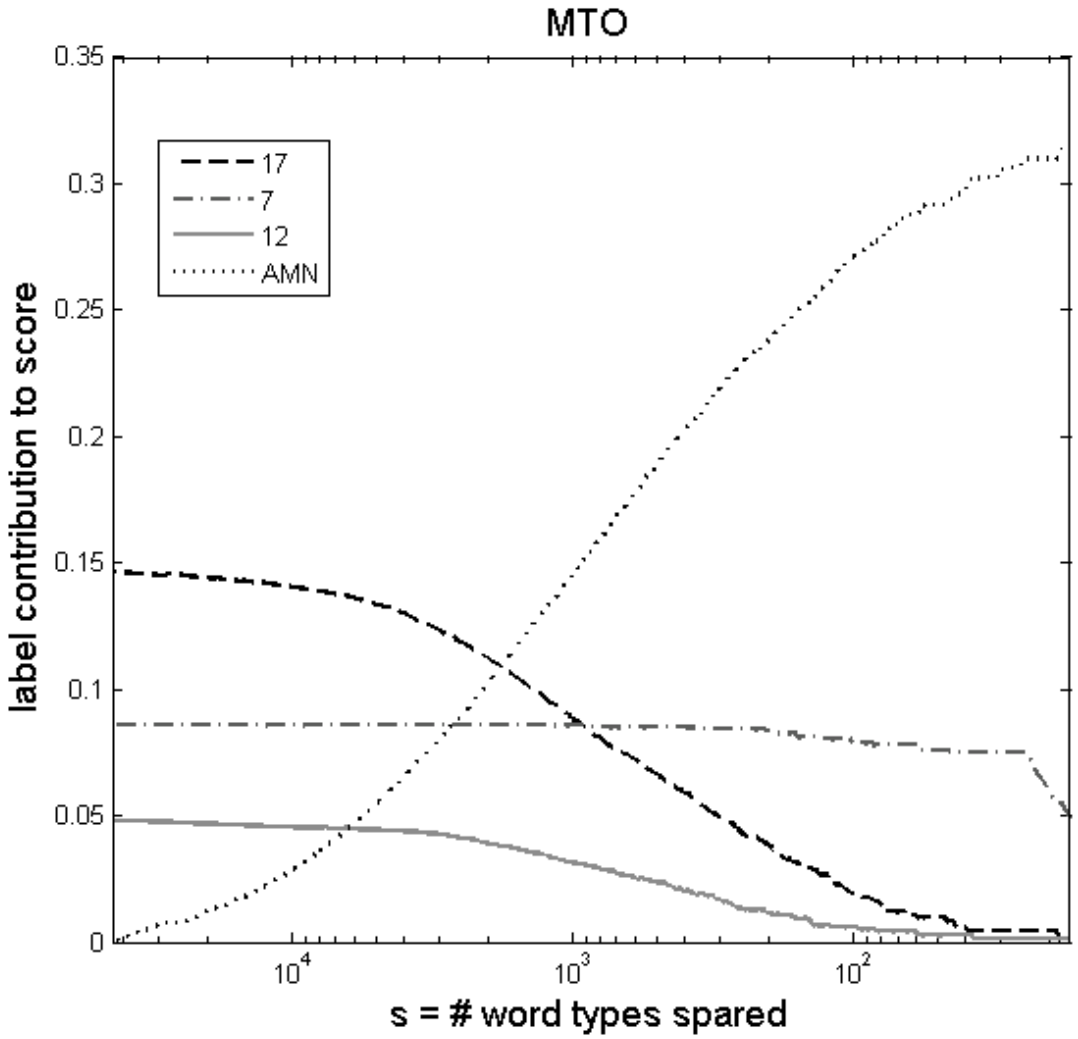

Figure 2.8: Contribution to the MTO score of the same HMM-VB, PTB 17 decimation data shown in figures 2.2 and 2.3.

This erratic behavior of the OTO components is in stark contrast to the MTO behavior seen in figure 2.2. Here, we see that the amnesia label, initially unpopulated, begins to contribute to the score immediately. This contribution is small at first as the number of decimated tokens grows only slowly. While the amnesia label's contribution grows, the contributions of labels 17 and 12 (and others not shown) decrease steadily.

The amnesia label acquires mostly nouns throughout decimation, so it is immediately tagged as a noun label and this *never* changes. As decimation proceeds, the change to MTO score comes from two sources. First, rare nouns that were incorrectly labeled by the model get decimated to the amnesia label, causing an increase in the score. (Rare nouns that were already labeled as nouns swap labels but not tags, so the score is not affected.) Second, sufficiently rare words of tags other than nouns (e.g. adjectives, verbs, and adverbs) that were formerly tagged correctly get tagged as nouns after decimation, causing a decrease in the score. As seen in figure 2.2 the net effect of these two changes early in decimation is a very small (less than 1%) increase in overall MTO score. This increase is not surprising, when we consider that most rare words are nouns, and rare words are the most difficult to tag correctly. Since most are nouns, we most often stand to gain by decimating a rare type into a noun-tagged amnesia label. Also, since rare words are difficult to tag correctly, it is likely that many decimated non-nouns were not tagged correctly prior to their decimation, so no price is paid for moving them to a noun label. In the end, slightly decimating this model with this tagset does lead to a minuscule improvement in the labeling, but as we move away from the rarest words and begin to decimate more common words, the combined MTO score curve begins to decrease. This decrease comes about because most of the open-class words that join the amnesia label are either correctly labeled nouns – and these do not help the score when switched – or are correctly labeled words of other tags – and these hurt the score when switched.

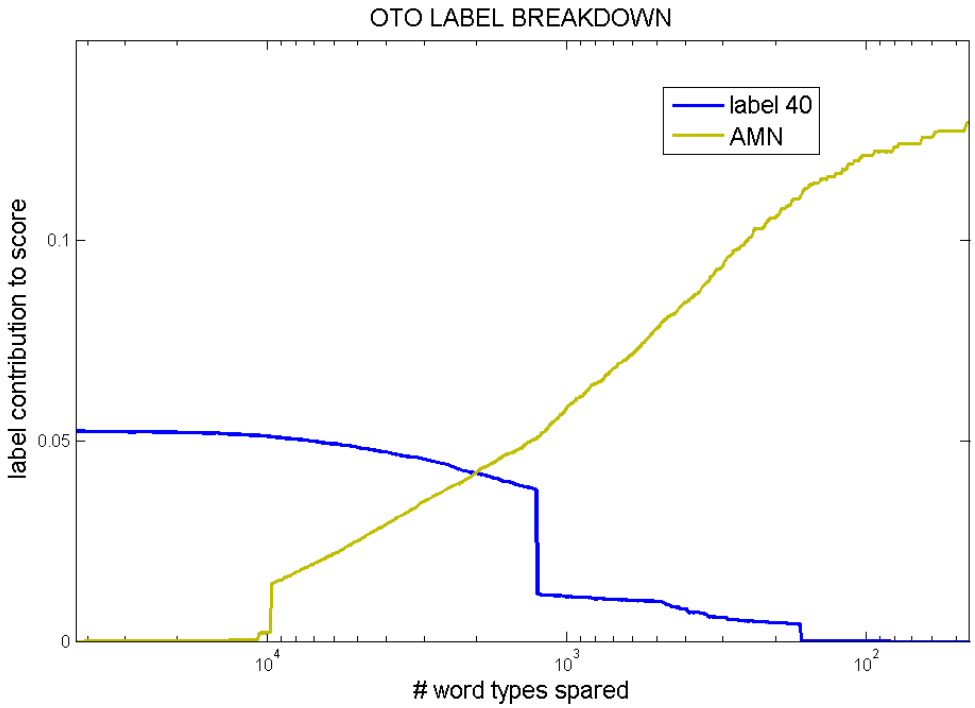

Figure 2.9: Contribution to the total OTO score of two labels for SVD2 PTB45. We see a sudden drop in the contribution of a label with no corresponding rise in the amnesia label, producing a discontinuous drop in the OTO score.

We see the undesirable discontinuous behavior of OTO in figure 2.9, which corresponds to figure 2.3, describing the decimation of SVD2 under PTB45. In this figure, we focus only on two labels to understand the discontinuity. In this example, there is no three-way fight for tags. Label 40, which contains primarily singular common nouns and proper nouns, is initially given the common noun tag. It steadily loses tokens throughout decimation, but keeps its common noun tag until only about 1500 undecimated types remain. At roughly 10 thousand undecimated types, the AMN label acquires the proper-noun tag. From this point on, its score contribution steadily rises, as more proper nouns are added, while the contribution of label 40 slowly falls as its types

are moved to the AMN label. At about 1500 spared types, these labels switch tags, because the number of common nouns has just begun to exceed the number of proper nouns in the AMN label, and this label contains far more of both than does label 40. This switch does not change the contribution of the AMN label, because the difference between its two kinds of nouns is small. However, the contribution of label 40 immediately drops by over 3%, because it has far fewer proper nouns than common nouns remaining. The resulting discontinuity is apparent in the combined OTO score shown in figure 2.3. However, we again emphasize that this drop is a result of the greedy nature of the OTO search. A non-greedy OTO would recognize that AMN should continue to hold the proper noun tag, leaving the common noun tag for label 40 until such time that their combined contribution begins to become higher, then switching the tags and resulting in a smooth transition.

Unlike OTO, under the MTO map, labels never compete for a single tag. Small changes in the labeling of the data, therefore, cannot produce discontinuous changes in the MTO score, neither are they able to yield rapid changes in the slope of the overall MTO curve. This smooth behavior is obviously desirable when studying the effect of model parameters, making MTO the preferred local measure.

Summary

The most obvious conclusion from this investigation is that VI (and NVI), despite its frequent use for unsupervised POS tagging and nice property as a map-free metric, exhibits behavior that is so directly contrary to what intuition would demand that it

should not be used at all. This conclusion holds at least until agreement with the gold standard is significantly improved, but the necessary improvement may not be possible. Recall that driving any tagger to find exclusively POS regularities without supervision is simply not possible. For this reason, VI may never become a useful evaluation tool for this task.

Similarly clear is that MTO performs well under these tests. It consistently ranks taggers above the MFW baseline, and it is sensitive to decimation in a way that closely matches intuition. Of course, it may be possible that MTO will fail to behave well under a future test, in which case, like VI, it may turn out to be unreliable. Until such a test is devised, we recommend using MTO because of the consistent, stable behavior we demonstrate.

Although OTO is seemingly not as flawed as VI, the greedy mapping does produce erratic behavior during decimation. Furthermore, the OTO scores of the un-decimated models either fail to beat the MFW baseline or only narrowly defeat it. If this criterion fails to score an actual tagger over a trivial baseline assignment, it is difficult to recommend that it should ever be used as a basis for preferring one model to another. Its freedom to give discontinuous scores to models that differ only slightly also makes it difficult to trust when evaluating how a single model responds to changes in its parameters. Like VI, it may be that, as taggers improve, OTO will become more reliable. In fact, even the greedy search would be completely smooth when the labels match the tags extremely well, but again, there is no indication that an unsupervised tagger can produce such close agreement.

Chapter 3: An Iterative SVD-and-Clustering Algorithm for POS Tagging

Introduction

In this chapter we describe two implementations of a novel algorithm for unsupervised POS tagging that relies on singular value decomposition (SVD). When applied to the WSJ corpus, this algorithm represents a significant advancement in state-of-the-art performance. We begin with a brief discussion of SVD and its use for dimensionality reduction. SVD has a rich history in computational linguistics, and it was first used for POS tagging in Schütze (1995). It has drawn considerably more attention for its use in information retrieval by latent semantic analysis (Deerwester et al. 1990).

With the foundation of SVD established, we move to a general introduction of the algorithm itself and discuss its parameters. This introduction is followed by a description of the two implementations. The first approach, SVD2, is conceptually simple, not requiring the full generality of the algorithm, and resembles somewhat the previous work of Schütze (1995) although it differs in many details of implementation. We demonstrate that even this simple implementation performs significantly better than the recent HMM's of Gao and Johnson (2008) and Graca et al. (2009) on the reduced tagset (PTB17) of Smith and Eisner, but SVD2 achieves only minimally better performance on the full tagset (PTB45). The second implementation, ISVD, is a completely novel algorithm. It takes advantage of the full capability of the approach, and exhibits a dramatic jump in the PTB45 score over both SVD2 and HMM levels of performance.

Singular Value Decomposition

SVD is a well studied matrix factorization technique dating back to the mid-nineteenth century. (See Stewart 1992 for a nice discussion of its development and history.) We state without proof the central theorem as it applies to the field of real numbers (although an equivalent result holds for complex-valued matrices as well).

> *Theorem: Given any real-valued matrix, M, there exists a decomposition:*
>
> $$M = USV^T$$
>
> *such that U and V are orthogonal (i.e. unitary) matrices, and S is a non-negative matrix whose non-zero elements are located on the diagonal. (The superscript T denotes the matrix transpose.)*

The columns of U and V are referred to as left and right singular vectors respectively while the diagonal elements of S are known as singular values. The decomposition is generally not unique even when ignoring the flexibility in the choice of signs as well as possible permutations in the order of the singular values and vectors. It is conventional to oder the singular values by decreasing magnitude along the diagonal of S. The decomposition is generally not unique, in part because any of the columns of U can be multiplied by negative one provided the corresponding column of V is given the same multiplication. Even up to these permutations in sign, the decomposition may not be unique. If singular values are repeated, then any orthonormal basis can be chosen for the

space spanned by the associated left singular vectors. The same is true for the associated right singular vectors. Finally, if the matrix M is not square, then some of its left or right singular vectors will have no associated singular value because the number of singular values is equal to the minimum of the number of rows and the number of columns in M. The "extra" vectors span a space, and any orthonormal set of vectors spanning the same space could be used instead.

Numerical techniques for approximating singular values and singular vectors are readily available in most scientific computing packages. Most of these algorithms have the useful feature that they can compute the singular values and their associated singular vectors sequentially in descending order of the magnitude of the singular value. As we will see, often only a small number of these dominant singular values and their vectors are needed in our algorithm so this sequential computation represents a significant savings in run-time. When seeking the decomposition of a sparse matrix, a stochastic algorithm is available and may be required if the full matrix is too large to decompose without it. The conventional algorithm is deterministic and is based on the QR factorization method (see Smith et al. 1976).

An important property of SVD known as the Eckart-Young theorem and proved in Eckart and Young (1936) can be exploited for POS tagging. The theorem states that SVD produce a (non-unique) solution to the problem of matrix approximation by a lower-rank matrix. Given a matrix M of rank ρ, find a matrix (whose size matches M) of rank $r < \rho$ that best approximates M under the Frobenius norm (i.e. least-square difference). If Ω is the set of rank r matrices whose size matches the size of M, then we seek M^*:

$$M^* = \underset{\tilde{M} \in \Omega}{argmin} \sum_{i,j} (M_{ij} - \tilde{M}_{ij})^2$$

A non-unique solution to this problem arises from simply truncating the SVD by disregarding all the singular values smaller than the r^{th} singular value.

If: $M = USV^T$

then: $M^* = US^*V^T$

with: $$S^* = \begin{bmatrix} s_{11} & & & & \\ & s_{22} & & \mathbf{0} & \\ & & \ddots & & \\ & & & s_{rr} & \\ & \mathbf{0} & & 0 & \\ & & & & 0 \end{bmatrix}.$$

This application of SVD to approximate one matrix by a lower-dimensional facsimile is exploited to extract latent features in our tagging algorithm in much the same way as it is used for latent semantic analysis in document retrieval and many other applications, e.g. in molecular biology (see Wall et al. 2003 and Yeung et al. 2002). We develop this idea in more detail in the following section.

POS Tagging Using Latent Descriptor Vectors

Here, we describe a new algorithm for POS tagging that relies on SVD as well as cluster reassignment using the principles in the well-known *k*-means clustering algorithm (see Steinhaus 1956 and Lloyd 1957). To more easily understand the idea behind our approach, it is useful to first consider a naïve approach to tagging that creates vectors to describe the context of each word, and then clusters these vectors.

Suppose we first convert the corpus of N word types into a large N by N matrix, B, of bigram counts (i.e. B_{ij} counts the number of times word type j follows word type i in the corpus.) By concatenating the transpose of B with B itself, we then produce a matrix whose rows are vectors that describe the left and right context distribution of a word type (i.e., these rows describe how often every word type appears before and after the word type that corresponds to the row). We could then attempt to cluster these row vectors to group together words with similar context. Under the usual assumption that left and right context strongly inform POS, we might expect that the resulting clusters would correlate well with POS tags. This approach fails for two important reasons, which our algorithm attempts to resolve.

The most important reason for the failure of the naïve approach is that virtually no two words' contexts should be expected to be particularly similar. Why? Most context distributions are extremely sensitive to semantic roles, and some are even sensitive to rules or constraints that are unrelated to POS. As an example of semantic dependence, "food" and "car" are both common singular nouns, but semantic themes dictate that, apart from preceding articles and determiners, the contexts of these two words are quite different. Additionally, even very similar, very frequent words like "a" and "an" can have drastically different contexts due, in this case, to phonological rules. In fact, the right context vectors of these two particular words are as different as possible, even though these two words are arguably the most linguistically similar pair in the English language. The vectors differ so drastically because, by rule, any word following "a" can never follow "an" and vice-versa. This rule implies that the right contexts of these words are

orthogonal. (Note that these vectors are non-negative, so orthogonal indicates the maximum possible separation of the angles which are used for clustering.) Instead of looking for clusters based on the context of observed neighboring words,our algorithm attempts to describe the context with respect to an extracted set of latent features of the language, thus alleviating the problem.

A second issue with the aforementioned naïve approach is that over half the word types in the WSJ corpus (even after excluding capitalization) are used only once or twice. Not only is it difficult to classify such words based on so few observations, but also the dedication to these rare words of so many components of every context vector muddles the process of correctly classifying even the more common word types. These rare words effectively act as noise to any clustering effort. Again, the use of latent descriptors considerably alleviates the problem.

We outline the iterative algorithm in the following subsections, emphasizing the general framework. Later, we describe the two particular implementations in detail. Our iterative algorithm addresses the aforementioned difficulties by incorporating the following key features:

- The use of SVD for dimensionality reduction to extract latent features;
- A gradual refinenement of both the descriptor vectors and the cluster assignments, so that each informs the other;
- The incorporation of progressively less frequent word types in each iteration.

Building descriptors vectors

At the core of our algorithm is the descriptor vector. Every word type has a

descriptor, and every descriptor has a left and right half which measure its left and right context with respect to an ever changing collection of latent features. These descriptor vectors are dynamic entities which self-organize as the iterative algorithm proceeds.

The output of each iteration is a vector of cluster assignments, A_t, which stores the number of the cluster containing each word type. A_t is available at the start of the subsequent iteration, $t+1$, to help build new descriptor vectors. These new descriptor vectors store the context as measured with respect to this set of assignments. How does this process work? The left half of a word type's descriptor vector has one component for each cluster, as does the right half. Each component of the left half corresponds to a particular cluster. This component counts the number of tokens from that cluster that occur as an immediate left neighbor of the vector's word type. For example, suppose our corpus contains the phrase: *my dog ran away*. In response to this phrase, what happens to the vector for the word *ran*? Suppose these four types are currently assigned to clusters numbered one through four respectively. The second component of the left half vector for *ran* would increase by one because *dog* is currently assigned to the second cluster and *dog* is the left neighbor of *ran*. Similarly, the fourth component of the right descriptor increases by one because right neighbor, *away,* is in cluster four. Every token of *ran* in the corpus has neighboring words that give rise to a similar bump in the descriptor vector.

At the start of each iteration, the entire corpus is examined in this fashion so that a new descriptor vector for each type is constructed. The number of components of each half of the descriptor must therefore match the number of clusters from the previous iteration. Similarly, the precise direction of the descriptor vector depends on the exact

state of the assignments. In this way, the assignments always inform the descriptor vectors.

