MIXTURE MODELS FOR NATURAL LANGUAGE PROCESSING

Robert G. Thomas III

A DISSERTATION
PRESENTED TO THE FACULTY
OF PRINCETON UNIVERSITY
IN CANDIDACY FOR THE DEGREE
OF DOCTOR OF PHILOSOPHY

RECOMMENDED FOR ACCEPTANCE
BY THE DEPARTMENT OF
COMPUTER SCIENCE

June 1998

Abstract

Building computer systems that interact with humans via natural language has long been a goal of computer scientists. Although such systems may be ubiquitous by the 21st century, there are many problems facing computer scientists today that must be solved for such a vision to be realized. This thesis focuses on two such problems: natural language text modeling and phonetic baseform generation.

The goal of natural language text modeling is to characterize the statistical structure of natural language text. This goal is tantamount to constructing a statistical model that assigns probabilities to text strings that accurately reflect their likelihood of occurring. We present a theoretical and empirical analysis of four novel model classes for natural language text as well as an examination of the ramifications of modeling choices such as model order, parameter tying scheme, and withholding algorithm. Our work shows that a new model class, the hierarchical nonemitting model[46], is both empirically and theoretically superior to the widely-used classic model proposed by Jelinek and Mercer.[27] We also demonstrate that the use of higher-order models, frequency and codelength-based tying schemes, and k-fold cross-estimation lead to superior models.

Phonetic baseform generation is the process of generating a phonetic representation, known as a baseform, for arbitrary orthographic strings. We propose a new model, the hidden alignment model, that is based directly upon the model proposed by Yvon. [61] Our evaluation demonstrates that Yvon's model is superior from an empirical perspective. We explore reasons for why this is true and offer alternative approaches that may strengthen the model class.

Acknowledgments

Many people have influenced my life and work. I'd like to first thank my advisor Eric Ristad whose support, guidance, and friendship have made my years at Princeton very worthwhile. I consider myself incredibly lucky to have had the privilege of learning from such an insightful and hard-working individual as Eric. Much of the work reported in this thesis is the direct result of collaborating with Eric. I'd also like to thank the people at IBM TJ Watson Research Center who graciously allowed me to spend two summers learning about statistical models. In particular, I'd like to thank Lalit Bahl, Salim Roukos, Todd Ward, Harry Printz, Martin Franz, and Carl de Marcken (MIT). They each helped me develop a deep appreciation for the field.

People at Princeton have always been a great support to me. I have fond memories of wasting time with (in alphabetical order) Amar Andalkar, Ross Bunker, Sharon Chen, Leonard Cohen, Saud Hashimi, Jason Hildreth, Helen Hu, Ed Lau, Ken Mahieu, Jon Mcallister, Kyle Weston, Masi Yamada, and Mike Yang (and others!) as an undergraduate and Yuqun Chen, Drew Dean, Chris Dunworth, Stef Damianakis, Patrick Min, Mukund Raghavachari, Hongzhang Shan, Kevin Skadron, and Dan Wallach as a graduate student. Princeton is truly a wonderful place to learn, teach, and have fun.

I'd like to thank my two readers, Andrew Appel and Perry Cook, and my two non-readers, Ed Felten and Andrea La Paugh, for providing excellent advice along the way. Their suggestions significantly enhanced my thesis. Of course, any remaining errors are due to me alone.

Finally I'd like to thank my family. My wife Heather has been a great source of support throughout my time at Princeton. I don't know, and don't want to know, how my life would be different without her. I have also benefited immensely from my parents' love and the value they placed upon an education. My father instilled a love of problem solving, computers and programming in me at a very early age. For this, I am eternally grateful.

Contents

Chapter 1

Introduction

Many people feel that their lives have been improved by technologies that simplify complex computations, minimize the need for human intervention in mundane or dangerous tasks, and even expand their social and financial horizons. In recent years, for example, technologies such as the World Wide Web have made the visions of world-wide electronic commerce and publishing a reality.

Automatic natural language processing, that is, building hardware and software systems that allow the computer-human interface to be governed by natural language, is another application that has substantial potential to improve our lives. The primary source of its potential resides in the observation that the current form of human-machine interaction is inefficient, limiting, unnatural, and for some, unhealthy. The vast majority of people currently interface with their personal computer or workstation using a keyboard and a mouse. This paradigm is inefficient as many people can not type as quickly as they can speak. Allowing people to speak, rather than type, to their computer would greatly increase their productivity.

The current interface also limits the scope of computers. Computer systems could replace many human operators with the advent of successful natural language processing systems. For example, it would possible to place phone calls based upon speech if it were possible to generate phonetic representations for arbitrary proper names and recognize the speech of callers. More advanced applications like automatic voice reservation systems would become viable with the additional capability of limited-domain semantic processing.

In this thesis, we focus on the construction of computationally efficient, probabilistic models that serve as the foundation for many of these natural language processing systems. The remainder of this chapter will provide a high-level discussion of the problems examined in this thesis, motivating the importance of these problems by considering relevant applications.

1.1 Applications

A model of natural language text that is able to assign accurate probabilities is important in many real world applications. We discuss three here: speech recognition, text compression, and spell checking. While these three applications are extremely useful and important, other applications may also require an accurate natural language text model, e.g. text routing/classification, predicting user input[5], and automatic user customization.

We also introduce the problem of generating phonetic representations, known as baseforms, for proper names. Successful models of baseforms would allow such applications as voice-dialing to be realized.

Speech Recognition

The goal of speech recognition is to transcribe an acoustic signal to the intended text. While the acoustic signal carries information pertinent to the transcription task, other knowledge about the real world and natural language is also vital. In fact, recognition systems that perform the transcription conditioned on only the acoustic signal will likely have poor performance. Dismal performance is likely because the acoustic signal is either provably ambiguous, e.g. *to* vs. *too* vs. *two*, or is practically ambiguous due to microphone/background noise or sloppy pronunciation by the speaker. For example, the string `The Dow dropped by two points yesterday` is acoustically ambiguous. A speech recognizer based on acoustic evidence alone may assign equal probability to each of the following strings:

1. The Dow dropped by too points yesterday
2. The Dow dropped by to points yesterday
3. The Dow dropped by two points yesterday

In practice, accurate models of natural language text are combined with acoustic signal models to generate the transcription. From a high level point of view, natural language text models can be seen as a scorer for the hypotheses generated by the acoustic model. Thus, the speech recognizer would be able to properly choose `The Dow dropped by two points yesterday` from the earlier example because the natural language text model should assign it high probability. A more formal description of exactly how the scores are combined is given in the next chapter.

Text Compression

Text is stored in files and transferred over networks and modems. Both of these cost money – disk space for the files and bandwidth costs for the transfer of information. With disks and networks becoming bigger, cheaper, and faster one may think that the future of text compression is becoming less important. While this belief may be true to a degree, it is also is true that people push the envelope: always wanting to store or send more information.

Text compression can be viewed as a two-stage process, (1) building a model of the text and (2) encoding the text relative to the model. The second stage of the process has been "solved." Arithmetic coding [1, 24, 35, 39] allows the encoding of the message to approach arbitrarily close to the entropy of the text conditioned upon the model. Thus, the real challenge is to construct a good model of the text. The primary property of a good model of text is that it encodes messages using fewer bits than other possible models, implying that it has a better representation of the structure of the underlying text. Consider the extreme case: if a model could determine, with certainty, which piece of text was to be transmitted it would not need to be transmitted. Thus, any improvement in being able to model natural language text strings can be directly translated into an improved text compression system.

Spell Checking

The task of discovering misspellings has been greatly simplified in the past couple of decades by the advent of spell checking functions for word processors and text editors. A typical spell checker's definition of a misspelling is a token that is not in its dictionary. Other errors such misspelling an intended word such that the misspelling happens to be in the dictionary, omitting a word, or reversing the placement of words are often not brought to the attention of the user. This inadequacy of spell checkers can be partially resolved through the use of natural language text models. Since errors of the aforementioned form lead to unlikely word sequences, incorporating natural language text models into the spell checking programs would allow these revised programs to highlight the unlikely words to users.

Automatic Dialing System

Generating accurate baseforms, phonetic representations, for arbitrary strings is important for applications such as speech synthesis and voice-initiated dialing. A voice-initiated dialing system must be able to match the user's utterance with the closest telephone entry. Since speech recognition vocabularies are small, e.g. 20,000 words, it is often not possible to compare the text output of a speech recognition system to the text entries in the phone book since most of the phone book entries are not contained in the speech recognizer vocabulary. Instead, it is usually necessary for the system to compare phonetic representations of the dictionary entries to the speech utterance, and these phonetic representations must be generated automatically. If a superior baseform generation model is constructed, the accuracy of a voice dialing system would be improved.

1.2 Problem Descriptions

1.2.1 Natural Language Text Modeling

When two humans that speak the same natural language communicate with one another, they have excellent models of what the other is going to say. From a probabilistic view, we would say that they have an accurate probabilistic model of the particular natural language. A probabilistic model ϕ of natural language text assigns any message M drawn from a finite alphabet of atomic units A a probability $P(M|\phi, S, A)$ representing the degree to which the model believes that M could have been generated by the source S, i.e. the speaker. Other variables such as time and topic could also be used to condition the prediction.

Any software system that seeks to understand, generate, or transcribe natural language text must have some model of that language. We hypothesize that the model must be probabilistic in nature because representations of the degree to which something is possible seem pertinent. Researchers have yet to develop a full computational model of natural language that assigns accurate probabilities to natural language text. In general, restricted probabilistic finite-state automatons are utilized to compute the probabilities.

These restricted finite-state models are provably too weak to model all language phenomenon. On the positive side, they capture local regularities well and massive amounts of natural language text are available to estimate the free parameters of these models. The *Wall Street Journal* from 1987-1994 is currently available for a fee as are numerous other text corpora, including World Wide Web content and Usenet articles. As a result, researchers often decide to compromise model strength for restricted finite-state models that are theoretically well-understood and can be estimated in polynomial time.

In chapter two, we propose four novel finite-state mixture models that expand upon the classic mixture Markov model proposed by Jelinek and Mercer in 1980. The models we consider improve upon the classic mixture model in either a theoretical or empirical sense. The success of these models lie in their novel uses of the mixture parameters. In addition, we consider the problem of how to properly construct mixture models from finite data. We perform and analyze a large collection of experiments that shed light on the impact of modeling decisions such as model class, parameter tying, model order, and withholding algorithm. We now give a brief description of each of these choices.

Model Class

A model class is a formal description of a particular set of models: the number and type of model parameters as well as how these parameters are utilized. The choice of model class is probably the single most important decision made by the modeler, for it represents the assumptions made about the source by the modeler. As noted, we introduce several novel model classes and establish the empirical and theoretical characteristics of these models. Our results demonstrate that some of the new models classes are empirically and theoretically superior to the classic bigram and trigram models.

Parameter Tying

Parameter tying is a technique that reduces the number of free parameters in the model. The central motivation behind parameter tying is the fundamental tension that exists between model size and the model's ability to account for a set of data. A model with too few parameters is unable to capture important characteristics of the data while a model that has too many parameters has too many degrees of freedom, making it difficult to generalize from the data. Our results show that frequency-based and codelength tying schemes are to be preferred.

Model Order

Model order defines the number of symbols used to condition symbol predictions. Researchers have traditionally used bigram and trigram models because they did not have access to enough memory or CPU power to support higher order models. The continuing decline in memory and CPU prices motivates the utilization of higher order models. Our empirical results show that higher order models should be used if the computational resources can be afforded.

Withholding Algorithm

A withholding algorithm is used to circumvent the problem of estimating model parameters from a finite collection of text. Since any finite collection of text is a *sample* of the natural language, the modeler can not be certain that the string statistics of the sample are representative of the language as a whole. To combat this problem, researchers typically divide the collection of text into two sections – one from which the string statistics are gathered and the other is used to determine which string statistics are reliable. Two different withholding algorithms are presented: forward estimation withholding and k-fold cross-estimation withholding. The empirical results show that the application of the k-fold cross-estimation withholding algorithm consistently results in superior natural language text models. Nonetheless, forward estimation withholding is nearly as good and can often be used as a substitute to k-fold cross-estimation.

1.2.2 Phonetic Baseform Generation

A baseform is a phonetic representation of an orthographic string. Most baseform generation models are comprised of hand-generated rules, or are non-probabilistic in nature. Hand-generated models are time-consuming endeavors and they must be regenerated for each language. Most non-probabilistic models, on the other hand, acquire and produce baseforms automatically, thereby eliminating the need for human intervention beyond the programming stage. Yet, hand-generated models often give superior performance to models that automatically acquire their knowledge from some training data.

The third chapter of this thesis develops a probabilistic model of baseform generation. Partially motivated by the substantial successes of probabilistic models in other natural language processing domains, the model places a successful non-probabilistic model developed by Yvon[61] in a probabilistic framework. The model, referred to as the hidden alignment model, generates baseforms by analogy: utilizing large, overlapping substring matches between the given word and words in a training dictionary of ASCII strings aligned with their baseform. The aligned dictionary used in this thesis was constructed at IBM TJ Watson Research Lab and consists of proper names of people and places. The formal characterization of the problem requires a substantial number of definitions and will therefore be postponed until the third chapter.

Although we believe probabilistic models of baseform generation have the potential to outperform non-probabilistic and hand-generated ones, the hidden alignment model substantially underperforms Yvon's non-probabilistic algorithm. We investigate why this is true and offer advice on how to improve the model.

1.3 Mixture Models

The models considered in this thesis are restricted finite-state mixture models. A mixture model is a model that constructs a distribution by combining, or mixing, a set of distributions. For example, suppose n distributions were defined, $\delta_1, \delta_2, \ldots \delta_n$ such that δ_k conditioned the prediction of the next symbol on the previous k symbols. A new distribution δ could be defined such that $\delta = \kappa_1\delta_1 + \kappa_2\delta_2 + \ldots + \kappa_n\delta_n$ where the mixture parameters κ_i satisfy $\kappa_1 + \kappa_2 + \ldots + \kappa_n = 1$.

Mixture models are used to minimize the impact of distributions estimated from insufficient data as the model is not certain that any particular distribution is better than another based upon the data. Rather, the model has imperfect evidence from which to estimate the degree to which a particular distribution should be utilized. Thus, a mixture model can be viewed as blending distributions together rather than choosing one particular distribution. The notion of blending is surprisingly powerful as it allows models to handle novel situations in a reasonable manner. This ability to deal with novel events is discussed extensively in the next chapter.

1.4 Thesis Outline

This chapter is followed by an in-depth, technical study of the models used for natural language text. As noted, we propose four novel mixture models, examining their theoretical and empirical strengths relative to the popular models currently used in the field. Our results show that the hierarchical nonemitting model is strictly superior to the classic model in both theoretical and empirical evaluations. We also demonstrate that higher order models constructed with frequency or codelength-based tying schemes using the k-fold cross estimation withholding scheme perform the best.

The third chapter examines the problem of generating a phonetic baseform for an arbitrary orthographic string. The hidden alignment model is outlined and compared to a nonstochastic model proposed by Yvon. [61] The empirical evaluation of the models shows that Yvon's model is better than the hidden alignment model. We discuss why this is true and offer suggestions for future research.

The concluding chapter reiterates the results presented in chapters two and three, offering a more general overview of this work.

Chapter 2

Natural Language Text Modeling

2.1 Introduction

Most human beings have the ability to generate and understand a complex weave of external signals commonly referred to as natural language. Although many animals have some form of communication, humans appear to have the most highly developed system in the animal kingdom. As a result, scientists from diverse academic backgrounds are focused on developing a model of the machinery that allows humans to communicate with one another. For some, the question of how this machinery developed is of the utmost importance as it lends evidence to the precise way in which the human brain functions. Computer scientists, however, are primarily interested in constructing software systems that allow computers to understand and generate natural language in order to simplify user interfaces and expand the realm of problems for which computers can be used. Thus, the models proposed by computer scientists tend to be engineering abstractions that are not necessarily accurate theories of natural language or of the human brain.

The most common external manifestations of the internal generative process of language are text and speech. In addition to text and speech, psychological experiments and intuition are utilized by researchers to guide their understanding of the generative process. In this chapter, we explore the use of large probabilistic mixture models for modeling natural language text.

We investigate the utility of alternative interpretations of mixture parameters. These alternative interpretations manifest themselves in novel model classes that outperform, in both a theoretical and empirical sense, some of the most popular models used in the field.[27] In addition, we explore the impact of withholding algorithm, parameter tying scheme, and model order for both character and word-based models, over a large number of experimental situations. These experiments strongly suggest that the modeler must be attentive to detail as many modeling decisions can have a significant impact on the model's predictive acumen.

Although finite order mixture models are provably incapable of capturing certain linguistic phenomenon like anaphora[37], a successful model is, nonetheless, able to assign accurate probabilities to many strings drawn from the natural language. Such a model can be extremely useful for real-world applications such as speech and handwriting recognition, spelling correction, and natural language text compression.

2.2 Problem Definition

The goal of modeling natural language text stochastically is to construct a model that assigns accurate probabilities to strings drawn from a particular natural language. Before proceeding, we formalize some important concepts. A natural language string x is drawn from a finite alphabet A. A stochastic natural language model ϕ is drawn from some model class $\mathcal{M}$. A model class $\mathcal{M}$ is a formal description of a particular set of models: the number and type of model parameters as well as how these parameters are utilized. The free parameters of the natural language model ϕ are usually estimated from a finite string, $O = o_1 \ldots o_T$ referred to as the training corpus. A reasonable parameter estimation algorithm takes a polynomial number of atomic computational steps with respect to the size of the training corpus and the number of free parameters in the model.

The predictive quality of two models, ϕ and ϕ', can be compared by calculating the cross-entropy of a very long representative string of the language $Y = y_1 \ldots y_M$ with respect to these models. The cross-entropy of a string Y with respect to ϕ is given by:

$$H(Y \mid \phi) = -\log_2 \ P(Y \mid \phi) \tag{1}$$

The cross-entropy of a message defines the number of bits needed to encode the message relative to some model. $H(Y \mid \phi) < H(Y \mid \phi')$ implies that ϕ is a better model of Y because ϕ betters models the "statistical structure" of Y. [57] If Y is a very long representative string of the natural language, $H(Y \mid \phi) < H(Y \mid \phi')$ also strengthens the assertion that ϕ is better than ϕ' at modeling the particular natural language. This comparison is possible due to the fundamental relationship that exists between maximizing the probability assigned to a particular string and minimizing the number of bits needed to encode that string. If we construct a model that assigns probability $\mathcal{P}$ to the string Y, we can encode that string, relative to the model, using $-\log_2 \mathcal{P}$ bits.[56] Conversely, if we can encode Y using $\mathcal{B}$ bits relative to some model, that model ϕ can be thought of as setting:

$$P(Y|\phi) = 2^{-\mathcal{B}} \tag{2}$$

Thus, striving to encode Y with fewer bits is equivalent to increasing the probability the model assigns to Y.

The goal of this chapter is to develop large finite-state mixture models that assign high probability to representative strings of English.

2.3 Model Class Definitions

This section focuses on the definition of the various model classes considered in this chapter. The basic Markov model class is first defined which helps to motivate the classic, nonemitting, and nonuniform model classes. The formal definition of the basic and mixture models will be coupled with a presentation of the string generation, string probability evaluation, and parameter estimation algorithms.

2.3.1 Basic Markov Models

The first step in deriving a Markov model is to note that the probability that any model, Markov or not, assigns to a string x_1^M, $P(x_1^M \mid \phi)$, can be re-written as a product of conditional probabilities:

$$P(x_1^M \mid \phi) = \prod_{t=1}^{M} P(x_t \mid x_1^{t-1}, \phi) \tag{3}$$

If a set of distributions is defined for each possible history x_1^{t-1}, it is possible to compute the probability of an arbitrary string by multiplying the probabilities of the appropriate conditional distributions. These distributions are rarely known as a matter of fact and often must be estimated from a training corpus of text. Because this training corpus is finite, there is no guarantee that the corpus accurately reflects the underlying source. For example, some strings that could occur in the language will not occur simply because the training corpus is a *sample* of the particular natural language. The likelihood that a particular valid string will not be witnessed in the training corpus grows as the length of the string grows. Consider a corpus of length T: the number of strings of length n in this corpus is at most $T - n + 1$. Of course, the number of strings drawn from A of length n grows as $|A|^n$. In general, $|A|^n >> T - n + 1$ and therefore it is often impossible to accurately construct distributions conditioned on long strings because there is scant evidence from the training corpus to induce such distributions. As a result, it is generally necessary to introduce a function, h, that maps the entire history to its n-symbol suffix. n is chosen by the modeler with the belief that there is enough evidence to construct accurate distributions conditioned on the previous n symbols.

A Markov model assumes that the future symbol only depends on the past n symbols. Thus, the Markov mapping function is defined as:

$$h : x_1^t \rightarrow x_{t+1-n}^t \tag{4}$$

We can then rewrite the product of conditional probabilities as:

$$P(x_1^M \mid \phi) = \prod_{t=1}^{M} P(x_t \mid h(x_1^{t-1}), \phi) \tag{5}$$

This dependency assumption, know as an order n Markov assumption, allows models to be built that can be estimated well and require feasible amounts of memory because there is adequate data to estimate the conditional distributions of the model.

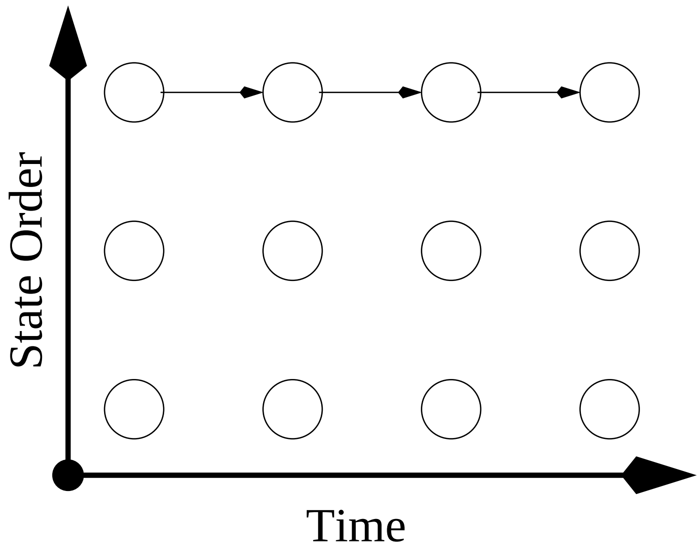

Figure 1: A graphical depiction of the basic Markov model. The probability assigned to a string is completely dependent on the highest order state applicable at each step.

A basic Markov model $\phi = \langle n, A, \delta \rangle$ consists of a model order n, a finite alphabet A, and a state output probability function $\delta : A^n \times A \rightarrow [0, 1]$. The model order dictates how many symbols in the past sequence condition the prediction. The state output probability function specifies the probability of a particular symbol given the past n symbols. Please examine figure 1 for a graphical representation of the model. A basic Markov model can be used to calculate the probability of a string x_1^M in the following manner, similar to equation 5:

$$P(x_1^M \mid \phi) = \prod_{t=1}^{M} \delta(x_t \mid x_{t-n}^{t-1}, \phi) \tag{6}$$

While equation 6 is technically sound, it is necessary to ensure that $\delta(x_i \mid x_{t-n}^{t-1}) > 0$ for all x_t that are possible continuations of x_{t-n}^{t-1}. Otherwise, the model will assign zero probability to a valid string in the language. Encoding a message that occurs with zero probability requires an infinite number of bits. $\delta(x_t \mid x_{t-n}^{t-1})$ may equal zero because the parameters are usually estimated from a finite text corpus. Typically, frequency information gathered from this corpus governs the estimation process. Thus, a string x_{t-n}^{t-1} could occur zero times in the corpus and still be a possible string of the language. The simplest solution to the zero-frequency problem is to modify the δ function directly by forcing all probabilities to be greater than zero. For example, one could simply treat each event as occurring once more than it actually did. This method is referred to as Laplace's Law of Succession.[32]

2.3.2 Backoff Models

An alternative solution to the zero-frequency problem is to allow the model to "back off" to lower order states when a higher order state has not witnessed a particular event. [12, 30, 41, 34] Such a model is called, not surprisingly, a backoff model. Backoff models are attractive because they are computationally efficient. The parameters of a backoff model are usually estimated directly from the frequency statistics of the training text. The lack of an expensive estimation procedure allows for online adaptation to be performed quickly. A backoff model $\phi = \langle n, A, \delta \rangle$ consists of a model order n, a finite alphabet A and a state output probability function $\delta : A^n \times A \to [0, 1]$ where

$$\delta(\sigma|x_1^m, \phi) = \begin{cases} f(\sigma, x_1^m, \phi) & \text{if } c(\sigma|x_1^m) > 0 \\ \alpha(\sigma, x_1^m, \phi)\delta(\sigma|x_2^m) & \text{otherwise} \end{cases} \tag{7}$$

where f is some function that determines the probability with which σ follows x_1^m contingent upon the fact that σ has followed x_1^m in the training corpus, α is the amount of probability mass to be redistributed to the unseen event, and $c(\sigma|x_1^m)$ is the number of times σ followed x_1^m in the training corpus. With δ defined, we can compute $P(x_1^M|\phi)$ using equation 6.

2.3.3 Mixture Markov Models

The basic Markov model assigns probability to the next symbol x_t conditioned on the n-symbol suffix x_{t-n}^{t-1}. The backoff model assigns probability to the next symbol x_t based on the longest suffix, x_{t-j}^{t-1} such that $0 \le j \le n \wedge c(x_t|x_{t-j}^{t-1}) > 0$. A different approach is to condition the current prediction on x_{t-n}^{t-1} and the n proper suffixes of x_{t-n}^{t-1}. A mixture Markov model combines information from $n+1$ separate conditional distributions to determine $P(x_t \mid x_{t-n}^{t-1}, \phi)$. [27, 10, 26, 36, 46, 50, 38, 33] Formally, a mixture Markov model assumes:

$$P(x_t \mid x_1^{t-1}, \phi) \approx \sum_{i=0}^{n} \kappa_i P(x_t \mid x_{t-i}^{t-1}, \phi)$$

such that[1]

$$\sum_{i=0}^{n} \kappa_i = 1$$

A mixture parameter is associated with each state and it dictates the degree to which that state is used to condition the prediction. This idea leads to the formal definition of the mixture Markov model. A mixture Markov model $\phi = \langle n, A, \delta, \lambda \rangle$ consists of a model order n, a finite alphabet A, a set of state output probability functions $\delta = \{\delta_i : 0 \leq i \leq n\}$ for each state order such that $\delta_i : A^i \times A \rightarrow [0, 1]$, and a set of mixture parameter functions that define the mixture parameters of ϕ, $\lambda = \{\lambda_i : 0 \leq i \leq n\}$ such that $\lambda_i : A^i \rightarrow [0, 1]$. Under this paradigm, the δ functions are usually the empirical distributions of the training corpus, and the λ parameters are adjusted so as to maximize the performance of the model on a withheld data set. The maximization process is carried out with the expectation-maximization algorithm.[2] The mixture parameters define the degree to which a particular history is used to predict the next symbol. The mixture Markov model differs from the basic Markov model in that multiple histories are used to assign probability to the next symbol whereas the basic Markov model uses only one history.

We consider five mixture Markov models in this chapter: the classic model[27], the hierarchical nonemitting model[46], the general nonemitting model, the general nonuniform model, and the collapsed nonuniform model.[50] The classic mixture Markov model was proposed by Jelinek and Mercer in 1980. The remaining four models were devised by Ristad and Thomas. These models differ significantly from one another in how they utilize the interpolation parameters, the number of interpolation parameters in the model, and how symbols are predicted at each step. The

[1] To ensure no event is assigned zero probability, some modelers include a order -1 state that assigns each symbol uniform probability. Alternatively, one could simply add a small probability to each event from the order zero state. In addition, we assume that $0 > \kappa > 1$.

[2] This estimation process will be discussed in section 2.4.4

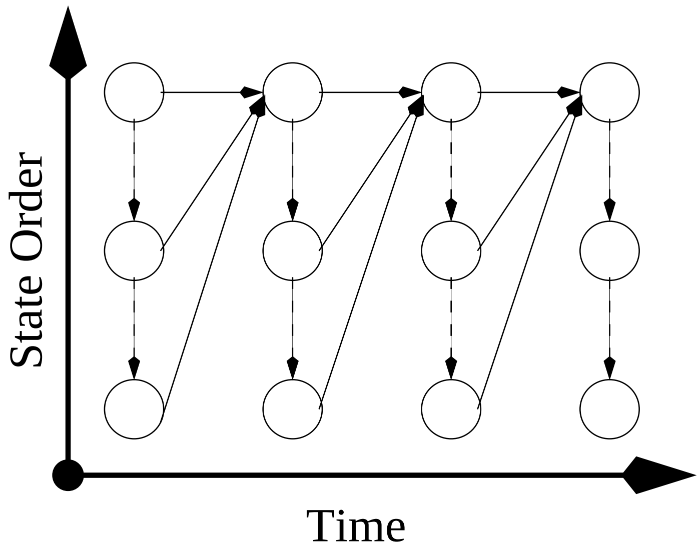

Figure 2: A graphical depiction of an order two classic Markov model. The dashed lines represent intra-step blending while the inter-step lines represent the emission of a symbol along with a transition to a new state. Notice that the classic model's inter-state transitions always end at the highest order state.

models are now discussed individually by examining the probability that the model utilizes a particular context and the probability that a symbol is generated at a chosen context.

2.3.4 Classic Model

Jelinek and Mercer view the mixture parameters as dictating the degree to which the probability assignment of a higher order state is smoothed with that of a lower order state. The probability that the first $t-1$ symbols are generated, and the particular string x_{t-i}^{t-1} is used at step t to condition the prediction is given by:

$$P(x_1^{t-1}, s_t = x_{t-i}^{t-1} \mid \phi) = P(x_1^{t-1}|\phi)\lambda(x_{t-i}^{t-1}) \prod_{k=i+1}^{n} (1 - \lambda(x_{t-k}^{t-1})) \tag{8}$$

where s_t denotes the state at step t and $P(x_1^{t-1}|\phi)$ is given by:

$$P(x_1^{t-1}|\phi) = \sum_{i=0}^{n} P(x_1^{t-2}, s_{t-1} = x_{t-1-i}^{t-2}|\phi)\delta(x_{t-1}|x_{t-1-i}^{t-2}, \phi) \tag{9}$$

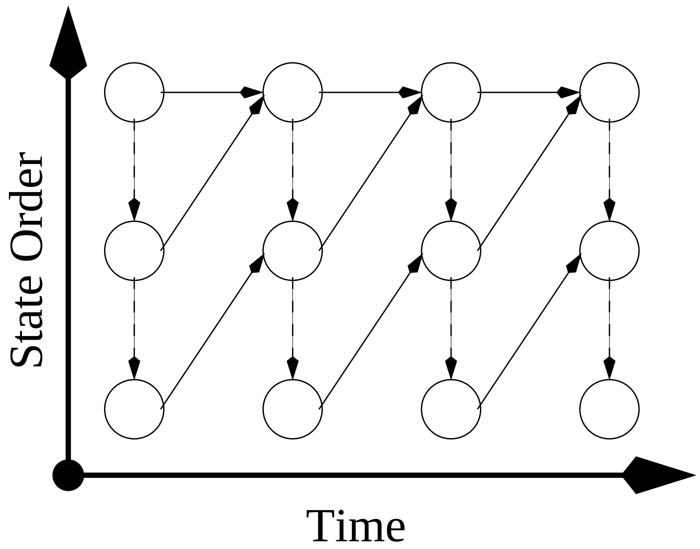

Figure 3: A graphical depiction of an order two hierarchical nonemitting Markov model. The hierarchical nonemitting model has inter-step transitions from states of order i to states of order $\max(n, i+1)$. Contrast this with the classic model where the inter-step transitions from the state at step t always terminate at the state of order n at step $t+1$.

The probability of using a particular state is comprised of three factors: (1) $P(x_1^{t-1}|\phi)$, the probability of reaching the t'th step, (2) $\lambda(x_{t-i}^{t-1})$, the degree to which model believes the string x_{t-i}^{t-1} should be used to condition the prediction instead of some shorter proper suffix of x_{t-n}^{t-1} and (3) $\prod_{k=i+1}^{n}(1-\lambda(x_{t-k}^{t-1}))$ represents the probability that none of the longer suffixes of x_{t-i}^{t-1} were used for conditioning purposes.

The probability that a particular symbol σ is generated at step t from state x_{t-i}^{t-1} is given by $\delta(x_t \mid x_{t-i}^{t-1}, \phi)$. Refer to figure 2 for a graphical depiction of the classic model's use of the mixture parameters.

2.3.5 Hierarchical Nonemitting Model

The hierarchical nonemitting Markov mixture model utilizes the λ parameters to dictate the likelihood of making a nonemitting transition. A hierarchical nonemitting transition is a intra-step transition at step t from the state x_{t-i}^{t-1} of order i to the state $x_{t-(i-1)}^{t-1}$ of order $i-1$ that is not accompanied by the emission of a symbol. A hierarchical nonemitting transition is permanent in nature implying that only states of order $\min(n, (i-1)+1)$ can be used for conditioning at step $t+1$. Contrast this with the classic model where inter-step transitions at step t do not impact the consideration of conditioning strings at step $t+1$. Figure 3 gives a graphical realization of the hierarchical nonemitting model.

The model is hierarchical in the sense that there is an implicit hierarchy defined over the set of conditioning strings. The state $x_{t-(i-1)}^{t-1}$ of order $i-1$ will be used for conditioning if the state x_{t-i}^{t-1} of order i was passed over or the order $i-2$ state at step $t-1$ transitions to the order $i-1$ state at step t with non-zero probability.

The probability that the first $t-1$ symbols are generated and a particular string x_{t-i}^{t-1} at step t is used to condition the prediction is given by (assuming $n \geq 1$):

$$
\begin{aligned}
&P(x_1^{t-1}, s_t = x_{t-i}^{t-1} \mid \phi) = \\
&\begin{cases}
\lambda(x_{t-i}^{t-1})[P(x_1^{t-2}, s_{t-1} = x_{t-1-n}^{t-2} \mid \phi)\delta(x_{t-1} \mid x_{t-1-n}^{t-2}, \phi) + \\
\qquad P(x_1^{t-2}, s_{t-1} = x_{t-1-(n-1)}^{t-2} \mid \phi)\delta(x_{t-1} \mid x_{t-1-(n-1)}^{t-2}, \phi)] & i = n \\
\lambda(x_{t-i}^{t-1})[P(x_1^{t-1}, s_t = x_{t-(i+1)}^{t-1} | \phi)\frac{1-\lambda(x_{t-(i+1)}^{t-1})}{\lambda(x_{t-(i+1)}^{t-1})} + \\
\qquad P(x_1^{t-2}, s_{t-1} = x_{t-1-(i-1)}^{t-2} \mid \phi)\delta(x_{t-1} \mid x_{t-1-(i-1)}^{t-2}, \phi)] & 0 > i > n \\
P(x_1^{t-1}, s_t = x_{t-(i+1)}^{t-1} | \phi)\frac{1-\lambda(x_{t-(i+1)}^{t-1})}{\lambda(x_{t-(i+1)}^{t-1})} & i = 0
\end{cases}
\end{aligned}
\tag{10}
$$

If $i = n$, there are two transitions that lead to the order n state at step t. Both the order n and order $n-1$ states at step $t-1$ may emit a symbol and transition to the order n state at step t. These dual transitions are reflected in equation 10. There are also two transitions leading to the state of order i if $0 > i > n$. One transition originates in the order $i-1$ state at step $t-1$. The other other corresponds to being in the order $i+1$ state at step t and deciding to make a nonemitting transition to the order i state at step t. Finally, there is only one way to reach the order zero state at step t, being in the order 1 state and deciding to make a nonemitting transition to the order zero state.

The probability that the hierarchical model assigns to a symbol, σ, in the state x_{t-i}^{t-1} is given by: $\delta(\sigma \mid x_{t-i}^{t-1}, \phi)$.

The hierarchical model's structure allows it to learn points of conditional independence. A point of conditional independence is said to occur when the future symbol does not depend on the previous k symbols for $0 > k \geq n$. Ends of sentences, paragraphs, chapter, and novels are all clear examples of potential points of conditional independence. If points of conditional independence occur frequently in natural discourse, a model that is able to learn these points will perform substantially better than an equivalent order classic model as the classic model always assumes that the future is conditioned, to some degree, on the previous n symbols. Ultimately, the degree to which points of conditional dependencies occur is an empirical question.

The structure of the hierarchical nonemitting model class can also be viewed as providing a means for reducing the effective number of free parameters in the model. Although a mixture parameter is associated with each state in the model, these states are only utilized if a hierarchical nonemitting model has learned that these states are beneficial. In essence, a hierarchical nonemitting model can effectively remove free parameters by making nonemitting transitions that cause these parameters not to be utilized in practice. The classic model is unable to emulate this dynamic form of mixture parameter pruning because the state order choice at step t does not impact the set of allowable states at step $t+1$.

2.3.6 General Nonemitting Model

The general nonemitting model relaxes a fundamental constraint of the hierarchical nonemitting model. A state of order k at step t may transition to the state of any order at step $t+1$ thereby eliminating the need for intra-state transitions.[3] This means it is possible for the model to *learn* which state to state transitions are beneficial as opposed to establishing an *a priori* hierarchy. Of course, this freedom greatly increases the number of free parameters in the model since each state could possibly be connected to a large number of states, whereas the models considered hereunto have one mixture parameter per state. Thus, the λ parameters are re-defined to be

[3] This is the simplest interpretation of the mixture parameters. An alternative interpretation, that motivates the model class name, is to think of each general nonemitting transition as a series of intra-step hierarchical nonemitting transitions followed by an inter-step transition.

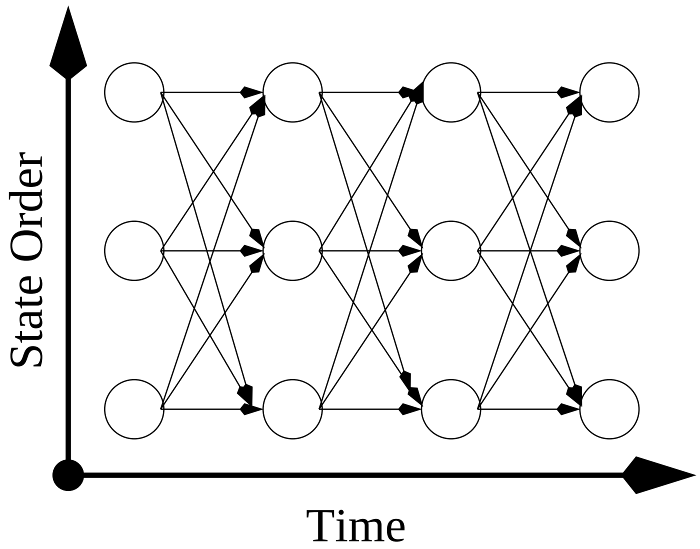

Figure 4: A graphical depiction of the general nonemitting Markov model. The general nonemitting model has no intra-step transitions. Inter-step transitions are allowed between a state of order less than or equal to n at step t to a state of any order less than or equal to n at step $t+1$.

$\lambda : A^{\leq n} \times A^{\leq n} \rightarrow [0,1]$. Figure 4 demonstrates the general nonemitting model's parameter utilization.

The probability that a state of order i, x_{t-i}^{t-1}, is used at step t depends on the set of contexts that could have been utilized at step $t-1$. Formally:

$$P(x_1^{t-1}, s_t = x_{t-i}^{t-1} \mid \phi) = \\ \sum_{j=0}^{n} P(x_1^{t-2}, s_{t-1} = x_{t-1-j}^{t-2} | \phi) P(s_t = x_{t-i}^{t-1} | s_{t-1} = x_{t-1-j}^{t-2}, \sigma = x_{t-1}) \delta(x_{t-1} \mid x_{t-1-j}^{t-2}, \phi) \tag{11}$$

where

$$P(s_t = x_{t-i}^{t} | s_{t-1} = x_{t-j}^{t-1}, \sigma = x_{t-1}) = \frac{\lambda(x_{t-1-j}^{t-2} \rightarrow x_{t-i}^{t-1})}{\sum_{k=0}^{n} \lambda(x_{t-1-j}^{t-2} \rightarrow x_{t-k}^{t-1})} \tag{12}$$

The probability that a symbol σ is generated conditioned on the state x_{t-i}^{t-1} is given by $\delta(\sigma \mid x_{t-i}^{t-1}, \phi)$.

The general nonemitting model can be viewed as reaching a particular state s_t at step t with some probability $P(x_1^{t-1}, s_t = x_{t-i}^{t-1} \mid \phi)$, making a symbol prediction with probability $\delta(\sigma \mid x_{t-i}^{t-1}, \phi)$, and transitioning to a state at step $t+1$ conditioned

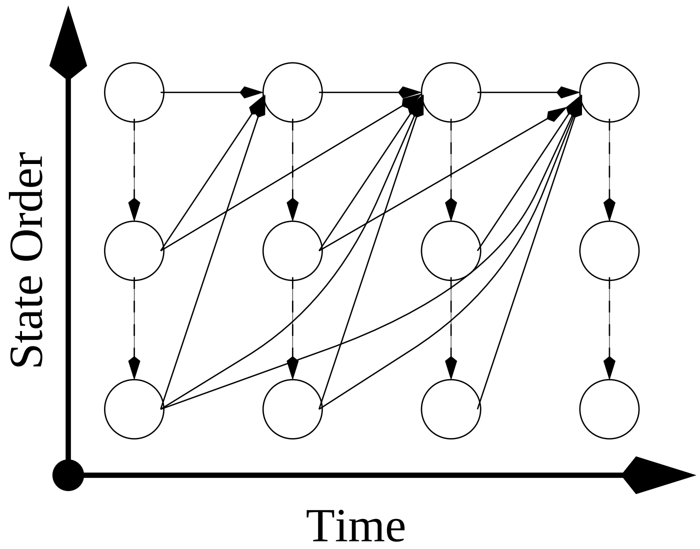

Figure 5: A graphical depiction of an order 2 nonuniform model. The multi-step transitions correspond to multi-unit predictions. For example, the order zero state at step t can transition to the order n state at step $t+1$, $t+2$, ..., $t+n+1$.

on the symbol prediction. Given the symbol predicted, there are at most $n+1$ possible states to which the model may transition. Any such transition occurs with probability: $P(s_t = x_{t-i}^{t-1} | s_{t-1} = x_{t-1-j}^{t-2}, \sigma = x_{t-1})$.

2.3.7 General Nonuniform Model

The general nonuniform model utilizes two sets of mixture parameters: one set of context mixture parameters and one set of prediction length parameters. The central feature that differentiates the general nonuniform model from other models considered so far is that it has the ability to make nonuniform length predictions according to the prediction length parameters. The ability to make nonuniform parameters gives the model the potential to better capture multi-unit linguistic constructs, e.g. *the United States of America*. Refer to figure 5 for a graphical description of the nonuniform model.

The probability that a particular context, x_{t-i}^{t-1}, is used at step t is given by:

$$P(x_1^{t-1}, s_t = x_{t-i}^{t-1} \mid \phi) = \\ \begin{cases} \lambda(x_{t-n}^{t-1}) \sum_{j=1}^{n+1} \sum_{k=0}^{n+1-j} [P(x_1^{t-1-j}, s_{t-j} = x_{t-j-k}^{t-1-j} | \phi) \rho(j \mid x_{t-j-k}^{t-1-j}) \\ \qquad \prod_{m=1}^{j-1} [(1 - \rho(j \mid x_{t-j-k}^{t-1-j})] \delta(x_{t-(j-1)}^{t-1} \mid x_{t-j-k}^{t-1-j}, \phi)] & i = n \\ \lambda(x_{t-i}^{t-1}) P(x_1^{t-1}, s_t = x_{t-(i+1)}^{t-1} | \phi) \frac{1-\lambda(x_{t-(i+1)}^{t-1})}{\lambda(x_{t-(i+1)}^{t-1})} & i < n \end{cases} \tag{13}$$

Although equation 13 appears complicated, it's conceptually straightforward. Since inter-state transitions can be of non-unit length, the model must consider each possible scenario where a prediction, multi-unit or not, can cause the model to reach state x_{t-n}^{t-1} at step t. It's interesting that the general nonuniform model can be viewed, on the one hand, as behaving like the classic model when the general nonuniform model believes the future is conditionally dependent on the previous n symbols – in which case it will make a unit-length prediction with high probability. On the other hand, if the model believes the future is not conditionally dependent on all of the past n symbols, the nonuniform model can choose to make a multi-length prediction.

The probability that a particular sequence of symbols σ_1^j is generated conditioned on the state x_{t-i}^{t-1} is given by:

$$P(\sigma_1^j \mid s_t = x_{t-i}^{t-1}, \phi) = \rho(j \mid x_{t-i}^{t-1}) \delta(\sigma_1^j \mid x_{t-i}^{t-1}, \phi) \prod_{k=1}^{j-1} (1 - \rho(k \mid x_{t-i}^{t-1})) \tag{14}$$

2.3.8 Collapsed Nonuniform Model

The collapsed nonuniform model collapses the two sets of mixture parameters used by the general nonuniform model into one set. It uses the λ parameters to determine which context to use as well as what length prediction to make.

The probability that a particular context, x_{t-i}^{t-1} is used at step t is given by:

$$P(x_1^{t-1}, s_t = x_{t-i}^{t-1} \mid \phi) = \\ \begin{cases} \lambda(x_{t-n}^{t-1}) \sum_{j=1}^{n+1} \sum_{k=0}^{n+1-j} [P(x_1^{t-j}, s_{t-j} = x_{t-j-k}^{t-1-j} | \phi) (1 - \lambda(x_{t-j-k}^{t-1+j}) \\ \qquad \prod_{m=1}^{j-1} [\lambda(x_{t-j-k}^{t-1+m})] \delta(x_{t-(j-1)}^{t-1} \mid x_{t-j-k}^{t-1-j}, \phi)] & i = n \\ \lambda(x_{t-i}^{t-1}) P(x_1^{t-1}, s_t = x_{t-(i+1)}^{t-1} | \phi) \frac{1-\lambda(x_{t-(i+1)}^{t-1})}{\lambda(x_{t-(i+1)}^{t-1})} & i < n \end{cases} \tag{15}$$

The probability that a sequence of symbols σ_1^j is generated conditioned on ith order state at step t is given by:

$$P(\sigma_1^j \mid s_t = x_{t-i}^{t-1}, \phi) = (1 - \lambda(x_{t-i}^{t-1+j}))\delta(\sigma_1^j \mid x_{t-i}^{t-1}, \phi) \prod_{k=1}^{j-1} \lambda(x_{t-i}^{t-1+k}) \qquad (16)$$

2.4 Model Class Algorithms

Explicit algorithmic descriptions of three important model operations – string generation, string probability evaluation, and parameter estimation – are now presented for each model class. These descriptions allow the computational efficiency of the model classes to be assessed while serving to highlight the fundamental differences between the model classes.

The efficiency claims made in this section are based on several assumptions. First, we assume $\delta(\sigma|x)$ can be computed in $O(1)$ time. This is a reasonable claim if the δ values are stored in an hash table of adequate size. [4] Second, we assume certain mathematical products and sums can be calculated in an efficient manner. For example, a loop of the form:

$p = 1$
for $i = 1$ to n
    $p = \lambda(x_{t-i}^{t-1}) \prod_{j=i+1}^{\min(t-1,n-1)}(1 - \lambda(x_{t-j}^{t-1}))$

is common throughout many of the algorithms supplied. As written, it appears that the calculation of p takes $O(n^2)$ time. In fact, the loop can be re-written as:

$p = 1$
$q = 1$
for $i = 1$ to n
    $p = (q)\lambda(x_{t-i}^{t-1})$
    $q = (q)(1 - \lambda(x_{t-i}^{t-1}))$

[4] This claim also assumes the hash function is computed in $O(1)$ time, which means the number of operations in the hash function can not be a function of the string size. Thus, one must choose the number of operations in the hash function not conditioned on n. For sake of this argument, we choose to compute the hash value based on the first 10 symbols of x.

so as to be performed in $O(n)$ time. We do not re-write these product assignments as we believe that the former pseudo-code description is more clear. Nonetheless, our efficiency claims rely on the fact that an actual implementation would contain a re-written version of the product assignment.

2.4.1 String Generation Algorithms

Each of the natural language models considered in this chapter can be repeatedly sampled to generate a string. Cleary, Bell, and Whitten[5] generated text randomly with character-based basic Markov models of various orders and we include their results to give a feel for how model order impacts model quality.

- Order 0 model: fsn'iaad ir Intns hynci,.aais oayimh t n .at oeotc fheotyi t afrtgt oidtsO, wrr thraeoe rdaFr ce.g psNo is.emahntawe,ei t etaodgdna- &em r n nd fih an f tpteaaInmas ss n t"bar o be um oon tsrcs et mi ithyoitt h u ans w vsgr tn heaacrY.d erfdut y c, a,m ¡hra Pieodn nyeSrsoto oea nlorseo j r s t w ge g E ikdeAJ .I eeTJiahednn ,ngaosl dshoHo eh seelm G os threen nrgifeo,edsoht tgt n til a issnin"abi"h nht.e bs co efhetntoilgevtnnadrssaa ka dfnssiivb kuniseeao M41 h acdchnr onoal ie a lhehtr web Yolo aere mblefeuom eomtlkIo h oattogodrinl aw Blbe.

- Order 5 Model: number diness, and it also light of still try and among Presidential discussion is department-transcended "at they maker and for liquor in an impudents to each chemistry is that American denying it did not feel I mustached through to the budget, son which the fragment on optically should not even work before that he was ridiculous little black-body involved the workable of write: "The Loard Steak a line (on 5 cubic century. When the bleaches suggest connection, and they were that, but you". The route whatever second left Americans will done a m the cold.

- Order 11 model: papal pronouncements to the appeal, said that he'd left the lighter fluid, ha ha"? asked the same number of temptation to the word 'violent'.

> "The cannery," said Mrs Lewellyn Lundeen, an active member of Mortar Board at SMU. Her husband, who is the Michelangelo could not quite come to be taxed, or for a married could enroll in the mornings, I was informed. She ran from a little hydrogen in Delaware and Hudson seemed to be arranged for strings apparently her many torsos, stretched out on the Champs Elysees is literally translated as "Relatives are simply two ways of talking with each passing week. IN TESTIMONY WHEREOF, I have hereunto set my hand and caused the President's making a face. "What's he doing here"? "This afternoon. When he turns upon the pleader by state law.

The generation strings clearly demonstrate the ability of higher order models to capture regularities of English. Of course, as Cleary *et al.* commented: "even the order-11 model is far from perfect." Because of the models finite order, there is little connection between letters more than n positions away. This accounts for the wandering effect seen in the generation strings.

We now present the generation algorithms for each of the natural language models considered in the chapter.

Basic Model

The basic Markov model's generation algorithm, algorithm 1, emits a symbol based on the previous n symbols generated. Line 3 of basic-generate demonstrates that the choice of which symbol to emit is determined by the probability distribution defined by δ. Recall that the δ function defines the probability that a given symbol will follow the string x according to the model ϕ. The basic Markov model's generation algorithm runs in $O(T)$ time and $O(1)$ space.

Classic Model

Like the basic model, the classic model (algorithm 2) generates the next symbol based upon the previous n symbols of the already generated string. Unlike the basic model, the classic model exploits the $n+1$ suffixes of the history individually. The probability that a particular context is used to generate the next symbol depends on its λ value

Algorithm 1 Basic Markov Model Generation Algorithm

basic-generate(T, ϕ)
$x = \epsilon$
for $t = 1$ to T
    pick σ according to $\delta(\sigma \mid x_{t-\min(t-1,n)}^{t-1}, \phi)$
    $x = x \cdot \sigma$
return x

Algorithm 2 Classic Model Generation Algorithm

classic-generate(T, ϕ)
$x = \epsilon$
for $t = 1$ to T
    pick context length i in $[0,\min(\text{t-1,n})]$ according to
        $\lambda(x_{t-i}^{t-1}) \prod_{j=i+1}^{\min(t-1,n-1)} (1 - \lambda(x_{t-j}^{t-1}))$
    pick σ according to $\delta(\sigma \mid x_{t-i}^{t-1}, \phi)$
    $x = x \cdot \sigma$
return x

as well as the λ values of each of the longer contexts. The generation algorithm of the classic model runs in $O(nT)$ time and $O(1)$ space.

Hierarchical Nonemitting Model

Since hierarchical nonemitting transitions are permanent, the hierarchical nonemitting model must first choose a context (line 4 of algorithm 3). Line 5 of hierarchical-nonemitting-generate then chooses a symbol conditioned on the context, line 6 concatenates that symbol to the generation string, and line 7 calculates the longest applicable context at step $t + 1$. A string is generated by the hierarchical nonemitting model in $O(T)$ time and $O(1)$ space. The time bound is obtained using amortized analysis. Notice that it is not possible to consider $O(n)$ states at each step. In fact, n states are considered if and only if max-length in line 4 of hierarchical-nonemitting-generate

Algorithm 3 Hierarchical Nonemitting Model Generation Algorithm

```
hierarchical-nonemitting-generate(T, φ)
1.  x = ε
2.  max-length = 0
3.  for t = 1 to T
4.      pick context length i in [0, max-length] according to
            λ(x_{t-i}^{t-1}) ∏_{j=i+1}^{max-length} (1 − λ(x_{t-j}^{t-1}))
5.      pick σ according to δ(σ | x_{t-i}^{t-1}, φ)
6.      x = x · σ
7.      max-length = min(i + 1, n)
8.  return x
```

is n. max-length can equal n at most T/n times – giving a total time bound of $O(\frac{T}{n}n + (T - \frac{T}{n})1)$, or $O(T)$.

General Nonemitting Model

The string generation algorithm, algorithm 4, for the general nonemitting model is straightforward. At each step, a prediction is made (line 4) and the next context is chosen based on the state λ parameters. This can be seen in line 7 of general-nonemitting-generate. The computational properties of the general nonemitting model match those of the classic model: $O(nT)$ time and $O(1)$ space.

General Nonuniform Model

The general nonuniform model must choose the prediction length at each step in addition to the usual choices of context length and output symbols. The general nonuniform model first decides upon a context length (line 3 of algorithm 5), then it determines the prediction length at step t (line 4), and finally it chooses which symbols to generate (line 5). This algorithm generates a string in $O(n^2T)$ time and $O(1)$ space. The extra n factor is due to the additional choice of prediction length.