Dimensionality reduction

With a new set of descriptor vectors, the next step of the algorithm is to use SVD to force the vectors into a lower dimensional space. Each iteration, we are free to choose the target dimension, and the strategy for this choice differs in our two implementations. By forcing data into a lower dimension, the algorithm moves some vectors closer together in a way that captures particular latent features (both semantic and syntactic). (It should be noted that this SVD approach is a generalization of the commonly used principal component analysis, PCA. While PCA requires data to have zero mean so that the co-variance matrix can be analyzed rather than a correlation matrix, SVD is appropriate even for non-zero-mean data.)

We treat the left and right halves of a context vector separately and store them row-wise into matrices L and R. Our SVD theorem guarantees that matrix L has a decomposition of the form:

$$L = U_L S_L V_L^T$$

We truncate the matrix of singular values to get a low rank approximation of L producing S_L^* and the approximating matrix L^*. We are free to choose the rank of L^*, but it maintains the same size as L. In general each row vector of context data has been moved in this approximation, but moved as little as possible in the least-squares sense. Recall that this approximating matrix has the decomposition:

$$L^* = U_L S_L^* V_L^T$$

Because the matrix V_L is unitary and therefore an isometry, leaving it out of the multiplication (which gives us a new matrix we denote $L^{**)}$ does not affect the relative geometric relationships between the row vectors, only their absolute directions. Because we only need to cluster the vectors, only their relative positions matter so this final multiplication can be safely ignored.

$$L^{**} = U_L S_L^*$$

Now, the size of L^{**} still matches the size of the original data matrix, L. However, if r is the reduced rank (equal to the number of preserved singular values), then only the first r columns of S_L^* have non-zero elements, which in turn implies that only the first r columns of L^{**} have non-zero elements. Similarly, because only the first r rows of S_L^* have non-zero elements, only the first r singular vectors in U_L contribute to the final product of L^{**}. We keep the upper left r by r block of S_L^* which we denote with a ~ over the matrix, and we keep only the first r columns of U_L and using a similar notation. The resulting matrix, $\tilde{L}$, holds exactly the first r columns of L^{**}.

$$\tilde{L} = \tilde{U}_L \tilde{S}_L^*$$

This final matrix has several desirable properties. First, it is actually low dimensional rather than just low rank since each of its row vectors contain only r components, and r is typically small. Second, its vectors (if appended with zeros to put them into the original space) are isomorphic to the best low rank approximation of the original data, so clustering these vectors is equivalent to clustering the optimal approximations of the data. Finally, to compute this matrix, we only need to find the first r singular values of L and their associated left singular vectors – the full decomposition

need never be found. This truncation of the full SVD results in significant computational savings for large matrices. Identical treatment is given to the matrix of right descriptors resulting in:

$$\tilde{R} = \tilde{U}_R \tilde{S}_R^*$$

The rows of these resulting matrices are the low dimensional descriptor vectors for each word type. Each descriptor (both the left and right halves separately) is then normalized to unit length to facilitate clustering by angle. Finally, the two halves are concatenated creating a single vector for each type that lives in the Cartesian product of two r dimensional spheres, which we call the two-sphere. These updated descriptor vectors for this iteration, D_t, are then, in turn, used to inform the next set of cluster assignments.

$$D_t = \left[\text{unit}(\tilde{L}) \;\; \text{unit}(\tilde{R}) \right]$$

Updating assignments

The last step of the algorithm is to assign each new descriptor vector to a cluster. This assignment is done with one or more steps of a k-means style procedure. From the previous set of assignments, A_{t-1}, which placed every type into one of k clusters, we compute a centroid for each cluster, C_t. This centroid is the average of the newly updated descriptor of each its constituent types based on the previous iteration's A_t output. The average is weighted by type frequency so that the "noisy" rare words have less impact on the centroids. Each type is then assigned to the cluster with nearest centroid by angle. (This assignment represents a minor deviation from the introduction which, for

simplicity, implies Euclidean distance is used.) This centroid-and-assignment update defines one step of the *k*-means clustering algorithm. In classic *k*-means, these steps are, themselves, iterated until convergence. However, in our general framework, any number of these *k*-means steps may be used to refine $A_{t,n}$ before moving to the next iteration. Our choice for the number of steps to employ depends on the particular implementation. In other words, it may or may not be advantageous to run enough steps for *k*-means to converge each iteration. With the new assignments in hand, the iteration is complete, and we proceed to the beginning when new descriptor vectors are created.

Inclusion of rare word types

Although we have until now suppressed this detail from our discussion, we are free to gradually increase the number of word types that are included rather than including them all as described in the so-called naïve approach. This feature is important because it allows us to control the rate at which the rare words are included, attenuating the effect of any noise they might inject. Early exclusion of rare words prevents words with limited contextual information from "confusing" the algorithm in early iterations. However, any algorithm must eventually make an informed guess about these rare words despite their limited context so they must be included at some point.

Any word that is excluded in an iteration is not assigned to any cluster. At the start of the next iteration, therefore, it does not in any way affect the update of any descriptors vector. However, we can include this previously excluded word in this iteration by building its descriptor vector exactly as we build every other vector. Because this rare type was not previously assigned, this new vector does not affect the update of

the centroids, but it is available for clustering and must be assigned to one of the *k* choices. Again, the exact strategy for inclusion of rare words varies depending on implementation.

Implementation 1: The Two-Step Tagger (SVD2)

As outlined in the introduction, SVD2 is a very simple implementation of our algorithm in which only two iterations are performed, thereby forcing the vectors into an abrupt rather than gradual organization. Conceptually, it mirrors the two-step version of the approach of Schütze (1995) in which vectors are created, modified with SVD, and then clustered from scratch in each of the two steps. Interestingly, the two iterations of SVD2 prove sufficient for the PTB17 tagset but are insufficient for PTB45 as we later demonstrate with the second implementation, ISVD. Although the dimensionality-reduction technique in SVD2 is likely somewhat different than that employed by Schütze (his description is somewhat vague), and both the geometry and clustering approaches are certainly quite different, the general procedure is similar.

In this section, we place SVD2 within the framework of our more general algorithm described in the previous section by emphasizing the particular details of the implementation. We then discuss the results, which show that it produces strikingly impressive performance on the PTB17 tagset of the WSJ corpus when compared with the recent, state-of-the-art HMMs.

Iteration 1

We initialize SVD2 with an assignment that puts each word type in its own cluster, but in contrast to the earlier "naïve" approach, we limit the number of types to

include only the 1000 most frequent words. From this initial trivial set of assignments, we assemble a descriptor vector for every word type. These initial descriptors store the context with respect to observable words rather than latent variables, because that is all we have available. From this point forward, all types are included throughout both of the iterations of SVD2, despite the ability of the general algorithm to accommodate gradual introduction of rare types. Since all types are included, we must produce approximately fifty thousand left descriptor vectors each of which has 1000 components (based on the thousand initial singly occupied clusters) and a similar set of right descriptor vectors. These vectors are assembled row-wise into the matrices L and R.

Recall that dimensionality reduction of L and R is performed using SVD. In the first iteration, we force the data into $r = 100$ dimensions from the original $\rho = 1000$, to capture latent features. These new left and right descriptor vectors are normalized and concatenated to produce D_1 placing them on the two-sphere where they await a new assignment.

Recall also that the general framework allows the use of one or more k-means steps to update the assignments; here we choose to let k-means iterate until the assignments converge, so n runs as high as needed. We represent this as infinity, but in practice, a few dozen iterations typically yields convergence. (A much different choice is made in the later ISVD implementation.) The final detail lies in the choice of initial cluster centroids for the clustering update. We drop from the $k = 1000$ trivial clusters used for initialization to only $k = 500$ in the first iteration. We initialize these 500 new centroids using the new vectors of the 500 most frequent types. The initial centroids are

therefore taken to be the new descriptors of the 500 MFW. With these centroids in hand, *k*-means proceeds in the usual way (apart from the weighted averaging by type frequency in the centroid updates) until the set of assignments, $A_{1,\infty}$, converge.

Iteration 2

At this point, the first iteration has ended, and the second and final iteration begins. We use the new assignments of the fully re-clustered types, $A_{1,\infty}$, to make new descriptor vectors which are accumulated by counting how often each type is neighbored by any type from each cluster. Conceptually, we consider the assignments made in iteration one as an initial guess at a fine grained syntactic categorization. In the second iteration, we use this initial guess to build new left and right descriptors, which measure context with respect to this categorization, and we re-cluster these new descriptors into POS categories.

These newly created left and right descriptor vectors now have 500 components that correspond to the $k = 500$ clusters. We again use SVD for dimensionality reduction this time from $\rho = 500$ dimensions down to $r = 300$. This reduction represents a much less drastic change in the data than the dimensionality reduction of the first iteration (from 1000 to 100 dimensions) because the new descriptor vectors are "better informed" having been built from inferred categories rather than the original 1000 MFW counts. The new vectors, D_2, are less noisy, so more dimensions of latent features can be extracted.

These updated descriptors are again normalized to unit length and concatenated, and the algorithm is ready to update the cluster assignments. Because this is the final

iteration, we must jump from the previous 500 clusters down to the desired number of output labels for our tagger. We use either 50 (for PTB45) or 17 for (PTB17), depending on the tagset, to allow direct comparison with the previous HMM results; however, there is no particular constraint (either philosophical or numerical) to dictate this choice. Later, we discuss reasons why a much larger choice may be advantageous in a semi-supervised process we call prototype tagging. Just as in the first iteration, we use the new descriptors of the *k* most frequent words to initialize the centroids, and, just as in the first iteration, we allow the weighted version of *k*-means to iterate until convergence.

SVD2 results

In table 3.1 we compare the performance of SVD2 with the performance of the best previously published taggers, which use various versions of HMM's. We include one-to-one and VI numbers in this table for completeness, but as we discuss in the previous chapter, we reject these criteria as unreliable. (However, these measures agree with MTO that SVD2 is quite successful.)

	MTO		**OTO**		**VI**	
Model	**PTB17**	**PTB45**	**PTB17**	**PTB45**	**PTB17**	**PTB45**
SVD2	**0.731**	**0.665**	0.515	0.467	**2.36**	**3.80**
HMM-EM	0.647	0.621	0.431	0.405	3.86	4.48
HMM-VB	0.637	0.605	0.514	0.461	3.44	4.28
HMM-GS	0.674	0.660	0.466	**0.499**	3.46	4.04
HMM-Sparse(32)	0.702	0.654	0.495	0.445		
VEM (10^{-1},10^{-1})	0.682	0.546	**0.528**	0.460		

Table 3.1. Tagging accuracy under the best M-to-1 map, the greedy 1-to-1 map, and VI, for the full PTB45 tagset, and the reduced PTB17 tagset. HMM-EM, HMM-VB and HMM-GS show the best results from Gao and Johnson (2008); HMM-Sparse(32) and VEM (10-1,10-1) show the best results from Graça et al. (2009) which ignores VI.

The SVD2 algorithm is significantly better, according to MTO, than any of the

HMM's on PTB17, showing a jump in accuracy that ranges from 3% above Sparse(32) up to 10% above HMM-VB. Recall from the previous chapter that the MFW baseline for PTB17 is 62%. Our result represents a 37% relative increase over the baseline compared to Sparse(32), which is a substantial increase. Although it also performs better on PTB45 than the HMM's, the improvement over the best of the Gibbs sampler HMM's is a non-significant ½ %. We demonstrate with ISVD that SVD2 is too simple an implementation to fully capture the regularities on the PTB45 tagset, which is why its performance only matches the best of the HMM methods.

Interestingly, the MTO performance on PTB45 is nearly 12% higher than the VEM method, even though VEM has nearly as good an OTO score – yet another indication that these criteria are not compatible. The VEM method also produces a slightly higher OTO score on PTB17. Lastly, the VI score of SVD2 is dramatically better (lower) than the HMM's of Gao and Johnson. Graca et al. did not include this figure in their publication, and because we do not believe the metric is particularly well suited for this problem, we did not attempt to discover its value for inclusion in the table. As drastic as it is, we do not take the VI improvement too seriously, however, as even our VI score is still well above the VI of the MFW baseline.

Several figures in the previous chapter demonstrating the MTO performance of the tagging algorithms were constructed using the SVD2 method. We do not include them again here nor do we extend a discussion of these results beyond our discussion of table 3.1 because, as we will see in the following section, the more general ISVD algorithm is much superior to SVD2 (although somewhat more complicated).

Implementation 2: The Fully Iterative Tagger (ISVD)

Although we see that our general algorithm is quite successful for the PTB17 tagset when only two iterations are used, it fails to produce similar improvement over the HMM's for PTB45. We describe here a fully iterative implementation, ISVD, that is more in the spirit of the general framework of the algorithm as first described. This more involved implementation can be adjusted for either tagset, but the description included here is more appropriate for PTB45. Details for PTB17 are included briefly along the way but results are not significantly improved over the SVD2 implementation for this reduced tagset, because the simpler implementation is able to capture the PTB17 regularities quite well.

Recall that SVD2 simply builds descriptors, completely clusters them until convergence, then builds and clusters a second time. ISVD, by contrast, only gradually changes the cluster assignments from iteration to iteration, using only a single *k*-means step each iteration. It also gradually extracts more latent features, by slowly increasing the number of dimensions preserved by SVD. This more gradual process allows the descriptors and assignments to organize together into a globally more coherent arrangement, which scores much better than any of the the HMM approaches.

Word type inclusion

As with the two step version, ISVD is initialized by assigning each of the 1000 MFW's, each to its own cluster. The number of word types gradually increases each of the first few iterations until all types are included. We determine which types to include

each iteration based on how well populated their descriptor vectors are. This strategy prevents early inclusion of a type that has limited contextual information. For example, a word type with only 10 total tokens in the corpus cannot have a right or left context descriptor vector with a combined total of more than 10 entries. However, the actual number is potentially less than 10 because one or more of its tokens may have a right (or left) neighbor that has not yet been assigned to a cluster. Since the vectors count neighboring clusters, this un-clustered word makes no contribution. In the first iteration for example, it may be the case that all 10 tokens have right neighbors not among the 1000 MFW's that are initially assigned. In this case, even though the word has 10 tokens, it has an empty right context and thus is not a viable candidate for inclusion this iteration. However, some or all of its 10 right neighbors may have a well populated context in the first iteration so that they are included in this iteration. As a result of inclusion for clustering these types in the first iteration, the original word type which had no right context in the first iteration would then have a non-empty right context in the *second* iteration and might then be included for assignment. We use a "threshold schedule" defining how populated the context must be in any iteration for a type to be included. By gradually decreasing this threshold, progressively more words are included. We use an aggressive schedule that results in the inclusion of all types within the first six iterations. Typical threshold schedules are included in the performance figures included later in the chapter.

A typical iteration

We illustrate a typical iteration in figure 3.1 and describe each step in this section,

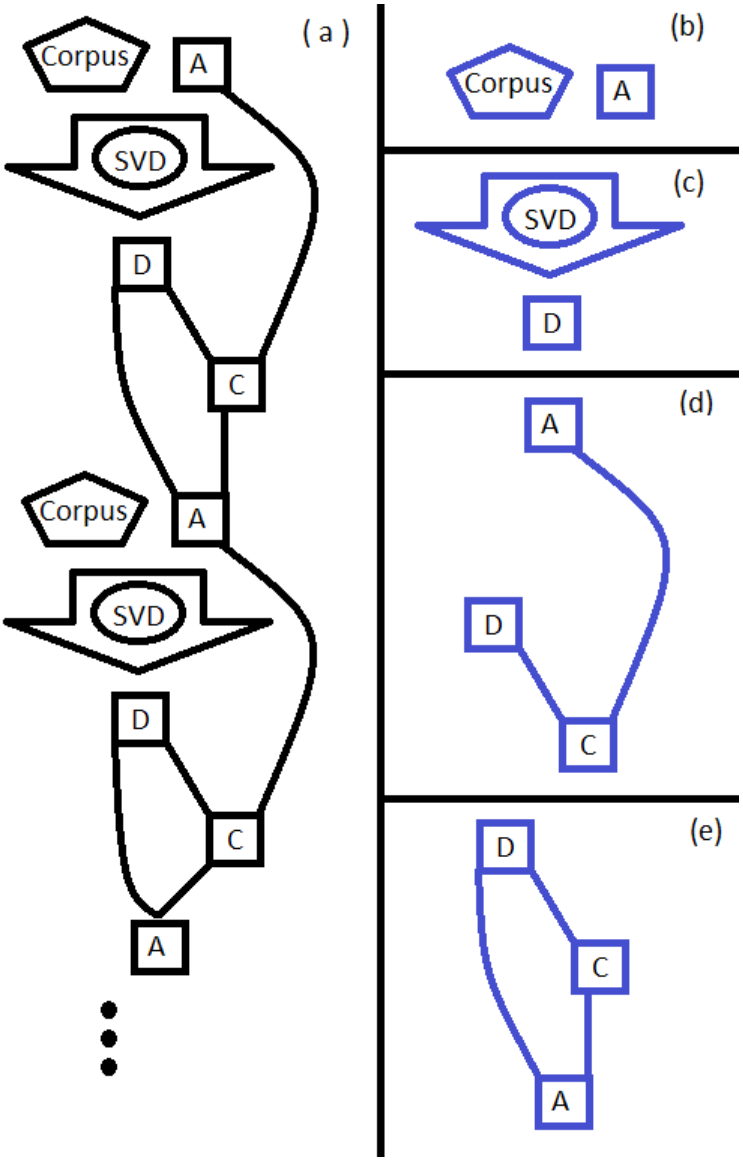

Figure 3.1: This figure illustrates the ISVD algorithm. A is the set of cluster assignments. D is the collection of descriptor vectors, C is the set of cluster centroids, and SVD indicates the dimensionality reduction. (a) *Shows the entire iterative algorithm.* (b) *Indicates that we use the current set of assignments along with the corpus to get a new context.* (c) *Here, the context is forced into a lower dimension using SVD to build updated descriptors.* (d) *The new descriptors are averaged according to the previous assignments to update the centroids.* (e) *The nearest centroid to each descriptor defines the new assignment.*

focusing on the PTB45 tagset. The first step of the iteration – panel (b) of the figure – is to accumulate descriptor vectors from the corpus for all word types based on the previous set of assignments, although, as we have just seen, underpopulated vectors may be ignored. Accordingly, we next determine which types have vectors sufficiently populated to be included in the iteration checking, their population against the current value of the threshold. The number of included words may increase from the previous iteration but cannot decrease.

These vectors of counts are then forced into a lower dimension using SVD, which is shown in panel (c), and we must specify the number of dimensions to use. We choose a general strategy of coarse to fine so that the number of dimensions gradually increases from iteration to iteration. In the first iteration, the initialized 1000 dimensional vectors are forced into 50 dimensions. From this point on the left and right half context vectors always maintain 50 components because we maintain 50 clusters throughout the process. In each successive iteration, the SVD dimension is one more than the last, starting with 10 and ending with 33. This schedule is also included in the performance figures described later. We attempt to force the vectors into a very low dimensional space first to extract the most salient latent features early in the process. Then, the progressively less important features are gradually incorporated as the process is refined in later iterations.

The final step is cluster re-assignment. In SVD2, at this point in the iteration, *k*-means was run until convergence. In ISVD, however, only one *k*-means-type step is used, so that the assignments evolve gradually as the vectors become more well-informed. The lone *k*-means step proceeds as follows. Based on the assignments of the

previous iteration and the new descriptor vectors produced from the SVD, new centroids are computed by averaging the updated vectors, as seen in panel (d). With the new centroids, each word type included in this iteration is assigned to the cluster with the nearest centroid, as determined by angle and depicted in panel (e). This re-assignment completes the *k*-means step and also completes the iteration of ISVD. As we indicated in the discussion of SVD dimension, over twenty total iterations are used to allow for more gradual organization of the assignments and vectors.

Computation

Although achieving superior accuracy is the paramount goal, it is important to note the huge computational savings achieved by ISVD over the HMM's. While ISVD runs on a desktop PC in under a minute, the Gibbs sampler for the HMM can take as long as as day. Furthermore, ISVD (or SVD2) can be used to produce an arbitrarily fine-grained labeling of the data, with a minor increase in computational cost. An HMM would experience a more significant increase in computational cost due to the increased number of hidden states. (As we discuss in detail later, more hidden states in the HMM would likely also hurt accuracy, because even with only 50 states there are considerably more emission-probability parameters than tokens in the entire corpus. Adding more parameters to this over-parameterized model is not advantageous.)