Collapsed Nonuniform Model

Algorithm 4 General Nonemitting Model Generation Algorithm

general-nonemitting-generate(T, ϕ)
1. $x = \epsilon$
2. $\mathrm{c} = \epsilon$
3. for $t = 1$ to T
4.   pick symbol, σ according to $\delta(\sigma \mid c, \phi)$
5. $x = x \cdot \sigma$
6.   Let $C = \{c^m_{1+\max(0,n-m)}\sigma, \ldots, c^m_m\sigma, \sigma, \epsilon\}$
7.   pick new context, $c' \in C$ according to
   $\frac{\lambda(c \to c')}{\sum_{\hat{c} \in C} \lambda(c \to \hat{c})}$
8. $\mathrm{c} = \mathrm{c'}$
9. return x

Algorithm 5 General Nonuniform Model Generation Algorithm

general-nonuniform-generate(T, ϕ)
1. $x = \epsilon$
2. while $|x| < T$
3.   pick context length i in $[0,\min(t-1, n-1)]$ according to
   $\lambda(x^{t-1}_{t-i}) \prod_{i+1}^{j=\min(t-1,n)} (1 - \lambda(x^{t-1}_{t-j})$
4.   pick prediction length j in $[1, \max(\text{T-(t-1), n-i})]$ according to
   $\rho(j \mid x^{t-1}_{t-i}) \prod_{k=1}^{j-1} (1 - \rho(k \mid x^{t-1}_{t-i}))$
   note: $\rho(\max(T-(t-1), n-i) \mid x^{t-1}_{t-i}) = 1$
5.   pick symbols, σ^j_1, according to $\delta(\sigma^j_1 \mid x^{t-1}_{t-i}, j, \phi)$
6. $x = x \cdot \sigma^j_1$
7. return x

Algorithm 6 Collapsed Nonuniform Model Generation Algorithm

collapsed-nonuniform-generate(T, ϕ)
1. $x = \epsilon$
2. while $|x| < T$
3. pick context length i in $[0,\min((t-1), n-1)]$ according to $\lambda(x_{t-i}^{t-1}) \prod_{j=i+1}^{\min(t-1,n)} (1 - \lambda(x_{t-j}^{t-1})$
4. pick prediction length j in $[1, \max(\text{T-(t-1), n-i})]$ according to $(1 - \lambda(x_{t-i}^{t-1} y_1^j)) \prod_{k=1}^{j-1} (1 - \lambda(x_{t-i}^{t-1} y_1^k))$
   note: $\lambda(x_{t-i}^{t-1} y_1^{\max(T-(t-1),n-i)}) = 0$
5. pick symbols, σ_1^j, according to $\delta(\sigma_1^j \mid x_{t-i}^{t-1}, j, \phi)$
6. $x = x \cdot \sigma_1^j$
7. return x

The collapsed nonuniform model's generation algorithm, algorithm 6, is nearly identical to the general nonuniform model's version. The single difference is that the prediction length choice is made based upon the λ parameters instead of the ρ parameters of the general nonuniform model. Algorithm 6 runs in $O(n^2T)$ time and $O(1)$ space.

2.4.2 String Probability Evaluation Algorithms

Efficient algorithms that calculate the probability that a model ϕ generated a string x are an essential component of speech and handwriting recognition systems.

The goal of both handwriting or speech recognition systems is to transcribe a signal, either graphical or acoustic, to text. Using the signal in isolation for transcription can result in dismal transcription accuracy. This is not surprising as human beings share common knowledge about natural language and the real world and use that knowledge to improve their generation and comprehension skills. As a result, systems that exploit knowledge of natural language and the real world have the potential to yield better results. This knowledge is often encapsulated in a statistical natural language text model.

Formally, recognition systems attempt to find the most likely text transcription, x, of the, acoustic or graphic, signal S. From a probabilistic perspective, the goal is to find the transcription x that maximizes $P(x \mid S)$. Since there are two ways to re-write the joint distribution $P(x, S)$:

$$\begin{aligned} P(x,S) &\doteq P(x \mid S)P(S) \\ P(x,S) &\doteq P(S \mid x)P(x) \end{aligned}$$

we can calculate $P(x \mid S)$ as:

$$P(x \mid S) = \frac{P(S \mid x)P(x)}{P(S)} \tag{17}$$

by setting the two possible decompositions of the joint distribution equal to one another and dividing by $P(S)$. Equation 17 is an example of the use of Bayes' Rule. Because we're interested in finding the x that maximizes $P(x \mid S)$ we can ignore the $P(S)$ factor in the denominator, the language model can be utilized to calculate $P(x)$, and the signal model calculates $P(S \mid x)$.

We now show how to calculate $P(x)$ for each of the mixture Markov models.

Basic Model

The probability assigned to a string x_1^T by a basic model is simply the product of conditional probabilities and is performed in $O(T)$ time and $O(1)$ space. Algorithm

Algorithm 7 Basic Markov Model Evaluation Algorithm

basic-evaluate(x_1^T, ϕ)
$p = 1$
for $t = 1$ to T
    $p \mathrel{\cdot}= \delta(x_t \mid x_{t-n}^{t-1}, \phi)$
return p

Algorithm 8 Classic Model Evaluation Algorithm

classic-evaluate(x_1^T, ϕ)
$\mathrm{p} = 1$
for $t = 1$ to T
    $\mathrm{p} \mathrel{\cdot}=$ classic-symbol-prob(x_1^{t-1}, x_t, ϕ)
return p

classic-symbol-prob(x_1^t, σ, ϕ)
$\mathrm{p} = 0$
for $i = \min(t, n)$ downto 0
    $\mathrm{p} \mathrel{+}= \lambda(x_{t-i}^t)\delta(\sigma \mid x_{t-i}^t, \phi) \prod_{j=i+1}^{\min(t,n)} (1 - \lambda(x_{t+1-j}^t))$
return p

7 contains the pseudo-code for the basic model's probability assignment.

Classic Model

The evaluation algorithm, algorithm 8, for the classic mixture Markov model is very similar to basic-evaluate. The fundamental difference is that the probability that the classic model generated a particular symbol depends on the set of $n + 1$ suffixes – instead of just the order n suffix. This adds an extra factor of n to the time complexity, $O(nT)$. The classic model uses $O(1)$ space.

Algorithm 9 Hierarchical Nonemitting Model Evaluation Algorithm

hierarchical-nonemitting-evaluate(x_1^T, ϕ)
1. for $i = n$ downto 0
2. $\quad p_{t+1}[i] = 0.0$
3. $\quad p_t[i] = 0.0$
4. $p_t[0] = 1.0$
5. for $t = 1$ to T
6. $\quad$ for $i = \mathsf{min}(t, n)$ downto 0
7. $\quad\quad p_{t+1}[\mathsf{min}(n,i+1)] \mathrel{+}= p_t[i]\ \lambda(x_{t-i}^{t-1})\delta(x_t \mid x_{t-i}^{t-1}, \phi)$
8. $\quad\quad$ if($i > 0$)
9. $\quad\quad\quad p_t[i-1] \mathrel{+}= p_t[i]\ (1 - \lambda(x_{t-i}^{t-1}))$
10. $\quad$ for $i = 0$ to n $p_t[i] = p_{t+1}[i]$; $p_{t+1}[i] = 0$
11. $\mathrm{p} = 0$
12. for $i = n$ downto 0
13. $\quad \mathrm{p} \mathrel{+}= p_t[i]$
14. return p

Hierarchical Nonemitting Model

The probability evaluation algorithm for the hierarchical nonemitting mixture Markov model is now outlined (algorithm 9). Since a prediction made from a state of order k at step t leads to states of order less than or equal to $\min(n, k+1)$ to be considered at step $t+1$, it is necessary to keep track of the probability of reaching a particular state. Viewed from another perspective, a state of order k at step t can be reached in two ways: (1) a direct transition from a state at step $t-1$ or (2) a nonemitting transition from a state of order $k+1$ at step t. The hierarchical nonemitting model's evaluation algorithm runs in $O(nT)$ time and uses $O(n)$ space. The p_{t+1} and p_t arrays keep track of the probability of reaching a particular state at steps $t+1$ and t respectively. Line 7 calculates the impact of transition from a state of order i at step t to a state of order $\min(n, i+1)$ at step $t+1$. This calculation depends on the probability of reaching that state, choosing that state to condition the prediction, and generating the symbol.

Lines 12-14 calculates the sum of reaching any state at step $T+1$. The sum

Algorithm 10 General Nonemitting Model Evaluation Algorithm

general-nonemitting-evaluate(x_1^T, ϕ)
1. for $i = n$ downto 0
2. $\quad p_{t+1}[i] = 0.0$
3. $\quad p_t[i] = 0.0$
4. $p_t[0] = 1.0$
5. for $t = 1$ to T
6. $\quad$ for $i = \min(t, n)$ downto 0
7. $\quad\quad$ for $j = \min(t+1, n)$ downto 0
8. $\quad\quad\quad p_{t+1}[j] \mathrel{+}= p_t[i] \frac{\lambda(x_{t-i}^{t-1} \to x_{t-j}^{t})}{\sum_{k=0}^{\min(t+1,n)} \lambda(x_{t-i}^{t-1} \to x_{t+1-k}^{t})} \delta(x_t \mid x_{t-i}^{t-1}, \phi)$
9. $\quad$ for $i = 0$ to n $p_t[i] = p_{t+1}[i]$; $p_{t+1}[i] = 0$
10. $\mathrm{p} = 0$
11. for $i = n$ downto 0
12. $\quad \mathrm{p} \mathrel{+}= p_t[i]$
13. return p

represents the probability of ϕ generating x since any generation path must end at step $T + 1$.

General Nonemitting Model

The general nonemitting model's evaluation algorithm, algorithm 10, calculates the probability that ϕ generated x. Line 8 of general-nonemitting-evaluate calculates the probability of transitioning from a state of order i at step t to the state of order j at step $t + 1$. The algorithm runs in $O(n^2T)$ time and uses $O(n)$ space. As was true for the hierarchical nonemitting model, the sum over the probability of reaching any state at step $T + 1$ represents the probability that general model ϕ generated x.

General Nonuniform Model

The general nonuniform evaluation algorithm also employs dynamic programming techniques to make the computation efficient (algorithm 12). A table of size $O(T)$

Algorithm 11 General Nonuniform Model Evaluation Algorithm

general-nonuniform-evaluate(x_1^T, ϕ)
1. for $t = 1$ to $T + 1$
2. $\quad$ $\mathrm{p}[t] = 0$
3. $\mathrm{p}[0] = 1$
4. for $t = 1$ to T
5. $\quad$ for $i = \min(t, n)$ downto 0
6. $\quad\quad$ for $j = 1$ to $\min(n + 1 - i,\ t - i)$
7. $\quad\quad\quad$ $p_{event} = \mathrm{p}[t]\ \lambda(x_{t-i}^{t-1})\ \rho(j \mid x_{t-i}^{t-1})\delta(x_t^{t+j-1} \mid x_{t-i}^{t-1}, \phi)$ $[\prod_{k=i+1}^{\min(t,n)}(1 - \lambda(x_{t-k}^{t-1}))]$ $[\prod_{k=1}^{j-1}(1 - \rho(k \mid x_{t-i}^{t-1}))]$
8. $\quad\quad\quad$ $\mathrm{p}[t + j]$ += p_{event}
9. return $\mathrm{p}[T + 1]$

tracks the probability of generating the first $t - 1$ symbols. Line 7 of general-nonuniform-evaluate demonstrates the impact of choosing a context of length i at time t and making a prediction of length j. The table size can be reduced to $O(n)$ with a reduction in clarity and the algorithm runs in $O(n^2T)$ time. [5]

Collapsed Nonuniform Model

The collapsed nonuniform model's evaluation algorithm is nearly identical to the general nonuniform model's and runs in $O(n^2T)$ time and $O(T)$ space. The space complexity can be reduced to $O(n)$ with a degradation in clarity. Line 7 of collapsed-nonuniform-evaluate utilizes the λ parameters to determine both the context and prediction length whereas general-nonuniform-evaluate utilizes the ρ parameters to determine prediction length.

[5] Since a nonuniform prediction can only impact states $n + 1$ steps away, a table of size $O(n)$ is sufficient to store the probabilities. Such a table, however, would need to be updated at each step making the algorithm less clear.

Algorithm 12 Collapsed Nonuniform Model Evaluation Algorithm

collapsed-nonuniform-evaluate(x_1^T, ϕ)
for $t = 1$ to $T + 1$
    $\mathrm{p}[t] = 0$
$\mathrm{p}[0] = 1$
for $t = 1$ to T
    for $i = \min(t, n)$ downto 0
        for $j = 1$ to $\min(n + 1 - i,\ t - i)$
            $p_{event} = \ \mathrm{p}[t]\ \lambda(x_{t-i}^{t-1})\ (1 - \lambda(x_{t-i}^{t+j-1}))\delta(x_t^{t+j-1} \mid x_{t-i}^{t-1}, \phi)$
            $[\prod_{k=i+1}^{\min(t,n)}(1 - \lambda(x_{t-k}^{t-1}))]$
            $[\prod_{k=1}^{j-1} \lambda(x_{t-i}^{t+k-1}))]$
            $\mathrm{p}[t + j]$ += p_{event}
return $\mathrm{p}[T + 1]$

2.4.3 String Probability Evaluation: An Example

We will now examine the manner in which the different model classes assign probability to strings by considering the basic, classic, and hierarchical nonemitting models.

Assume the following: the finite alphabet $A = \{0, 1\}$, $n = 2$, and the δ and λ parameters are set to equivalent values for each of the models. The δ and λ values were chosen arbitrarily and are given in tables 1-2. The models will assign probability to the string 0110 in this example.

Basic Model

The basic model's probability assignment is simply the product of conditional probabilities defined by the δ function.

$$\begin{aligned} P_b(0110) &= \delta(0|\epsilon)\delta(1|0)\delta(1|01)\delta(0|11) \\ &= (1/2)(1/4)(2/3)(1/8) \\ &= 1/96 \\ &= .0104 \end{aligned}$$

See figure 6 for a graphical interpretation of the probability assignment.

Classic Model

The classic model's probability assignment is a bit more difficult because each of the three distributions, representing n =0,1, or 2, must be utilized for the probability assignment at each step:

$$\begin{aligned} P_c(0110) &= \prod_{t=1}^{4}[\sum_{i=0}^{n} P_c(s_t = x_{t-i}^{t-1})\delta(x_t|x_{t-i}^{t-1})] \\ &= \delta(0|\epsilon) \cdot \\ & \quad [\lambda(0)\delta(1|0) + (1-\lambda(0))\delta(1|\epsilon)] \cdot \\ & \quad [\lambda(01)\delta(1|01) + (1-\lambda(01))\lambda(1)\delta(1|1) + (1-\lambda(01))(1-\lambda(1))\delta(1|\epsilon)] \cdot \\ & \quad [\lambda(11)\delta(0|11) + (1-\lambda(11))\lambda(1)\delta(0|1) + (1-\lambda(11))(1-\lambda(1))\delta(0|\epsilon)] \cdot \end{aligned}$$

δ Values		
$\delta(0 \mid \epsilon) = 1/2$	$\delta(0 \mid 0) = 3/4$	$\delta(0 \mid 00) = 8/9$
$\delta(1 \mid \epsilon) = 1/2$	$\delta(1 \mid 0) = 1/4$	$\delta(1 \mid 00) = 1/9$
	$\delta(0 \mid 1) = 1/5$	$\delta(0 \mid 01) = 1/3$
	$\delta(1 \mid 1) = 4/5$	$\delta(1 \mid 01) = 2/3$
		$\delta(0 \mid 10) = 1/2$
		$\delta(1 \mid 10) = 1/2$
		$\delta(0 \mid 11) = 1/8$
		$\delta(1 \mid 11) = 7/8$

Table 1: The δ values

λ Values		
$\lambda(\epsilon) = 1$	$\lambda(0) = 3/4$	$\lambda(00) = 1/4$
	$\lambda(1) = 7/8$	$\lambda(01) = 1/5$
		$\lambda(10) = 7/8$
		$\lambda(11) = 1/9$

Table 2: The λ values

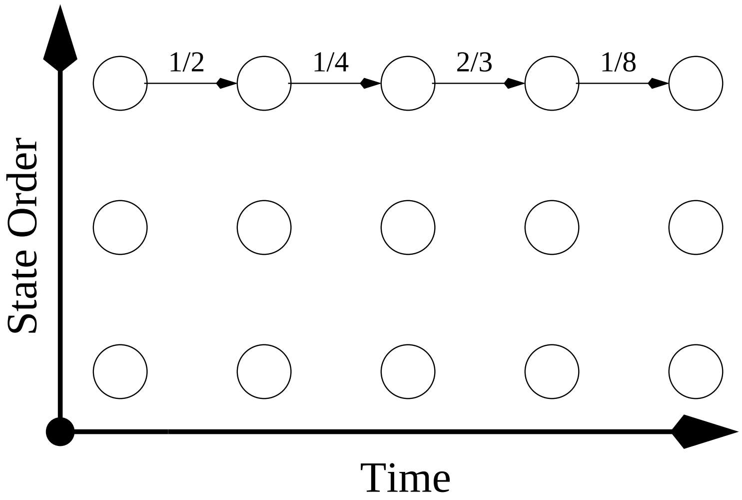

Figure 6: A graphical depiction of the basic model example. The transitions are labeled with the appropriate probabilities from the δ table.

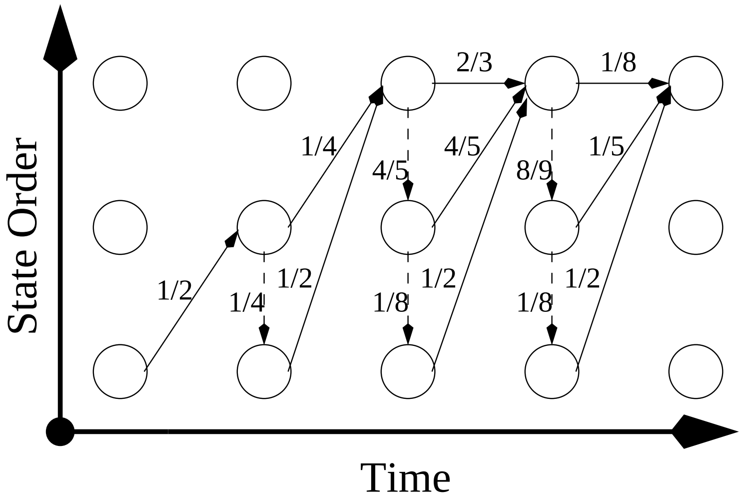

Figure 7: A graphical depiction of the classic model example. The dashed lines represent the blending of higher order and lower order states. These transitions are labeled with probabilities from the λ table. The inter-step lines represent the emission of a symbol and transition to a new state. These transitions are labeled with probabilities from the δ table.

$$
\begin{aligned}
&= (1/2) \cdot \\
&\quad [(3/4)(1/4) + (1/4)(1/2)] \cdot \\
&\quad [(1/5)(2/3) + (4/5)(7/8)(4/5) + (4/5)(1/8)(1/2)] \cdot \\
&\quad [(1/9)(1/8) + (8/9)(7/8)(1/5) + (8/9)(1/8)(1/2)] \\
&= (1/2)(5/16)(67/75)(9/40) \\
&= 603/19200 \\
&= .0314
\end{aligned}
$$

Refer to figure 7 for a visual depiction of the classic model's probability assignment.

Hierarchical Nonemitting Model

The easiest way to calculate the probability that the hierarchical nonemitting model assigns to a particular string is to calculate the probability of being in each state at every step. The sum over the probability of reaching every state for a particular step t

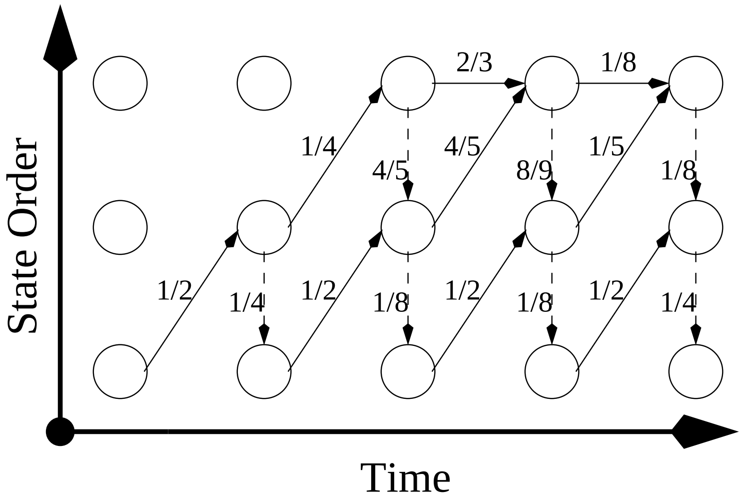

Figure 8: A graphical depiction of the hierarchical nonemitting model example. The dashed lines represent hierarchical nonemitting transitions labeled with probabilities from the λ table. The inter-step transitions correspond to the emission of a symbol and transition to a new state. These transitions are labeled with probabilities from the δ table.

is the probability the model assigns to the first t symbols. A graphical representation of the hierarchical nonemitting model's probability assignment is presented in figure 8.

$$P_h(s_1 = \epsilon) = 1$$

$$\begin{aligned}
P_h(s_2 = 0) &= P_h(s_1 = \epsilon)\delta(0|\epsilon)\lambda(1) \\
&= (1)(1/2)(7/8) \\
&= 7/16 \\
P_h(s_2 = \epsilon) &= P_h(s_2 = 0)\frac{1-\lambda(1)}{\lambda(1)} \\
&= 7/16\frac{1/8}{7/8} \\
&= 1/16
\end{aligned}$$

$$\begin{aligned}
P_h(s_3 = 01) &= P_h(s_2 = 0)\delta(1|0)\lambda(01) \\
&= (7/16)(1/4)(1/5) \\
&= 7/320 \\
P_h(s_3 = 1) &= \lambda(1)[P_h(s_2 = \epsilon)\delta(1|\epsilon) + P_h(s_3 = 01)\frac{1-\lambda(01)}{\lambda(01)}] \\
&= (7/8)[(1/16)(1/2) + (7/320)\frac{4/5}{1/5}] \\
&= (7/8)[(1/32) + (7/80)] \\
&= 133/1280 \\
P_h(s_3 = \epsilon) &= P_h(s_3 = 1)\frac{1/8}{7/8} \\
&= 19/1280
\end{aligned}$$

$$\begin{aligned}
P_h(s_4 = 11) &= \lambda(11)[P_h(s_3 = 01)\delta(0|01) + P_h(s_3 = 1)\delta(0|1)] \\
&= (1/9)[(7/320)(1/3) + (133/1280)(1/5)] \\
&= (1/9)[7/960 + 133/6400] \\
&= 539/172800 \\
P_h(s_4 = 1) &= \lambda(1)[P_h(s_3 = \epsilon)\delta(0|\epsilon) + P_h(s_4 = 11)\frac{1-\lambda(11)}{\lambda(11)}] \\
&= (7/8)[(19/1280)(1/2) + (539/172800)\frac{8/9}{1/9}] \\
&= (7/8)[19/2560 + 4312/172800 \\
&= 539/21600 \\
P_h(s_4 = \epsilon) &= P_h(s_4 = 1)\frac{1-\lambda(1)}{\lambda(1)} \\
&= (539/21600)\frac{1/8}{7/8} \\
&= 77/21600
\end{aligned}$$

$$\begin{aligned}
P_h(s_5 = 10) &= \lambda(10)[P_h(s_4 = 11) + P_h(s_4 = 1)] \\
&= (7/8)[539/172800 + 539/21600]
\end{aligned}$$

$$
\begin{aligned}
&= 33957/1382400 \\
&= .0246 \\
P_h(s_5 = 0) &= \lambda(0)[P_h(s_4 = \epsilon) + P_h(s_5 = 10)\frac{1-\lambda(10)}{\lambda(10)}] \\
&= 3/4[77/21600 + 33957/1382400\frac{1/8}{7/8}] \\
&= 3/4[68453/9676800] \\
&= 205359/38707200 \\
&= .0053 \\
P_h(s_5 = \epsilon) &= P_h(s_5 = 0)\frac{1-\lambda(0)}{\lambda(0)} \\
&= 205359/38707200\frac{1/4}{3/4} \\
&= 205359/116121600 \\
&= .0018
\end{aligned}
$$

$$
\begin{aligned}
\text{Thus, } P_h(0110) &= .0246 + .0053 + .0018 \\
&= .0317
\end{aligned}
$$

So, for this simple example $P_h(0110) = .0317 > P_c(0110) = .0314 > P_b(0110) = .0104$.

2.4.4 Parameter Estimation Algorithms

The state output, δ, and mixture, λ, parameters must accurately reflect the natural language else the evaluation and generation algorithms will give poor results. The parameter estimation algorithms defined here allow the parameters of the model to be adjusted to increase the probability assigned to a given string, x, by the model. Assuming this string is representative of the language, it is hoped that the model will assign reasonably accurate probabilities to other strings drawn from the natural language.

Algorithm 13 Basic Markov Model Estimation Algorithm

basic-estimate$(x, n+1, \phi)$
1. $Y = \{y : y \in \text{Substrings}(x) \wedge \mid y| = n+1\}$
2. $\forall\ y_1^{n+1} \in Y$
3. $\quad \delta(y_{n+1} \mid y_1^n) \doteq \frac{\mathrm{C}(y_1^{n+1})}{\mathrm{C}(y_1^n)}$

The δ parameters are typically the maximum-likelihood estimates of the training corpus, $P_{ML}(\sigma|x) = \frac{c(\sigma|x)}{c(x)}$ where $c(x)$ is the number of times the string x occurred in the training corpus. The λ parameters are estimated using expectation maximization, an iterative algorithm that repeatedly increases, or maintains, the likelihood that the model generated a string x.

Basic Model

The basic Markov model estimation algorithm, algorithm 13, sets the δ parameters to the maximum-likelihood estimates of the string x. Line 1 defines the set of substrings of length $n+1$ of the string x. For each of these strings, it is possible to estimate the probability that the final symbol, y_{n+1}, of a particular substring y_1^{n+1} occurs after the first n symbol y_1^n by counting the number of times each event occurs. basic-estimate runs in $O(T)$ time and $O(1)$ space.

Expectation Maximization

The expectation maximization algorithm is used to estimate the mixture parameters of the mixture models considered in this chapter (see [3, 4] for a detailed description of the algorithm). The expectation maximization algorithm has two steps: the expectation step that calculates the expected number of times each hidden event occurs in the model, and the maximization step that estimates the mixture parameters based on the accumulated expectations. A hidden event is an event that is not observable from the model's activity. For example, the choice of state order at each step is a hidden event.

Although each of the upcoming estimation algorithms are slightly different from one another, they all calculate $E_p[v]$ and $E_p[\bar{v}]$ (relative to some probability distribution p), for all hidden events v. $E_p[v]$ is the expected number of times that v occurred. We define $E_p[\bar{v}]$ to be the expected number of times the event v did not occur when it could have occurred, e.g. a state of order k was used instead of a state of order $k+1$. These accumulated expectations can be used as the basis for estimating $\lambda(v)$:

$$\lambda(v) = \frac{E_p[v]}{E_p[v] + E_p[\bar{v}]} \tag{18}$$

Given a model $\phi = \langle n, A, \lambda, \delta \rangle$ and a training string L for the λ parameters, it is possible to construct a model $\hat{\phi} = \left\langle n, A, \hat{\lambda}, \delta \right\rangle$ such that $P(L|\hat{\phi}) \geq P(L|\phi)$ where the $\hat{\lambda}$ parameters are the result of the expectation maximization algorithm. Yianilos[60] proves that the expectation maximization estimate $\hat{\phi}$ satisfies $P(L|\hat{\phi}) \geq P(L|\phi)$ for finite growth models. A finite growth model is a "directed acyclic graph with a single source and sink together with a collection of non-negative bounded weight function w_i that are associated with edges – one per edge. The value of a source-sink path is the product of the weights along it and the FGM's value is the sum over all such paths of their values[53]." Since the estimation problem for the mixture models considered in this thesis may be reduced to the estimation problem for finite growth models, by inspection, we may conclude that EM improves or maintains the current parameter set relative to L.

Classic Model

The classic model's estimation algorithm, algorithm 14, must determine both the δ and λ parameters. classic-estimate has three arguments: the training string D to estimate the δ parameters, the training string L to estimate the λ parameters, and the model ϕ. From a high level view, the estimation function (1) sets the δ parameters to be the maximum-likelihood state output parameters, and (2) adjusts the λ parameters so as to increase or maintain model performance. Step (2) is performed holding the δ parameters constant to the values established in step (1). Lines 1-2 of classic-estimate set the δ parameters to be the maximum-likelihood transition parameters of D. Since the classic model uses contexts of length 0 to n, we must call basic-estimate $n+1$ times, once for each length. Lines 4-6 initialize the λ accumulation parameters, λ^+ and λ^-.