ISVD results

In this section we describe the results of our ISVD experiments in detail, but we begin simply with the best performance we have achieved. Using the full ISVD, we can, of course, at least duplicate the performance of the special-case SVD2 algorithm, and not

surprisingly, the performance is improved. In fact, the performance under the PTB45 tagset is significantly increased from 66.5% using SVD2 (which essentially equaled the HMM-GS performance) to 71.5%. This increase is so substantial that ISVD's PTB45 score is actually higher than the PTB17 score of any of the HMM based approaches; this is impressive, considering that the reduced tagset represents an easier problem. ISVD implementation does not dramatically improve the PTB17 score over SVD2, boosting the score from the previous 73.1% up to 74.5%; however, the SVD2 score already represented a substantial increase over the HMM approaches.

Figure 3.2 shows the PTB45 score as a function of iteration along with the fraction of word tokens included and the SVD dimension used each iteration. We point out that the algorithm is initialized using only the 1000 most frequent types, but these types account for about three quarters of the tokens in the corpus. In the next iteration, about 6000 types are included, which amounts to over 90% of the corpus. The number of included types then grows quickly, so that every type is included after just six iterations.

Each half of the initial descriptor vector is 1000-dimensional, and in the first iteration SVD is used to reduce the dimensionality to 50. At this point, the number of clusters is also fifty so that during the second iteration, the descriptors only have 50 dimensions. We then adopt the coarse to fine strategy for dimensionality reduction, so that data is initially forced into 10 dimensions and then gradually more in each iteration, until we reach 33, at which point the process has converged.

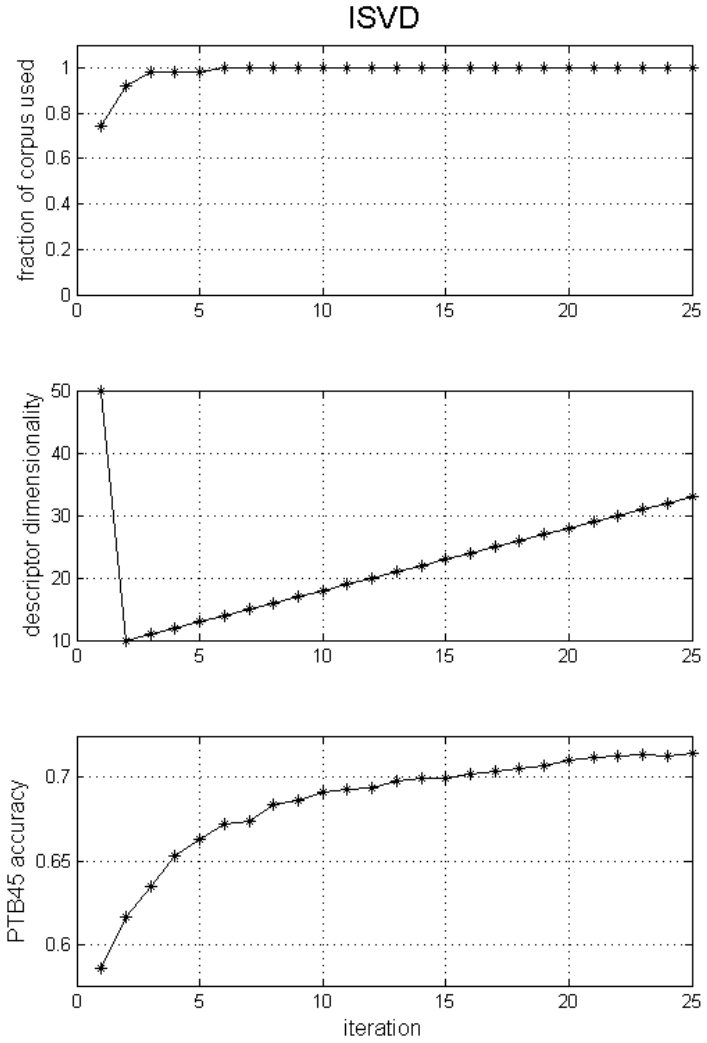

Figure 3.2: The bottom plot shows the MTO accuracy of PTB45 each iteration of the ISVD algorithm. We see a steady but gradual increase with asymptote at just under 72%. The top plot shows the fraction of tokens the algorithm includes each iteration. We see that this fraction increases quickly, reaching complete inclusion by iteration six. The middle plot indicates the SVD dimension schedule. After the initialization step, when we use 50 dimensions, the dimensionality gradually increases from 10 to 33.

In figure 3.3, we see a similar set of plots for the ISVD algorithm applied to the PTB17 tagset. After only six iterations, the algorithm has achieved its maximum performance of slightly less than 75%. The threshold schedule for including rare types is identical to the PTB45 process, so that after these six iterations every type is included. The important difference for PTB17 is that because we only have 17 clusters rather than the 50 for PTB45, the descriptor vectors now have about 1/3 as many components. These descriptor vectors start in a lower dimensional space, so the choice of SVD dimensionality reduction must change. We take the initial 1000 dimensional vectors immediately down to 20 dimensions. Then, with the vectors clustered into 17 groups, we again use the coarse to fine strategy, gradually increasing from 8 dimensions to 12 in the sixth iteration. Further iterations do not significantly affect the score.

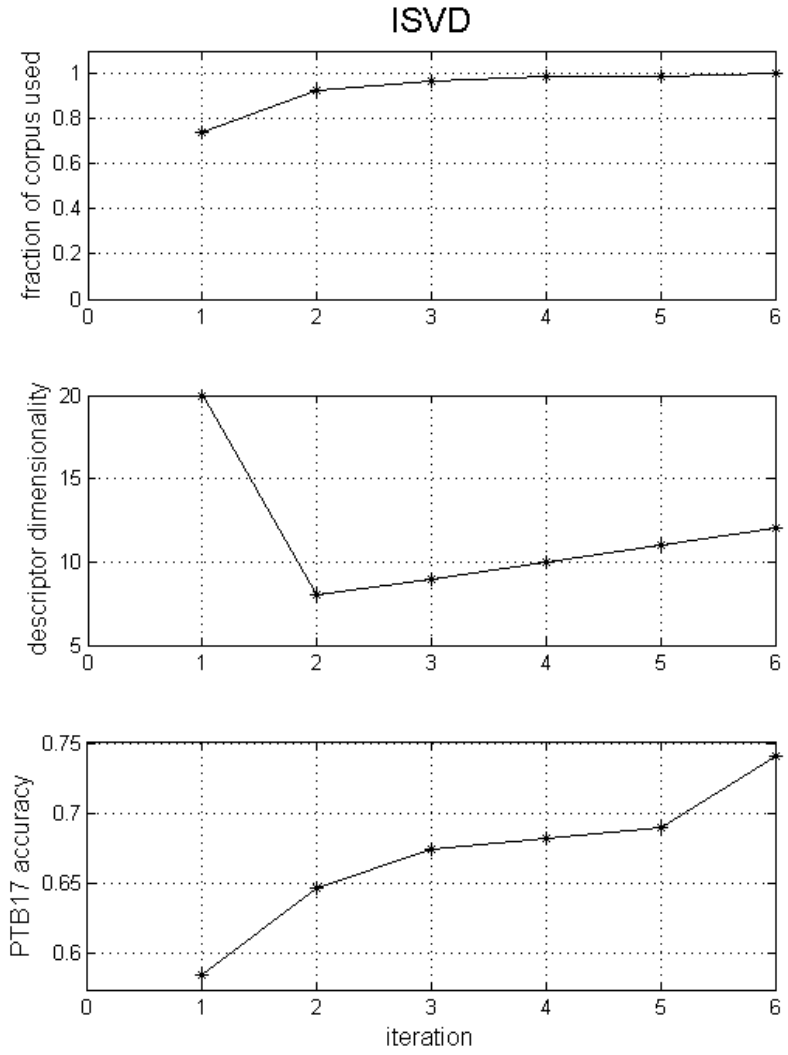

Figure 3.3: We show the performance of ISVD on the PTB17 tagset. Only 6 iterations are needed for the algorithm to achieve its maximum performance of just under 75% (bottom plot). With more iterations, the score remains basically fixed. The fraction of included tokens (top) is identical to the PTB45 algorithm. The SVD dimension (middle) goes from an initialization of 20 down to 8 then gradually increases to 12. (Since only 17 clusters are used, the maximum possible dimension after initialization is just 17.)

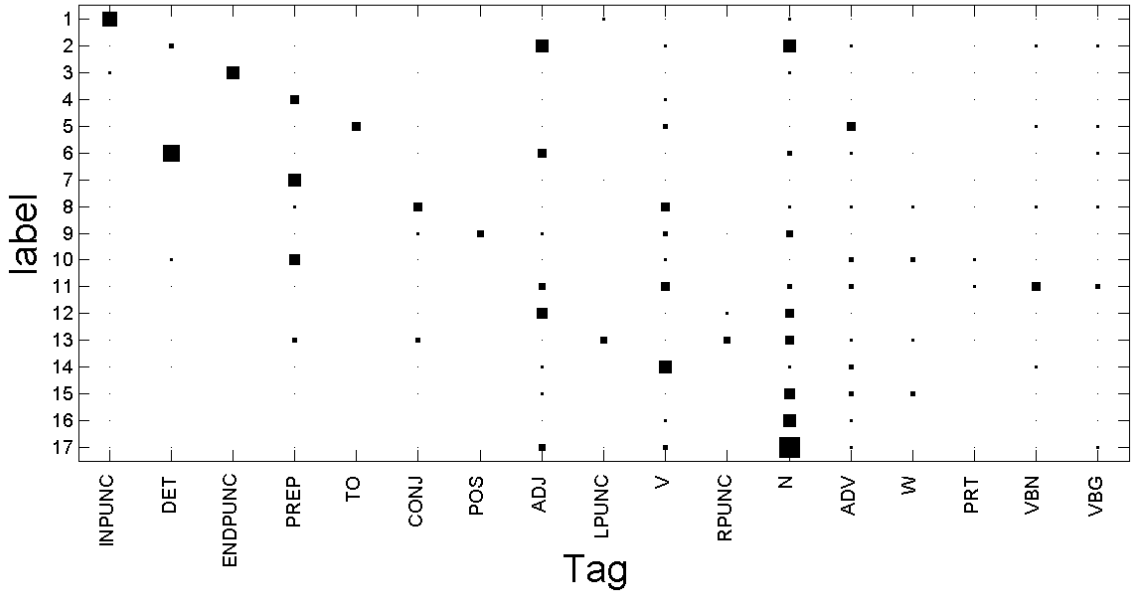

Figure 3.4: This confusion matrix of the ISVD algorithm on the PTB17 tagset demonstrates how the algorithm labels and mislabels the word types. Each square corresponds to tokens of a particular tag (column) that ISVD assigns to a particular label (row). The size of the square indicates the relative quantity of such tokens.

The confusion matrix in figure 3.4 is a helpful way to examine the performance of the algorithm without the application of any kind of mapping which is required to produce an accuracy score. To achieve a MTO score of 100%, a confusion matrix would need at most one square in every row, indicating that each label contains tokens of only one tag. To achieve an OTO score of 100%, the matrix would need at most one square in every row and column.

We include only the confusion matrix for PTB17 because the corresponding figure for PTB45 is, ironically, too confusing, due to the large number of labels and tags. Fortunately, there are no important qualitative differences between the two figures. What stands out in figure 3.4 is the difficulty in tagging open-class words (N, ADJ, ADV, VBN,

and VNG). Such difficulty is expected because most open-class words are rarely seen so we have limited contextual information on which to base our assignment. Furthermore, the semantic themes play a dominant role in governing many open-class words. An unsupervised tagger can only hope to extract the most salient statistical regularities, and it is likely that some of these regularities are semantic, leading to a labeling that is not necessarily based on POS.

The confusion between N and ADJ is particularly large. Several labels include many tokens of each. Part of the reason for this overlap is undoubtedly explained by the tendency to string together consecutive nouns or consecutive adjectives as in the phrase *the big tall friendly school bus driver*. This common feature of English tends to make the context of nouns and adjectives surprisingly similar, leading to confusion in the tagging. Such structure causes a similar difficulty in an HMM approach, because adjectives are likely to transition to both adjectives and nouns. The correct tagging sequence for the previous phrase is (DET, ADJ, ADJ, ADJ, N, N, N), but if any of these words were rarely seen in the corpus a tagging like (DET, ADJ, ADJ, ADJ, ADJ, ADJ, N) would be likely as well, because the model only considers immediate neighbors and does not realize it has tagged five consecutive adjectives.

The similar labeling of VBG and VBN (present and past participles) indicates that ISVD has not learned to distinguish these tags from each other. The somewhat uniform presence of tokens of these tags in nearly every label indicates that ISVD has not learned to distinguish well these tags from anything else either. As a result, under the MTO map, no label is ever mapped to either tag. Not surprisingly though, these two POS tags are

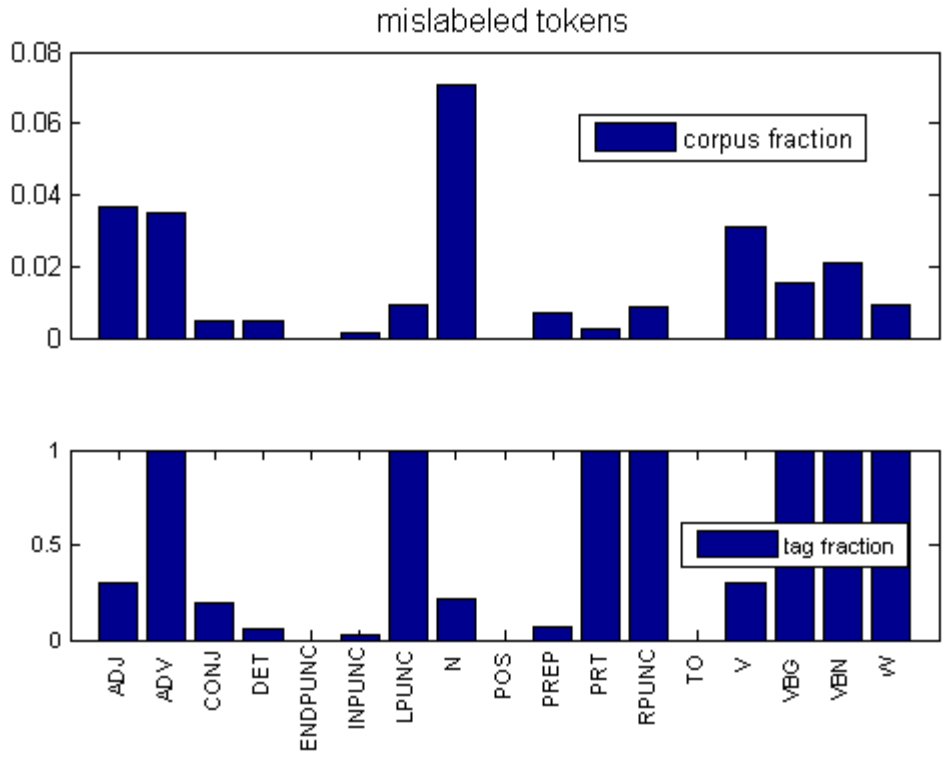

Figure 3.5: The top graph shows, for each tag, the fraction of all tokens in the corpus that are mislabeled by ISVD for the PTB17 tagset. The bottom graph shows the number of mislabeled tokens relative to each tag; in other words, for any tag, it shows the number of mislabeled tokens divided by the total number of tokens with that tag. (The top graph divides instead by the total size of the corpus.)

relatively infrequent, so completely missing them has only a minor impact on accuracy. Figure 3.5 illustrates this analysis nicely.

The bar graphs of figure 3.5 (PTB17) and 3.6 (PTB45) help demonstrate which tags are well accounted for by ISVD and which are not. The upper graph in each figure shows the number of tokens of each tag that are incorrectly labeled by ISVD and divides this number by the total number of tokens in the corpus. The lower graph divides this

number instead by the total number of tokens with that tag, giving a relative fraction of mismatched tokens. From the upper graph for PTB17, we immediately see that mislabeled nouns are the dominant source of inaccuracy, followed by adjectives, adverbs, and verbs. Again, we see that open-class words are the most frequently mislabeled words.

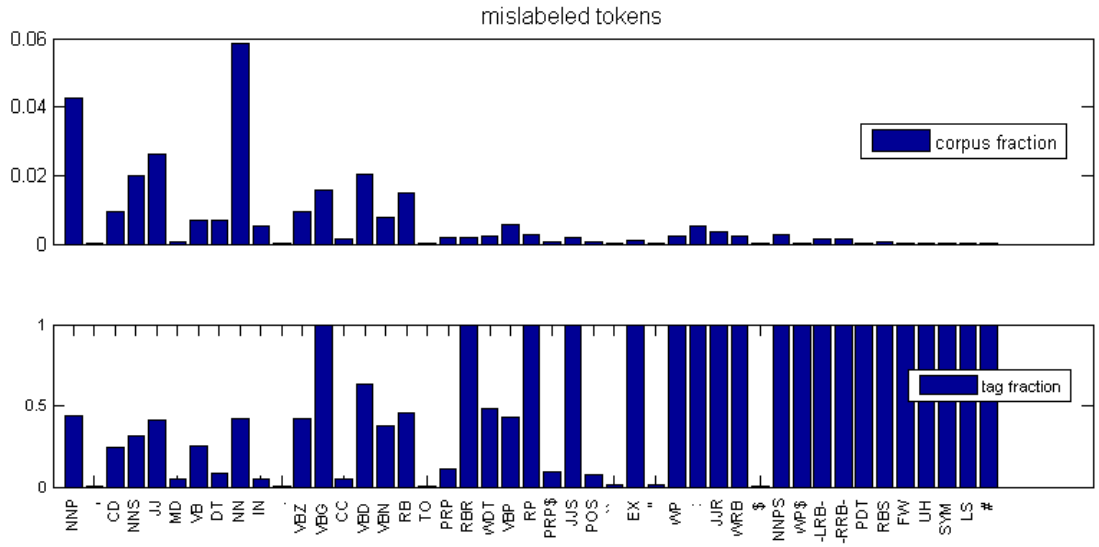

Figure 3.6: A mislabeling plot for PTB45. The upper and lower graphs are analogous to those in figure 3.5.

The bottom graph tells a much different story. We see that we actually mislabel only 20% of the nouns while we mislabel all of the adverbs (because no labels are mapped to the ADV tag). We mislabel about 40% of the adjectives and verbs as well. While the top graph indicates that nouns are the biggest source of inaccuracy, the bottom plot indicates we tag nouns relatively well compared to other open-class words. This figure also indicates that three closed-class tags – end punctuation, the possessive markers, and to – are perfectly accounted for with no mislabeled tokens. Other less common closed-classes tags, including left and right punctuation, WH words, and

particles are completely ignored, so that all such tokens are mislabeled, yet each accounts for no more than 1% of the corpus. The remaining more common closed-class tags – conjunctions, prepositions, determiners, and interior punctuation – are very rarely mislabeled, so that they play a minor role in the total inaccuracy.

The story is similar in figure 3.6 for PTB45. Again, we see a very large number of infrequent tags that are completely unaccounted for, so that they have 100% relative mislabeling, but each plays a very minor role in the overall inaccuracy of the algorithm. There are several closed-class tags that are extremely well accounted for, so that they have very small relative and absolute mismatch scores. The largest absolute errors again come from the NN and NNP tags (common and proper singular nouns) but in PTB45, they also have nearly identical relative mislabeling as does JJ (base adjectives). This increase in the relative mislabeling is largely due to the fact that ISVD does not distinguish well between these two types of singular nouns so that some common nouns are labeled as proper nouns and vice-versa. Such "mistakes" are not surprising, considering the similarity in context between these two tags.