Algorithm 14 Classic Model Parameter Estimation Algorithm

classic-estimate(D, L, ϕ)
for $m = 1$ to $n + 1$
    basic-estimate(D, m, ϕ)
$Y = \{y : y \in \text{Substrings}(L)\}$
$\forall\ y \in Y$
    $\lambda^+(y) = 0$
    $\lambda^-(y) = 0$
for $t = 1$ to $|L|$
    $\text{event-sum} = 0$
    for $i = \min(t, n)$ downto 0
        $\text{event-sum} \mathrel{+}= \lambda(L_{t-i}^{t-1})\delta(L_t \mid L_{t-i}^{t-1}, \phi) \prod_{j=i+1}^{\min(t,n)} (1 - \lambda(L_{t-j}^{t-1}))$
    for $i = \min(t, n)$ downto 0
        $p_{event} = \delta(L_t \mid L_{t-i}^{t-1}, \phi) \prod_{j=i+1}^{\min(t,n)} (1 - \lambda(x_{t-j}^{t-1}))$
        $\lambda^+(L_{t-1-i}^{t-1}) \mathrel{+}= \frac{\lambda(L_{t-i}^{t-1}) p_{event}}{\text{event-sum}}$
        $\lambda^-(L_{t-1-i}^{t-1}) \mathrel{+}= \frac{(1-\lambda(L_{t-1-i}^{t-1})) p_{event}}{\text{event-sum}}$
classic-maximize(Y, ϕ)

classic-maximize(Y, ϕ)
$\forall\ y \in Y$
    $\lambda(y) = \frac{\lambda^+(y)}{\lambda^+(y)+\lambda^-(y)}$
    $\lambda^+(y) = 0$
    $\lambda^-(y) = 0$

λ^+ accumulates the expected number of times a particular context, z, was used when it could have been used. λ^- accumulates the expected number of times a particular context, z, was not used when it could have been used. Lines 8-10 calculates the probability of the symbol L_t at step t. Lines 12-14 accumulates the expectation that a particular context of length i was used at step t to generate L_t. Line 15 calls classic-maximize which estimates the λ parameters based on the λ^+ and λ^- accumulators. classic-estimate can be called repeatedly until the λ values converge. The estimation algorithm for the classic model runs in $O(nT)$ time (per iteration) and $O(nT)$ space.

Hierarchical Nonemitting Model

The parameter estimation algorithm for the hierarchical nonemitting model's λ parameters requires a forward and backward pass through the training string (algorithm 15). The forward pass of the algorithm calculates:

$$\alpha[t, i] \doteq P(h = x_1^{t-1}, c = x_{t-i}^{t-1}, v = x_t \mid T, \phi) \tag{19}$$

$\alpha[t, i]$ represents the probability of generating the first $t-1$ symbols, at step t choosing a context c of length i, and generating the tth symbol.

The backward pass of the algorithm calculates:

$$\beta[t, i] \doteq P(x_t^T \mid h = x_1^{t-1}, c = x_{t-i}^{t-1}, v = x_t, T, \phi) \tag{20}$$

$\beta[t, i]$ represents the probability of generating the final $T - t$ symbols given that the first $t - 1$ symbols have already been generated, and at step t a context of length i is utilized.

The α and β tables can be used to calculate the probability that a context of length i was used to predict symbol x_t at step t:

$$\begin{aligned} \gamma[t, i] &\doteq P(c = x_{t-i}^{t-i}, v = x_t \mid x, \phi) & (21) \\ &= \frac{\alpha[t, i]\beta[t + 1, \min(n, i + 1)]}{P(x \mid \phi)} & (22) \end{aligned}$$

The γ values can be accumulated so as to compute the expected number of times a particular state x was utilized when it could have been versus the expected number

Algorithm 15 Hierarchical Nonemitting Model Parameter Estimation Algorithm

hierarchical-nonemitting-estimate(D, L, ϕ)
- for $m = 1$ to $n + 1$
    - basic-estimate(D, m, ϕ)
- $\alpha =$ hierarchical-nonemitting-forward(L, ϕ)
- $\beta =$ hierarchical-nonemitting-backward(L, ϕ)
- $\langle \lambda^+, \lambda^- \rangle =$ hierarchical-nonemitting-expectations(L, α, β, ϕ)
- hierarchical-nonemitting-maximization$(L, \lambda^+, \lambda^-, \phi)$

hierarchical-nonemitting-forward(x_1^T, ϕ)
- for $t = 1$ to $T + 1$
    - for $i = n$ downto 0
        - $\alpha[\mathrm{t,i}] = 0$
- $\alpha[0{,}0] = 1$
- for $t = 1$ to T
    - for $i =$ min(t, n) downto 0
        - $p_{event} = \alpha[t,i]\lambda(x_{t-i}^{t-1})\delta(x_t \mid x_{t-i}^{t-1}, \phi) \prod_{j=i+1}^{\min(t,n)} (1 - \lambda(x_{t-j}^{t-1}))$
        - $\alpha[t{+}1,\min(i{+}1,n)] \mathrel{+}= p_{event}$
        - if$(i > 0)$ $\alpha[t, i-1] \mathrel{+}= (1.0\text{-}\lambda(x_{t-i}^{t-1}))\ \alpha[t,i]$
- return α

hierarchical-nonemitting-backward(x_1^T, ϕ)
- for $t = 1$ to T
    - for $i = n$ downto 0
        - $\beta[t,i] = 0$
- for $i = n$ downto 0
    - $\beta[t+1, i] = 1$
- for $t = T$ downto 1
    - for $i = 0$ to min(t, n)
        - if $(i > 0)$ $\beta[t,i] = (1\text{-}\lambda(x_{t-i}^{t-1}))\ \beta[t, i-1]$
        - $\beta[t,i] \mathrel{+}= \beta[t+1, \min(i+1, n)]\ \lambda(x_{t-i}^{t-1})\ \delta(x_t \mid x_{t-i}^{t-1}, \phi)$
- return β

Algorithm 15 cont'd Hierarchical Nonemitting Model Parameter Estimation Algorithm

hierarchical-nonemitting-expectations($x_1^T, \alpha, \beta, \phi$)
1. $p =$ hierarchical-nonemitting-evaluate(x, ϕ)
2. for $t = 1$ to T
3. previous-contexts $= \{\}$
4. for $i = \min(t, n)$ downto 0
5. $\gamma = \frac{\alpha[t,i]\beta[t+1,\min(i+1,n)]}{p}$
6. $\lambda^+(x_{t-i}^{t-1}) \mathrel{+}= \gamma$
7. $\forall\, c' \in$ previous-contexts $[\lambda^-(c') \mathrel{+}= \gamma]$
8.   previous-contexts $=$ previous-contexts $\cup\, x_{t-i}^{t-1}$
9. return $\langle \lambda^+, \lambda^- \rangle$

hierarchical-nonemitting-maximization($x, \lambda^+, \lambda^-, \phi$)
1. $\forall\, x' \in$ Substrings(x)
2. $\lambda(x') = \frac{\lambda^+(x')}{\lambda^+(x')+\lambda^-(x')}$
3. $\lambda^+(x') = 0;\ \lambda^-(x') = 0$

of times a lower order state was used instead. hierarchical-nonemitting-estimate first calls basic-estimate so that the δ parameters of the model are set to the maximum-likelihood transition parameters of D. Line 3 builds the α table in the forward pass. Line 4 builds the β table in the backward pass. Line 5 calculates and accumulates the γ values in the λ^+ and λ^- tables. Finally, line 6 of hierarchical-nonemitting-estimate calls hierarchical-nonemitting-maximization which adjusts the λ parameters according to the accumulated γ values, thereby increasing (or maintaining) the probability the model ϕ generated L. The estimation algorithm for the hierarchical-nonemitting model runs in $O(n^2T)$ time and $O(nT)$ space.

hierarchical-nonemitting-forward constructs the α table. Lines 7-8 of this algorithm show how the probability of using the state of order i at step t impacts the probability of reaching the state of order $\min(n, i+1)$ at step $t+1$.

hierarchical-nonemitting-backward builds the β table. Lines 8-9 enumerate the two ways of using the state of order i at step t. Line 9 is straightforward as it's clear that if the model uses a state of order i at step t then it must impact the state of order $\min(n, i+1)$ at step $t+1$. Line 8 represents the nonemitting transition from the state of order i at step t to the state of order $i-1$ at step t.

General Nonemitting Model

The parameter estimation algorithm for the general nonemitting model is very similar to the parameter estimation algorithm for the hierarchical nonemitting model. There is a forward pass that generates the α table such that the α entries represent equation 19. The backward pass produces the β tables where the β values are given by equation 20. The γ variable is given by:

$$\gamma[t,i,j] \doteq P(c_t = x_{t-i}^{t-1}, c_{t+1} = x_{t-j}^{t}, v = x_t \mid x, \phi) \tag{23}$$

$$= \frac{\alpha[t,i]\lambda(c_t \rightarrow c_{t+1})\delta(x_t|c_t)\beta[t+1,j]}{P(x \mid \phi)} \tag{24}$$

The γ's keep track of the probability of using a state of order i at step t, generating symbol x_t, and transitioning to a state of order j at step $t+1$. This additional dimension is a manifestation of the difference between the general and hierarchical nonemitting models. The hierarchical nonemitting model forces a particular transition to made, whereas the general nonemitting model has the freedom to choose the state to which it will transition. general-nonemitting-estimate, algorithm 16, runs in $O(n^2T)$ time and uses $O(nT)$ space.

General Nonuniform Model

The general nonuniform model's parameter estimation algorithm calculates $\alpha[t,i]$ during the forward pass:

$$\alpha[t,i] \doteq P(h = x_1^{t-1}, c = x_{t-i}^{t-1} \mid \phi) \tag{25}$$

where $\alpha[t,i]$ represents the probability of using the order i context at step t. The backward pass calculates:

$$\beta[t,i] \doteq P(x_t^T \mid h = x_1^{t-1}, c = x_{t-i}^{t-i}, \phi) \tag{26}$$

where $\beta[t,i]$ defines the probability of generating the final $T-t$ symbols of x given that the first $t-1$ symbols have been generated and the order i state was used at step t to condition the prediction.

Algorithm 16 General Nonemitting Model Parameter Estimation Algorithm

general-nonemitting-estimate(D, L, ϕ)
for $m = 1$ to $n + 1$
    basic-estimate(D, m, ϕ)
$\alpha =$ general-nonemitting-forward(L, ϕ)
$\beta =$ general-nonemitting-backward(L, ϕ)
$\langle \lambda^+, \lambda^- \rangle =$ general-nonemitting-expectations(L, α, β, ϕ)
general-nonemitting-maximization$(L, \lambda^+, \lambda^-, \phi)$

general-nonemitting-forward(x_1^T, ϕ)
for $t = 1$ to $T + 1$
    for $i = n$ downto 0
        $\alpha[\text{t,i}] = 0$
$\alpha[0{,}0] = 1$
for $t = 1$ to T
    for $i =$ min(t, n) downto 0
        for $j =$ min$(t + 1, n)$ downto 0
            $\alpha[t + 1, j] \mathrel{+}= \alpha[t, i]\ \lambda(x_{t-i}^{t-1} \rightarrow x_{t+1-j}^{t})\ \delta(x_t \mid x_{t-i}^{t-1}, \phi)$
return α

general-nonemitting-backward(x_1^T, ϕ)
for $t = 1$ to T
    for $i = n$ downto 0
        $\beta[t, i] = 0$
for $i = n$ downto 0
    $\beta[t + 1, i] = 1$
for $t = T$ downto 1
    for $i = 0$ to min(t, n)
        for $j = 0$ to min$(t + 1, n)$
            $\beta[t, i] \mathrel{+}= \beta[t + 1, j)\ \lambda(x_{t-i}^{t-1} \rightarrow x_{t+1-j}^{j})\ \delta(x_t \mid x_{t-i}^{t-1}, \phi)$
return β

Algorithm 16 cont'd General Nonemitting Model Parameter Estimation Algorithm

general-nonemitting-expectations($x_1^T, \alpha, \beta, \phi$)
prob-x = hierarchical-nonemitting-evaluate(x, ϕ)
for $t = 1$ to T
  for $i =$ min(t, n) downto 0
    λ-sum $= 0$
    for $j =$ min$(t+1, n)$ downto 0
      $\gamma = \frac{\alpha[t,i]\lambda(x_{t-i}^{t-1} \to x_{t-j}^{t})\delta(x_t|x_{t-i}^{t-1})\beta[t+1,j]}{\text{prob-x}}$
      $\lambda^+(x_{t-i}^{t-1} \to x_{t+1-j}^{t})$ += γ
      $\lambda^-(x_{t-i}^{t-1} \to x_{t+1-j}^{t})$ -= γ
      λ-sum += γ
    for $j = n$ downto 0
      $\lambda^-(x_{t-i}^{t-1} \to x_{t+1-j}^{t})$ += λ-sum
return $\langle \lambda^+, \lambda^- \rangle$

general-nonemitting-maximization($x, \lambda^+, \lambda^-, \phi$)
$\forall\ x' \in \text{Substrings}(x)$
  $\lambda(x') = \frac{\lambda^+(x)}{\lambda^+(x)+\lambda^-(x)}$
  $\lambda^+(x') = 0$
  $\lambda^-(x') = 0$

As before, the α and β values can be combined to calculate the probability that a state of order i was used at step t to condition the prediction. With the general nonuniform model, we must also calculate the probability that a prediction of length j was made from the state or order i at step t. Therefore,

$$\gamma[t,i,j] = \frac{\alpha[t,i]\rho(j|x_{t-i}^{t-1})\delta(x_t^{t+j-1}|x_{t-i}^{t-1})\prod_{k=1}^{j-1}(1-\rho(k|x_{t-i}^{t-1}))\beta[t+j]}{P(x|\phi)} \tag{27}$$

The estimation process for the general nonuniform model takes $O(n^3T)$ time and uses $O(nT)$ space (algorithm 17). The β table can be made one-dimensional as the likelihood of predicting the remaining symbols in equation 27 is equivalent to $\sum_{i=0}^{n}\beta[t+j,i]$.

Collapsed Nonuniform Model

The collapsed nonuniform parameter estimation algorithm, algorithm 18, differs from the general nonuniform's algorithm in that the λ parameters of the collapsed nonuniform model serve the same purpose as both the λ and ρ parameters of the general nonuniform model. That is, the collapsed nonuniform model's λ dictate the degree to which a particular context is used as well as the likelihood that a prediction of length j is made. As a result the α, β, and γ equations have the ρ parameters replaced with $1-\lambda$'s. Algorithm 18 has the same computational properties as 17.

Algorithm 17 General Nonuniform Model Parameter Estimation Algorithm

general-nonuniform-estimate(D, L, ϕ)
general-nonuniform-delta-estimate(D, ϕ)
$\alpha =$ general-nonuniform-forward(L, ϕ)
$\beta =$ general-nonuniform-backward(L, ϕ)
$\langle \lambda^+, \lambda^-, \rho^+, \rho^- \rangle =$ general-nonuniform-expectations(L, α, β, ϕ)
general-nonuniform-maximization$(L, \lambda^+, \lambda^-, \rho^+, \rho^-, \phi)$

general-nonuniform-delta-estimate(x, ϕ)
$Y = \{y : y \in \mathrm{Substrings}(x) \wedge |y| = n+1\}$
$\forall\ y = y_1^{n+1} \in Y$
    for $t = 0$ to $n+1$
        for $i = 0$ to min(t, n)
            for $j = 1$ to $n+1-i$
                $\delta(y_t^{t+j-1} \mid y_{t-i}^{t-1}, \phi) \doteq \frac{\mathrm{c}(y_t^{t+j-1})}{\mathrm{c}(y_{t-i}^{t-1})}$

general-nonuniform-forward(x_1^T, ϕ)
    for $t = 1$ to $T+1$
        $\mathrm{p[t]} = 0$
        for $i = n$ downto 0
            $\alpha[\mathrm{t,i}] = 0$
    $\mathrm{p}[0] = 1$
    for $t = 1$ to T
        for $i =$ min(t, n) downto 0
            for $j = 1$ to min$(T - t, n+1-i)$
                $p_{event} = \mathrm{p}[t]\ \lambda(x_{t-i}^{t-1})\ \prod_{k=i+1}^{\min(t,n)}(1 - \lambda(x_{t-k}^{t-1}))$
                $\alpha[t, i]$ += p_{event}
                p_{event} ·= $\delta(x_t^{t+j-1} \mid x_{t-i}^{t-1}, \phi)\rho(j \mid x_{t-i}^{t-1}) \prod_{k=1}^{j-1}(1 - \rho(k \mid x_{t-i}^{t-1}))$
                $\mathrm{p}[t+j]$ += p_{event}
    return α

general-nonuniform-backward(x_1^T, ϕ)
    for $t = 1$ to $T+1$
        $\beta[t] = 0$
    $\beta[t+1] = 1$
    for $t = T$ downto 1
        for $i =$ min(t, n) downto 0
            for $j = 1$ to min$(T - t, n+1-i)$
                $\beta[t]$ += $\beta[t+j)\ \lambda(x_{t-i}^{t-1})\ \prod_{k=i+1}^{\min(t,n)}(1 - \lambda(x_{t-k}^{t-1}))$
                $\delta(x_t^{t+j-1} \mid x_{t-i}^{t-1}, \phi)\ \rho(j \mid x_{t-i}^{t-1}) \prod_{k=1}^{j-1}(1 - \rho(k \mid x_{t-i}^{t-1}))$
    return β

Algorithm 17 cont'd General Nonuniform Model Parameter Estimation Algorithm

general-nonuniform-expectations$(x_1^T, \alpha, \beta, \phi)$

$p =$ general-nonuniform-evaluate(x, ϕ)
for $t = 1$ to T
    previous-contexts $= \{\}$
    for $i = \min(t, n)$ downto 0
        previous-predictions $= \{\}$
        for $j = 1$ to $\min(T - t, n + 1 - i)$
            $\gamma = \frac{\alpha[t,i]\rho(j \mid x_{t-i}^{t-1}))\delta(x_t^{t+j-1} \mid x_{t-i}^{t-1}),\phi)\prod_{k=1}^{j-1}(1-\rho(k \mid x_{t-i}^{t-1}))\beta[t+j]}{p}$
            $\lambda^+(x_{t-i}^{t-1}) \mathrel{+}= \gamma$
            $\rho^+(j \mid x_{t-i}^{t-1}) \mathrel{+}= \gamma$
            $\forall p' \in$ previous-predictions
                $\rho^-(p') \mathrel{+}= \gamma$
            previous-predictions $=$ previous-predictions $\cup \left\langle x_{t-i}^{t-1}, j \right\rangle$
            $\forall c' \in$ previous-contexts
                $\lambda^-(c') \mathrel{+}= \gamma$
        previous-contexts $=$ previous-contexts $\cup\, x_{t-i}^{t-1}$
return $\langle \lambda^+, \lambda^-, \rho^+, \rho^- \rangle$

general-nonuniform-maximization$(x, \lambda^+, \lambda^-, \rho^+, \rho^-, \phi)$

$\forall\, x' \in$ Substrings(x)
    $\lambda(x') = \frac{\lambda^+(x')}{\lambda^+(x') + \lambda^-(x')}$
    for $j = 1$ to $n + 1 - |x'|$
        $\rho(j \mid x') = \frac{\rho^+(j \mid x')}{\rho^+(j \mid x') + \rho^-(j \mid x')}$
        $\rho^+(j \mid x') = 0$
        $\rho^-(j \mid x') = 0$
    $\lambda^+(x') = 0$
    $\lambda^-(x') = 0$

Algorithm 18 Collapsed Nonuniform Model Parameter Estimation Algorithm

collapsed-nonuniform-estimate(D, L, ϕ)
collapsed-nonuniform-delta-estimate(D, ϕ)
$\alpha =$ collapsed-nonuniform-forward(L, ϕ)
$\beta =$ collapsed-nonuniform-backward(L, ϕ)
$\langle \lambda^+, \lambda^- \rangle =$ collapsed-nonuniform-expectations(L, α, β, ϕ)
collapsed-nonuniform-maximization$(L, \lambda^+, \lambda^-, \phi)$

collapsed-nonuniform-delta-estimate(x, ϕ)
$Y = \{y : y \in \mathrm{Substrings}(x) \wedge |y| = n+1\}$
$\forall\ y = y_1 \ldots y_{n+1} \in Y$
for $t = 0$ to $n+1$
  for $i = 0$ to min(t, n)
    for $j = 1$ to $n + 1 - i$
      $\delta(y_t^{t+j-1} \mid y_{t-i}^{t-1}, \phi) \doteq \frac{\mathrm{c}(y_t^{t+j-1})}{\mathrm{c}(y_{t-i}^{t-1})}$

collapsed-nonuniform-forward(x_1^T, ϕ)
for $t = 1$ to $T + 1$
  $\mathrm{p[t]} = 0$
  for $i = n$ downto 0
    $\alpha[\mathrm{t,i}] = 0$
$\mathrm{p}[0] = 1$
for $t = 1$ to T
  for $i =$ min(t, n) downto 0
    for $j = 1$ to min$(T - t, n + 1 - i)$
      $p_{event} = \mathrm{p}[t]\ \lambda(x_{t-i}^{t-1}) \prod_{k=i+1}^{\min(t,n)} (1 - \lambda(x_{t-k}^{t-1}))$
      $\alpha[t, i] \mathrel{+}= p_{event}$
      $p_{event} \cdot= \delta(x_t^{t+j-1} \mid x_{t-i}^{t-1}, \phi)(1 - \lambda(x_{t-i}^{t+j-1})) \prod_{k=1^{j-1}} \lambda(x_{t-i}^{t+k-1})$
      $\mathrm{p}[t + j] \mathrel{+}= p_{event}$
return α

collapsed-nonuniform-backward(x_1^T, ϕ)
for $t = 1$ to $T + 1$
  $\beta[t] = 0$
$\beta[t + 1] = 1$
for $t = T$ downto 1
  for $i =$ min(t, n) downto 0
    for $j = 1$ to min$(T - t, n + 1 - i)$
      $\beta[t] = \beta[t + j]\ \lambda(x_{t-i}^{t-1}) \prod_{k=i+1}^{\min(t,n)} (1 - \lambda(x_{t-k}^{t-1}))$
      $\qquad \delta(x_t^{t+j-1} \mid x_{t-i}^{t-1}, \phi)\ (1 - \lambda(x_{t-i}^{t+j-1})) \prod_{k=1^{j-1}} \lambda(x_{t-i}^{t+k-1})$
return β

Algorithm 18 cont'd Collapsed Nonuniform Model Parameter Estimation Algorithm

collapsed-nonuniform-expectations$(x_1^T, \alpha, \beta, \phi)$

prob-x = collapsed-nonuniform-evaluate(x, ϕ)

for $t = 1$ to T

  previous-contexts $= \{\}$

  for $i = \min(t, n)$ downto 0

    previous-predictions $= \{\}$

    for $j = 1$ to $\min(T - t, n + 1 - i)$

      $\gamma = \frac{\alpha[t,i](1-\lambda(x_{t-i}^{t+j-1}))\delta(x_t^{t+j-1} \mid x_{t-i}^{t-1}),\phi) \prod_{k=1}^{j-1}(1-\lambda(x_{t-i}^{t-k-1}))\beta[t+j]}{\text{prob-x}}$

      $\lambda^+(x_{t-i}^{t-1}) \mathrel{+}= \gamma$

      $\lambda^-(x_{t-i}^{t+j-1} \times j) \mathrel{+}= \gamma$

      $\forall p' \in$ previous-predictions

        $\lambda^+(p') \mathrel{+}= \gamma$

      previous-predictions = previous-predictions $\cup \left\langle x_{t-i}^{t-1}, j \right\rangle$

      $\forall c' \in$ previous-contexts

        $\lambda^-(c') \mathrel{+}= \gamma$

    previous-contexts = previous-contexts $\cup\, x_{t-i}^{t-1}$

return $\langle \lambda^+, \lambda^- \rangle$

collapsed-nonuniform-maximization$(x, \lambda^+, \lambda^-, \phi)$

$\forall\, x' \in \text{Substrings}(x)$

  $\lambda(x') = \frac{\lambda^+(x')}{\lambda^+(x') + \lambda^-(x')}$

  $\lambda^+(x') = 0$

  $\lambda^-(x') = 0$

2.5 Theoretical Results

This section explores how powerful, in a theoretic sense, each model class is relative to one another. A model class $\mathcal{X}$ is more powerful than a model class $\mathcal{Y}$ if $\mathcal{X}$ can represent a superset of the set of probability functions that $\mathcal{Y}$ can represent. Two model classes, $\mathcal{X}$ and $\mathcal{Y}$, are equivalent if any model from class $\mathcal{X}$ can be transformed into a model of class $\mathcal{Y}$, and vice-versa, that assigns the same probability to all possible strings drawn from the alphabet.

The theoretical results given in this section do not necessarily have empirical consequences. For example, two model classes $\mathcal{X}$ and $\mathcal{Y}$ could be proven to be theoretically equivalent yet models from class $\mathcal{X}$ could empirically out-perform models from a class $\mathcal{Y}$. Such a disparity could arise from the difference in parameter estimation schemes or sensitivity to the amount of available training data.

2.5.1 Basic and Classic Markov Models

We begin by showing the basic and classic Markov model classes are theoretically equivalent – that is they are able to represent the same class of probability functions. In practice, the classic Markov model is empirically superior because its parameters are more easily estimated.

Lemma 1 $\forall \phi \exists \phi' \forall x \in A^\star [P_c(x_1^T|\phi) = P_b(x_1^T|\phi')]$

Proof: By definition:

$$P_b(x_1^T|\phi') = \prod_{t=1}^{T} P_b(x_t|x_{t-n}^{t-1}, \phi')$$

where

$$P_b(x_t|x_{t-n}^{t-1}, \phi') = \delta'(x_t|x_{t-n}^{t-1}, \phi')$$

and

$$P_c(x_1^T|\phi) = \prod_{t=1}^{T} P_c(x_t|x_{t-n}^{t-1}, \phi)$$

where

$$P_c(x_t|x_{t-n}^{t-1},\phi') = \sum_{i=0}^{n} \delta(x_t|x_{t-i}^{t-1},\phi')\lambda(x_{t-i}^{t-1}) \prod_{j=i+1}^{n} (1-\lambda(x_{t-j}^{t-1},\phi'))$$

If we set:

$$\delta'(x_t|x_{t-n}^{t-1}) = \sum_{i=0}^{n} \delta(x_t|x_{t-i}^{t-1},\phi')\lambda(x_{t-i}^{t-1}) \prod_{j=i+1}^{n} (1-\lambda(x_{t-j}^{t-1},\phi'))$$

then $P_b(x_t|x_{t-n}^{t-1},\phi') = P_c(x_t|x_{t-n}^{t-1},\phi)$ $\forall[x_t$ and $x_{t-n}^{t-1}]$. Thus, $P_b(x_1^T|\phi') = P_c(x_1^T|\phi)$. $\Box$

Lemma 2 $\forall\phi\exists\phi'\forall x \in A^\star[P_c(x|\phi') = P_b(x|\phi)]$

Proof: If we set:

$$\lambda'(x^n) = 1$$

and

$$\delta'(x_t|x_{t-n}^{t-1},\phi') = \delta(x_t|x_{t-n}^{t-1},\phi)$$

then $P_b(x_t|x_{t-n}^{t-1},\phi) = P_c(x_t|x_{t-n}^{t-1},\phi')$ $\forall[x_t$ and $x_{t-n}^{t-1}]$. Thus, $P_b(x_1^T|\phi) = P_c(x_1^T|\phi')$. $\Box$

Theorem 1 *The class of classic mixture Markov models is equivalent to the class of basic Markov models.*

Proof: By lemmas 1 and 2. $\Box$

2.5.2 Nonemitting Models

We now show the class of hierarchical nonemitting model is strictly more powerful than the class of basic and classic Markov models. Lemma 3 establishes that any basic Markov model can be transformed into an equivalent hierarchical nonemitting model but lemma 4 demonstrates that there exists a hierarchical model that cannot be transformed into an equivalent basic Markov model. These two lemmas allow the conclusion that the class of hierarchical nonemitting models is strictly more powerful than the class of basic Markov models because it can represent a superset of the basic

Markov model distributions on strings. Furthermore, since the class of basic Markov models is equivalent to the class of classic mixture models, we may conclude that the hierarchical nonemitting model class is also strictly more powerful than the classic mixture model class.