In the final mislabeling plot, figure 3.7, we illustrate how some POS tags can be captured with additional labels while these tags are ignored when an insufficient number of labels is used. As we have seen in the previous bar graphs, some tags of both open and closed class are completely ignored (100% relative mislabeling) by our model on the PTB17 tagset when k = 17 labels are used. Right and left punctuation are two such closed class labels while present and past participles are examples of ignored open class labels. When the number of labels used increases from k = 17 to k = 50 (still using

PTB17 tagset) we see very different behavior in these four examples. The closed class tags are well accounted for by this finer grained labeling; hence, the relative mislabeled fraction drops from 100% to below 20% for both kinds of punctuation. Past participles are somewhat accounted for which drops the VBN fraction from 100% to about 60%. VBG, on the other hand, remains completely ignored.

Figure 3.7: This mislabeling figure is for ISVD on PTB17. It illustrates how the mislabeled words by tag change as the number of labels increases: k = 17, 50, 1000, 43766. The largest value assigns an individual label to each type with no clustering.

We also see on this figure the problem with a non-disambiguating model. The bar corresponding to k = 43766 gives a different label to every word type. The 10 tags that have a non-zero bar for this value of k all have word types with some tokens that

continue to be mislabeled due to ambiguity. We see from the lower set of bars that 40% of the particles (PRT) are impossible for our model to capture as are roughly one quarter of the so called "WH words" (W). Fortunately, the absolute contribution of this ambiguity is small as seen in the upper graph of the figure.

When an extremely large value of k is used (k = 1000), we see that the closed class tags are captured as well as ambiguity will permit, but even with this large number, some open class words still have a gap between their high k score and the ambiguity limit. These word types are likely to have descriptor vectors that are simply unlike any other group of words of the same POS tag. This kind of mistake can easily happen if a type's distribution is somewhat sparse and its usage simply makes it look to the model like the wrong POS. Any mistakes in the annotation would be in this same category.

Robustness

Although the particular parameters have been fine tuned to optimize performance, the algorithm is reasonably robust to most design choices. Small deviations from the parameters described in this section decrease performance, but typically, only by a few percent. There is one notable exception. When running the PTB45 experiment, it appears to be important to reduce the dimensionality down to at least as few as 10 dimensions before allowing it to increase. Decreasing this parameter below 10 causes a small decrease in performance with scores typically around 70%. However, increasing it to 11 or more causes a larger drop so that scores are only about 65%. There is some indication that injecting a small amount of noise into the process can make the algorithm less sensitive to this particular parameter, but this increased robustness comes at the cost

of a small decrease in accuracy. The exact reasons behind this particular sensitivity were not explored.

Prototype Supervised Tagging

One reason why unsupervised treatment of the POS tagging problem is important is that many languages lack the annotated data required to train a supervised algorithm. We demonstrate that both ISVD and SVD2 can be used along with a post-processing step that incorporates weak supervision to yield very high POS tagging performance.

Our algorithm produces a set of cluster assignments (i.e. labels) for each word type. We then use the unconstrained MTO criteria to find the best tag/label correspondence to compute an accuracy for evaluation purposes. We can amend this process slightly to incorporate what we call prototype supervision. Until now, we have assumed a choice of k labels that is very nearly the same as the number of POS tags in the tagset. This choice is important to allow fair comparison between different taggers since the MTO score depends heavily on the number of labels used. An MTO comparison is simply inappropriate if two taggers differ in the number of labels they produce. (This property of MTO is not shared by either OTO or VI which can be fairly used regardless of the number of labels. Unfortunately, as we have seen, these criteria have other more serious drawbacks.) Since we require comparison to the unsupervised HMM's, it is essential that we choose the same number of labels used by those models.

Under prototype supervision, we instead use a large number of labels (e.g. 500) rather than 17 or 50. From each of the 500 labels, we select the most frequent word

which we call the label's *prototype*. We propose then querying an expert in the language for a POS tag of each prototype. The expert's chosen tag is then assigned to all tokens of all types that share the prototype's label. This limited supervision is confined to post-processing, and, if annotation is available, it allows direct evaluation of the accuracy of the tagger, since each type now has an inferred tag from the tagset instead of just a generic label.

To investigate this process on the WSJ corpus, we replace the query of the expert with a query of the annotated tags themselves, with the expectation that an expert's choice would consistently match this result. In other words, for each prototype word, we find its most frequent annotated tag in the corpus and use this in place of the expert's assigned tag. In a sense, our experts are the linguists from the University of Pennsylvania who produced the annotation. Of course, when unsupervised methods such as ours are truly needed, the expert query is the only viable option because no such annotated data would exist, but, here, we use the annotation query for proof of concept. The prototype query of the annotation differs from the optimal MTO map because, for any label, we use the tag that best fits only the label's most frequent word rather than the tag that is the overall best fit for all the label's words.

Figure 3.8 shows the performance of ISVD incorporating prototype supervision as the number of labels (and therefore prototypes) increases. We provide, for comparison, the accuracy score when the optimal MTO map is used, which incorporates information about all of the 43,000 types in contrast to the prototype map which uses information

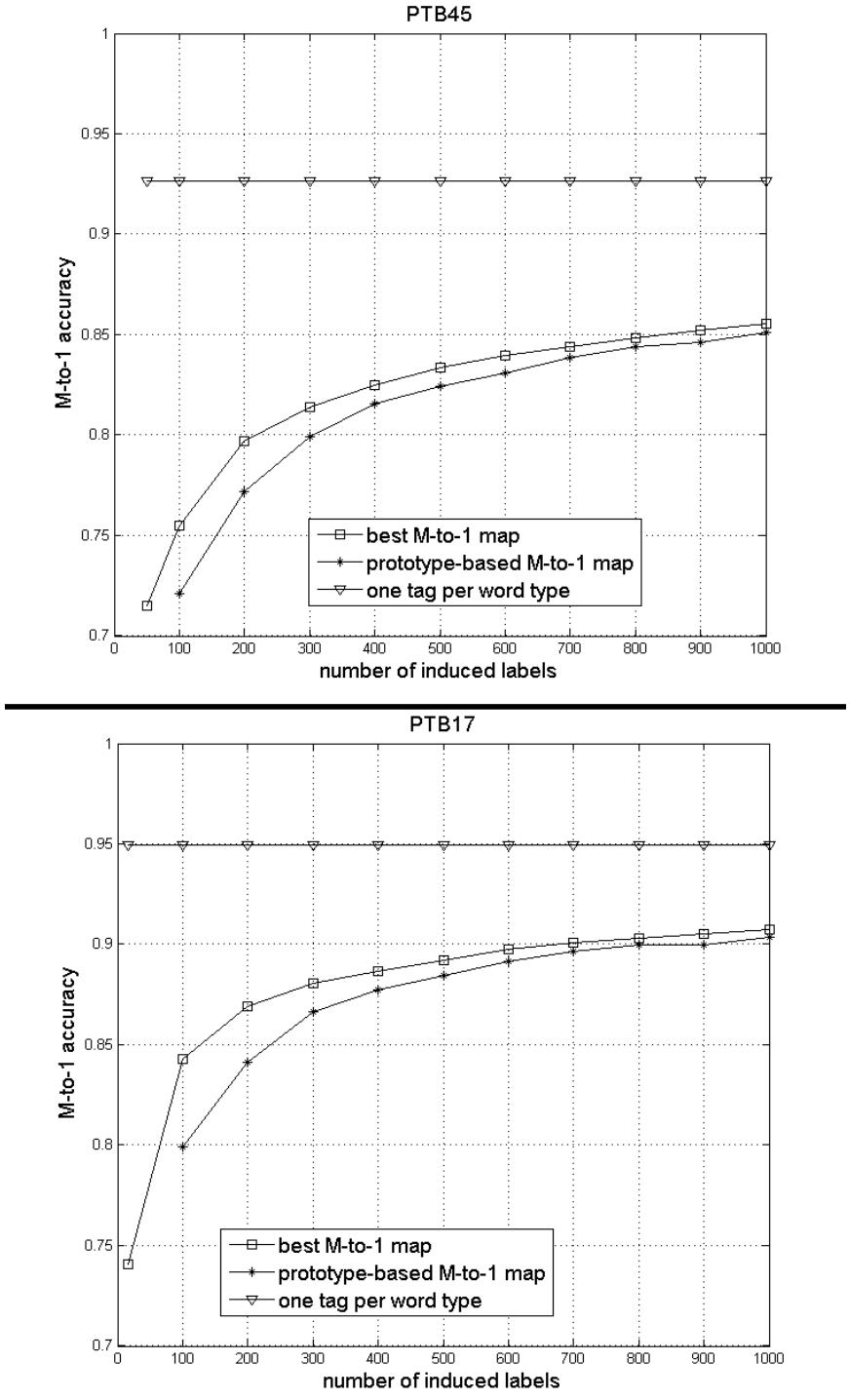

Figure 3.8: Comparison of the prototype map and optimal M-to-1 map for ISVD on PTB45 (top) and SVD2 on PTB17 (bottom). The one tag per word benchmark is the best possible accuracy for any non-disambiguating tagger.

only about the prototypes. We also show the theoretical upper limit for a model that does not attempt to disambiguate. This limit is calculated by assigning all tokens of each type the most often used tag for this type. For each type then, the largest possible number of tokens agree with the annotated tagging. Figure 3.8 demonstrates that very high performance can be achieved with minimal supervision. The time and expense required for an expert to pick the best tag for a list of a few hundred types pales in comparison to the effort required to annotate an entire corpus or even to build a lexicon containing all possible tags for all types. We see that when more than roughly 300 labels are used, the gap in accuracy between the optimal MTO map and the prototype map has significantly narrowed. With 300 labels, we see 80% accuracy on PTB45 and about 87% on PTB17. As unannotated data becomes increasingly available in the information age, a prototype-based approach is likely to become even more useful, and our algorithm is well suited for this task because the computational burden does not significantly increase as the number of labels increase. The HMMs certainly do not share this property, making prototype tagging quite difficult.

Discussion

SVD2 vs. ISVD

The primary benefit of the ISVD algorithm over SVD2 is that it allows the clustering to evolve simultaneously with the descriptor vectors over many iterations. Recall that SVD2 commits to a clustering (by allowing *k*-means to converge), builds descriptors based on that clustering, manipulates the vectors using SVD, and finally

commits to an entirely new clustering. ISVD, on the other hand, only slightly changes the current state of the clustering assignments each iteration. Comparing SVD2 to ISVD for PTB45 demonstrates that this simultaneous discovery of descriptors and vectors can be very beneficial.

There is, however, one drawback of ISVD when compared to SVD2: the potential for increased computational cost of ISVD stemming from the repeated use of the SVD calculation. For small numbers of clusters, ISVD actually outperforms SVD2 because of the way rare words are incorporated, but in prototype tagging, with a very large number of clusters, it is possible to choose parameters that cause ISVD to be more time consuming. Even with this potential burden, the savings when compared to the cost of the HMM algorithms is dramatic.

Disambiguation

The more interesting comparison is between the descriptor vector approach of our algorithm compared to the generative approach of the HMM's. Most notable of the many differences in the approaches is that we do not attempt to disambiguate tokens of the same type. While an HMM is designed to pick a tag for every token, our approach is limited to clustering descriptor vectors, and, as described, we only allow one vector per word type. Every token of a single type must therefore be given the same tag. This limitation seems, at first, to be very severe. In the WSJ corpus under PTB45, roughly half the types have tokens of more than one tag. These differences cause approximately 8% of the tokens to have tags that do not match their types' most frequent tag. As a result, our non-disambiguating approach cannot possibly achieve better than 92%

accuracy while the HMM could, in principle, score 100%.

Given this limitation, one might wonder why our approach achieves significantly higher accuracy than the various HMM approaches. Part of the explanation lies in the possibility that the HMM's may not be taking advantage of their ability to disambiguate. Using the data provided by Mark Johnson for the HMM-VB model, we explored this possibility, by crippling the HMM's ability to disambiguate. For every type in the corpus, we found the most frequent tag assigned by the HMM, which we call the type's *modal tag*.. We then created a new labeling that forced every token given a tag by the HMM other than its type's modal tag to accept the modal tag instead. In this alternative tagging, every token of a particular type has the same tag, just like in ISVD and SVD2. One might expect that, by limiting the HMM model in this way, the accuracy would suffer; however, performance actually increased for both PTB17 and PTB45 on each of the two dozen data sets we were given. Scores consistently increased by an average of 1.5% on PTB45 and 2.6% on PTB17. While we were not able to test the several other HMM approaches described in this chapter, no one has demonstrated that any HMM (or any other unsupervised model) disambiguates successfully. Assuming the behavior of HMM-VB is not unique, our algorithm's inherent inability to disambiguate may not, in practice, differentiate it from any other approach.

Over-parameterized vs. over-constrained

Upon more careful examination of the HMM approach, it is not so surprising that this very powerful approach has limited success. The HMM is primarily parameterized by emission probabilities, $P(X = x_i \mid Y = y_j)$, which represent the conditional probability

of a word type given a hidden POS state. By fixing the POS state, we define a collection of probability vectors each of which has one component for every word type in the corpus. If we store these vectors, column-wise, into an emission probability matrix, it is useful to consider each row of that resulting matrix, which gives probabilities involving a fixed word type. (The row vectors need not sum to one, and the rows are not themselves distributions.) An important structural property of natural language is that this collection of probabilities is extremely sparse. In other words, the vast majority of types are used as only one or two POS and only very rare types occur as more than three POS. In an HMM with 50 states, therefore, we expect at least 47 of the columns to be zero. Furthermore, as already noted, under PTB45 over 92% of the tokens in the WSJ corpus are used as their dominant POS, and of course this fraction increases under PTB17. So while at least 47 of the components should be zero, often one of the three non-zero components should dominate the other two. Discovering this extremely skewed distribution proves to be a daunting task for the HMM.

Contrast this difficulty with our approach in which only one POS state is possible for every word type. Clearly, our model is over-constrained, but even with this over-constraining, our model can, in principle, account for 92% of the data. This constraint leaves for our model only to discover which is the one tag for each word type, amounting to roughly fifty thousand unknowns. The HMM on the other hand has fifty parameters for each of the roughly fifty thousand types – a staggering total of 2.5 million unknowns – which must be learned from a data set that contains only half that number of tokens. In light of this vast over-parameterization, it is not so surprising that our model performs

quite well by comparison, despite its being mildly over-constrained.

Not surprisingly, the EM treatment of the HMM does fail to capture the desired sparse structure of these emission probabilities. A Bayesian formulation affords the use of a prior, and the HMM-VB and HMM-GS approaches do use a Dirichlet prior in an attempt to facilitate the learning of this sparse structure. This choice of prior attempts to produce a zero emission probability for many word types, given a fixed POS state. This approach only indirectly promotes the desired structure by attempting to populate columns of the aforementioned matrix with zeros rather rows. Furthermore, even the number of zeros in a column varies drastically depending on the tag. It is very difficult for such an approach to be successful on both open-class (noun, verb, adjective, adverb, etc.) and closed-class (punctuation, determiner, preposition, etc.) tags. For example, a closed-class POS such as the determiners can only have a handful of types, so only a handful of the nearly fifty thousand components of this column vector should be non-zero, while an open-class like noun must have many thousands non-zero types. Capturing these very different distributions with a single prior is difficult, and even with the open and closed classes correctly accounted for, this approach does not guarantee that each individual row of the emission probability matrix will be sparse.

The HMM's of Graca et at. (2009) attempt to overcome the difficulty in characterizing the sparse nature of the type/tag distribution by using a modification of the posterior regularization framework, which attempts to guide the variational EM algorithm toward a set of soft constraints on the posterior distribution. The HMM-Sparse line of Table 3.1 indicates that this approach is successful in improving the overall MTO

accuracy of the HMM, but it still falls well short of the performance of ISVD. This difference in performance indicates that while posterior regularization does help remedy the drastic over-parameterization of the HMM, it is still not enough to make the approach competitive with our over-constrained algorithm.

More than just a tag

It should be pointed out that the true output of our algorithm is not just a POS assignment but rather a high-dimensional vector whose geometric position describes its type's use more richly than just a POS description. These vectors are clustered into POS tags because we want to be able to apply this approach to a classic natural language processing problem, but the information encapsulated by the vectors is deeper. These vectors, for example, could immediately be used for sub-categorization or for hierarchical clustering. An HMM, by contrast, would need to be completely re-trained if we wanted to describe the corpus using more (or fewer) hidden states.

Future work

Although our approach is over-constrained and cannot disambiguate, we see it as only the first step to be followed by a targeted effort to disambiguate word types in a subsequent step. There are a number of possible approaches that could be used. One is to simply use the results of ISVD to initialize an HMM; this might help the HMM find a suitably sparse set of parameters to explain the data. A second idea is to identify candidate types that would benefit from disambiguation based on the position of their descriptor vectors (and possibly their frequencies). A descriptor vector that is well-explained as the combination of two or more cluster centroid vectors is likely to be an

ambiguous type. We propose giving such a type a second descriptor vector and then using the two vectors to try to disambiguate tokens of this type.

Chapter 4: Biologically Motivated Modeling

In this chapter, we describe a novel approach to modeling in natural language, which relies on mechanisms loosely related to biological processes. We describe a dynamic process that manipulates descriptor vectors similar to the descriptors of the algorithm of the previous chapter, but we relate the vectors themselves as well as their dynamics to neural processes. This new approach is applied to the unsupervised POS tagging problem and then extended to a context-free-grammar learning algorithm. While traditional unsupervised inference in natural language processing focuses predominantly on an algorithm's performance, in this chapter we instead focus on exploring models whose behavior may well have more direct implications on human language acquisition and processing.

We begin with a description of just such a model for the unsupervised POS tagging problem. This model relies on two competing mechanisms to produce a dynamic process that leads to the self-organization of the collection of descriptor vectors of all word types. The geometry of the organized system lends itself to clustering, and this clustering is shown to correlate remarkably well with gold-standard POS tags, according to the usual MTO criterion. We relate the vectors themselves as well as the dynamic forces between them to the basic neural mechanisms of synaptic plasticity and inhibition.

This model can be easily extended to allow multiple descriptor vectors for each word type, thus allowing for ambiguity of types; however, the performance of the algorithm suffers under this extension and we briefly examine reasons why. Lastly, we

compare this approach to the Generalized Hebbian Algorithm (GHA) for SVD described in Gorrell (2006), by showing that its two chief forces actually approximate a matrix decomposition algorithm.

In addition to the application for POS tagging, we include an extension of these ideas to the problem of unsupervised grammar induction. While the ideas are not yet sufficiently mature to be applied to natural language data, we demonstrate the capability to learn a collection of simple context-free grammars. We discuss several limitations of this approach for unsupervised learning of grammars in natural language.

We argue that this approach to modeling is valuable for several reasons. First, experimental neuroscience is limited in its ability to explore complex phenomena like language processing, due to limitations in the nature of the data that can be collected. If the neural mechanisms required for processing language take place on very fine spatial and temporal scales and involve a large quantity of neurons acting together, there is a technological barrier that currently precludes the possibility of direct observation and study. Multi-electrode recoding allows collections of individual cells to be studied on very fine time scales, but can only isolate the activity of perhaps a few hundred neurons. fMRI allows the observation of activity in much larger regions, but cannot measure the activity of individual cells. Computational modeling, which does not suffer from the same limitations, may be able to aid in understanding how the brain processes language.

A second reason why this approach is important is that the state-of-the-art in unsupervised algorithms for many natural language processing problems significantly lags human levels of performance. While still a distant goal, we hope that by better

understanding and capturing the functionality of the key neural mechanisms, we may someday dramatically advance the state of the art in unsupervised inference. Although achieving superior performance is not the immediate goal of the work presented in this chapter, we show that our neurally-inspired model achieves excellent performance, and in fact outperforms all HMM-based approaches. We thus feel justified in formulating the hope that emphasizing neural mechanisms in model design may eventually help to push the performance of algorithms closer to human levels.

Hebbian Model for Unsupervised POS Tagging

We begin with an outline of the algorithm, which is immediately followed by a thorough description of each element. This outline demonstrates the relative simplicity of this model in comparison to the ISVD method of the previous chapter.

Hebbian POS Tagger
Initialize descriptor vectors i.i.d uniform
Loop sequentially over bigrams in the corpus
-Apply Hebbian attraction to vectors of types in the bigram
-Apply orthogonalization to these same vectors
Terminate the algorithm with *k*-means clustering of vectors
Evaluate with MTO mapping to POS gold standard

Descriptor Vectors

In the POS-tagging approach of the previous chapter, dynamic descriptor vectors are used to capture the context distributions of word types. Descriptor vectors play the same role in our so-called Hebbian model. The key difference is that, in the previous

approaches, the descriptor vectors stored contextual information, derived from the corpus in the form of bigram counts. In contrast, in the Hebbian model, the vectors serve as guideposts that direct other descriptor vectors into an organized state, obeying a sequential dynamics in response to the local interactions defined by the data in the corpus.

These high-dimensional descriptor vectors are an abstraction of complex spatiotemporal neural activity patterns in the brain. For example, suppose we focus on a collection of 10 specific neurons that, together, might be selective for a particular word type, and we count the number of action potentials in each neuron over the course of a relatively long window (e.g. one hundred milliseconds) for 10 consecutive windows. These ten counts over ten windows produce a 100-dimensional activity vector describing firing rates. Similarly, we might examine action potentials in a very large collection of neurons (e.g. five hundred cells) for 100 ms by counting the total number of spikes fired by all cells each millisecond, thus measuring synchronous activity. This activity vector has the same size (100), but captures a very different kind of spatiotemporal activity pattern. Rather than focusing on the precise biological interpretation of our descriptor vectors, we choose instead to leave a gap of abstraction, without committing to a single kind of activity pattern; however, bridging this gap by demonstrating how these ideas can be implemented in an actual network of neurons is an important direction for future work. As in our ISVD and SVD2 algorithms, we again use a left and right half vector for each word type.

Hebbian attraction

As with the SVD-based models of the previous chapter, the simplest implementation of the Hebbian algorithm uses a single descriptor vector (with left and right halves) for each type, although we later discuss the extension to multiple vectors. The learning algorithm is sequential, in an effort to more realistically approximate natural language acquisition. Data is thus presented to the algorithm one sentence, or even one bigram (i.e. a pair of consecutive words), at a time, and vectors are altered only when a token of their word type is encountered.

The central mechanism in this algorithm is an attraction term that we relate to Hebbian synaptic plasticity – the well studied principle that synaptic strength increases when a presynaptic neuron fires immediately prior to the firing of the postsynaptic neuron. Hebb's Law is often paraphrased as "neurons that fire together wire together." Because our descriptor vectors are an abstract representation of activity patterns in a collection of neurons, we implement this Hebbian principle as a simple attraction between vectors that "fire together," and this attraction is depicted in figure 4.1.

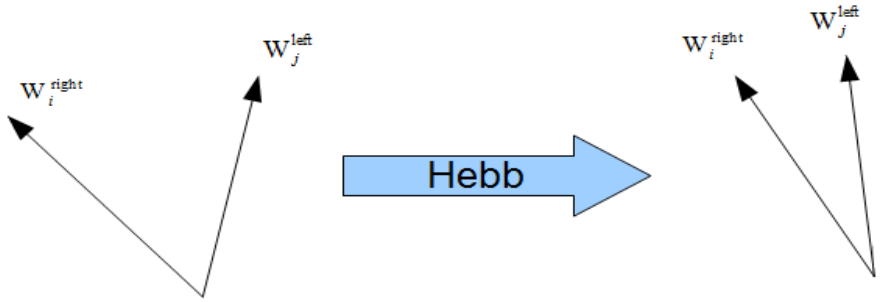

Figure 4.1: If a pair of words, (W_i, W_j), is seen together in the corpus, the Hebbian attraction moves the right half vector of word i toward the left half vector of word j and vice versa. Thus, a word's right half of the vector concerns its right context, and its left half vector concerns its left context.

If bigram, (W_i, W_j) appears in the corpus and is presented to the algorithm, the appropriate halves of their descriptor vectors for these two words "bind together" and are attracted. Specifically, the right half of the vector belonging to the first word, W_i, and the left half of the vector belonging to the second word, W_j, are mutually attracted:

$$W_i^{\text{right}} = \alpha * W_i^{\text{right}} + (1-\alpha) * W_j^{\text{left}}$$

$$W_j^{\text{left}} = \alpha * W_j^{\text{left}} + (1-\alpha) * W_i^{\text{right}}$$

Each half vector is renormalized to unit length after the "Hebbian" update. The idea behind this Hebbian attraction is simple. Each descriptor vector has a left and a right half whose value is governed by the left and right context of its associated word type. When a pair of words is seen together, the right vector of the first word becomes more similar to the left vector of the second word and vice versa. This behavior allows the indirect attraction of words with similar context over the course of many bigrams. Suppose the corpus somewhere contained each of the following three bigrams: "*black dog*", "*white dog*", and "*white cat*". When the first bigram is encountered, the right half of *black* moves toward the left half of *dog* and vice versa. When the next bigram is encountered later in the corpus, the right half of *white* also moves toward the left half of *dog*, thereby indirectly moving it toward the right half of *black*, causing *black* and *white* to resemble one another more closely. Finally, when the third bigram is encountered, the left half of *cat* moves toward the right half of *white*, which is an indirect move toward the left half of *dog*.

This attraction can be even more indirect, as exemplified by a corpus containing: "*old car*", "*fast car*", "*fast runner*", "*Olympic runner*". Through this sequence of

bigrams, *old* and *Olympic* are indirectly attracted, even though the corpus may contain no instances of the same word type following both *old* and *Olympic* (i.e. they have completely disjoint right contexts). The goal of this Hebbian attraction is to promote the self-organization of the descriptor vectors by bringing together words with similar contexts. However, just as in the SVD-based approaches, this context needs to capture latent features, and the indirect attractive mechanism described above accomplishes this goal by generalizing beyond the observed words. It causes certain directions in the high dimensional space to gradually become general attractors for words with similar generalized contexts.

Inhibitory orthogonalization

The obvious drawback of this Hebbian attraction is that, if left unchecked, all vectors would eventually converge to a single direction in space. This convergence happens because all word types are weakly attracted through long sequences of indirect interactions. Such convergence would be catastrophic, since a state in which all vectors are parallel carries no useful information. Such a state corresponds to neural activity in which every collection of neurons fires in exactly the same pattern.

We envision an inhibitory mechanism to control this tendency, and we implement this inhibitory mechanism in our model as an orthogonalization force. Each time a word type is encountered, its right and left half vectors are given a small orthogonalizing push. The direction of the push is determined by the mean of a small number of additional vectors, selected at random according to the type frequency of the words in the corpus. For example, suppose *dog* is one of the two word types in a bigram that is presented to

the algorithm. Each half of the vector for *dog,* which we denote *d* (deviating slightly from consistent notation by using no subscript to indicate the type), must be orthogonalized. We randomly sample a small number of types from the corpus in such a way that the probability of selecting a type is equal to its relative frequency in the corpus. With these randomly drawn word types in hand, we compute the mean of their vectors, μ.. The two halves of *d* are then pushed orthogonally to the two halves of μ. More precisely, a fraction of the component of *x* that is parallel to μ is subtracted, leaving the orthogonal component to define more dominantly the direction of the resulting vector: (angle brackets indicate a vector dot product)

$$d_{\text{left}}^{\text{new}} = d_{\text{left}}^{\text{old}} - \beta \left\langle d_{\text{left}}^{\text{old}}, \mu_{\text{left}} \right\rangle \mu_{\text{left}}$$

$$d_{\text{right}}^{\text{new}} = d_{\text{right}}^{\text{old}} - \beta \left\langle d_{\text{right}}^{\text{old}}, \mu_{\text{right}} \right\rangle \mu_{\text{right}}$$

After a portion of the parallel components are removed, the vectors are renormalized. The coefficient of orthogonalization, β, is also a parameter of the model, and must be selected with some care to ensure a stable balance between the Hebbian attraction and orthogonalization. If β is too small, the algorithm "over-generalizes," meaning that too many vectors converge to the same point in space. If β is too large, the algorithm never organizes, and all vectors continue to drift around the unit sphere without organizing into clusters.

Type frequency adjustment

The algorithm's performance can be improved by multiplying both learning parameters, α and β, by a frequency adjustment factor, ζ, each time attraction and orthogonalization

are applied.

$$\zeta_j = \frac{1}{1 + 0.1 * \#(j)}$$

#(j) counts the total number of times word type (W_j) has been previously encountered in the corpus. This adjustment allows a vector to change more rapidly early in "learning" when little is known about it, but to change less once its context has been "well understood" from many examples. After the tenth instance of the type, the movement of the vector is half what it was the first instance. After twenty instances, the effect is one third the original effect, and so on. Incorporating this term allows the coefficients α and β to be larger in magnitude so that organization can develop more quickly, but then become stable late in the process. Using this term reduces the number of times the corpus must be presented.

Initialization and normalization

The left and right vectors are, once again, constrained so that they maintain unit length throughout the algorithm, so that similarity can be measured by simple dot products, but this treatment is for computational convenience and plays no role in the biological interpretation. Unlike our previous algorithms, which initialize the vectors by counting adjacent types throughout the corpus, we now initialize our half-vectors by assuming them to be independent, identically distributed (IID) random variables distributed uniformly on a high-dimensional unit sphere. This choice of initialization reflects the "ignorance" of a brain that has no prior exposure to the language.

Clustering

This algorithm does not directly yield a discrete categorization; rather, it produces a geometric representation for each word type, so that each word type's vector is a continuous variable in a high-dimensional space. Again, this representation is potentially much richer than a simple assigned label, but because we want to apply this model to a classic natural language processing task, we cluster the vectors, so that we can infer POS tags. Clustering is performed using a weighted *k*-means algorithm.

The number of clusters, *k*, is chosen either for the purpose of direct comparison with other algorithms, or a very large value can be chosen for prototype supervision as described in the previous chapter. Cluster centroids are initialized using the final descriptor vectors of each of the *k* most frequent types, in much the same way as the SVD2 and ISVD implementations. Many-to-one (MTO) accuracy is then computed in the standard way for evaluation.

Multiple vectors

Remarkably, Hebbian update and orthogonalization are the only two mechanisms needed for the model to organize well. This simple model however, just like the SVD-based algorithms, inherently lacks the power to disambiguate, as only a single vector per word type is available. However, unlike our previous algorithm, there is an obvious extension of the model that permits disambiguation – the inclusion of multiple vectors for each word type. When a token is encountered, the model must then decide which of the candidate vectors should be used. When using multiple vectors, we consider an entire sentence simultaneously (rather than one bigram at a time) and find the optimal sequence

of vectors that best "explains" all the words in the sentence. The choice of optimal sequence is made to maximize the total sum of dot products of all bigrams in the sentence.

When we do not use multiple vectors for a type to allow for disambiguation, the distinction between a single vector with two halves versus a different vector for each half is unimportant. With multiple vectors, however, the distinction is important as a left-right pair must always be used together as one vector. We do not allow the left half of a type's first vector to be used in combination with the right half of its second vector. Such use would imply that the token occurs in one sense according to its left context and another sense according to its right context. Instead, we enforce a single interpretation for each token by requiring the same left-right pair must always be used together.

For example, suppose we allow two vectors per type and our corpus has a sentence of length ten tokens. In this case, there are 1024 possible sequences of vectors that could be used to represent the sentence. We use the subscript *i* here to indicate the position in the sentence in a temporary deviation from consistent notation. We define an objective function, *F*, for the sentence, which takes as an input any of the 1024 possible sequence of vectors for that sentence. When maximized, the objective function provides the optimal sequence of vectors.

$$F\left(\left\{ W_i^{v_i} \right\}_{i=1}^{10}\right) = \sum_{i=1}^{9} \left\langle W_i^{v_i,\,\text{right}} , W_{i+1}^{v_{i+1},\,\text{left}} \right\rangle$$

We write such a sequence as $\{v_1, v_2, \ldots, v_{10}\}$ where each v indicates the binary choice for a word's preferred descriptor. (If we allowed three or more vectors per type, then the v's

could take any of the three or more values.) This objective function simply sums the dot products of the half-vectors used for each bigram. So if v_i represents a particular choice among word i 's possible vectors (i.e. vector 1, 2, etc), then $w_i^{v_i}$, denotes the current value of the dynamic high-dimensional vector corresponding to that choice of descriptor vector. The additional superscript in the following equation indicates the appropriate choice of left or right half of each vector.

The goal then is to find the *argmax* of F over all possible sequences, $\{v_1, v_2, \ldots, v_{10}\}$. The original algorithm must be amended to perform this maximization. This is done through a simple implementation of the Viterbi algorithm, i.e., dynamic programming. Because of the efficiency of dynamic programming, the required maximization does not significantly impact performance compared to the original, single vector per word type, implementation.

Again, maximizing the sum of the dot products is a natural criterion for selecting the optimal sequence of descriptor vectors. Once learning has progressed, we expect that the optimal pair for a bigram in a commonly used construction will be nearly parallel because the vectors would have been directly or indirectly attracted to the same direction in space. On the other hand, any two vectors with a low dot product must have had little direct or indirect Hebbian attraction throughout learning, indicating that their use in the current sentence is either incorrect or must represent a novel construction. Early in learning, every presentation is somewhat novel, so the algorithm must break the symmetries early on to allow a more consistent choice of descriptor vectors later in the process.

Once the optimal choice of vectors is selected, the appropriate halves of the selected vectors are again attracted in the Hebbian style. These vectors are also given the same orthogonalizing push to avoid over-generalization. Lastly, when one of the candidate vectors is used for a particular type, it is helpful to give a small orthogonalizing push to the unused candidate or candidates, moving them away from the selected vector. This push encourages the unused vectors to take on a different grammatical role from the role played by the optimal vector. The goal is for these other candidates to eventually converge to a different cluster and therefore a different POS tag, which can then explain alternate uses of the ambiguous word type.

Once the vectors have stabilized, they are again clustered using weighted *k*-means. This clustering yields a label for each of the candidate vectors for every word type. An additional pass through the corpus is needed (again using Viterbi on each sentence) to find the optimal sequence of the finalized descriptors that explain the data. The label of the optimal descriptor vector is then assigned to its token. The MTO map is again constructed to build a correspondence between the labels and the tags, and the MTO accuracy is computed.

Hebbian POS tagging results

The non-disambiguating version of the Hebbian algorithm performs remarkably well, which may be somewhat surprising considering the very simple, biologically inspired nature of the algorithm. In fact, this approach out-performs the HMM's, and is competitive with the full ISVD algorithm. As shown in figure 4.1, the Hebbian tagger achieves 68% accuracy on PTB45 – a 2% increase over the best of the HMM results but a

3% drop from ISVD. It also achieves 74% on PTB17 – equaling the score of ISVD, at a 4% increase over the best of the HMM's. Scores are obtained with $\alpha = 0.5$ and $\beta = 1$.

Model	PTB17	PTB45
Hebb	**0.74**	0.68
ISVD	**0.74**	**0.71**
HMM-EM	0.65	0.62
HMM-VB	0.64	0.61
HMM-GS	0.68	0.66
HMM-Sparse(32)	0.7	0.65

Table 4.1. Tagging accuracy under the best M-to-1 map. HMM-EM, HMM-VB and HMM-GS show the best results from Gao and Johnson (2008); HMM-Sparse(32) shows the best results from Graça et al. (2009). ISVD is the second implementation of the algorithm from the previous chapter. Hebb is the non-disambiguating biological model.

It should be noted that we only proceed through the corpus one time to produce these results. We can improved the performance slightly (roughly 1% on average) by making the coefficients much smaller, removing the frequency adjustment term, and iterating several times through the corpus. Furthermore, these values represent the mean of repeated experiments. The standard deviation is roughly 1% and results from the random initialization and sampling of vectors for orthogonalization.

Significant efforts were expended to tune the Hebbian model in order to allow disambiguation through the use of multiple vectors, but these efforts remained unsuccessful. Indeed the extended model achieves scores in the low 60% range for PTB45 and only slightly better for PTB17: it has thus far been unable to match the performance typical of the best HMM's, and lags well behind the non-disambiguating Hebbian model. This failure is not terribly surprising, in light of our discussion in the

previous chapter. Namely, HMM's may actually perform worse using their disambiguating tags than they would if modal tags were used instead, and we also noted that 92% of the corpus (PTB45) can be explained without disambiguation. This may well be just another example of Occam's razor: "entities must not be multiplied beyond necessity," or “the simplest explanation is the best.” For the vast majority of word types, necessity only demands a single tag, and this simple explanation seems to perform the best.

Robustness

Without the frequency adjustment term, the Hebbian model is not particularly robust, as performance becomes somewhat sensitive to the coefficients of attraction and orthogonalization. The frequency adjustment makes the model more robust but comes at a small cost in accuracy. Without it, relatively small changes in these coefficients can lead to a performance drop of several percent. First, changing the ratio of these parameters too much can cause the model to fail, by allowing all vectors to converge to the same point; this happens when the Hebbian coefficient is too large compared with the orthogonalization coefficient. On the other hand, when this same ratio is too small, significant organization can fail to progress from the start.

Not only is the ratio between these parameters important so that a proper balance be struck, their individual magnitudes are important as well. If the parameters are too large, vectors change too much each time their word types are encountered. Early in learning, this causes vectors to “over-commit,” preventing them from effectively integrating information across multiple tokens. If the magnitudes of the parameters are

too small, organization is very slow, because the algorithm must encounter many tokens of a type to allow its vector to move sufficiently far to be assured of reaching its desired cluster even if the direction of attraction is completely consistent (which of course it generally is not). This behavior is particularly problematic for rare words, which may have only one or two tokens in the corpus. In fact, it is partially for this reason that we must run the algorithm over the corpus multiple times as previously noted when the frequency adjustment term is omitted.

Fortunately the model is so simple, having only two parameters, that, despite this lack of robustness, it is possible to tune the parameters to achieve excellent performance. However, modifying this model to make it more powerful can make it very difficult to tune. Our inability to achieve disambiguation may well be an example of this parameter sensitivity. As a second example of an extension of the algorithm, we have examined including a functional dependency of the learning coefficients, α and β, on the type frequency of each word in the bigram. If a novel word occurs alongside a frequently observed word, it stands to reason that the novel word's vector should be changed more than the vector of the word that is, presumably, already well-understood, and by more than it would change were its neighbor equally rare. Implementing this change should allow the model to capture the important "one-shot learning" capability of natural language learners. While we have been able to improve performance by including frequency dependence on the single word being updated (the frequency adjustment factor previously described), we have not been able to find a satisfactory way of including a dependence of our learning parameters, α and β, on the frequencies of both word types

that make up the bigram. Hence, we have been unable to demonstrate "one-shot learning"; we attribute this difficulty to the lack of robustness of the algorithm. Despite the general lack of robustness, however, we believe that the impressive performance of the Hebbian algorithm is a testament to this novel approach to modeling in natural language processing. Achieving state-of-the-art performance on PTB17 and a level of performance on PTB45 that is only surpassed by our own ISVD algorithm indicates that this approach is indeed a powerful one.

Matrix factorization

In this section, we relate our Hebbian algorithm to a matrix factorization algorithm reminiscent of the Generalized Hebbian algorithm (GHA) of Sanger (1989), later extended by Gorrell (2006). The Gorrell version of GHA attempts to estimate the singular value decomposition of a data set based on a sequence of single observations of word pairs for latent semantic analysis. It includes a Hebbian-style term with orthogonalization to incrementally estimate one singular vector at a time, in the decomposition of the matrix of bigram frequencies. Here, we relate these same two dominant mechanisms of our algorithm to a different decomposition of the bigram frequency matrix. Unlike GHA, it is not possible to directly produce our algorithm from this kind of derivation because our algorithm is designed for a much different task than estimation of the SVD.

In particular, our algorithm must gradually change each descriptor vector (one per word type), moving the system towards organization. GHA builds a singular vector for each latent feature dimension, and it builds them one at a time in order of decreasing

singular value. Loosely speaking, GHA builds column vectors (of the SVD) one at a time, while our algorithm builds the row vectors (of a different factorization) all together by updating them one at a time.