Lemma 3 $\forall\phi\exists\phi'\forall x \in A^\star[P_h(x|\phi') = P_b(x|\phi)]$

Proof: By definition:

$$P_b(x_1^T|\phi) = \prod_{t=1}^{T} P_b(x_t|x_{t-n}^{t-1}, \phi)$$

where

$$P_b(x_t|x_{t-n}^{t-1}, \phi) = \delta(x_t|x_{t-n}^{t-1}, \phi)$$

and

$$P_h(x_1^T|\phi') = \sum_{i=0}^{n} P_h(x_1^{T-1}, s_t = x_{T-i}^{T-1}|\phi')\delta'(x_T|x_{T-i}^{T-1}, \phi')$$

where

$$P_h(x_1^{t-1}, s_t = x_{t-i}^{t-1}|\phi') =$$
$$\begin{cases} \lambda'(x_{t-i}^{t-1})[P_h(x_1^{t-2}, s_{t-1} = x_{t-1-n-1}^{t-2}|\phi')\delta'(x_{t-1}|x_{t-1-n-1}^{t-2}, \phi')+ & \\ \qquad P_h(x_1^{t-2}, s_{t-1} = x_{t-1-n}^{t-2}|\phi')\delta'(x_{t-1}|x_{t-1-n}^{t-2}, \phi')] & \text{if } i = n \\ \lambda'(x_{t-i}^{t-1})[P_h(x_1^{t-1}, s_t = x_{t-(i+1)}^{t-1}|\phi')\frac{(1-\lambda'(x_{t-(i+1)}^{t-1}))}{\lambda'(x_{t-(i+1)}^{t-1})}+ & \\ \qquad P_h(x_1^{t-2}, s_{t-1} = x_{t-1-i-1}^{t-2}|\phi')\delta'(x_{t-1}|x_{t-1-i-1}^{t-2}, \phi')] & 0 > i > n \\ P_h(x_1^{t-1}, s_t = x_{t-(i+1)}^{t-1}|\phi')\frac{(1-\lambda'(x_{t-(i+1)}^{t-1}))}{\lambda'(x_{t-(i+1)}^{t-1})} & i = 0 \end{cases}$$

If we set $\lambda(x^n) = 1$ and $\lambda(x^{<n}) = 0$ then:

$$P_h(x_1^{t-1}, s_t = x_{t-i}^{t-1}|\phi') = \begin{cases} P_h(x_1^{t-2}, s_{t-1} = x_{t-1-n}^{t-2}|\phi')\delta'(x_{t-1}|x_{t-1-n}^{t-2}, \phi') & \text{if } i = n \\ 0 & \text{otherwise} \end{cases}$$

Next, we set $\delta'(\sigma|x^n, \phi') = \delta(\sigma|x^n, \phi)\ \forall[\sigma, x^n]$. Thus,

$$P_h(x_1^{t-1}, s_t = x_{t-i}^{t-1}|\phi') = \begin{cases} P_h(x_1^{t-2}, s_{t-1} = x_{t-1-n}^{t-2}|\phi')\delta(x_{t-1}|x_{t-1-n}^{t-2}, \phi') & \text{if } i = n \\ 0 & \text{otherwise} \end{cases}$$

Since all $P_h(x_1^{t-1}, s_t = x_{t-i}^{t-1}|\phi') = 0$ for $i < n$, then:

$$P_h(x_1^T|\phi') = P_h(x_1^{T-1-n}, s_T = x_{T-n}^{T-1}|\phi')\delta(x_T|x_{T-n}^{T-1}, \phi')$$

If we unroll the recursion:

$$P_h(x_1^T|\phi') = \prod_{i=1}^{T} \delta(x_t|x_{t-n}^{t-1}, \phi) = P_b(x_1^T|\phi)$$

□

Lemma 4 $\exists\phi\forall\phi'\exists x \in A^\star[P_h(x|\phi) \neq P_b(x|\phi')]$

Proof: The idea of the proof is that the hierarchical nonemitting model can encode unbounded dependencies in its state distribution. Consider the order-2 hierarchical nonemitting model with A = $\{0, 1\}$, $\lambda(0) = 1$, $\lambda(1) = 0$, $\lambda(01) = 1$, and $\lambda(11) = 1$ on strings in $A1^\star A$. When $x_1 = 0$ x_2 will be predicted using $\delta(x_2|x_1)$ and all subsequent x_t will be predicted by the order-2 state $\delta(x_t|x_{t-2}^{t-1})$. When $x_0 = 1$ all subsequent x_t will be predicted by the order 0 state $\delta(x_t|\epsilon)$. Thus $P_h(A|01^p, \phi) \neq P_h(A|11^p, \phi)$ by construction for $p > n$ if $\delta(x_t|x_{t-2}^{t-1}, \phi) \neq \delta(x_t|x_{t-1}^{t-1}, \phi) \neq \delta(x_t|\epsilon, \phi)$.

The basic Markov model, $\phi' = \langle n', A, \delta' \rangle$, is incapable of representing this unbounded dependency. Consider the hierarchical model described in the previous paragraph and the strings $01^{n'+1}A$ and $11^{n'+1}A$. For the hierarchical model, $P_h(A|01^{n'+1}, \phi) \neq P_h(A|11^{n'+1}, \phi)$ by construction. Yet, for the basic Markov model $P_b(A|01^{n'+1}, \phi') = P_b(A|11^{n'+1}, \phi)$. Thus $0 \geq n' > \infty, \exists x \in A^{\geq n'+1}[P_h(x|\phi) \neq P_b(x|\phi')]$. □

Theorem 2 *The class of hierarchical nonemitting models is strictly more powerful than the class of basic Markov models*

Proof: By lemmas 3 and 4. [46]

We may then conclude that the hierarchical nonemitting model class is also more powerful than the class of classic mixture Markov models.

Corollary 1 *The class of hierarchical nonemitting mixture Markov models is strictly more powerful than the class of classic mixture Markov models.*

Proof: By theorems 1 and 2.

Lemma 5 demonstrates that any hierarchical nonemitting model can be converted to a general nonemitting model that assigns equivalent probability to all strings. This allows us to conclude, in theorem 3, that the general nonemitting model is strictly more powerful than the class of basic and classic Markov models. The question of equivalence between the hierarchical nonemitting and general nonemitting models is still an open question.

Lemma 5 $\forall \phi \exists \phi' \forall x \in A^\star [P_n(x|\phi') = P_h(x|\phi)]$

By definition:

$$P_h(x_1^T|\phi) = \sum_{i=0}^{n} P_h(x_1^T, s_{T+1} = x_{T+1-i}^T|\phi)$$

where

$$\begin{array}{l} P_h(x_1^{t-1}, s_t = x_{t-i}^{t-1}) = \\ \begin{cases} \lambda(x_{t-i}^{t-1})[P_h(x_1^{t-2}, s_{t-1} = x_{t-1-n}^{t-2}|\phi)\delta(x_{t-1}|x_{t-1-n}^{t-2}, \phi) + \\ \qquad P_h(x_1^{t-2}, s_{t-1} = x_{t-1-(n-1)}^{t-2}|\phi)\delta(x_{t-1}|x_{t-1-(n-1)}^{t-2}, \phi)] & \text{if } i = n \\ \lambda(x_{t-i}^{t-1})[P_h(x_1^{t-1}, s_t = x_{t-(i+1)}^{t-1}|\phi)\frac{(1-\lambda(x_{t-(i+1)}^{t-1}))}{\lambda(x_{t-(i+1)}^{t-1})} + \\ \qquad P_h(s_{t-1} = x_{t-1-(i-1)}^{t-2}|\phi)\delta(x_{t-1}|x_{t-1-(i-1)}^{t-2}, \phi)] & \text{if } 0 > i > n \\ P_h(x_1^{t-1}, s_t = x_{t-(i+1)}^{t-1}|\phi)\frac{(1-\lambda(x_{t-(i+1)}^{t-1}))}{\lambda(x_{t-(i+1)}^{t-1})} & \text{otherwise} \end{cases} \end{array}$$

and

$$P_n(x_1^T|\phi') = \sum_{i=0}^{n} P_n(x_1^T, s_{T+1} = x_{T+1-i}^T|\phi')$$

where

$$\begin{array}{l} P_n(x_1^{t-1}, s_t = x_{t-i}^{t-1}|\phi') = \\ \qquad \sum_{j=0}^{n} P_n(x_1^{t-2}, s_{t-1} = x_{t-1-j}^{t-2}|\phi')\frac{\lambda'(x_{t-1-j}^{t-2} \to x_{t-i}^{t-1})}{\sum_{k=0}^{n}\lambda'(x_{t-1-j}^{t-2} \to x_{t-k}^{t-1})}\delta'(x_{t-1}|x_{t-1-j}^{t-2}, \phi') \end{array}$$

If we set:

$$\lambda'(x_{t-1-j}^{t-1} \to x_{t-i}^{t-1}) = \begin{cases} \lambda(x_{t-i}^{t-1}) & \text{if } j = n \vee j+1 = i \\ \prod_{k=i+1}^{n}[(1-\lambda(x_{t-k}^{t-1}))]\lambda(x_{t-i}^{t-1}) & \text{if } j+1 > i \\ 0 & \text{otherwise} \end{cases}$$

and:

$$\delta'(x_{t-1}|x_{t-1-i}^{t-2},\phi') = \delta(x_{t-1}|x_{t-1-i}^{t-2},\phi)\sum_{j=0}^{n}\lambda'(x_{t-1-i}^{t-2} \to x_{t-j}^{t-1})$$

then

$$P_n(x_1^{t-1}, s_t = x_{t-i}^{t-1}|\phi') =$$
$$\begin{cases} \lambda(x_{t-n}^{t-1})[P_n(x_1^{t-2}, s_{t-1} = x_{t-1-n}^{t-2}|\phi')\delta(x_{t-1}|x_{t-1-n}^{t-2},\phi) + \\ \qquad P_n(x_1^{t-2}, s_{t-1} = x_{t-1-n+1}^{t-2})\delta(x_{t-1}|x_{t-1-n+1}^{t-2},\phi)] & \text{if } i = n \\ \lambda(x_{t-i}^{t-1})[P_n(x_1^{t-2}, s_{t-1} = x_{t-1-(i-1)}^{t-2}|\phi')\delta(x_{t-1}|x_{t-1-(i-1)}^{t-2},\phi) + \\ \quad \sum_{j=i}^{n}[P_n(x_1^{t-2}, s_{t-1} = x_{t-1-j}^{t-2}|\phi')\delta(x_{t-1}|x_{t-1-j}^{t-2},\phi)\prod_{k=min(n,j+1)}^{n}(1-\lambda(x_{t-k}^{t-1}))]] & \text{if } n > i > 0 \\ \sum_{j=0}^{n}[P_n(x_1^{t-2}, s_{t-1} = x_{t-1-j}^{t-2}|\phi')\delta(x_{t-1}|x_{t-1-j}^{t-2},\phi)\prod_{k=1}^{n}(1-\lambda(x_{t-k}^{t-1}))] & \text{if } i = 0 \end{cases}$$

We note that:

$$P_n(x_1^{t-1}, s_t = x_{t-(i+1)}^{t-1}|\phi')\frac{1-\lambda(x_{t-(i+1)}^{t-1})}{\lambda(x_{t-(i+1)}^{t-1})} =$$
$$\sum_{j=i}^{n}[P_h(x_1^{t-2}, s_{t-1} = x_{t-1-j}^{t-2})\delta(x_{t-1}|x_{t-1-j}^{t-2},\phi)\prod_{k=1}^{n}(1-\lambda(x_{t-k}^{t-1}))]$$

and therefore $P_h(x_1^{t-1}, s_t = x_{t-i}^{t-1}|\phi') = P_n(x_1^{t-1}, s_t = x_{t-i}^{t-1}|\phi)$. Thus, $\forall\phi, \exists\phi', \forall x \in A^\star, [P_h(x|\phi) = P_n(x|\phi')$. $\Box$

Theorem 3 *The class of general nonemitting models is strictly more powerful than the class of basic Markov and classic mixture Markov models because it is able to to represent a larger class of probability distributions on strings.*

Proof: By theorems 1, 2, and lemma 5.

2.5.3 Nonuniform Models

The focus is now placed on the strength of the nonuniform models. Theorem 4 concludes that the general and collapsed nonuniform models can represent equivalent sets of probability distributions. The relationship between the nonuniform model class and the nonemitting model classes is still undefined.

Lemma 6 $\forall\phi\exists\phi'\forall x \in A^\star[P_g(x|\phi') = P_o(x|\phi)]$

Proof: We can transform any collapsed nonuniform model ϕ to a general nonuniform model ϕ' that assigns equal probability to all strings by setting $\lambda'(x) = \lambda(x), \forall x$. Then we set $\rho'(j|x) = c, \forall x, j$ for $0 < c < 1$. Finally, we set:

$$\delta'(y^j|x^i) = \frac{1}{c(1-c)^{j-1}}\delta(y^j|x^i)(1-\lambda(x^iy^j))\prod_{k=1}^{j-1}\lambda(x^iy_1^k)$$

Thus, the ρ' parameters are ignored and the δ' function encodes the δ function and the prediction parameters given by $(1-\lambda(x^iy^j))\prod_{k=1}^{j-1}\lambda(x^iy_1^k)$. Note the c chosen must be the same over the entire model. $\Box$

Lemma 7 $\forall\phi\exists\phi'\forall x \in A^\star[P_o(x|\phi') = P_g(x|\phi)]$

Proof: We can transform any general nonuniform model ϕ to a collapsed nonuniform model that assigns the same probability to strings as the general nonuniform model by (1) setting $\lambda'(x) = \lambda(x), \forall x$ and (2):

$$\delta'(y^j|x^i) = \frac{1}{(1-\lambda(x^iy^j))\prod_{k=1}^{j-1}\lambda(x^iy_1^k)}\delta(y^j|x^i)\rho(j|x^i)$$

Here, the ρ function is encoded along with the δ function using the δ' function. The λ parameters that dictate the probability that the collapsed nonuniform model makes a multi-unit prediction are canceled out. $\Box$.

Theorem 4 *The class of general nonuniform models is equivalent to the class of collapsed nonuniform models.*

Proof: By lemmas 6 and 7. □

Lemma 8 establishes that any basic Markov model can be converted to an equivalent general nonuniform model. The analogous lemma establishing that there exists a general nonuniform model that is not equivalent to any basic Markov model is not provided although we strongly believe this to be the case.

Lemma 8 $\forall\phi\exists\phi'\forall x \in A^\star[P_g(x|\phi') = P_b(x|\phi)]$

Proof: First, set $\rho'(1|x) = 1\ \forall x$. Next set $\lambda'(x^n) = 1\ \forall x^n$. Finally, set $\delta'(y|x^i) = \delta(y|x^i)$. □

2.6 The Art of Parameter Estimation

The goal of parameter estimation, with respect to natural language text modeling, is to determine the value of the model's free parameters so as to maximize the performance of the model. The term performance typically refers to the model's predictive accuracy and one can estimate this accuracy by calculating the cross-entropy of a long string x that is representative of the natural language. The parameter settings of ϕ that maximizes $P(x|\phi)$ is deemed optimal. If the parameter estimation process can be conditioned on x, the task of finding optimal parameter values is straightforward as the relative frequencies of substrings of x can be used directly, i.e. $P_{ML}(x_t|y_{t-n}^{t-1}) = \frac{c(x_t|x_{t-n}^{t-1})}{c(x_{t-n}^{t-1})}$. Models developed in this chapter, however, do not as a general rule have access to x during the parameter estimation process. Rather, the model parameters must be estimated based upon another string y that has an undefined relationship to x.[6]. Since both x and y are finite in length, there is a low likelihood that the utilization of the maximum likelihood distribution of y would result in an accurate probability assignment to x as the maximum-likelihood distributions conditioned on y may assign zero probability to events that actually occur in x.

This chasm between the training string, y, and the testing string, x, poses a vexing challenge to the modeler: how does the model assign accurate probabilities to

[6]Usually, one can at least assume that x and y are drawn from the same natural language.

previously unseen strings based upon insufficient statistics? Because the relationship between x and y is undefined, it is impossible to define an algorithm that produces the maximum-likelihood estimator of x based on y with certainty. Nonetheless, this problem still exists and the predictive quality of a model often rests upon the modeler's ability to cope with this uncertainty.

In addition to moving towards mixture models, researchers have traditionally employed a small set of techniques to improve the quality of parameter estimation. These techniques can be considered to be ad-hoc in nature but they unquestionably improve model quality and any empirical evaluation that ignored such techniques would be void of value. As such we introduce and discuss two important schemes commonly used in the field: data withholding and mixture parameter tying.

2.6.1 Withholding Algorithms

Withholding algorithms simulate the occurrence of novel strings by dividing the training string into two substrings: one training substring and one validation substring. Thus, it is possible to ascertain, to a degree, the robustness of the probability distributions defined by the relative frequencies of the training string by examining their accuracy in assigning probability to the newly constructed validation string. This artificially constructed validation string is typically called the withheld string.

In general, the training data, O, is first divided into two sections: D and L. The δ functions are estimated directly from D and the λ, and possibly ρ, parameters are estimated so as to maximize the performance of the model on L given that the δ parameters were estimated from D. L is the validation string.

Withholding algorithms differ on how they divide O. The forward estimation algorithm divides O into D and L such that D is generally 85-95% of O and L is the remainder of the data. This algorithm is fast and can lead to reasonable parameter estimates if the number of parameters is relatively small compared to the length of the training string. k-fold cross estimation divides O into k, usually equally-sized, blocks. Each of these blocks is treated as L in turn and D is repeatedly adjusted to account for L. For each block, an expectation step is carried out over L. After each of

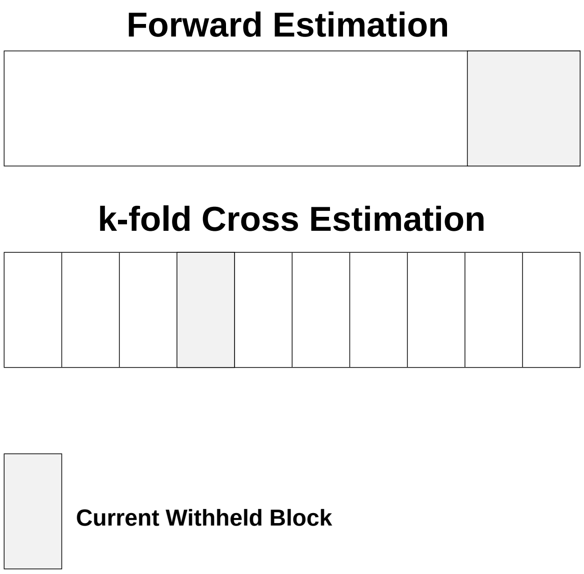

Figure 9: A graphical depiction of the difference between forward and cross estimation.

the blocks has been utilized as the withheld data, the maximization process is carried out considering the accumulated values from all of the expectation steps. k-fold cross estimation generally yields superior performance relative to forward estimation but is roughly k times slower than forward estimation. The difference between forward and k-fold cross estimation can be seen in figure 9.

2.6.2 Parameter Tying

Parameter tying is another popular technique used to improve the accuracy of probability assignments made by the model. $\lambda(x)$ is tied to $\lambda(y)$ if the states representing the strings x and y share the same λ parameter.[7] Tying the λ parameters of states x and y is desirable if the robustness of the resulting λ parameter is increased. The robustness of the resulting mixture parameter may increase because the amount of data to estimate the parameter will increase and therefore the estimation of the parameter could be more accurate. We can estimate the increase in robustness by comparing the

[7] We will slightly abuse notation in this section. We will use x to refer to both the string x and the state that represents the string x.

performance of model ϕ on some string z where x and y have separate λ parameters to the model performance of model $\hat{\phi}$ on z where x and y share one λ parameter. In principle, we could find the best tying scheme for a particular withheld data set by simply enumerating all possible tying schemes and determining which $\hat{\phi}$ maximizes model performance on that data set. Unfortunately, the number of possible tying schemes is huge. In general the number of ways to tie N strings with exactly k λ parameters is given by $S(N,k)$:

$$\begin{aligned} S(N,k) &= S(N-1,k-1) + kS(N-1,k) && (28) \\ &= \frac{1}{k!}\sum_{i=0}^{k}(-1)^i \begin{pmatrix} k \\ i \end{pmatrix} (k-i)^N && (29) \\ &= \sum_{i=0}^{k} \frac{(-1)^i}{i!} \frac{(k-i)^N}{(k-i)!} && (30) \end{aligned}$$

These $S(N,k)$'s are called *Stirling numbers of the second kind.*[6] The number of possible tying schemes for N strings, $T(N)$, is therefore:

$$T(N) = \sum_{k=1}^{N} S(N,k) \tag{31}$$

since the number of λ parameters is between 1 and N. A gross lower bound on this number for $N \geq 2$ and $k \geq 2$ is $\Omega(2^N)$. This bound can be obtained by considering arbitrary N for $k = 2$.

Clearly, it is impossible to enumerate all possible tying schemes in polynomial time. Given the impossibility of a complete enumeration, modelers tend to use a single or a small set of tying schemes motivated by a simple, overriding heuristic: tie the mixture parameters of two states, x_1^m and y_1^m, if the modeler is confident that the relationship between x_1^m and x_2^m and y_1^m and y_2^m, respectively, are similar. This is a reasonable heuristic as the mixture parameters represent the relative confidence possessed by the model to favor x_1^m (or y_1^m) over x_2^m (or y_2^m). Thus, it may make sense to tie states, x_1^m and y_1^m, if they have a similar relationship with their maximal suffix. The foremost question then becomes, how does the modeler measure the similarity in relationship two states have with their maximal suffix?

We consider nine different tying schemes in this chapter (this includes the scheme of no parameter tying). In particular we consider tying mixture parameters based on the discrete attributes of frequency and state order; continuous attributes such as codelength, divergence, and entropy; and joint attributes such as frequency $\times$ order, frequency $\times$ diversity, and frequency $\times$ order $\times$ diversity where diversity is the number of non-zero probability transitions from a particular state.

The discrete tying schemes tie the mixture parameter for states x and y if the desired attribute of state x is exactly the same as the attribute for y. If the tying scheme is joint, every attribute must be equal.

For continuous tying schemes such as codelength, we quantize the value of the attribute and tie the mixture parameters of state x and y if the quantized values of the attributes are exactly the same. Fixed-width quantization is the only quantization scheme considered – i.e. each quantum state is of an equal width r. Variable-width quantization schemes could certainly be used.

Frequency-Based Schemes

Frequency, frequency $\times$ order, frequency $\times$ diversity, and frequency $\times$ order $\times$ diversity are frequency-based tying schemes. Frequency-based schemes are motivated by the assumption that states that have similar frequencies will tend to have state output probability distributions that are roughly equivalent in terms of accuracy. If this assumption is correct, the model should have a similar level of confidence that either state should be utilized when possible. Since this is true, tying the mixture parameters of these states together should increase the robustness of the estimate since more data will be available to estimate the mixture parameter.

The joint frequency-based tying schemes impose further restrictions on which states are to be tied. For example, the frequency $\times$ order scheme requires that two states have the same frequency and the same order. This scheme alleviates the possibility of tying two states that have the same frequency but vastly different orders, e.g. it's difficult to motivate the tying of the states representing the strings q and *the United States of* even if both strings occurred c times in some training corpus.

Order Scheme

The order scheme is based on the assumption that models should be equally confident in using states of the same order. This assumption is clearly not correct in all cases, for example the states representing *ther* and *zqmg* should probably not be relied upon equivalently for typical English text simply because they have the same number of symbols. In normal English text, *ther* will occur many more times than *zqmg* and therefore the model should be more confident in the accuracy of *ther*'s state output probability distribution. Thus, it's probably the case that $\lambda(ther) > \lambda(zqmg)$.

Entropy Scheme

The entropy, $H(x_1^i)$ of a state, x_1^i is:

$$H(x_1^i) = -\sum_{a \in A} \delta(a|x_1^i) \log_2 \delta(a|x_1^i) \tag{32}$$

It's plausible to assume that if two state states have similar entropies, the reliability of their output distribution would also be similar. For example, if two states have entropies of zero, one could hypothesize that both distributions are very reliable as they can predict the next symbol with certainty and therefore their λ values should be one.

Divergence Scheme

The divergence, $D(x_1^i|\phi)$ of a state x_1^i is the Kullback-Liebler divergence between the state output probability distribution of x_1^i and the probability distribution of x_2^i given ϕ:

$$D(x_1^i|\phi) = \sum_{\sigma \in A} \delta(\sigma|x_1^i) \log_2 \frac{\delta(\sigma|x_1^i)}{P(\sigma|x_2^i, \phi)} \tag{33}$$

Two states that have nearly equivalent divergence values have similar relationships to their maximal suffix as their distributions better model potential continuations of the string to similar degrees.

Codelength Scheme

The codelength, $L(x_1^i|\phi)$ of a state x_1^i is the frequency-weighted Kullback-Liebler divergence between the state output probability distribution of x_1^i and the probability distribution defined by x_2^i given ϕ:

$$L(x_1^i|\phi) = \sum_{\sigma \in A} c(\sigma|x_1^i) \log_2 \frac{\delta(\sigma|x_1^i)}{P(\sigma|x_2^i, \phi)} \tag{34}$$

States representing the strings y_1^m and z_1^m that have nearly identical codelength values are similar as their addition to the model ϕ cause roughly equivalent reductions in the number of bits necessary to encode x_1^i. Since the marginal benefit of adding either string to the model is similar, it's plausible to assume that the degree to which the model uses either string y_1^m versus y_2^m or z_1^m versus z_2^m is likewise, similar.

The divergence and codelength tying schemes could be updated after each expectation-maximization(EM) iteration because the λ parameters of the model change after each maximization step. Hence, the probability that the that model assigns to a particular symbol given a string, $P(\cdot|x_2^i, \phi)$, may also change. This could impact which lambda parameters are tied together if the tying is updated after tying begins. We did not pursue such a dynamic tying scheme.

2.7 Empirical Results

The goal of this section is to determine the ramifications of model class, model order, parameter tying scheme, and withholding algorithm on the predictive capability of models. We evaluate model performance over five model classes, nine parameter tying schemes, two withholding algorithms, and up to nine model orders on three different corpora. For the WSJ 1989 corpus, we evaluated five model orders, 1-5, for a subtotal of 282 experiments. For the WSJ 1987-1989 corpus, we evaluate three model orders, 1-3, for a subtotal of 163 experiments. For the Brown corpus, we evaluate nine model orders, 1-9, for a subtotal of 502 experiments. In total, we report on 947 different natural language text models.

2.7.1 Experimental Paradigm

Experiments were performed on three widely-available corpora: the 6.2 million word WSJ 1989 corpus, the 42.3 million word WSJ 1987-1989 corpus, and the 6.0 million character Brown Corpus. We built word models for the WSJ experiments using verbalized punctuation and a vocabulary of approximately 20,000 words chosen from both the training and test sets. All out-of-vocabulary words were mapped to a single OOV symbol which we treated the same as any other symbol. We use a vocabulary of the 90 characters present in the Brown corpus for the character model runs.