For this formulation, we consider the language as a stochastic process that generates i.i.d. bigrams which are pairs of random variables, (X_1, X_2), taking values in the set of possible word types. We define the probability distribution that governs this process by the probability matrix B so that B_{ij} stores the probability of observing the bigram (W_i, W_j):

$$Pr\left(\left(X_1, X_2\right)=\left(W_i, W_j\right)\right)=B_{ij}$$

The object of the algorithm is to approximately factor the matrix B as a product of two matrices, L and R, which, as we will later see, come to represent the collection of left and right halves of our descriptor vectors.

$$B \approx R\,L^T$$

While B is a large square matrix with size determined by the number of possible types in the corpus, L and R have identical sizes with a number of rows to match B but far fewer columns. (This factorization implements dimensionality reduction reminiscent of the algorithms in the previous chapter.) The dot product of row i of matrix R with row j of matrix L gives an approximation of the probability stored in B_{ij}. Recall that the dot product of the left and right half of two vectors in our Hebbian algorithm captures the compatibility of two words in a bigram. (Our continual renormalization to unit length causes our vectors to no longer capture a true probability matrix, but the connection is clear nonetheless.)

In this formulation, we seek matrices *L* and *R* to minimize the Frobenius norm of the difference between the probability matrix and our decomposition. After randomly initializing the matrices, *L* and *R,* we use a gradient descent approach to refine them into a locally optimal decomposition. Gradient descent is performed on the objective function that computes the square of the norm of this matrix difference.

$$E(L, R) = \sum_{ij} (B_{ij} - [RL^T]_{ij})^2$$

We compute the gradient by differentiating with respect to each element of the unknown matrices.

$$\frac{\partial E}{\partial R_{ab}} = -2 \sum_{j} (B_{aj} - [RL^T]_{aj}) L_{jb}$$

$$\frac{\partial E}{\partial L_{ab}} = -2 \sum_{j} (B_{ja} - [RL^T]_{ja}) R_{jb}$$

To optimize, we move down the gradient based on a step size η. When written in matrix form, this gives the update for matrices *L* and *R*.

$$R := R + 2\eta (B - RL^T) L$$

$$L := R + 2\eta (B - RL^T)^T R$$

An sequential algorithm has access only to the current bigram and cannot actually incorporate *B* as is required in this update rule. Instead of using the true but unknown value of *B* each step, we choose to approximate it based on the lone observation of the current bigram. In other words, in response to the bigram indexed by (τ, ω), we estimate *B* by a one in position (τ, ω) with zeros elsewhere.

$$\hat{B}_{ij} = \delta_{i=\tau, j=\omega}$$

Our update rules are only slightly modified under this approximation, and the choice of the Kronecker delta for the estimate of B is unbiased, because, on average, it agrees with B.

$$R := R + 2\eta(\hat{B} - RL^T)L$$

$$L := L + 2\eta(\hat{B} - LR^T)^T R$$

It is useful to move away from the matrix notation to investigate this update.

$$R_{ab} := R_{ab} + 2\eta L_{\omega b}\delta_{a=\tau} - 2\eta \sum_{jp} R_{ap} L_{jp} L_{jb}$$

$$L_{ab} := L_{ab} + 2\eta R_{\tau b}\delta_{a=\omega} - 2\eta \sum_{jp} R_{jp} L_{ap} R_{jb}$$

These equations describe an update that consists of two components. The first is a Hebbian-style attraction which appears in the terms that include the Kronecker delta. This term in the first equation indicates that when updating the row of R that matches the first word in the bigram, we move in the direction of the row of L corresponding to the second word in the bigram. We see the familiar movement of the right half-vector of the first word moving toward the left half-vector of the second word! The term with the delta in the second equation is the reciprocal Hebbian step of the left half-vector of the second word toward the right half-vector of the first word.

The other term involving the sum in each equation is a little more difficult to interpret but represents an "all-to-all" orthogonalization. For example, each row of R is orthogonalized to each row of L by subtracting a portion of the parallel component. This process is remarkably similar in style to our simple orthogonalization force, but rather

than explore this term in detail in this form, we make an additional approximation. While the update requires a change to every term in each matrix, we instead implement a step that is only partially in the gradient direction. In particular, we choose to change only the rows involved in the Hebbian term in direct response to the bigram (τ, ω). We also write these equations in vector form by dropping the column subscripts. The single subscript indicates the index of the entire row vector. By recognizing that one of two sums in each equation represents a dot product, we can write the updates as:

$$R_\tau := R_\tau + 2\eta L_\omega - 2\eta \sum_j \langle R_\tau, L_j \rangle L_j$$

$$L_\omega := L_\omega + 2\eta R_\tau - 2\eta \sum_j \langle R_j, L_\omega \rangle R_j$$

We see that the Hebbian step for the right half vector is followed by a one-to-all orthogonalization. This differs from our Hebbian algorithm implementation in which we orthogonalize away the mean of a small number of randomly chosen components and scale the step by a larger factor than the one used for the Hebbian term to compensate.

While this factorization by gradient-descent approach does not exactly replicate our algorithm, it is surprisingly close, and it does help us interpret our biological mechanisms. The Hebbian attraction term enforces the probable co-occurrence of the words in the observed bigram. The orthogonalization term attempts to compensate for all the other possible bigrams unobserved during this presentation, making each of them less likely. Because we only update the rows of the observed words, we only "penalize" the unobserved bigrams that include one of the two words in the observed bigram. These roles in the gradient descent mirror the biological roles we envision for these forces with

the probabilistic notion of bigram likelihood replaced by the closely related idea of bigram compatibility. In particular, the Hebbian term promotes the "wiring together" of words that are observed together. The orthogonalization term prevents "over-wiring" by generally trying to keep all vectors apart. The link between this factorization algorithm and our Hebbian algorithm is quite close and therefore compelling.

Context Free Grammar Induction

**Note: see remark 5 in the preliminary dissertation pages regarding notation for the remainder of the chapter.*

While the unsupervised syntactic categorization of POS tagging is, in itself, a challenging problem, the problem of unsupervised grammar induction is considerably more ambitious. In this section, we discuss the application of a Hebbian-style model to context-free grammar (CFG) induction, which requires inferring not only the non-terminal "categories" but also the production rules that govern their use. We demonstrate the ability of this enhanced algorithm to learn several relatively simple grammars, and close with a discussion of challenges that must be overcome before this type of modeling can be applied to natural language data.

Context free grammars

A context-free grammar (CFG) is a set of recursive production rules that generate strings of terminal symbols. A CFG consists of four basic components:

- Terminal symbols are the characters that compose the strings generated by the grammar.
- Nonterminal symbols are placeholders for sequences of terminal and nonterminal

symbols.

- Production rules govern the rewriting of nonterminal symbols into strings of both nonterminal and terminal symbols.
- The start symbol is a special nonterminal symbol that initializes the process.

The process of generating a string of terminal symbols from a CFG is recursive.

- The start symbol is the "initial string."
- Given any string that contains at least one nonterminal symbol, pick the first nonterminal symbol and apply its production rule to rewrite that nonterminal into some combination of nonterminal and terminal symbols.
- Repeat this process until the string consists entirely of terminal symbols.

It is not necessary that this process actually terminate (depending on the production rules) so the general CFG framework is not constrained to produce only finite strings; however, for all examples considered in this work, the CFG's always produce finite, relatively short strings.

We later revisit the following most simple non-trivial CFG that produces bigrams (i.e. pairs of terminal symbols), but we introduce it now as an example of the CFG process. The production rule governing the start symbol, S, rewrites S into a pair of nonterminal symbols AB (i.e. $S \rightarrow AB$). The production rule for the nonterminal symbol A rewrites A into one of several terminal symbols $\{a_i\}$ (i.e. $A \rightarrow \{a_1, a_2, \ldots, a_n\}$). The production rule for B matches the rule for A but with a disjoint set of terminal symbols (i.e. $B \rightarrow \{b_1, b_2, \ldots, b_n\}$). This simple CFG produces a set of n^2 distinct bigrams of the

form (a_i, b_j). We typically assign a probability distribution (often uniform) to the choice of terminal symbols in the rewrite rules for A and B, so we actually have described a probabilistic context free grammar (or PCFG), which induces a distribution on the set of strings. This example is particularly simple because, other than the start symbol, nonterminal symbols are never rewritten into strings containing other nonterminal symbols. Also, there is no overlap between the two sets of terminal symbols, so that each is uniquely associated with a particular nonterminal symbol. Neither of these constraints are required for a more complex CFG, and removing either would make the CFG more difficult to "learn."

Syntax bricks and terminal bricks

At the heart of the extension of our Hebbian model to the problem of CFG learning is a new element that we refer to as a syntax brick, and a modification of the descriptor vector into what we call a terminal brick. Figure 4.2 illustrates these two types of bricks. A terminal brick holds a single descriptor vector that stores a geometric representation of a terminal symbol. A syntax brick holds three descriptor vectors: one on top, one on the bottom left, and one on the bottom right. Vectors on the top of any brick (terminal or syntax) are free to bind with vectors on the bottom of a syntax brick.

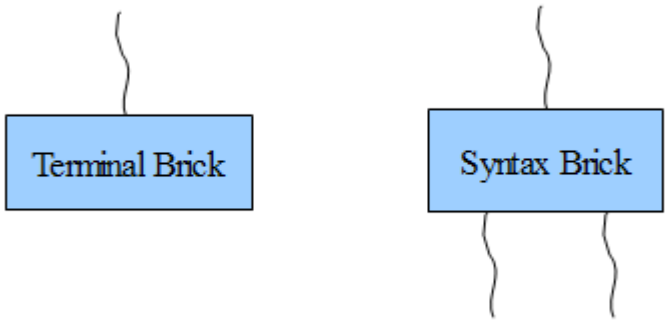

Figure 4.2: The terminal brick is a descriptor vector that represents a particular terminal symbol. The syntax brick is a collection of three descriptor vectors: a top vector, a left vector, and a right vector.

The key feature in the Hebbian model for POS tagging is the ability for word types to interact indirectly with similar types through the binding of descriptor vectors. (Recall the *old* / *Olympic* example.) This interaction comes about because left and right halves of the descriptor vector of each word retain a trace of all their past neighboring words. The syntax bricks serve the same function in this Hebbian CFG model. Figure 4.3 depicts a single syntax brick that has bound to two terminal symbols from the AB-type CFG described above.

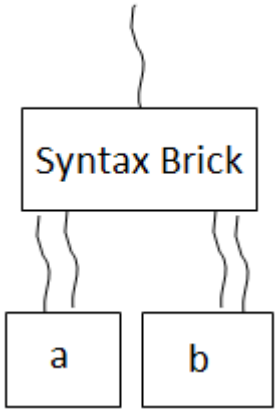

Figure 4.3: We show two terminal bricks bound to a syntax brick.

As a result of this binding, the same Hebbian-style attraction brings the bound vectors closer together. If the same syntax brick later binds with the two new terminal symbols in a new string produced by the same CFG, the Hebbian attraction will cause the same indirect interaction between the two terminal symbols.

The role of the top site on the syntax brick is to allow a syntax brick to bind to one of the bottom sites of another syntax brick so that we can combine syntax bricks and terminal bricks into a tree representation of the input string. Figure 4.4 shows two alternate trees for a single CFG string. Which tree actually forms depends entirely on the descriptor vectors as we later describe in the algorithm.

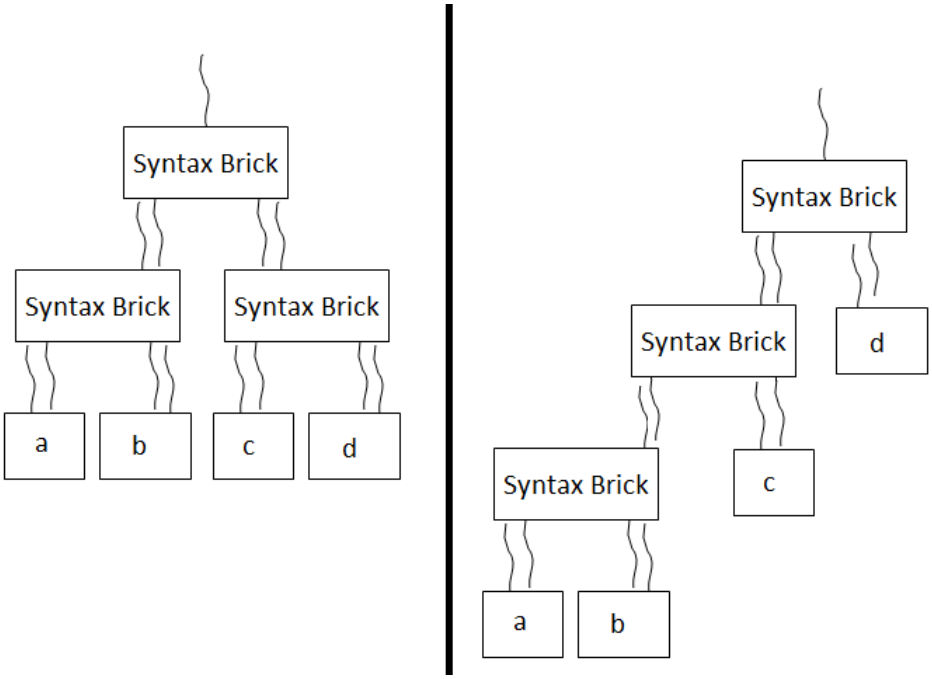

Figure 4.4: Two alternate trees that could form in response to the input string (a,b,c,d). The left tree is called center branching. The right tree is called left branching.

Data

We train the model to learn a grammar by repeatedly giving it examples of grammatical strings. We do not provide the model with ungrammatical strings. While negative examples with an indication of their impossibility could certainly be exploited for improved performance and would make learning easier, we rely exclusively on positive examples because natural language is acquired dominantly from exposure to positive examples. The model is given an excess of syntax bricks which could be used in order to build a tree. As learning progresses, it is common that some syntax bricks will fall out of favor, eventually no longer being used for any tree, while others will be used consistently and in the same role from presentation to presentation. This phasing out of bricks is feasible mathematically in this Hebbian CFG model because unlike the Hebbian

POS model, the vectors are constrained not on the unit sphere but rather inside the unit ball. We very gradually shrink the magnitude of unused vectors so that, eventually, they effectively disappear.

Outline

As with the Hebbian POS tagger, we present a brief outline of the Hebbian CFG model before describing the steps in more detail in the following sections.

Hebbian CFG Model
Initialize all descriptor vectors of bricks i.i.d uniform
Loop over CFG strings until vectors stabilize
-Build the "fittest possible" tree
-Apply Hebbian attraction to bonded vectors in the tree
-Apply orthogonalization to these same vectors
-Apply small vector contraction to syntax bricks
Evaluate by examining all bricks to determine whether the system has become selective to only grammatical strings

Hebbian attraction

There are two essential processes in our learning algorithm. First, we must specify how the trees are built in response to an input string. With the tree in place, we must then describe the dynamic response of the descriptor vectors in each brick to their use in the tree. We begin with this second aspect – the dynamic response of vectors – because it closely resembles the Hebbian POS tagging algorithm. Later, we describe the process of tree building, which is somewhat more complex.

Any two descriptors (represented by x and y in the following equation) that are bound together in the tree structure are attracted according to a Hebbian rule very similar to the one used for POS tagging.

$$x = x + \alpha\, y$$

$$y = y + \alpha\, x$$

This attraction applies to terminal bricks bound to syntax bricks as well as to syntax bricks bound together, but the coefficient depends on which type of bricks are bound. The coefficient is larger for the change of terminal brick vectors than for the change of syntax brick vectors, because the syntax bricks are used more frequently. These coefficients of attraction are free parameters. The optimal choice typically depends on the grammar. There is no equivalent to the bigram-count adjustment of the Hebbian algorithm.

Orthogonalization and noise

Not surprisingly, we again employ an orthogonalization force. We pick two vectors at random (uniformly) to orthogonalize. The orthogonalization could more properly be called a "push away" force. Each vector is moved in the direction of the vector difference as seen in figure 4.5. This push away force causes any two vectors that are somewhat parallel to become more orthogonal.

$$x = x + \epsilon\,(x - y)$$

$$y = y + \epsilon\,(y - x)$$

We often find it useful to orthogonalize more than just one pair of vectors, and the particular choice of how many works best depends on the grammar as does the best choice of ϵ.

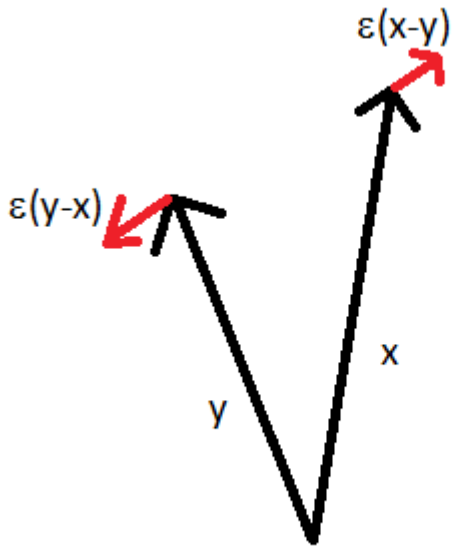

Figure 4.5: The vector y is pushed away from the vector x by adding to it a fraction of the vector difference (y-x). Similarly, x is pushed away from y by adding a fraction of (x-y). The red vectors indicate the direction of change for both x and y.

In addition to the Hebbian and orthogonalization forces that are also present in the POS tagging algorithm, this algorithm includes a small amount of noise to perturb shape vectors. This noise helps break symmetries early in the learning process. It can be omitted later in learning, to allow the vectors to converge to tighter clusters if desired.

Tree building

The most challenging part of this model is the process of tree building. The goal is to find the tree that maximizes the "fitness" of the entire construction. This step is analogous to the choice of an optimal sequence of descriptor vectors in the disambiguating version of the Hebbian POS tagger. We seek the particular selection of syntax bricks and their placement to maximize the total sum of dot products of all pairs of

bound vectors in the tree. The process of finding the optimal tree quickly becomes intractable as the length of the tree grows and the number of available syntax bricks grows because the number of possible branching patterns grows quickly and any the number of trees for each branching pattern is large.

To facilitate tree building we can employ a special root brick that, when used, always sits at the root (i.e. top in our depiction) of the tree. This brick is identical to every other syntax brick apart from the fact that it needs no top vector because it sits at the top. This brick mirrors the nonterminal start symbol of a CFG. Its inclusion facilitates learning by encouraging more consistent use of the same subset of the syntax bricks at the top of the tree.

We employ several techniques for building trees. If the input string is sufficiently short and the number of possible syntax bricks is small, it is possible to exhaustively search for the best tree. Unfortunately, very few CFG's are simple enough to be studied in this fashion. Another strategy is to choose the optimal tree out of a set of candidates built at random. With this approach, we build many trees using random placement of available syntax bricks. The fitness of the construction is evaluated by summing the dot products of all bound vectors, and the most fit tree is selected.

There are two drawbacks to this approach. First, uniform random placement of syntax bricks tends to bias the trees to be center-branching. Choosing a non-uniform sampling can help bias the algorithm away from center-branching trees, but the bias in the choice of distribution must be drastically increased as the strings get longer in order to compensate for the combinatorics which prefer center-branching trees. Second, for all

but the simplest CFG's, the number of possible trees is vast, so in order to have any confidence that even a good tree (much less the optimal tree) was constructed, a very large number of candidates must be sampled.

The strategy we generally employ is that of a greedy search. We build the tree one brick at a time, choosing the best brick from among several candidates. This approach is not without its own drawbacks. In particular, it continues to prefer center branching trees, and, as with any greedy search, there is no guarantee that an optimal (or even highly fit) tree will result. To improve on a basic greedy search, we actually run the greedy search many times and choose the best of the resulting constructions. This ad hoc tree building process allows us to demonstrate the capability of our bricks to learn several CFG's; however, it certainly limits the complexity of the CFG's that can be explored successfully. To extend the capability to more difficult CFG's, the tree building process itself would need to be carefully modeled, but such an endeavor lies well outside the scope of this work.

Example: AB or BA

The simplest possible CFG that can be explored by this approach is the AB grammar used as a previous illustration. This problem is trivial and presents no difficulty in learning. In particular, there is no interaction between terminals symbols that come from the nonterminal symbol A, and those that come from B. Some syntax brick is chosen to sit above an input string of the form (*a,b*). The *a* terminal brick binds to the bottom left vector in the syntax brick while the *b* brick binds to the bottom right vector. The next time the same syntax brick is used, we are guaranteed that its left site will again

bind to an *a* while its right site will bind to a *b*. In the end, all the *a*'s eventually indirectly interact with one another as do all the *b*'s, but neither interacts with the other. Not surprisingly, the *a*'s quickly converge to some direction in space as do the *b*'s. There is no tendency for these directions to be the same, and, in a high dimensional space, they will be nearly orthogonal with high probability.