In all experiments the first 90% of the corpus was used for training and the remaining 10% for testing. Per tradition, we report cross-perplexity values for the WSJ test corpora and cross-entropy (bits/char) values for the Brown experiments. The per symbol cross-entropy $H(M|\phi)$ of a test message $M = m_1, \ldots, m_{|M|}$ is:

$$H(M|\phi) = -\frac{1}{|M|} \log_2 \; P(M|\phi) \tag{35}$$

The per symbol perplexity, $\mathrm{PP}(M|\phi)$, of a test message is:

$$PP(M) = 2^{H(M|\phi)} \tag{36}$$

The results reported are after 10 iterations of the expectation-maximization (EM) algorithm. This last point warrants further clarification. We believe training should continue for a fixed number of EM iterations chosen prior to the experiment or, better yet, until EM converges. Reporting model performance after the EM iteration that maximizes model performance masks overfitting problems that many models exhibit. The choice of 10 iterations was seen as a compromise between computational cost and result quality.

We compare forward and k-fold cross estimation, nine mixture parameter tying schemes, up to five model classes, and different model orders for each corpus. The forward estimation experiments use the last 10% of the training corpus as the withheld block. The k-fold cross-estimation experiments use blocks that are approximately 250,000 symbols in size: 21 blocks were used for the Brown corpus, 22 blocks for the WSJ 1989 corpus, and 152 blocks for the WSJ 1987-1989 corpus. Preliminary experiments demonstrated that block size did not have a significant impact on performance

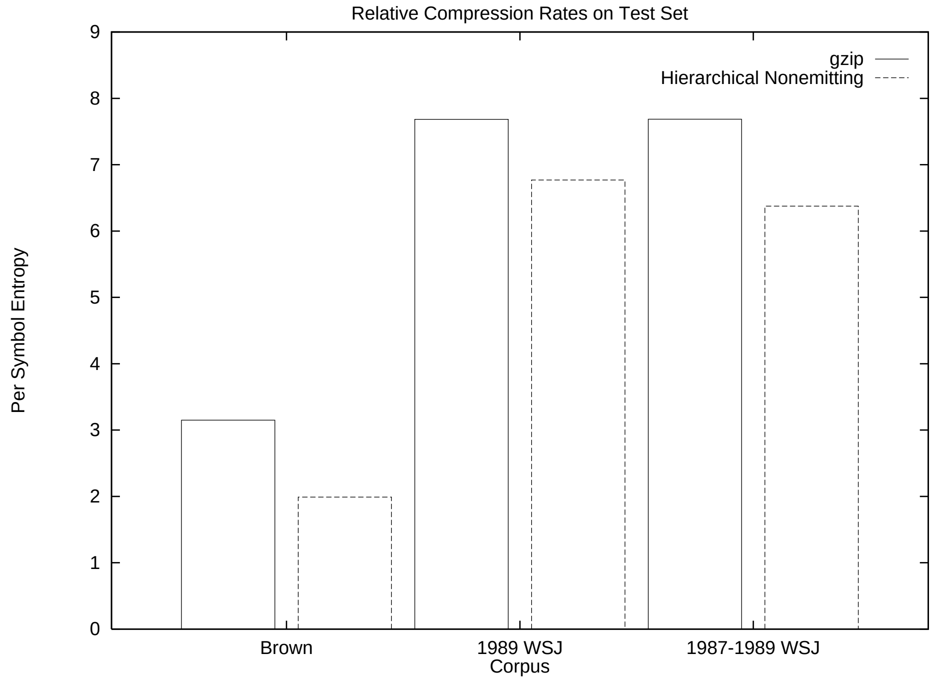

Figure 10: The relative compression rates of the test set material for gzip and the best hierarchical nonemitting models. This bar graph clearly demonstrates that the hierarchical nonemitting model gives superior compression rates relative to gzip. The Brown corpus was compressed using a 90 symbol alphabet, the 1989 WSJ had a 20293 symbol alphabet, and the 1987-1989 WSJ has a 20092 symbol alphabet.

and suggested that the given block sizes were good. The quantization width r of the codelength tying experiments was 1.0 while $r = 0.01$ for entropy and divergence tying. To avoid the possibility of assigning zero probability to an unseen continuation, the classic, hierarchical nonemitting, and collapsed nonuniform models smooth the order zero δ_0 distribution by adding $\frac{1}{16}$ to the frequency of each possible symbol. The general nonemitting and nonuniform models introduce an order -1 state that predicts each symbol with equal probability. Finally, all strings are included in the models – even those that occur only once.

2.7.2 A Comparison to gzip

We now compare the compression ability of the readily available gzip to the best hierarchical nonemitting models we were able to construct for all three corpora. Gzip is based on the LZ77 coding scheme introduced by Jacob Ziv and Abraham Lempel in 1977[64]. The family of Ziv and Lempel algorithms are dictionary-based compression approaches that are very popular primarily due the excellent real-world compression performance achieved with favorable time and space complexities. See Cleary, Bell, and Witten for a good in-depth description of many of the LZ algorithms[5].

The gzip algorithm encodes both the model and the uncompressed message in the compressed file, as any real-world compression system must. We are more interested, however, in the number of bits used to encode only the test message used in our experiments so that the predictive acumen of the models can be examined while ignoring the size of the respective models. To accomplish this, all reported gzip comparisons were made by (1) running gzip on the entire corpus to determine the number of bits needed to encode the entire corpus plus the model, $\mathcal{B}$, (2) running gzip on the training portion of the corpus to calculate the number of bits needed to encode the training corpus plus the model, $\mathcal{B}_{train}$ and (3) subtracting $\mathcal{B}_{train}$ from $\mathcal{B}$ to determine the number of bits needed to encode just the test corpus conditioned on the training corpus, $\mathcal{B}_{test}$. While the raw Brown corpus text was used to ascertain the entropy values reported in figure 10, the Wall Street Journal files were pre-processed by replacing each word with an integer representing its index in a dictionary. The dictionaries contained 20293 words for the 1989 WSJ Corpus and 20092 words for the 1987-1989 WSJ Corpus respectively. All out of vocabulary (OOV) words were mapped to a unique integer. The integers representing the dictionary entries were sorted according to the frequency with which a particular dictionary entry occcurred in the entire corpus. We also removed the impact of predicting spaces and newlines since the hierarchical nonemitting model did not make such predictions for the WSJ corpora.

Figure 10 clearly shows that the hierarchical nonemitting model is able to better model regularities in all three corpora. This strongly suggests that the hiearchical nonemitting model is a better model of natural language text than the LZ77 algorithm.

2.7.3 Summary of Results

Several results are contained in this section. We summarize these to focus their discussion.

- The hierarchical nonemitting and collapsed nonuniform models consistently outperform the classic model.
- Parameter tying can significantly improve model performance.
- Frequency based tying schemes (including codelength) work better than entropy, divergence, order, and no trying schemes.
- Withholding algorithms improve model quality.
- k-fold cross-estimation withholding yields superior results.
- Forward estimation withholding can closely approximate k-fold cross-estimation under certain conditions.
- Higher model orders are to be preferred for the hierarchical nonemitting model.

2.7.4 Explanation of Tables and Figures

An extensive set of tables and figures are presented throughout this section for purposes of examining the data from various perspectives.

Figures 11-14 describe the effect of parameter tying scheme on model performance as a function of corpus, withholding algorithm, and model class. These graphs examine the reduction or gain in entropy that is possible by employing an active tying scheme. It's clear from these tables that the frequency-based tying schemes consistently beat the order, divergence, entropy, and no tying schemes.

Table 3 contains the results of all classic, hierarchical nonemitting, and collapsed nonuniform models averaged over all tying schemes for both k-fold cross estimation and forward estimation. These results show that the hierarchical nonemitting model is, on the average, superior to either the classic or nonuniform models. The difference between the hierarchical nonemitting and classic model's perplexities approach 10% for higher orders. The table also demonstrates that k-fold cross-estimation is superior to forward estimation. The results in table 3 are visually represented in figures 15-20.

The best cross-perplexity and cross-entropy results for the classic, hierarchical nonemitting, and collapsed nonuniform models are presented in table 4. These results are presented for both k-fold cross-estimation and forward estimation. Table 4 clearly demonstrates that the hierarchical nonemitting and nonuniform models better the classic model in many experimental situations. The difference in model perplexities and entropies are rather small, however.

Table 5 isolates the ramifications of model class on cross-perplexity. The experiments were carried out using frequency $\times$ diversity tying over all three corpora, over all five model classes. This table demonstrates the superiority of the hierarchical nonemitting and collapsed nonuniform model classes for higher order models. The general nonemitting model performs well for lower-order models.

Figures 21-26 graph the relationship between the number of parameters contained in a model versus the performance of the model. There is one graph for each corpus $\times$ withholding algorithm. There is a point on the graph for every tying scheme for the highest order model for the classic, hierarchical nonemitting, and collapsed nonuniform model classes. These graphs show that there appears to be a "sweet spot" in the relationship between the number of model parameters and model performance. In other words, adding parameters does not always improve model perplexity.

The appendix of this chapter contains tables 7-16 that depict an unabridged, unfiltered, collection of all of the classic, hierarchical nonemitting, and collapsed nonuniform model experiments. The Wall Street Journal experiments are reported in entropy and perplexity.

We now consider the ramifications of modeling decisions such as model class, model order, parameter tying scheme, and withholding algorithm.

2.7.5 Impact of Parameter Tying Scheme

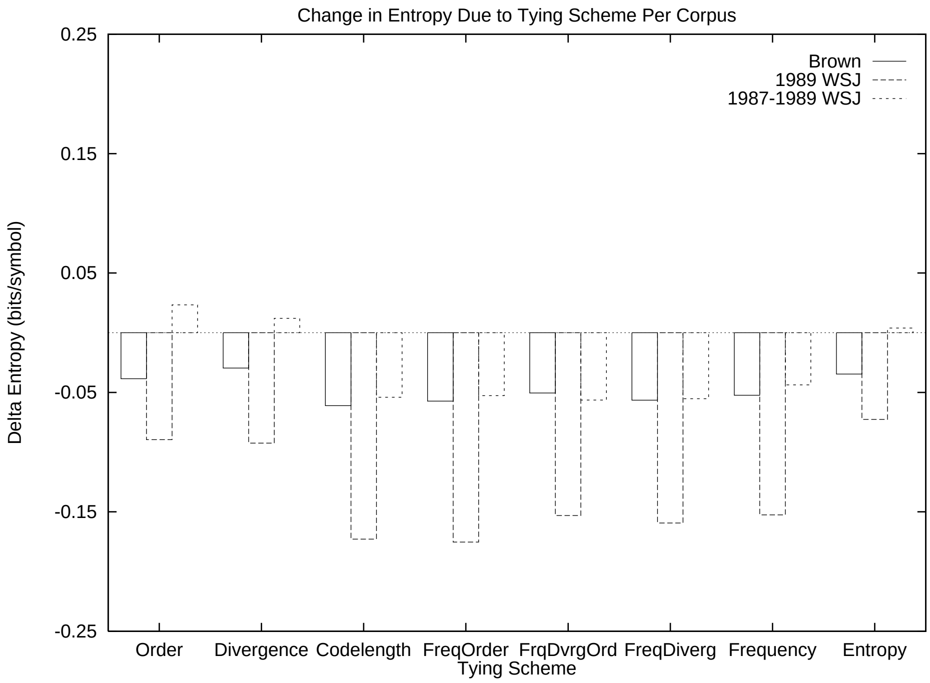

Figure 11: The difference in entropy according to tying scheme by corpus. The values were calculated by averaging the entropy reduction/increase over no parameter tying for all experiments for the same tying scheme and corpus. The graph shows that the 1989 WSJ models benefit most from parameter tying. The frequency-based and codelength schemes performed well.

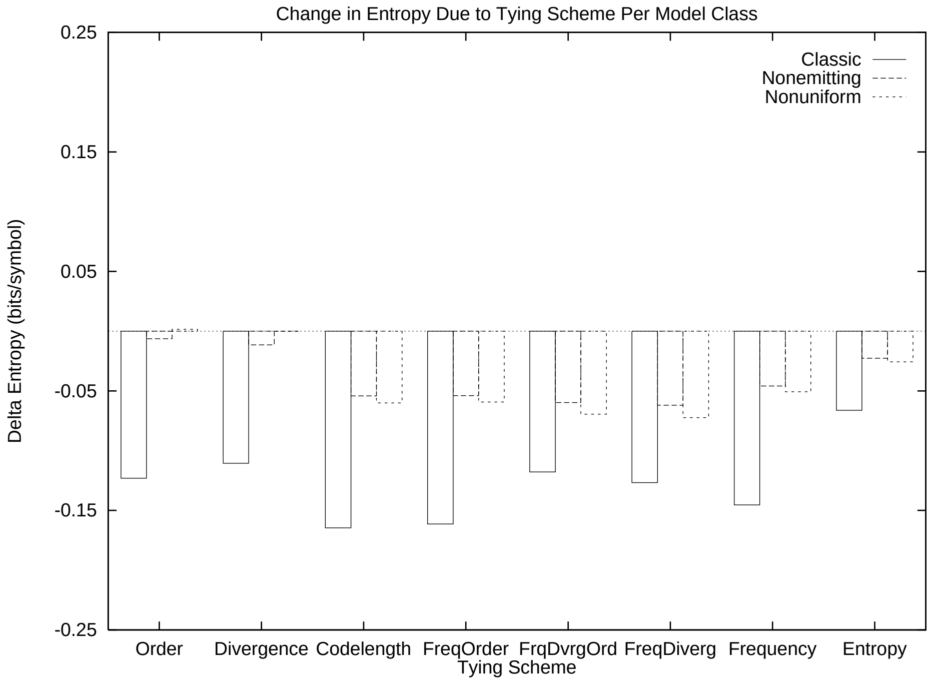

Figure 12: The difference in entropy according to tying scheme by model class. The values were calculated by averaging the entropy reduction/increase over no parameter tying for all experiments for the same tying scheme and model class. The graph shows that the classic model benefits substantially more from parameter tying than the other two model classes.

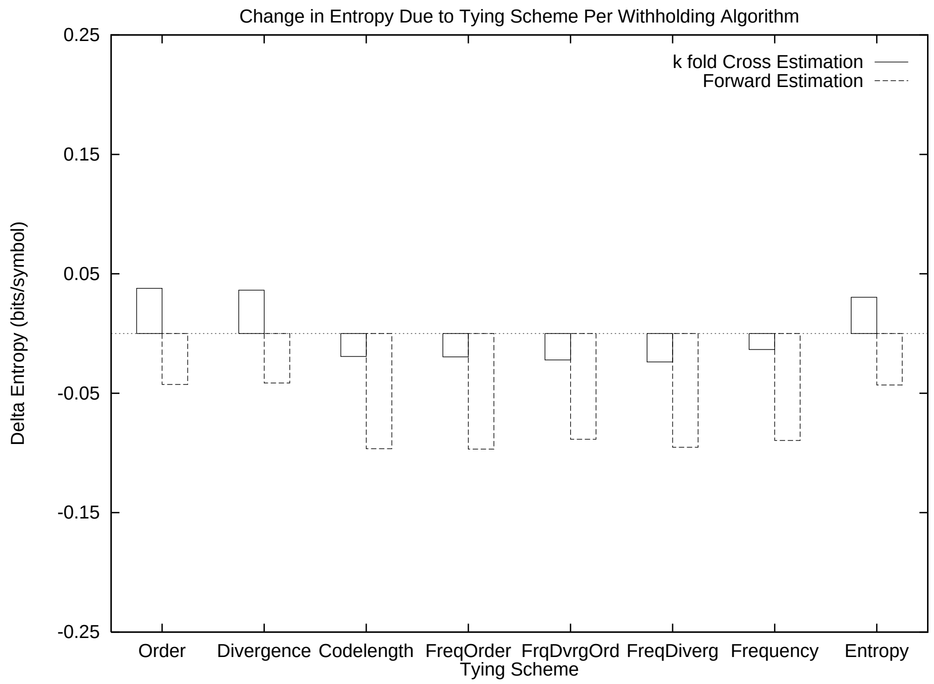

Figure 13: The difference in entropy according to tying scheme by model class. The values were calculated by averaging the entropy reduction/increase over no parameter tying for all experiments for the same tying scheme and withholding algorithm. The graph shows that the k-fold cross estimation algorithm's performance is actually damaged by non-frequency-based schemes whereas the forward estimation algorithm benefits from paramter tying.

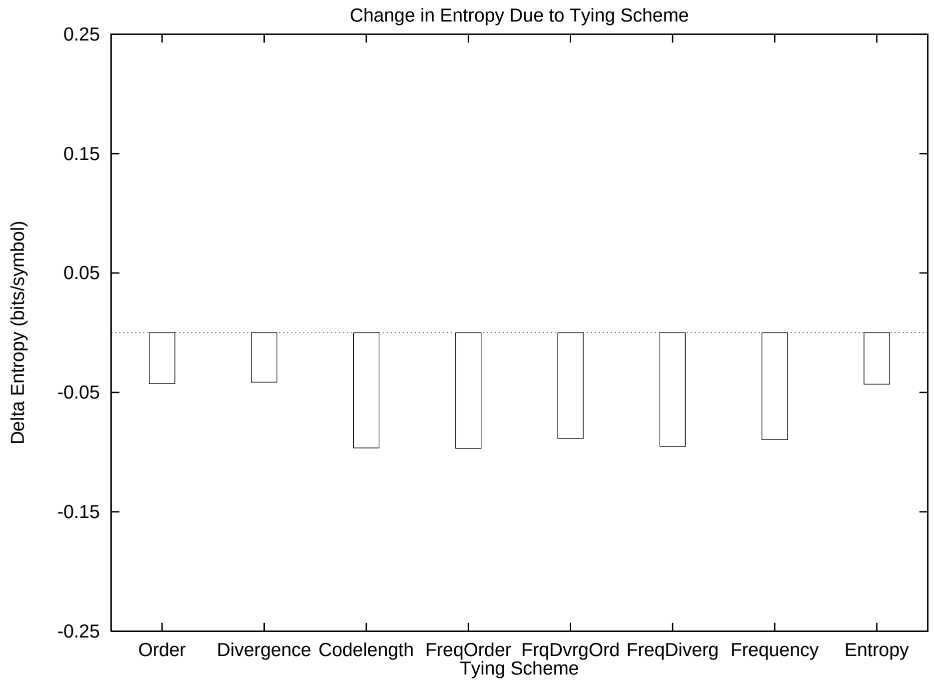

Figure 14: The difference in entropy according to tying scheme. The values were calculated by averaging the entropy reduction/increase over no parameter tying for all experiments for the same tying scheme. The graph shows that parameter tying helps on average and that the frequency-based and codelength schemes outperform the entropy, divergence, and entropy schemes.

The effect of parameter tying scheme can be substantial. For example, an order five classic model built for the WSJ 1989 using a frequency $\times$ order tying scheme outperforms the corresponding order five model with no parameter tying by over 39% measured by cross-perplexity. While typical reduction in cross-perplexity or cross-entropy is substantially less, the best models aggressively tie their parameters. Extensive parameter tying experiments were carried out over three corpora, three model classes, two withholding algorithms, various model orders, and nine parameter tying schemes. The general nonemitting and general nonuniform models were not included in the comparison due to computational limitations.

Overall, the frequency $\times$ diversity, codelength, and frequency $\times$ order schemes gave the best performance while the order, divergence, entropy and no tying schemes consistently underperformed the other schemes (see figures 11-14). The interaction between a parameter tying scheme and other model choices such as model class, withholding algorithm, and corpus were significant and these results are now discussed.

The frequency based tying schemes performed well for the Brown corpus experiments with the models built using the codelength scheme placing in first. The WSJ models were best served by the frequency $\times$ diversity scheme, with other frequency based schemes doing well.

If forward estimation withholding is used, parameter tying must be used to ensure the construction of quality models. A 16.9% reduction in entropy can be achieved by employing the codelength tying scheme instead of no tying for the order nine, forward estimation, Brown corpus model. A 39.2% reduction in perplexity can be realized by using frequency $\times$ order tying instead of no tying for the order five 1989 WSJ model estimated with forward estimation. Since parameter tying requires just a moderate implementation effort, it behooves modelers to seriously consider its use regardless of which withholding algorithm is employed.

The classic model sees significant gains from active parameter tying schemes while the hierarchical nonemitting model is the least sensitive to which tying scheme is used (tables 3,7-16). Thus, a good hierarchical nonemitting model is easier to build than a good classic model as the choice of tying scheme has less impact on the hierarchical nonemitting model.

As noted, divergence, order, and entropy tying schemes give poor results regardless of model class, corpus, or withholding algorithm. In addition, the no tying scheme performs abysmally if used with forward estimation. The poor performance of the order, entropy, and divergence schemes is easily explained. The order tying scheme restricts the model's freedom excessively as the number of free mixture parameters is n while the entropy tying scheme ties strings that have similar empirical distributions yet ignores frequency information.

For example, a string x that has a frequency of one will be tied to a string y that has a frequency of 100,000 and a symbol diversity of one. There is little reason to tie x and y's mixture parameters since one should be much more confident in the transition estimate provided by y than x. Simple modifications such as substituting Laplace's Law of Succession for the maximum-likelihood estimated of the δ functions would ameliorate this problem. In fact Laplace's law could be used temporarily – simply for purposes of calculating the empirical entropies for tying. Divergence, likewise, suffers from the lack of frequency data. The intuition that frequency plays a vital role in successful tying schemes is verified by the codelength scheme's outpeformance of the divergence scheme: since the codelength scheme is simply a frequency-weighted version of the divergence scheme.

The last point to be made in this subsection is that we did not investigate all sensible tying schemes due to computational limitations. Schemes such as frequency $\times$ entropy or frequency $\times$ codelength are easy to motivate and would have made for interesting comparisons. In fact, it seems likely that a scheme such as frequency $\times$ entropy would improve upon a related scheme such as frequency $\times$ diversity. Diversity can be viewed as an approximation to entropy as the entropy of a state is often intimately tied to the number of unique symbols that could be emitted from that state. In other words, it is often the case that the entropy of a state's conditional distribution moves in unison with the diversity of the state. In a sense both entropy and diversity approximate the uncertainty of predicting the next symbol conditioned on the state. Entropy, however, measures the degree of uncertainty more precisely than diversity, thereby suggesting that the frequency $\times$ entropy tying scheme would be likely to outperform the frequency $\times$ diversity tying scheme in many situations.

2.7.6 Impact of Withholding Algorithm

The k-fold cross-estimation withholding algorithm is superior to the forward estimation withholding algorithm (see tables 3,7-16). This result is in line with expectations as the cross-estimation withholding algorithm treats each section of the training corpus, in turn, as unseen data and the forward estimation withholding algorithm picks out one arbitrary section of the training corpus and treats it as the unseen data. Thus, modelers seeking the best possible cross-perplexity figures should unquestionably utilize the k-fold cross-estimation withholding algorithm.

Nonetheless, under many situations the two different withholding algorithms give nearly identical performance, suggesting that the substantial computational overhead of the k-fold cross-estimation withholding algorithm may not be a wise use of limited computational resources. We now examine when and why the k-fold cross-estimation algorithm is worthwhile and when it is not.

An example of the benefit of the k-fold cross-estimation withholding algorithm can be seen by analyzing the cross-perplexity results in table 5 for the general nonemitting model. Recall that the general nonemitting model has many more free parameters that it's hierarchical nonemitting cousin: $O(T^2n^2)$ vs. $O(Tn)$. This significant increase in parameters requires a commensurate amount of data to ensure that the mixture parameter estimation procedure is successful. The forward estimation algorithm uses only 10% of the available training corpus to estimate the mixture parameters for each model while the cross-estimation algorithm uses the entire corpus. This disparity in data size results in a substantial difference in cross-perplexity results. For example, cross-perplexity is reduced for the order four model on the WSJ1989 corpus by over 26.2% by opting for the k-fold cross-estimation withholding algorithm instead of the forward estimation withholding algorithm. Even more surprising is that these results hold for the excellent frequency $\times$ diversity tying scheme.

In addition, the use of k-fold cross-estimation is necessary to ensure good cross-perplexity numbers when used in combination with no parameter tying. This is especially true for the classic model. A 39.1% reduction in perplexity can be seen for the order five WSJ 1989 model trained with forward estimation and frequency $\times$ order tying as compared to the order five model with no parameter tying.

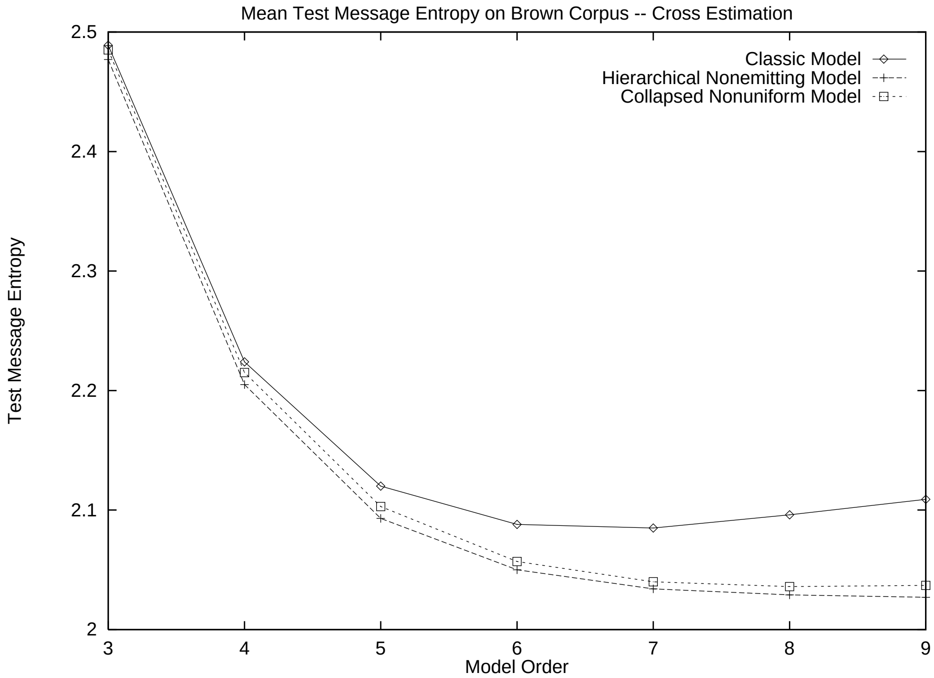

Figure 15: The mean test message entropy by model class on the Brown Corpus using cross estimation withholding.

Overall, the data suggests that the utility of the k-fold cross-estimation scheme increases as a function of the number of free parameters in the model as well as the amount of data available to train those parameters. Most models built from the Brown Corpus, for example, are relatively insensitive to the withholding algorithm used. This is because the number of free parameters in most models is relatively small and there is enough data for these parameters to be estimated. The general nonemitting model, however, bucks this trend and benefits greatly from the k-fold cross-estimation withholding algorithm – even for the Brown Corpus. Modelers must therefore be cognizant of the number of parameter in their models as well as the amount of data available to estimate those parameters. The likelihood that the cross-estimation withholding algorithm will give substantially better results than forward estimation increases as the number of parameters increase and the amount of data decreases.

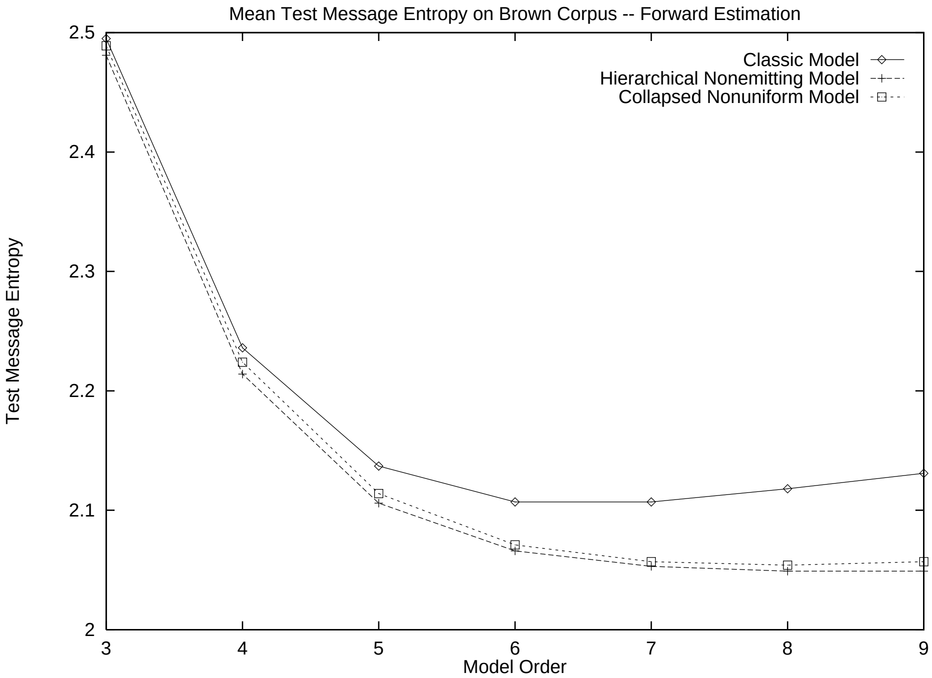

Figure 16: The mean test message entropy by model class on the Brown Corpus using forward estimation withholding.

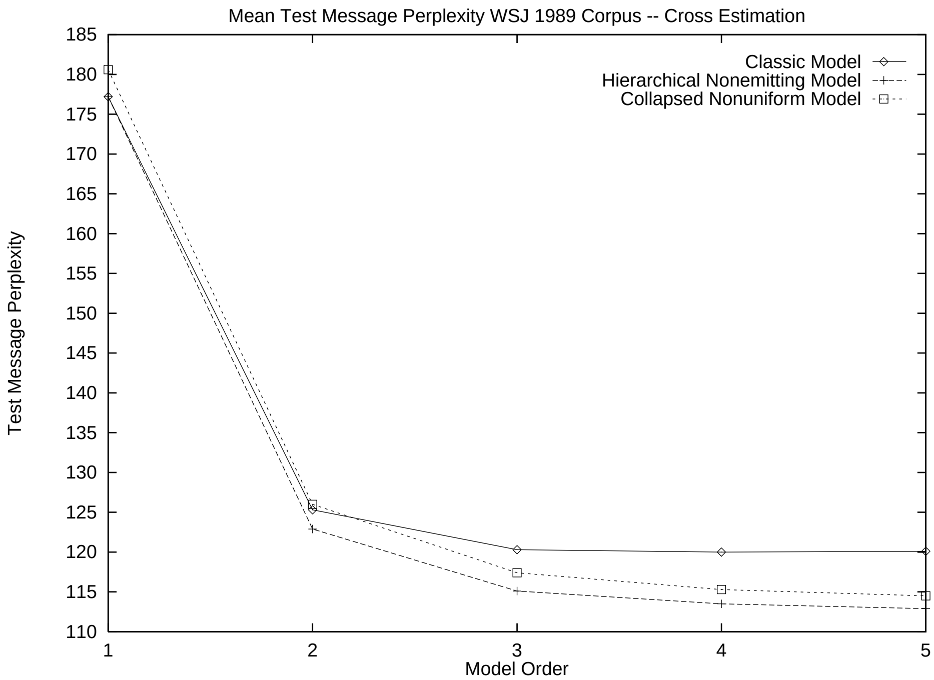

Figure 17: The mean test message perplexity by model class on the WSJ 1989 Corpus using cross estimation withholding.

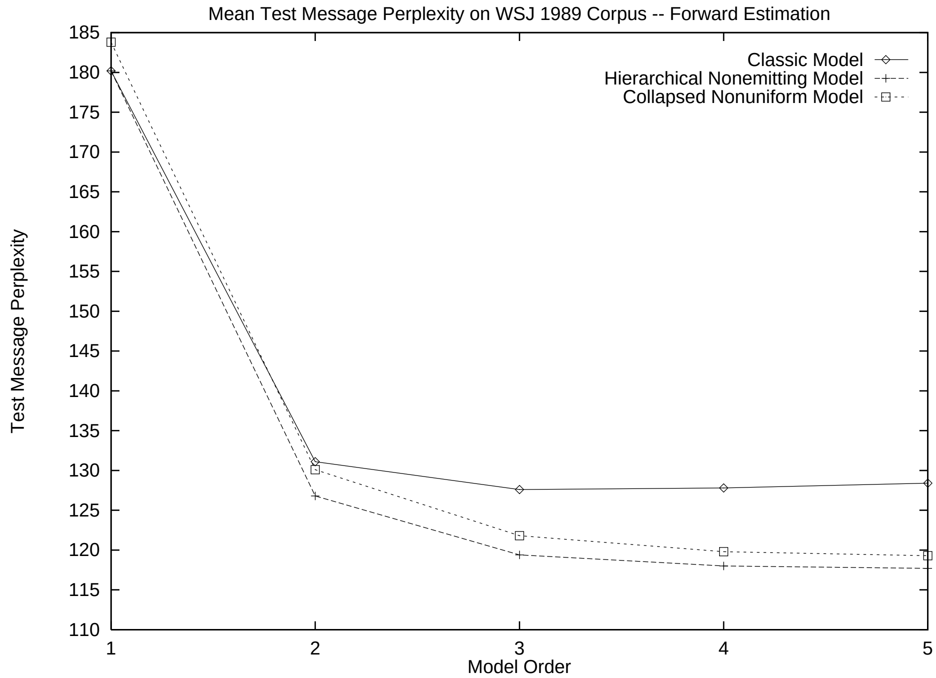

Figure 18: The mean test message perplexity by model class on the WSJ 1989 Corpus using forward estimation withholding.

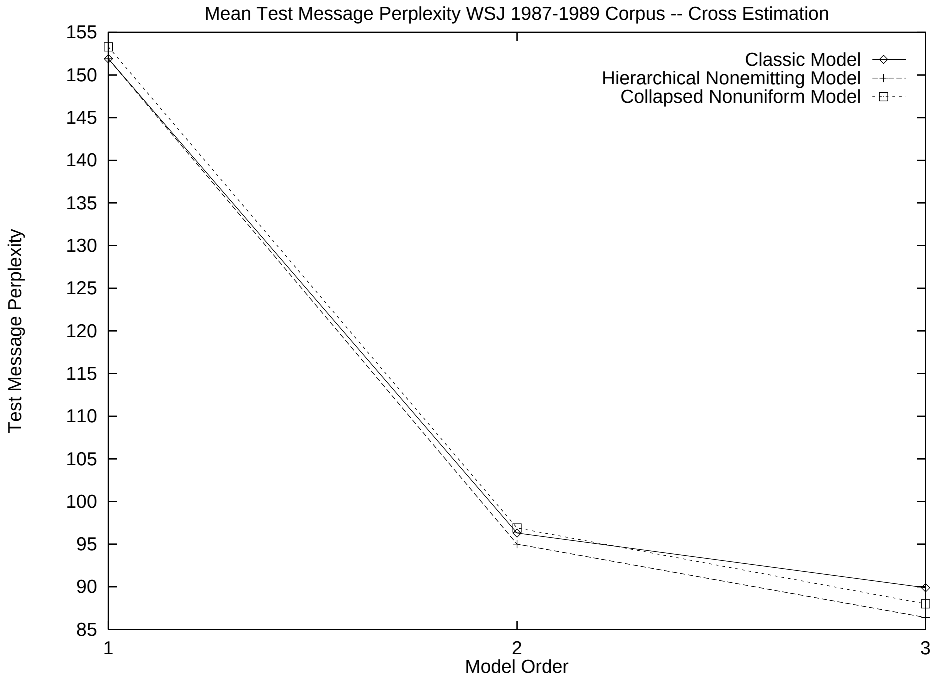

Figure 19: The mean test message perplexity by model class on the WSJ 1987-1989 Corpus using cross estimation withholding.

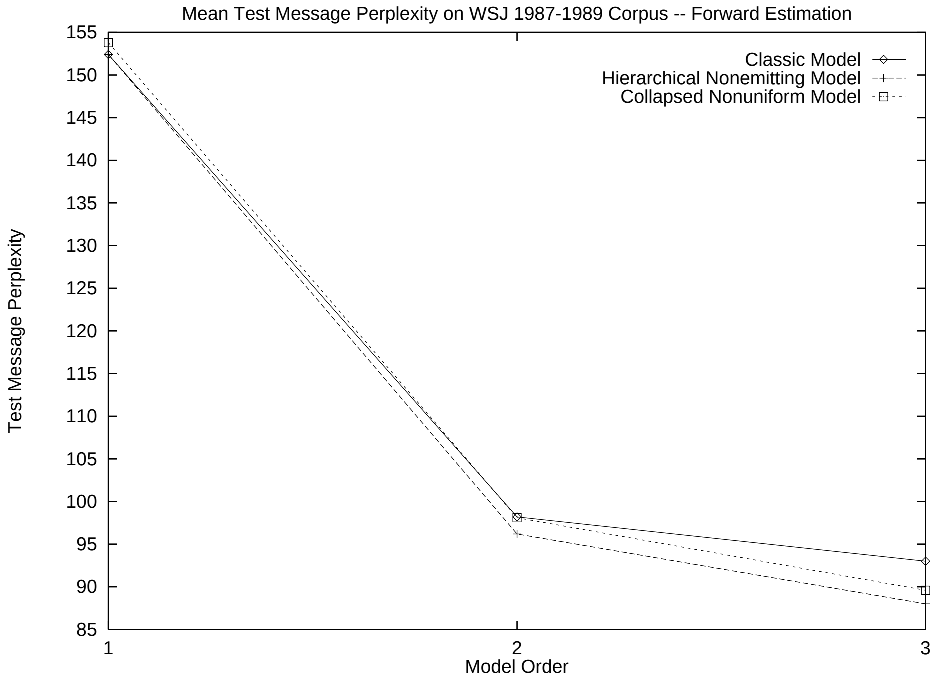

Figure 20: The mean test message perplexity by model class on the WSJ 1987-1989 Corpus using forward estimation withholding.

k-fold Cross-Estimation									
	WSJ 1987-1989			WSJ 1989			Brown		
		Hierch	Collap		Hierch	Collap		Hierch	Collap
N	Classic	Nonemit	Nonuni	Classic	Nonemit	Nonuni	Classic	Nonemit	Nonuni
1	**151.9**	**151.9**	153.3	**177.2**	**177.2**	180.6	**3.602**	**3.602**	3.607
2	96.3	**95.0**	96.9	125.3	**122.9**	126.0	2.950	**2.947**	2.952
3	89.9	**86.4**	88.0	120.3	**115.1**	117.4	2.489	**2.477**	2.485
4				120.0	**113.5**	115.3	2.224	**2.205**	2.215
5				120.1	**112.9**	114.5	2.120	**2.093**	2.103
6							2.088	**2.050**	2.057
7							2.085	**2.034**	2.040
8							2.096	**2.029**	2.036
9							2.109	**2.027**	2.037

Forward Estimation									
	WSJ 1987-1989			WSJ 1989			Brown		
		Hierch	Collap		Hierch	Collap		Hierch	Collap
N	Classic	Nonemit	Nonuni	Classic	Nonemit	Nonuni	Classic	Nonemit	Nonuni
1	152.4	152.4	153.8	180.2	180.2	183.8	3.603	3.603	3.609
2	98.2	96.2	98.1	131.1	126.8	130.1	2.951	2.947	2.952
3	93.0	88.0	89.6	127.6	119.4	121.8	2.495	2.481	2.489
4				127.8	118.0	119.8	2.236	2.214	2.224
5				128.4	117.7	119.3	2.137	2.106	2.114
6							2.107	2.066	2.071
7							2.107	2.053	2.057
8							2.118	2.049	2.054
9							2.131	2.049	2.057

Table 3: Perplexity and entropy results averaged over all tying schemes for the classic, hierarchical nonemitting, and collapsed nonuniform models over the three corpora. These results suggest the classic model is more sensitive to tying scheme due to the increase in disparity between the classic and hierarchical nonemitting models when compared to their respective numbers in table 9. The data also imply that k-fold cross estimation is better than forward estimation. The best model per order and corpus is in bold print.

2.7.7 Impact of Model Order

Increasing model order tends to reduce the cross-perplexity of a test message although with diminishing returns. Caution is warranted because increasing model order can actually increase perplexity because of the phenomenon of overfitting. Overfitting is said to occur when the test message perplexity of a model increases as model order increases beyond a certain point and this increase is attributed to the fact that there is too little data for too many parameters. Instead of learning the generative behavior of a particular context, which could be gleaned from many examples, the output distributions tend to memorize, or overfit, the one or small number of examples present in the data.

The hierarchical nonemitting model is virtually immune from this deficiency in our evaluation while the nonuniform model is afflicted by overfitting less often than the classic model. The classic and general nonuniform models both suffered from overtraining to a certain degree.

Still, the potential positive ramifications of increasing model order should not be overlooked. For example, a reduction in perplexity of 9.0% was realized by increasing the model order of a hierarchical nonemitting WSJ 1987-1989 model from 2 to 3, i.e. trigram to quadgram for the best nonhierarchical models.

Unfortunately, increasing model order can increase memory usage as well as increase training and evaluation time. The growth rate of memory usage for storing the training corpus $O = o_1 \ldots o_T$ is $O(nT)$ because for each model order, $0 > k \geq n$, there are up to $T - k$ unique strings in the corpus. An index into the corpus for each of these strings is stored in a hash table with each index taking $O(1)$ space.

2.7.8 Impact of Model Class

The hierarchical nonemitting model gives the best cross-perplexity results over many experiments. This reflects the hierarchical nonemitting model's relative insensitivity to parameter tying scheme, withholding algorithm, and corpus. The collapsed nonuniform model also performs very well even out-performing the hierarchical nonemitting model for some higher orders.

k-fold Cross-Estimation									
	WSJ 1987-1989			WSJ 1989			Brown		
		Hierch	Collap		Hierch	Collap		Hierch	Collap
N	Classic	Nonemit	Nonuni	Classic	Nonemit	Nonuni	Classic	Nonemit	Nonuni
1	**150.7**^{9}	**150.7**^{9}	151.7^{9}	**174.9**^{9}	**174.9**^{9}	177.4^{9}	**3.601**^{5}	**3. 601**^{5}	3.607^{4}
2	93.9^{9}	**92.0**^{9}	93.2^{9}	122.0^{1}	**118.9**^{9}	121.1^{9}	2.949^{1}	**2.946**^{1}	2.951^{1}
3	86.0^{8}	**83.0**^{9}	84.0^{9}	114.4^{8}	**111.0**^{9}	111.8^{7}	2.483^{1}	**2.473**^{9}	2.481^{9}
4				112.4^{8}	109.4^{9}	**109.3**^{9}	2.212^{1}	**2.192**^{9}	2.201^{9}
5				111.4^{1}	109.0^{7}	**108.4**^{9}	2.100^{1}	**2.075**^{9}	2.081^{7}
6							2.055^{1}	**2.027**^{9}	2.029^{7}
7							2.037^{1}	2.008^{9}	**2.007**^{7}
8							2.030^{1}	1.999^{9}	**1.996**^{7}
9							2.028^{1}	1.996^{9}	**1.992**^{7}

Forward Estimation									
	WSJ 1987-1989			WSJ 1989			Brown		
		Hierch	Collap		Hierch	Collap		Hierch	Collap
N	Classic	Nonemit	Nonuni	Classic	Nonemit	Nonuni	Classic	Nonemit	Nonuni
1	151.2^{1}	151.2^{1}	152.3^{1}	176.2^{8}	176.2^{8}	178.8^{8}	3.603^{2}	3.603^{2}	3.609^{1}
2	94.4^{1}	92.7^{7}	93.8^{7}	122.4^{8}	121.5^{7}	123.9^{7}	2.950^{4}	2.946^{1}	2.951^{1}
3	86.1^{8}	83.7^{7}	84.2^{7}	114.8^{8}	113.4^{7}	113.9^{7}	2.485^{1}	2.476^{1}	2.483^{1}
4				112.9^{8}	111.7^{8}	111.3^{7}	2.217^{1}	2.202^{7}	2.210^{7}
5				112.2^{8}	111.2^{8}	110.4^{7}	2.105^{1}	2.084^{7}	2.089^{7}
6							2.060^{1}	2.037^{7}	2.037^{7}
7							2.042^{1}	2.018^{7}	2.014^{7}
8							2.035^{1}	2.010^{7}	2.003^{7}
9							2.033^{1}	2.006^{7}	1.999^{7}

Table 4: The best perplexity and entropy results on all three corpora over all tying schemes for the classic, hierarchical nonemitting, and collapsed nonuniform models. The best values per order and corpus are in bold print. Notice that the k-fold cross estimation algorithm always outperforms forward estimation and that the nonemitting and nonuniform models consistently outperform the classic model for $n > 1$. The tying scheme that produced the best result is indicated with a superscript. Abbreviations: 1 = Codelength; 2 = Divergence e; 3 = Frequency; 4 = Order; 5 = Entropy; 6 = None; 7 = Frequency $\times$ Diversity; 8 = Frequency $\times$ Order; 9 = Frequ ency $\times$ Diversity $\times$ Order

Brown Corpus Forward Estimation					
N	Classic	Hierarchical Nonemitting	General Nonemitting	Collapsed Nonuniform	General Nonuniform
3	2.497	2.479	**2.469**	2.486	2.488
4	2.228	**2.202**	2.243	2.210	2.212
5	2.123	**2.084**	2.161	2.089	2.096
6	2.092	**2.037**	2.133	2.037	2.052
Brown Corpus Cross Estimation					
N	Classic	Hierarchical Nonemitting	General Nonemitting	Collapsed Nonuniform	General Nonuniform
3	2.486	2.473	**2.443**	2.481	2.480
4	2.218	2.192	**2.184**	2.201	2.200
5	2.112	**2.075**	2.077	2.081	2.084
6	2.082	**2.027**	2.035	2.029	2.041
WSJ 1989 Corpus Forward Estimation					
N	Classic	Hierarchical Nonemitting	General Nonemitting	Collapsed Nonuniform	General Nonuniform
2	126.5	**121.5**	161.3	123.9	126.2
3	122.9	**113.4**	158.1	113.9	126.1
4	120.9	**111.7**	158.0	111.3	128.3
WSJ 1989 Corpus Cross Estimation					
N	Classic	Hierarchical Nonemitting	General Nonemitting	Collapsed Nonuniform	General Nonuniform
2	122.8	**119.0**	124.7	121.3	123.4
3	119.0	**111.1**	117.8	111.8	123.9
4	117.2	109.5	116.5	**109.3**	126.2

Table 5: The impact of model class on predictive quality. This table demonstrates the empirical superiority of the hierarchical nonemitting model for the frequency $\times$ diversity parameter tying scheme.

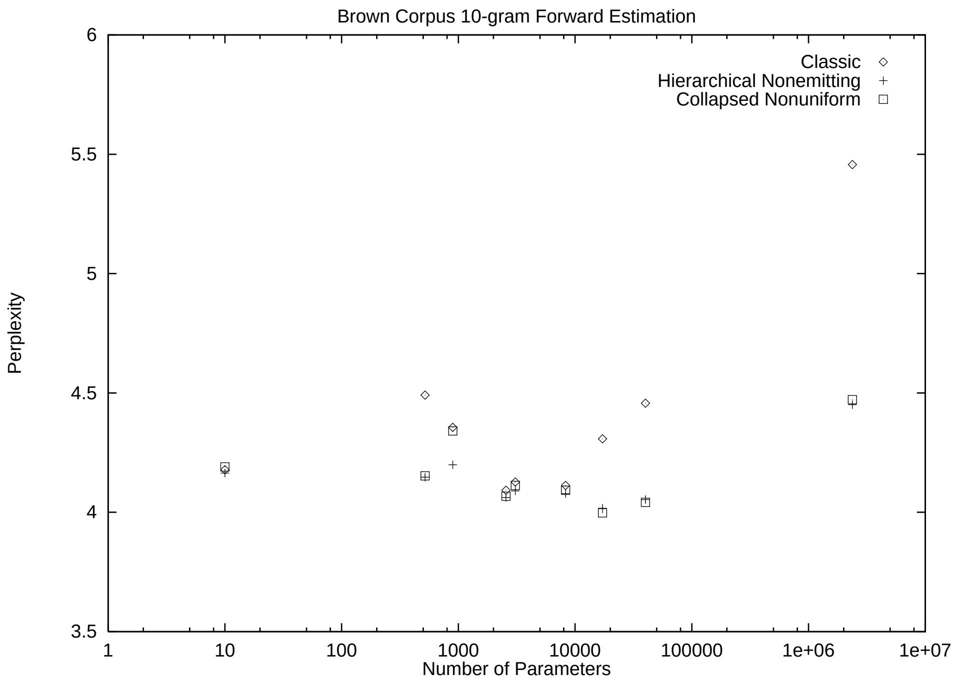

Figure 21: The impact of the number of free parameters on the cross-entropy of the Brown Corpus testing string. Each data point represents an order nine model estimated with the forward estimation withholding algorithm. There are points for each of the nine tying schemes. The classic model is unable to assimilate a large number of parameters, while the hierarchical nonemitting and nonuniform models do a much better job.

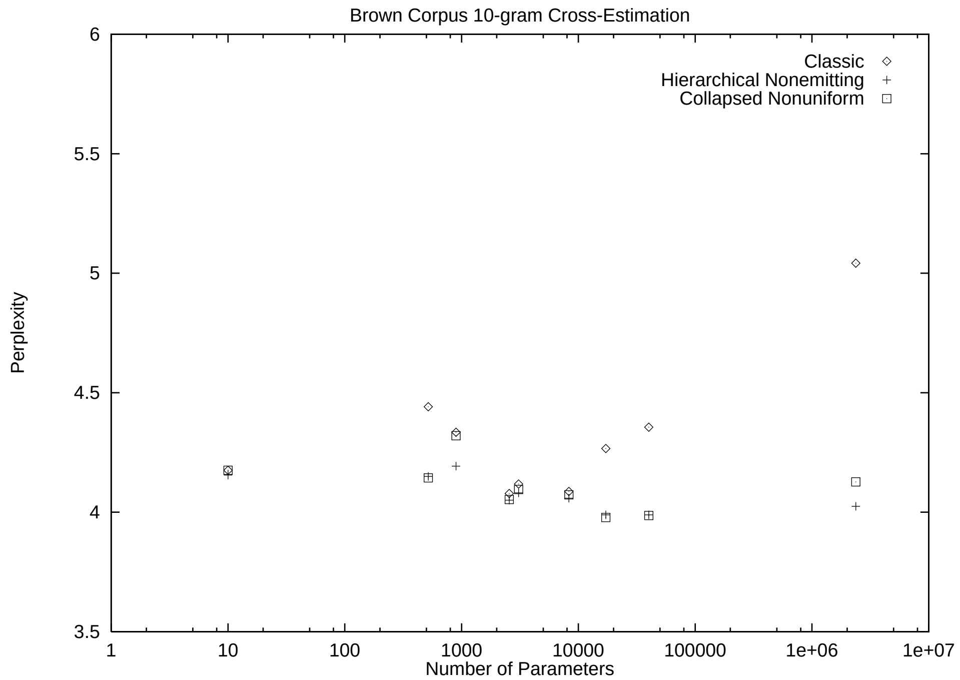

Figure 22: The impact of the number of free parameters on the cross-entropy of the Brown Corpus testing string. Each data point represents an order nine model estimated with the k-fold cross-estimation withholding algorithm. There are points for each of the nine tying schemes. This figure demonstrates the classic model's inability to absorb a large number of parameters.

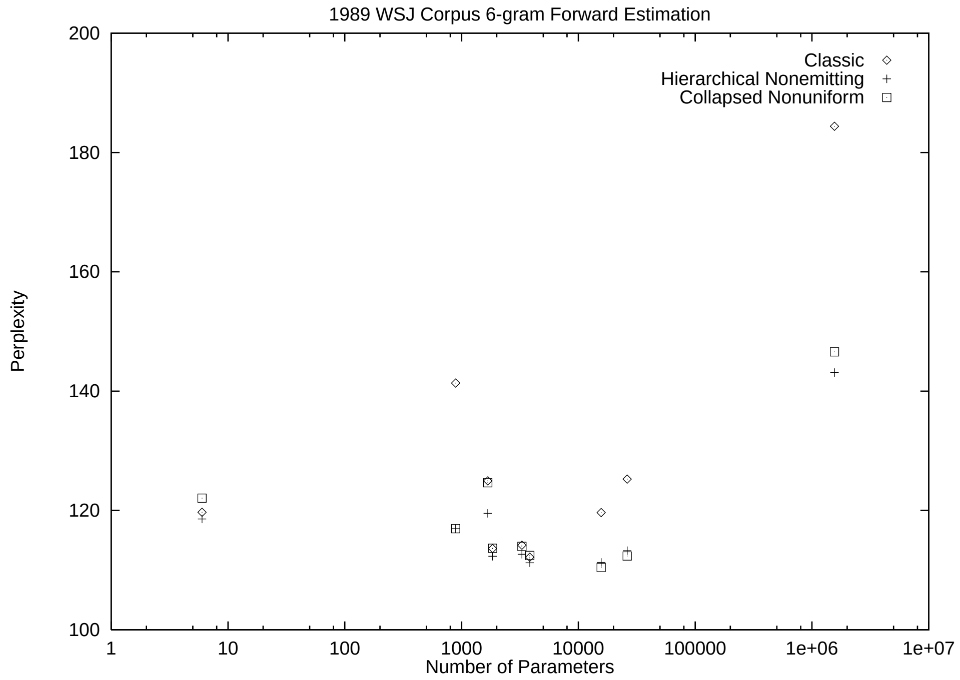

Figure 23: The impact of the number of free parameters on the cross-entropy of the 1989 WSJ testing string. Each data point represents an order five model estimated with the forward estimation withholding algorithm. There are points for each of the nine tying schemes. The graph demonstrates that none of the models are able to take advantage of a large number of parameters, with the classic model performing the worst.

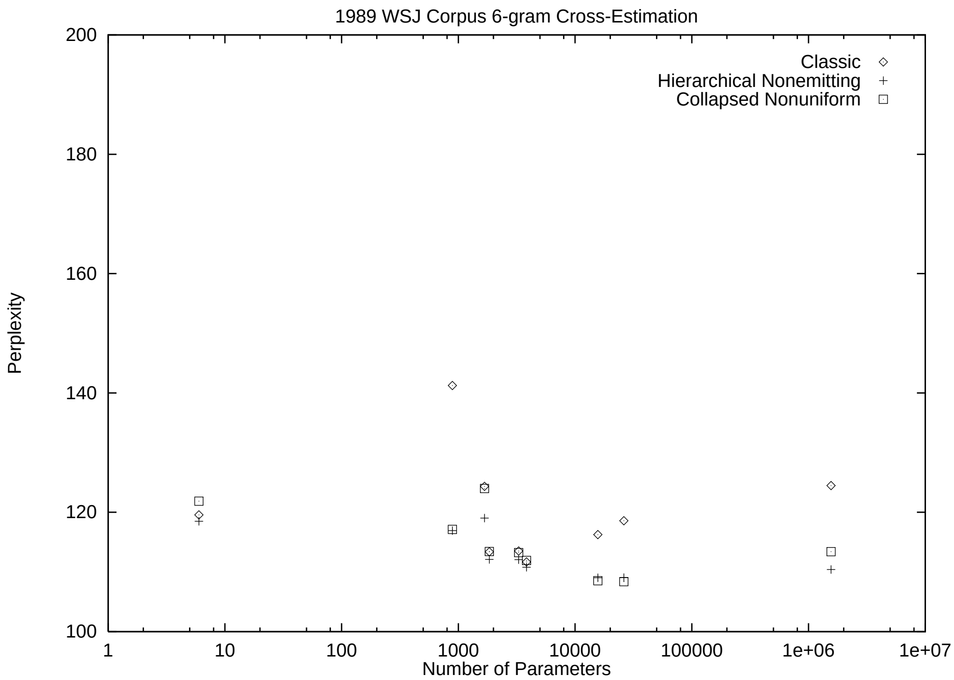

Figure 24: The impact of the number of free parameters on the cross-entropy of the 1989 WSJ testing string. Each data point represents an order five model estimated with the k-fold cross-estimation withholding algorithm. There are points for each of the nine tying schemes. This graph demonstrates that cross-estimation improves cross-perplexity, especially for higher order models. The classic model is still the least able to assimilate a large number of parameters.