The same is not true for the slightly more complicated CFG we describe as AB or BA. In this CFG, the start symbol is rewritten by a pair of nonterminals A and B, but they can occur in either order with equal probability. As with the AB grammar, A and B are re-written into disjoint collections of terminal symbols. This CFG always produces bigrams that include one terminal symbol of the A type and one of the B type, but these terminals occur in either order equally often.

Learning this grammar correctly would be impossible if only one syntax brick is available, because its left site would equally often bind to terminals of each kind. The same is true of its right site, so that, quickly, the terminals would all converge to a single direction in space and both the left and right vectors on the syntax brick would converge to the same direction. This configuration is a failure because, while the syntax brick is content to bind to an input of either form, AB or BA, it is equally content to bind to an input of either form AA or BB. It has failed to detect a difference in any of the terminal bricks. When our model fails to differentiate terminals of two or more different types, we say that it has over-generalized. Over-generalization is the most common failure of this model on difficult CFG's.

While employing only one syntax brick is guaranteed to cause over-

generalization, this grammar can be learned with two or more syntax bricks. The ideal solution is to have all the terminals of the A type converge to a single point. Similarly, all terminals of B should converge to a distinct point (presumably close to orthogonal to A). One syntax brick should have its left vector match the A direction while the right vector matches the B direction. A second syntax brick should have reversed left and right vectors from the first brick. In response to a bigram of the form (*a, b*), the first brick is employed to bind to the input. In response to the opposite type of bigram, the second brick is employed.

There are alternate stable configurations our algorithm can, in principle, achieve. For example, the A type terminals could all converge to one of two directions while the B types converge to a third direction. In this case, a minimum of four syntax bricks would need to emerge to explain all possible inputs. A terminals of either of the two converged directions could occur either before or after the B terminal in the input bigram, so we need one syntax brick to account for each of the four possible configurations. We describe behavior like this as under-generalization, because the model has failed to ascertain that there is no statistical difference between the two kinds of A's that it has classified.

In practice, under-generalization is extremely rare and is not problematic, since the model can still detect any non-grammatical string. In fact, across a wide range of parameters, the model converges to exactly the state we desire for the AB or BA grammar. Under some combination of parameters, the model can occasionally or consistently over-generalize. Using this simple example, it is easy to see why negative

examples would be so helpful. At initialization, the model does not know how many syntactic categories are present nor does it know the grammar rules. The model receives input like what is shown in table 4.2. With a few negative examples, we can quickly see that some strings are not grammatical and move our model away from a state that accepts them as grammatical, so that if our model is headed toward over generalization, we can pull it back using the negative evidence.

2 9	3 5	2 9	9 8	2 10
10 8	10 5	1 8	8 1	9 7
2 10	4 3	3 7	2 9	7 3
5 9	8 3	2 9	4 3	1 7
3 5	7 6	6 7	6 2	1 4
4 10	4 9	9 4	8 1	7 6
4 3	6 8	6 5	1 5	6 4
9 7	10 2	4 9	5 6	10 4
3 8	8 3	7 10	5 6	6 7
9 2	1 8	9 5	2 9	1 2

Table 4.2: Each column in the table is a collection of bigram strings from the grammar AB | BA. Discovering the correct explanation of this grammar is difficult for an unsupervised learner.

Without any negative evidence it is not surprising, especially after looking at the table, that the model or a person might decide that any bigram is acceptable. Even knowing the rules of the grammar, it takes a minute of examination to piece out which numbers are terminal symbols of type A and B. These are the types that go together in table 4.2: (2, 4, 5, 7, 8) and (1, 3, 6, 9, 10). In fact, the state in which every terminal vector is

parallel is always an attractor of our approach, regardless of the grammar. Noise and orthogonalization are needed to help direct the model into a different attractor that avoids over-generalization. For this grammar, this is not difficult to accomplish.

Example: ABCD

In this example, there are four non-terminal symbols. Each has a set of associated terminal symbols that are mutually disjoint. Grammatical strings are sequences of four terminals of the form (*a,b,c,d*). While the previous example only required one syntax brick to be used with each string, this example does require the construction of a tree involving three bricks. However, there is no particular branching pattern that must be employed to explain this simple grammar. Figure 4.6 shows the average fitness (i.e. dot product) of a bond in the constructed tree each iteration. (An iteration refers to a particular instantiated string which is presented to the algorithm for explanation, the process of constructing a tree , and all subsequent vector manipulation.)

For this example, there were ten terminals for each of the four nonterminal types, making ten thousand total possible strings. We gave the model a total of ten available syntax bricks (plus the root brick) so that even with these short strings, the number of possible trees is quite large. As the syntax bricks and terminals self-organize, the average fitness of a bond increases. By the time only four hundred strings have been presented, this bond fitness has already approached its maximum value of one.

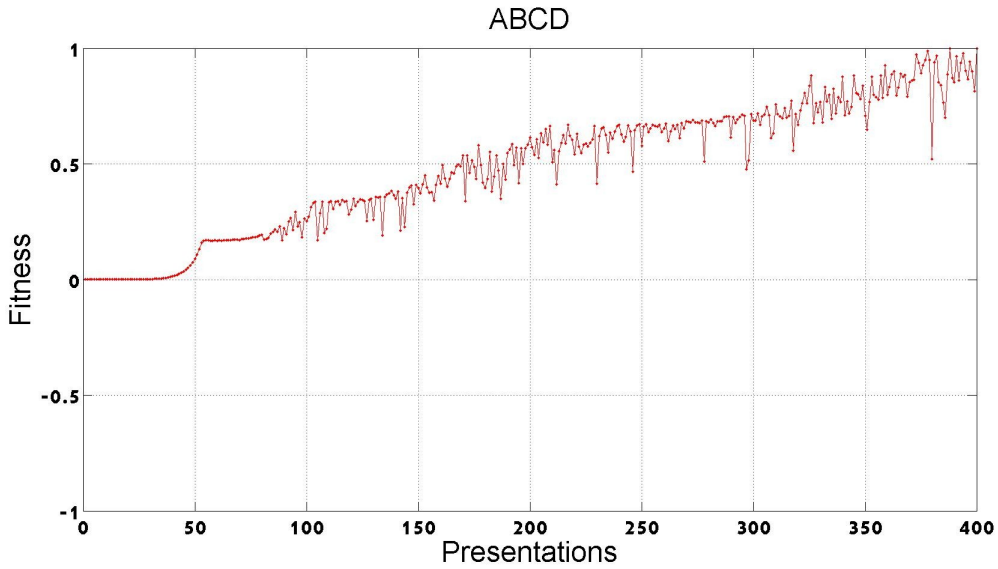

Figure 4.6: This plot shows the average fitness of a bond in the constructed tree for each string for the grammar ABCD.

The occasional dip in the fitness can be caused by one of two factors. First, we could see a terminal symbol that by random chance, has not been seen often and has not yet had the chance to converge. This explanation becomes increasingly less likely as more and more presentations occur. The more common explanation is that the greedy tree searching process simply failed to find a particularly good tree. As we see from this plot, the fitness is highly variable because of this difficulty; nevertheless, the algorithm still manages to converge to a desirable description of the CFG. In figure 4.7, we illustrate the way in which the syntax bricks organize to describe the grammar.

The complicated collection of circles, boxes, and arrows is a typical output of our algorithm. A box is a syntax brick and its three included circles are its three vectors. The letter and number of each vector indicate the index of the brick as well as the location of the vector on that brick. The arrows joining the bricks indicate high fitness between the vectors connected by the arrows. We see that syntax brick 11 (the root) binds on the right

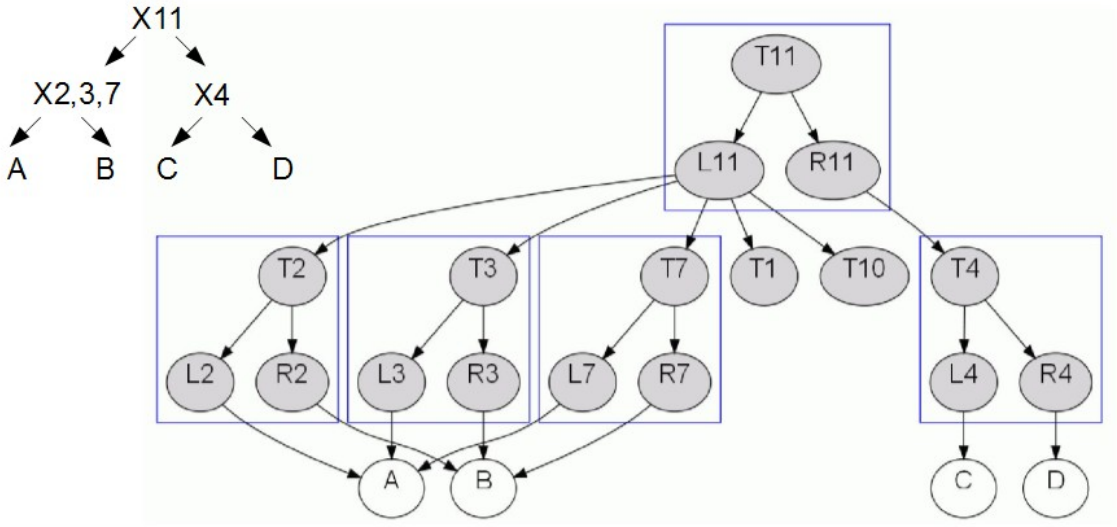

Figure 4.7: The top left illustration depicts the center branching tree that our model learns in response to the grammar ABCD. X refers to a syntax brick. Numbers following the X indicate the index number of the particular brick. We see that our model has created three redundant bricks - 2,3, and 7 - where only one of the three was needed. A, B, C, and D indicate the collection of terminal symbols. In the larger illustration, we show which vectors bind together well. Vectors are indicated by ovals. T, L, and R correspond to top, left, and bottom sites of a syntax brick, and again the following numbers index the brick. A box is drawn around the three vectors of a single brick. The arrows that exit the syntax bricks indicate that the vectors at either end of the arrow have high fitness.

to the top of syntax brick 4. On the left, it binds to the top of bricks 2, 3, and 7 as well as the top of 1 and 10. Bricks 1 and 10 do not bind downward with high fitness to any other bricks so they are not used in practice. Their inclusion in the figure is a residual effect of the fact that they were used earlier in learning, but later the model found that they were no longer needed. With more training, it is possible that 2, 3, or 7 may have a similar fate, because only one of the three is needed. We see that 2, 3, and 7 are identical bricks, because their left and right vectors have matching affinities for terminals of type A and B respectively. Similarly, the left and right vectors of brick 4 bind well with terminals of

type C and D respectively.

The small illustration at the top-left of the figure condenses this complicated plot into a simple description of the trees that the model can build. We see that the model has learned a center-branching representation for this grammar. Such a choice is not mandatory but neither is it surprising.

Example: AB or ABCC

This grammar is the final example we describe in detail, but it demonstrates the ability of the model to capture some interesting behavior. Again, we have ten terminal symbols for each of the three types of nonterminals. We have 20 syntax bricks available to the model, plus the root brick. We first show the plot of the fitness as a function of iteration.

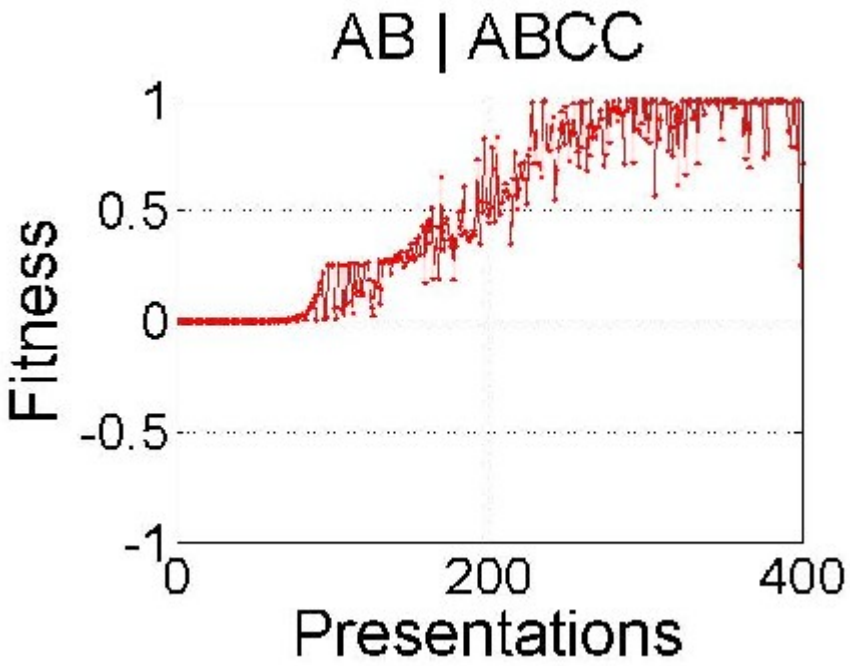

Figure 4.8: This plot illustrates the convergence of the algorithm to a solution of the AB or ACC grammar.

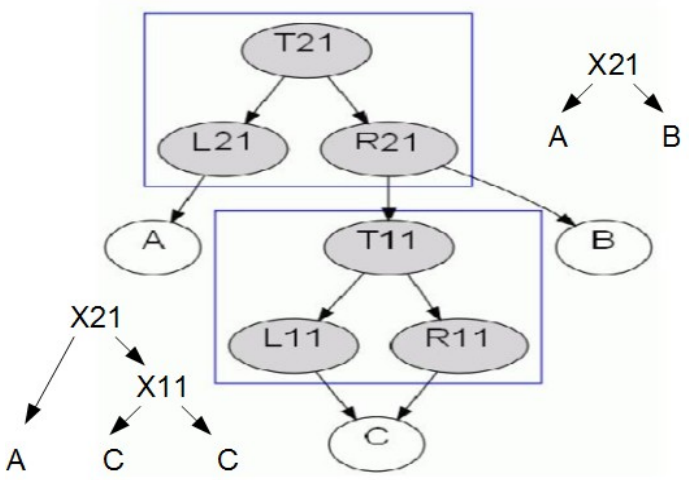

Figure 4.9: This figure illustrates the nature of the trees found in response to the AB or ACC grammar.

Just as in the previous example, we see that 400 presentations are sufficient for the algorithm to approach convergence. While the previous example required only about fifty presentations for the fitness to begin to grow, this grammar requires about twice that number. However, once learning takes hold, the algorithm generalizes the grammar quickly. In figure 4.10, we see the tree structure that is learned.

New to this example is the way in which the top vector of syntax brick 11 has converged in a way that matches the terminals of type B. We see that root brick 21 has its left vector always binding with a terminal of type A. Its right brick is equally happy to bind with terminals of type B or with the top vector of syntax brick 11. Both of brick 11's bottom sites bind to terminals of type C. This particular description points toward the following CFG formalism:

$$S \rightarrow AD$$
$$D \rightarrow \{B, CC\}$$
$$A \rightarrow \{a_1, a_2, ..., a_{10}\}$$
$$B \rightarrow \{b_1, b_2, ..., b_{10}\}$$
$$C \rightarrow \{c_1, c_2, ..., c_{10}\}$$

This description neither over- or under-generalizes, so exactly the correct set of strings will fit well into this framework. It is interesting that the model correctly generalized the similarity between the lone nonterminal, B, and the pair of nonterminals, CC.

Other examples

We studied a number of examples of CFG's with this model. Most were reliably learned, some were learned correctly, but not in every experiment. Others, we were unable to learn without strongly constraining our approach by drastically limiting the number of syntax bricks, training on only a subset of possible terminal symbols, or changing the learning parameters during training. Examples that we could learn reliably include AB or CD, ABAB or BABA, and alternating AB (recursive with maximum string length 5).

An example that proved to be quite difficult to learn reliably was S → {ABCD, ABD, BCD, BD}. This problem represents the kind of flexibility present in simple English constructions. Here, A represents adjectives, B represents nouns, C represents adverbs, and D represents verbs. This grammar describes four simple grammatical constructions in English. The difficulty in this grammar is that the model has a strong tendency to over-generalize, because several of the nonterminals have strongly overlapping contexts. With great effort and by drastically limiting the number of syntax

bricks, the model learned the description shown in figure 4.10.

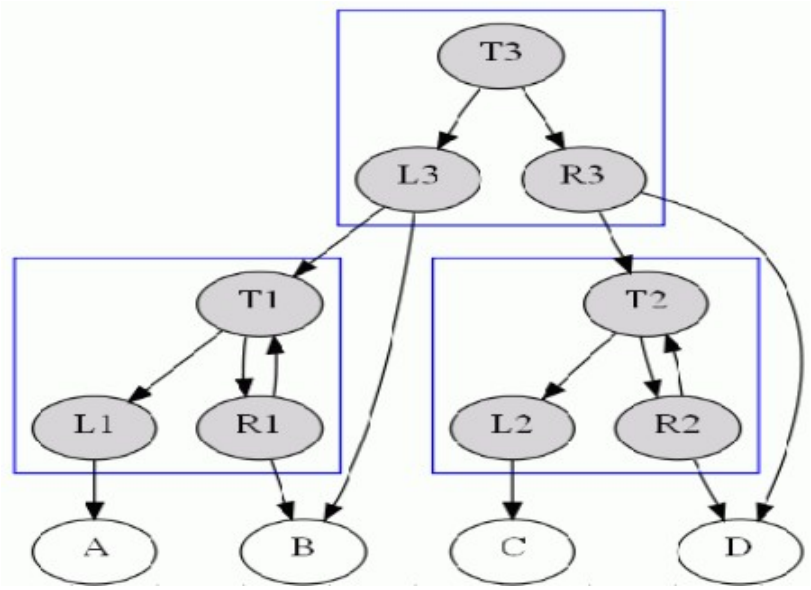

Figure 4.10: The response of the model to a particularly difficult grammar. The upward arrows within a syntax brick indicated that the vector on the bottom fits well with the vector on the top. This agreement can occur when the syntax brick can substitute for a terminal symbol it binds to.

A careful examination of figure 4.10 is required to verify that this description admits all grammatical strings and no ungrammatical ones. We see that the model has correctly discovered that the "ADJ-N" pairing can be used in place of a noun and the "ADV-V" paring can be used in place of a verb. While it is interesting that such grammars can be learned, the difficulty in learning this grammar again points to the lack of robustness inherent in this approach.

Discussion

While we would like to begin to employ this grammar induction algorithm on some simple natural language data, there are some fundamental limitations of our

approach that must first be overcome. Most notably, our model has no mechanism for nonlocal interactions or for agreement. An nice example of the need for both mechanisms is in Marcus (2000) which describes an experiment that is referred to as the Ga-Ti-Ga problem. In this experiment, infants are exposed to a grammar consisting of three sounds. The first and the last sound are always identical while the middle sound is always different. Ga-Ti-Ga is an example of a grammatical string. Ga-Ga-Ti is not. After exposure to this grammar for a period of time, the infants were then presented with novel sounds – sometimes arranged grammatically and sometimes not. The infants were consistently more attentive to the novel ungrammatical stimuli than to the grammatical ones. This experiment demonstrates that the brain, even in it earliest stages, is capable of learning this simple context-sensitive grammar. This problem is challenging because "sameness" is defined within a presentation in the form of agreement rather than across presentations as in categorization. It also involves non-adjacent bricks and our model has difficulty capturing non-local behavior.

Our model is not capable of learning this grammar because we have no mechanism in place to guarantee exact agreement between two terminals. Agreement plays a very important role in many natural languages particularly in number and gender, and our model is unable to account for it. A mechanism to account for this is needed before a serious investigation of natural language is possible.

As we previously noted, the most significant difficulty in our approach is the process of tree-building. The space of trees is vast, and our present implementation does not explore the space effectively. The solution is not to find a smart way to explore more

candidate trees, but rather to model the process of tree building itself. A successful step in this direction would almost certainly make this Hebbian approach to grammar induction more powerful and more successful. Despite these limitations, we find the successes of this biologically inspired approach intriguing especially in light of the more dramatic success of the Hebbian POS tagger. It is far from mature but points to a promising direction for future investigation.

Chapter 5: Conclusion

Unsupervised inference in the field of computational linguistics is extremely challenging. Models that work well on one natural language may not extend well to another. The vastly different statistical properties of open- and closed-class word types make it difficult to learn about both equally well. The prevalence of rare words makes is essential that the model be capable of interpreting some data based on very few instances. Although, in the information age, data from many languages in the form of text abounds on web pages and digital archives, in order to build and test algorithms it is necessary to have at least some annotated data that can be used to evaluate performance. Annotated data is relatively scarce by comparison, and its availability can significantly impact the model's design and performance.

In many cases, annotated data sets commonly in use may not be particularly well suited to unsupervised learning of a language's properties. The Wall Street Journal corpus, while used extensively in this dissertation, may be much less well suited as a starting point for unsupervised learning than a corpus of child speech for example. As is clear from our discussion of biological modeling, we believe that the ultimate key to success of unsupervised inference is attention to the process and mechanisms of human language acquisition, which is certainly the most successful example of (relatively) unsupervised linguistic inference. In light of this fact, working first with data taken from the Wall Street Journal may well limit the ability of a model to be successful. Unfortunately, it is exactly this kind of data that has been used extensively in the

literature in the past, and models are judged based on the success on just such data. There are several valid reasons why this may be the case. First, supervised methods for problems like POS tagging have been extremely successful on these complex data sets, and the community is interested in examining unsupervised methods by comparison. Also, many who are interested in unsupervised methods care first and foremost about application to specific language processing tasks rather than on the impact of unsupervised learning to our understanding of human acquisition, so performance of models on exactly this kind of data is critically important.

An important philosophical question is the extent to which one can learn anything without learning everything. This question is particularly relevant to the problem of POS tagging, in which the explicit goal is the inference of syntactic categories without an effort to induce the entire grammar. Several well known approaches to grammar induction take syntactic categorization as an input to the model, and attempt to learn the grammar with this starting point. Implicit in such approaches is the notion that unsupervised categorization without grammar induction is possible, which may well not be true. The same philosophical question applies to the question of whether it is possible to learn syntax without simultaneously learning semantics. Although many approaches, including ours, treat semantic themes as an obstacle and attempt to circumvent their effect to extract syntactic features, this approach may well be fundamentally flawed. Certainly, early in the process of human language acquisition, semantic learning dominates, with syntax coming much later. Of course modeling semantics is a daunting task, but it may be the case that unsupervised syntactic learning without attention to

semantics is destined to be very limited in its ability to succeed.

Despite all these reasons and more why unsupervised linguistic inference is challenging, we demonstrate in this dissertation that the previous state-of-the art approaches for unsupervised POS tagging still leave room for significant improvement. Our first result concerns the evaluation of these tagging models. Despite the large number of approaches that have been published over the past two decades, there has been no single accepted criterion for measuring performance. While a variety of approaches have been used, each has its own drawbacks or limitations. The source of the difficulty is twofold. First, unsupervised taggers have no access to the annotated tags created by the linguist. The categories that they create must be compared to these tags in a less direct way. If the tagging algorithms produced labels that correlated extremely well with the hand annotated tags, this process would be relatively simple, and it is likely that every criterion that has been used would yield consistent results. Unfortunately, this kind of close correlation has not been achieved, leaving the question of evaluation open.

Recently, three criteria have dominated the literature. As we began to explore the performance of our models for tagging, we quickly became aware of the potential danger involved in relying on two of these three criteria. From a mathematical viewpoint, the most desirable of these three is the information-theoretic metric known as variation of information, or VI. This metric produces a true distance between any two clusterings of a collection of data. It does not require any kind of map to be constructed between a model's labels and the linguist's POS tags, so it fits nicely within the framework of unsupervised tagging. Unfortunately, as we demonstrate, this criteria is extremely poorly

suited for comparing POS tagging methods that perform at or near the current state of the art. It scores the trivial clustering that emerges from the so-called most frequent word (MFW) baseline – which only attempts to capture the Zipfian nature of word-type distribution – a staggering 20% to 30% better than than current state of the art taggers. What's worse, it steadily improves the score as a model's ability to tag is crippled, as we demonstrate in a process we call decimation.

The two other commonly used criteria do require the construction of a map between the model's labeling and the hand annotated tags. Under the first criterion, one-to-one mapping or OTO, a greedy search is used to construct a map that allows at most one label to correspond to any tag. OTO is plagued by the same problems as VI (although they typically are less severe). In particular, there is a tendency to score genuine taggers slightly below the MFW baseline, which indicates that the labelings produced by these models are not "one-to-one," so that often several labels are dominated by words of the same POS tag. OTO is designed to severely punish this behavior, which results in a portrayal of these reasonable taggers as drastically unsound. In addition, the greedy search procedure used in OTO is dangerous, because it can produce two very different mappings with very different scores for two clusterings that are very similar. This property can cause erratic changes in performance, as a model's parameters are tuned. As a result, it is clear that relying on the OTO criterion to accurately compare two labelings is dangerous.

The third criterion also requires the construction of a map from the labels to the tags, but the mapping has no constraints. In particular, we are free to assign multiple

labels to the same tag (i.e. many-to one or MTO) if this benefits the accuracy score. This criterion is not without its own drawback. It is critical when using MTO not to compare two models unless they use the same number of labels, because using more labels will always improve a tagger's MTO score if the tagger is at all effective. Under MTO, there is no explicit penalty for using more labels, and taken to the extreme (when the number of labels matches the number of tokens in the corpus) a perfect MTO correspondence is possible. Under MTO, the unsupervised problem should properly be posed, "Find the unsupervised POS tagger that utilizes *k* labels so that the MTO accuracy is maximized." Without this restriction on the number of labels, comparison using MTO does not make sense.

On the other hand, given the tendency of models to produce labelings that are many-to-one in nature, this criterion ought to evaluate this kind of observed performance better than, for example, OTO. Accordingly, MTO accuracies of "informed" taggers are consistently higher than the "uninformed" MFW baseline, and we found no indication that the MTO criterion is unpredictable when decimating models. Quite on the opposite, MTO always indicated that steadily crippling a model steadily hurts its performance. For this reason, we encourage future work on the unsupervised POS tagging problem to rely on the MTO criterion, at least until the state of the art improves. If new evaluation criteria are developed (which would certainly be beneficial), we encourage that attention be paid to the kind of analysis described here, to ensure that these criteria behave in a manner that is consistent with the intuition of their users.

With a reliable evaluation criterion in hand, we turn our attention toward new

models for POS tagging that rely on dynamically self-organizing descriptor vectors to capture POS regularities. We present an algorithm that relies on SVD for dimensionality reduction and allows a gradual refinement of cluster assignments of the vectors, which in turn inform the dynamics of the vectors themselves. The simplest implementation of this process (SVD2) closely resembles the early work of Schütze from 1995, and we demonstrate that this simple approach is quite successful on the reduced tagset, but fails to significantly improve upon the best of the HMM's when the full tagset is used. Our ISVD implementation that takes full advantage of the power of the algorithm represents a significant advance in the state-of-the-art by achieving an impressive increase in MTO performance over the best HMM taggers for both the reduced and the full tagset. (We also obtain a similar improvement in accuracy under the other two most common criteria, but we obviously find this result less meaningful.)

Although accuracy is certainly the paramount goal, the huge reduction in computational complexity of this method (two to three orders of magnitude faster than various HMM implementations) is also impressive. The efficiency and flexibility of the algorithm makes it well-suited for producing fine-grained clustering of the data by using a large number of labels, in contrast to HMM's which do not scale well. By using more labels, each cluster becomes more homogeneous. The drawback is that a large number of generic labels may be less useful than a small number, depending on the application. However, we see this approach as valuable when paired with a post-processing step involving limited supervision. By querying an expert for the most likely POS tag of just one prototype word for each generic label, this procedure is able to produce a true POS

tagging that matches 80% to 90% of the tokens, depending on how many labels (and therefore how many expert queries) are needed for supervision.

Representing an entirely novel approach to modeling POS tagging, the Hebbian algorithm is a remarkably simple abstraction and approximation of several mechanisms in the brain. We relate the descriptor vectors to spatiotemporal patterns of neural activity, without committing to a particular type of pattern. We then describe a "Hebbian" mechanism that attracts vectors together to mimic the process of these underlying activity patterns becoming more similar due to Hebbian plasticity. We also include an orthogonalization mechanism that tends to keep random pairs of vectors apart, to avoid "over-generalization," in which vectors that do not belong together converge to the same point.

We implement these forces in an on-line algorithm that starts each vector in a random state, and allows it to change only when its type occurs in the corpus and only in response to its local context in that presentation. This algorithm is surprisingly successful at allowing the vectors to organize in a way that yields a very high MTO score for POS tagging. In fact, this algorithm duplicates the quantum leap demonstrated by ISVD on PTB17, and scores right in the middle of ISVD and the best of the HMM's on PTB45. Although this algorithm can, in principle, be used for disambiguation by allowing multiple vectors for each word type, we were unable to demonstrate that more vectors could improve the MTO score. We attribute this difficulty to several factors, including the lack of robustness of the model, and the fact that adding more vectors significantly increases the complexity of the model (i.e. the number of unknowns), while affording the

opportunity to improve the score only by the 8% (PTB45) of the tokens that are used with a tag that is not the most frequent for its type.

We conclude by describing a model for unsupervised inference of CFG's based on the same descriptor vectors and biologically inspired mechanisms as used for the Hebbian POS tagger; the operation of the model is illustrated on a number of "toy problems." This model relies on an auxiliary building block called a syntax brick, which allows the algorithm to construct a tree structure to describe any string of terminal symbols generated by the CFG. We show that this model can learn several simple CFG's after experiencing a relatively small number of example strings, compared to the total number of grammatical strings. Although this model is too simple to address real-size problems in natural language acquisition, it does demonstrate the power of this biologically motivated approach to self-organization, showing that it can capture some simple linguistic structure.

Although we connect the Hebbian POS tagging approach to a matrix factorization algorithm similar in spirit to the Generalized Hebbian Algorithm, our model actually evolved from a very different starting point. We began by working with a network of neurons and gradually moved from these neural-style models to progressively more abstract biological representations until reaching the implementation described in this dissertation. Although the early work produced some interesting preliminary results, it is not sufficiently mature for inclusion here. However, we anticipate that similar efforts will be the source of exciting future work as we seek to bridge the gap of abstraction between our model and a neural implementation.

Unsupervised inference in natural language processing is both challenging and important. Even on the distributional POS tagging problem in which the scope of linguistic structure being investigated is limited, state-of-the-art unsupervised algorithms are only capable of categorizing words in a way that produces modest agreement with the POS categories determined by linguists. It is likely that purely distributional approaches are inadequate for producing significantly better agreement than what we have demonstrated, so a paradigm shift toward models that build a more complete representation of the language may be required before unsupervised POS tagging accuracies improve significantly.

Supervised approaches, of course, do not suffer from the same difficulty, because their access to human annotated features allow them direct insight into non-distributional-based structure of the language. These methods are capable of exceptionally high accuracies, but are limited in their scope. Any natural language with scarce resources is unlikely to have the kind of annotated data required by a fully supervised method. Furthermore, these supervised approaches give little insight into the process of human language acquisition, which is arguably a mostly unsupervised process. Primarily for this reason, we see value in the continued study of unsupervised linguistic inference – particularly with an emphasis on biologically relevant models. Although these biologically inspired models may, for the foreseeable future, trail state-of-the-art algorithms in terms of performance, they can shed light on the process of human language acquisition and the neural mechanisms behind it.

References

Baum, L. 1972. An inequality and associated maximization technique in statistical estimation for probabilistic functions of a Markov process. *Inequalities* 3:1-8.

Beal, M. 2003. Variational algorithms for approximate Bayesian inference. *PhD. Thesis, Gatsby Computational Neuroscience Unit,* University College London.

Charniak, E.; Hendrickson, C.; Jacobson, N.; and Perkowitz, M. 1993. Equations for part of speech tagging. In *Proceedings of the Conference of the American Association for Artificial Intelligence* (AAA 1-93).

Church, K. 1988. A stochastic parts program and noun phrase parser for unrestricted text. In *Proceedings of the Second Conference on Applied Natural Language Processing, ACL*.

Clark, A. 2000. Inducing syntactic categories by context distribution clustering. In *Proceedings of CoNLL-2000 and LLL-2000.* Lisbon, Portugal.

Clark, A. 2003. Combining distributional and morphological information for part of speech induction. In *Proceedings of the Tenth Annual Meeting of the European Association for Computational Linguistics.*

Collins, M. 2002. Discriminative training methods for hidden markov models: Theory and experiments with perceptron algorithms. In *Proceedings of the ACL-2002 conference on empirical methods in natural language processing*. Volume 10.

Cover, T.; Thomas J. 2006. *Elements of information theory*, 2nd edition. John Wiley & Sons, Inc. Hoboken, NJ.

Cutting, D.; Kupiec, J.; Pedersen, J.; and Sibun, P. 1992. A practical part-of-speech tagger. In *Proceedings of the Third Conference on Applied Natural Language Processing,* ACL, Trento, Italy.

Deerwester, S.; Dumais, S.; Furnas, T.; and Harshman, R. 1990. Indexing by latent semantic analysis. *Journal of the American Society for Information Science* 41, 6, 391–407.

DeMarcken, C. 1990. Parsing the lob corpus. In *Proceedings of the 1990 Conference of the Association for Computational Linguistics.*

Derose, S. 1988. Grammatical category disambiguation by statistical optimization. *Computational Linguistics 14*.

Eckart, C. and Young, G. 1936. The approximation of one matrix by another of lower rank. *Psychometrika*. 1:211-218.

Elworthy, D. 1994. Does Baum-Welch re-estimation help taggers. In *Proceedings of the Fourth Conference on Applied Natural Language Processing, ACL*.

Gao, J. and Johnson, M. 2008. A comparison of Bayesian estimators for unsupervised hidden Markov model POS taggers. In *Proceedings of the 2008 Conference on Empirical Methods in Natural Language Processing*, pages 344–352.

Geman, S. and Geman, D. 1984. Stochastic relaxation, Gibbs distributions, and the Bayesian restoration of images. *IEEE Transactions on Pattern Analysis and Machine Intelligence,* 6(6).

Goldwater, S. and Griffiths, T. 2007. A fully Bayesian approach to unsupervised part-of-speech tagging. In *Proceedings of the 45th Annual Meeting of the Association of Computational Linguistics.*

Gorrell, G. 2006. Generalized Hebbian algorithm for incremental singular value decomposition in natural language processing. In *EACL 2006.*

Graca, J.; Ganchev, K.; Taskar, B.; and Pereira, F. 2009. Posterior vs. parameter sparsity in latent variable models. In *Proceedings of NIPS*.

Haghighi, A. and Klein, D. 2006. Prototype-driven learning for sequence models. In *Proceedings of the Human Language Technology Conference of the NAACL, Main Conference*, pages 320–327, New York City, USA, June. Association for Computational Linguistics.

Headden, W.; McClosky, D.; and Charniak. E. 2008. Evaluating unsupervised part-of speech tagging for grammar induction. In *Proceedings of the International Conference on Computational Linguistics (COLING 2008)*.

Jelinek, F. 1985. Self-Organized Language Modeling for Speech Recognition. In *Impact of Processing Techniques on Communication.*

Johnson, M. 2007. Why doesn't EM find good HMM POS-taggers? In *Proceedings of the 2007 Joint Conference on Empirical Methods in Natural Language Processing and Computational Natural Language Learning (EMNLP-CoNLL)*, pages 296–305.

Klien, D. and Manning, C. 2002. A generative constituent-context model for improved grammar induction. In *Proceedings of the Association for Computational Linguistics (ACL).*

Klien, D. and Manning, S. 2004. Corpus-based induction of syntactic structure: Models of dependency and constituency. In *Proceedings of the Association for Computational Linguistics (ACL)*.

Kupiec, J. 1992. Robust part-of-speech tagging using a hidden Markov model. *Computer Speech and Language 6.*

Lin, Y.; Chiang, T.; and Su, K. 1994. Automatic model refinement with an application to tagging. In *Proceedings of the 15th International Conference on Computational Linguistics.*

Lloyd, S. 1957. Least square quantization in PCM. *Bell Telephone Laboratories Paper.* Published in journal much later: Lloyd, S. 1982. Least squares quantization in PCM. *IEEE Transactions on Information Theory* 28 (2): 129–137. doi:10.1109/TIT.1982.

Marcus, G. F. 2000. Pabiku and Ga Ti Ga: Two mechanisms infants use to learn about the world. *Current Directions in Psychological Science*, 9, 145-147.

Marcus M.; Santorini B.; and Marcinkiewicz M. 1993. Building a Large Annotated Corpus of English: The Penn Treebank. *Computational Linguistics*, Volume 19, Number 2.

Meilă. M. 2003. Comparing clusterings by the variation of information. In *Bernhard Schölkopf and Manfred K. Warmuth, editors, COLT 2003: The Sixteenth Annual Conference on Learning Theory, volume 2777 of Lecture Notes in Computer Science*, pages 173–187. Springer.

Merialdo, B. 1994. Tagging English text with a probabilistic model. *Computational Linguistics*. Volume 20, Issue 2.

Murakami, J.; Yamatomo, H.; and Sagayama, S. 1993. The possibility for acquisition of statistical network grammar using ergodic HMM. In *Proceedings of Eurospeech* 93, pp. 1327–1330.

Reichart, R. and Rappoport, A. 2009. The NVI clustering evaluation measure. In *Proceedings of the Thirteenth Conference on Computational Natural Language Learning (CoNLL)*, pages 165–173.

Sanger, T. 1989. Optimal unsupervised learning in a single layer linear feed forward neural network. *Neural Networks*, vol. 2, pp. 459–473.

Schütze, H. 1995. Distributional part-of-speech tagging. In *Proceedings of the Seventh Conference on European chapter of the Association for Computational Linguistics*, pages 141–148.

Schutze, H. and Singer, Y. 1994. Part of speech tagging using a variable memory Markov model. In *Proceedings of the Association for Computational Linguistics*.

Smith, N. and Eisner, J. 2005. Contrastive estimation: Training log-linear models on unlabeled data. In *Proceedings of the 43rd Annual Meeting of the Association for Computational Linguistics (ACL)*, pages 354–362.

Smith B.; Boyle, J.; Dongarra, J.; Garbow, B.; Ikebe, Y.;Klema, V.; and Moler. C. Matrix eigensystem routines. *EISPACK Guide, Lecture Notes in Computer Science*. Volume 6. Springer-Verlag, New York, 1976.

Steinhaus, H. 1956. Sur la division des corps matériels en parties (in French). *Bull. Acad. Polon. Sci.* 4 (12): 801–804.

Stewart, G. 1992. On the early history of the singular value decomposition. UMIACS-TR-92-31, CS-TR-2855. Also, appeared in *SIAM Review*, 35 (1993) 551-566.

Toutanova, K.; Klein, D.; Manning, C.; and Singer, Y. 2003. Feature-rich part-of-speech tagging with a cyclic dependency network. In *Proceedings of HLT-NAACL 2003*, pages 252-259.

Wall, M.; Rechtsteiner, A.; and Rocha, L. 2003. Singular value decomposition and principal component analysis. In *A Practical Approach to Microarray Data Analysis*. D.P. Berrar, W. Dubitzky, M. Granzow, eds. pp. 91-109, Kluwer: Norwell, MA (2003). LANL LA-UR-02-4001.

Weischedel, R.; Meteer, M.; Schwartz, R.; Ramshaw, L.; and Palmucci, J. 1993. Coping with ambiguity and unknown words through probabilistic models. *Computational Linguistics.*

Yeung, M.; Tegner, J.; and Collins, J. 2002. Reverse engineering gene networks using singular value decomposition and robust regression. In *Proceedings of the National Academy of Science* 99, 6163–6168.