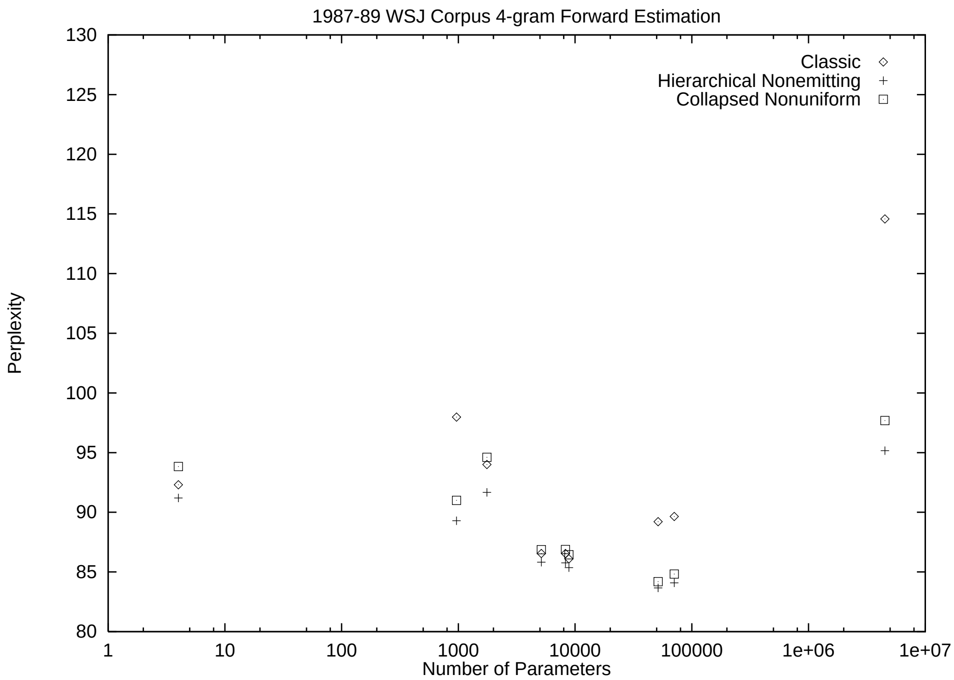

Figure 25: The impact of the number of free parameters on the cross-entropy of the 1987-1989 WSJ testing string. Each data point represents an order three model estimated with the forward estimation withholding algorithm. There are points for each of the nine tying schemes. The data again show that there is a point after which additional parameters adversely effect cross perplexity. The classic model peforms substantially worse than the collapsed nonuniform or hierarchical nonemitting models.

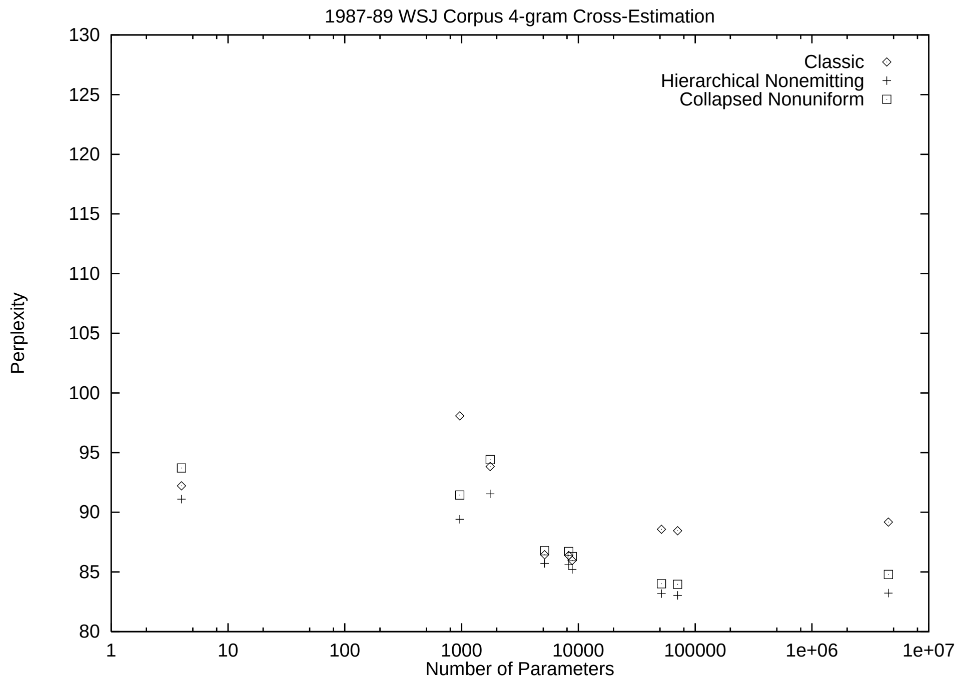

Figure 26: The impact of the number of free parameters on the cross-entropy of the 1987-1989 WSJ testing string. Each data point represents an order three model estimated with the k-fold cross-estimation withholding algorithm. There are points for each of the nine tying schemes. k-fold cross-estimation allows the large number of free parameters to be absorbed into the models, although the models perform slightly worse with no tying.

The classic, general nonemitting, and general nonuniform model's empirical performance are disappointing. Although the general nonemitting model is the best lower order model for the Brown corpus, the number of parameters in the general nonemitting model is overwhelming and causes the perplexity to increase quickly as model order increases. The general nonuniform model has similar difficulties. In fact, the general nonuniform model has n times more parameters than the collapsed nonuniform model and the prediction parameters are less precise than those of the collapsed model since they are only conditioned on prediction length. The classic model's underperformance was suggested by the theoretical results and is reassuring as it suggests that the widely used trigram model (classic model with $n = 2$) can be improved upon by changing the manner in which the mixture parameters are utilized and estimated.

The frequency with which the best hierarchical nonemitting model outperforms the best classic model is surprising. For $n > 1$, the best hierarchical nonemitting model always betters the best classic model regardless of the corpus and withholding algorithm. This is a rather amazing fact considering that each of the models is estimated with the same string statistics, and has the same number of parameters. That being said, the reduction in performance is rather small for the best models. For example, a perplexity reduction of 3.5% is gained by using the hierarchical nonemitting model instead of the classic model with k-fold cross estimation for the WSJ 1987-1989 corpus with $n = 3$. As noted earlier, the best collapsed nonuniform model tends to beat the best nonhierarchical nonemitting model for $n > 3$ for the WSJ 1989 corpus and for $n > 6$ for the Brown corpus by very small margins. The collapsed nonuniform model is able to reduce perplexity by a maximum of .72% compared to the hierarchical nonemitting model when $n = 5$ with forward estimation for the WSJ-1989 corpus.

Table 3 shows that the hierarchical nonemitting model's performance is least influenced by tying scheme. In other words, the hierarchical nonemitting model tends to perform well regardless of tying scheme. This is an important statistic because it can difficult to know *a priori* which parameter tying scheme will minimize perplexity.

For the WSJ 1989 corpus, the hierarchical nonemitting model reduces average perplexity by 6.0% compared to the classic model with k-fold cross-estimation for $n = 5$. It should also be noted that the classic model shows disturbing signs of overtraining for the WSJ 1989 corpus when $n > 3$ and the Brown corpus for $n > 6$.

While the hierarchical nonemitting and general nonuniform models tend to reduce perplexity, it should be noted that they require additional space and time. In addition, an efficient implementation of the collapsed nonuniform model is difficult.

The superiority of the hierarchical nonemitting and collapsed nonuniform model classes (relative to the classic model) can be seen in figures 21-26. These figures plot the impact that the number of free parameters has on model performance. These figures clearly demonstrate two things. First, that there appears to exist an optimal number of free parameters for each model. More free parameters is not always better. The graphs also show that the classic model's performance suffers significantly more than the other two model classes when the number of free parameters grow large.

2.8 Future Work

There are many research issues in this area that are still left open. We now propose several lines of future research that could prove to be fruitful.

2.8.1 Parameter Tying

It's clear from the data that parameter tying is a powerful technique. In fact, it was the case that some parameter tying scheme always improved model performance relative to no parameter tying. This fact motivates research into more robust tying techniques that could improve the predictive capability of these natural language text models.

First, we advocate establishing that certain tying schemes are inherently superior to others. Such an evaluation would require that the application of the tying schemes resulted in models with the same number of parameters. Figures 21-26 suggests a strong relationship between the number of parameters in the model and model

Algorithm 19 Arbitary Parameter Tying Scheme

arbitrary-tie(X, τ, N, ϕ)
1. $\forall x \in X$
2. class = pseudo-random(1,N)
3. $\tau(x)$ = class
4. return τ

performance. To determine whether the increase in model performance is caused solely (or mainly) by a simple reduction in free parameters, an arbitrary tying scheme should be evaluated. See algorithm 19 for an example of such a scheme (X is the set of strings in the model ϕ, τ is the tying function that maps a state that represents the string x to its class, and N is the desired number of classes).

Assuming that arbitrary tying schemes do not work as well as the heuristical techniques proposed, it makes sense to pursue alternative schemes. We now propose a simple greedy technique that can construct a tying scheme based upon the estimated benefit of tying two parameters together (algorithm 20). We assume the expectation step of the expectation-maximization algorithm has been run and therefore the λ^+ and λ^- values are available. The algorithm simply attempts to tie each state to another state (beginning with the most frequently occurring states) and calculating the reduction in codelength. If this reduction is greater than the user-specified ϵ, the two parameters are, at least temporarily, tied together. The reduction in codelength can be easily estimated by simply enumerating the symbols that are possible continuations of the states.

Finally, we note that tying techniques such as codelength and divergence attempt to directly estimate the relationship between a particular state and the maximal suffix of that state. Other techniques such as the frequency-based schemes, diversity, and entropy seek to estimate the reliability of the state output distribution based upon the state's characteristics, ignoring the characteristics of the maximal suffix. Since the λ parameters dictate the degree to which a particular state's output distribution is used in lieu of a distribution belonging to a suffix of that state, it seems to that

Algorithm 20 Greedy Parameter Tying Scheme

greedy-tie$(X, \lambda^+, \lambda^-, \tau, \epsilon, \phi)$
1. S = array sorted, descending order, by $c(x)$ $\forall x \in X$
2. for $i = 0$ to $|S|$
3. for j = i+1 to $|S|$
4. $L = \sum_{\sigma \in A} c(\sigma|S[i], \phi) \log_2 P(\sigma|S[i], \tau, \phi) +$ $\sum_{\sigma \in A} c(\sigma|S[j], \phi) \log_2 P(\sigma|S[j], \tau, \phi)$
5. old-class = $\tau(S[j])$
6. $\tau(S[j]) = \tau(S[i])$
7. $\hat{L} = \sum_{\sigma \in A} c(\sigma|S[i], \phi) \log_2 P(\sigma|S[i], \tau, \phi) +$ $\sum_{\sigma \in A} c(\sigma|S[j], \phi) \log_2 P(\sigma|S[j], \tau, \phi)$
8. if (L - $\hat{L} \leq \epsilon$)
9. $\tau(S[j])$ = old-class

the motivation behind the codelength and divergence schemes is more sound. Thus, we predict that schemes such as frequency $\times$ codelength or frequency $\times$ divergence will outperform the schemes investigated in this thesis.

2.8.2 Theoretical Questions

There are a couple of theoretical loose ends in this thesis that need to be discussed. First, the examination into the relative strength of model classes is not finished. For example, we suspect that the general nonemitting model is strictly more powerful than the hierarchical nonemitting model. We do not provide a proof for this conjecture.

In addition, a theoretical investigation into the relationship between the optimal number of free parameters relative to alphabet size, model class, and corpus size is sorely needed in our opinion.

2.9 Summary

The theoretical and empirical results suggest that the hierarchical nonemitting model is an improvement over the widely-used classic model proposed by Jelinek and Mercer in 1980. These results are especially exciting as the hierarchical nonemitting model uses the same number of parameters and these parameters are estimated in a similar manner. While it's true that the estimation algorithm for the hierarchical nonemitting model takes an extra factor of n time relative to the classic model it uses the same order of space overall.

The results, more importantly, establish a clear way of thinking about how to compare two model classes in terms of theoretical and empirical characteristics. While there have been many empirical evaluations of different models, the theoretical analysis of model classes is lacking in the literature. In addition, an empirical comparison on this type of scale, i.e. over various model classes, model orders, withholding algorithms, and corpora, is unprecented to our knowledge.

In conclusion, we believe that this chapter lends evidence to the assertion that mixture models are an effective way to model natural language text for engineering applications. Mixture models provide a way to model local regularities well and capture knowledge of the real-world in an efficient manner.

Chapter 3

Phonetic Baseform Generation

3.1 Introduction

Phonetic baseform generation is the process of generating a phonetic representation, commonly known as a baseform, for an orthographic string. Many important engineering applications rely on a module that is able to accurately generate baseforms for arbitrary orthographic strings. A voice dialing system, for example, initiates a phone call based upon a spoken utterance. For such a system to be successful, the system needs to determine the intended phone book entry conditioned upon the spoken signal. This task can be tackled by using one of two methods. One method is to transcribe the speech to text and then perform an orthographic search for a phone book match to the transcribed string. An alternative approach is to first generate phonetic representations for each of the phone book entries, and perform a phonetic search over the phonetic representations from the phone book with regard to the spoken utterance.

The speech-to-text first method is conceptually clear, but it can difficult to implement in practice. This is because most speech recognition systems are word-based, that is they assume each "token" in the speech stream is transcribed to word from a pre-defined dictionary. Such an approach requires an extremely large dictionary of words, and, most likely, many spoken utterances demonstrating the pronunciation of these words would also be needed to properly estimate the model's parameters.

The latter approach, generating phonetic representations for the phone book entries, is generally preferred as each orthographic symbol (or sequence of symbols) can be pronounced no more than 10 different ways. Thus, the difficulty in building such a system designing the rules that govern the situations under which certain pronunciations are to be used.

These governing rules can be hand-determined or "learned" automatically from a collection of text aligned with phonetic transcriptions. The advantage of building these rules by hand is increased accuracy as humans are able to draw upon their vast knowledge of pronunciations to construct a set of rules that properly govern the pronunciation of words in their language. The downsides to such an approach are that constructing a set of rules by hand is very time consuming and must be repeated for each new language. An automatic system can overcome, at least partially, these two related weaknesses and also has the potential to equal or surpass human's ability to construct accurate pronunciations.

3.1.1 Chapter Outline

This chapter focuses on models that atomically generate phonetic representations for arbitrary orthographic strings. In the next subsection we introduce the terminology and formalisms used throughout the chapter. This subsection is followed by a discussion of the models proposed by Dedina and Nusbaum [17] and Yvon [61] for automatic baseform generation. From a high-level view, both of these models construct baseforms by exploiting known pronunciations of similar words in a pre-constructed dictionary of orthographic strings aligned with their respective baseforms. These models, along with the successes of mixture modeling in natural language text modeling, help to motivate the hidden alignment model which is discussed in the following section. A hidden alignment model is a novel stochastic model that tries to "learn" which orthographic strings are beneficial for conditioning baseform predictions and exploits this knowledge in the generation process. We then examine the theoretical and empirical qualities of the hidden alignment model relative to the Yvon's model. Overall, the empirical results are disappointing because the simple model proposed by Yvon substantially outperforms the mixture model. We discuss various reasons why this is true and offer guidelines for future work in the area.

3.1.2 Terminology

A grapheme a is a symbol in the orthographic alphabet A. A phoneme b is a symbol in the phoneme alphabet B. A string of graphemes $w \in A^\star$ will be referred to as a orthographic string. A string of phonemes $v \in B^\star$ will be referred to as a phonetic string or a baseform.

A basic alignment is a grapheme-baseform pair that defines the phonetic representation of an individual grapheme. We denote the set of legal basic alignments as $C \subset A \times B^\star$. A complete alignment is a sequence of basic alignments that define the phonetic representation of a orthographic string. We denote the set of legal complete alignments as $E \subset C^\star$. A partial alignment is a subsequence of a complete alignment. We will use the symbol x to refer to a basic alignment, the symbol y to refer to a partial alignment, and the symbol z to represent a complete alignment. $x.a$ and $x.v$ correspond to the grapheme and baseform components of a basic alignment. $z.w$ and $z.v$ correspond to the orthographic and baseform components of a complete alignment. $z.w$ and $z.v$ may be addressed using standard notation, e.g. $z.w_i$ is the ith grapheme and $z.v_j$ is the baseform string that corresponds to the jth grapheme.

All of the models discussed in the chapter rely on a, possibly hand-generated, dictionary of complete alignments. We will refer to this dictionary as the alignment dictionary, D.

3.2 Previous Work

Humans generate baseforms for orthographic strings via some hidden process. [1] The generative process is termed hidden because it is not possible to view the computation

[1] One could certainly make an argument that humans do not pronounce words by first generating "baseforms" and then generating speech based upon those baseforms. Nonetheless, humans are able to generate baseforms for orthographic strings and the process by which they generate these baseforms is hidden.

explicitly – assuming the process is even computable in nature. As such, it is necessary for the engineer to postulate a possible generative process which manifests itself in a generative model. We now examine two analogy-based algorithms that motivate our stochastic model.

Dedina and Nusbaum

Dedina and Nusbaum [17] generate a baseform for an unknown word, w, by exploiting orthographic entries in D that are similar to w. For example, if $w =$ `hello` and D contained the orthographic entries `help` and `mellow` aligned with their respective baseforms, Dedina and Nusbaum may merge the baseforms:

$$\begin{array}{l} \boxed{\texttt{hel}}\texttt{p} \\ \texttt{me}\boxed{\texttt{llo}}\texttt{w} \end{array}$$

to determine the baseform for `hello`. This idea leads to the algorithm 21.

Informally, the algorithm first tries to find long substrings, along with their respective phonetic representations, in the dictionary that match substrings of w. If these substrings are found, the next step is to try to find the longest substrings that overlap by exactly one symbol at the orthographic and phonetic level. Any set of substrings that (1) overlap by one symbol orthographically and phonetically, and (2) completely "cover" w are a possible solution. The best solution is the one that contains the fewest number of substrings.

dn-generate returns a baseform for the orthographic string w based upon the aligned dictionary D. If the orthographic string is contained in the aligned dictionary, that baseform is returned. If the orthographic string is not present in the dictionary, it is necessary to construct a baseform from partial matches in the alignment dictionary. In step 2 of dn-generate, partial-alignments determines the alignments of orthographic *substrings* in D that match the substrings of w and returns those substrings in addition to their respective baseforms. These partial alignments define the limits of the search space: i.e. the eventual baseform will be composed of these partial alignments. Step 3 of dn-generate is the best-path function invocation. best-path first creates a non-cyclical directed graph such that the vertices are the partial alignments

Algorithm 21 Dedina and Nusbaum's Generation Algorithm

dn-generate(w, D)
1. if $w \in D$ return baseform(D, w)
2. $W \times V =$ partial-alignments(w, D)
3. $\Pi =$ best-path$(W \times V)$
4. $v =$ path2baseform(Π)
5. return v

partial-alignments(w, D)
1. alignments $= \{\}$
2. foreach substring, w', of w
3.   foreach orthographic entry, d, of D
4.   foreach substring, d', of d
5.   if $(w' = d')$
6.   alignments $\cup = d' \times$ baseform(D, d')
7. return alignments

best-path$(W \times V)$
1. dgraph $= \langle V = \{\}, E = \{\} \rangle$
2. dgraph $=$ insert-vertex(dgraph, `source`)
3. dgraph $=$ insert-vertex(dgraph, `sink`)
4. foreach partial alignment, $y \in W \times V$
5.   dgraph $=$ insert-vertex(dgraph, y)
6. foreach pair of vertices, v_i and v_j, in dgraph
7.   if (overlap-by-one(v_i, v_j))
8.   dgraph $=$ insert-edge(dgraph, v_i, v_j)
9. $P = \{p : p = p_1 \ldots p_k \text{ s.t. } p_1 = \texttt{source} \wedge p_k = \texttt{sink}\}$
10. $\Pi = \min_P |p|$
11. return Π

from D. Two special vertices are introduced to designate the `source` and `sink` points of the graph. An edge is introduced from vertex v_i to v_j if the last symbol in the orthographic and phoneme strings that correspond to v_i are the same as the first symbol in the orthographic and phoneme strings that corresponds to v_j. In addition, an edge is constructed between `source` and the vertex v_i if v_i's orthographic string matches a prefix of w. Likewise, an edge is constructed between v_i and `sink` if v_i's orthographic string matches a suffix of w. Any path that begins at source and ends at sink is a possible generation path for a baseform. The path with the minimum number of edges is considered to be the best generation path. If two paths have the same number of edges the one with the most frequent alignments is considered more likely. See figure 27 for a visual representation of a Dedina and Nusbaum graph.

Dedina and Nusbaum view the generative process as one driven by analogy. The language user generates a baseform for an arbitrary orthographic string by utilizing known alignments in their lexicon. The desire to favor minimum edge paths is sensible since the likelihood of mistakes is probably greatest at the point of overlap [61].

Yvon

Francois Yvon [61] extended Dedina and Nusbaum's work by considering multiple symbol overlaps. In line 7 of the best-path function, Dedina and Nusbaum impose the overlap-by-one constraint. Yvon relaxes this constraint by allowing for alignments that overlap by more than one symbol. This relaxation allows Yvon to construct a baseform for $w =$ `hello` by merging the baseforms

$$\begin{array}{l} \boxed{\texttt{hell}}\texttt{ish} \\ \texttt{f}\boxed{\texttt{ello}}\texttt{w} \end{array}$$

Dedina and Nusbaum, on the other hand, may have used

$$\begin{array}{l} \boxed{\texttt{hell}}\texttt{ish} \\ \texttt{fel}\boxed{\texttt{lo}}\texttt{w} \end{array}$$

to construct the baseform for `hello` since the overlap must be exactly one symbol long. Yvon's generative algorithm is outlined in algorithm 22. Figure 27 shows possible directed graphs for Dedina and Nusbaum and Yvon's algorithms. The difference

Algorithm 22 *Yvon's Generation Algorithm*

```
yvon-generate(w, D)
    if w ∈ D return baseform(D, w)
    W × V = partial-alignments(w, D)
    Π = best-path(W × V)
    x = path2baseform(Π)
    return x

partial-alignments(w, D)
    alignments = {}
    foreach substring, w', of w
        foreach orthographic entry, d, of D
            foreach substring, d', of d
                if (w' = d')
                    alignments ∪= d' × baseform(D, d')
    return alignments

best-path(W × V)
    dgraph = ⟨V = {}, E = {}⟩
    dgraph = insert-vertex(dgraph, source)
    dgraph = insert-vertex(dgraph, sink)
    foreach partial alignment, y ∈ W × V
        dgraph = insert-vertex(dgraph, y)
    foreach pair of vertices, v_i and v_j, in dgraph
        if overlap-by-atleast-one(v_i, v_j)
            dgraph = insert-edge(dgraph, v_i, v_j)
    P = {p : p = p_1 ... p_k s.t. p_1 = source ∧ p_k = sink}
    Π = max_P Σ_{y∈p} length(y)/|p|
    return Π
```

between Yvon's and Dedina and Nusbaum's algorithms is isolated to lines 7 and 11 of the best-path function. First, line 7 relaxes the single symbol overlap constraint. The predicate overlap-by-atleast-one is satisfied if and only if the last k symbols of the orthographic and phoneme strings of v_i agree with the first k symbols of the orthographic and phoneme strings of v_j ($k > 0$). Line 11 now favors the path with the longest mean alignment length. Thus, a path with m alignments could be favored over a path with n alignments ($m > n$) if the m-path tended to have longer alignments that overlapped significantly. Yvon's relaxation of the single symbol overlap constraint is useful because it allows alignments to be combined that agree on a significant number of graphemes and phonemes, thereby decreasing the risk that an error will occur at the overlap site.

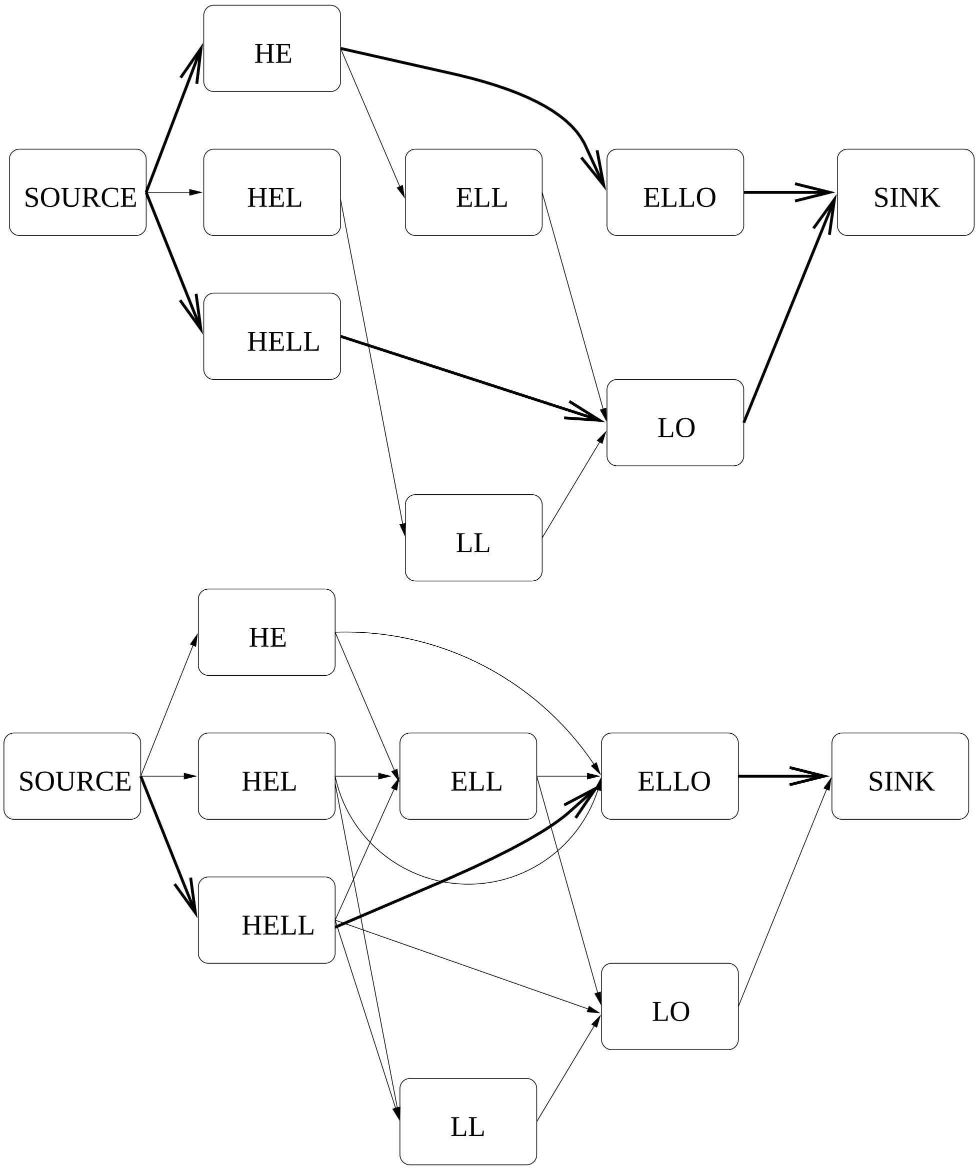

Figure 27: Generating a baseform for the orthographic string **hello**: these figures highlight the difference between Dedina *et. al* and Yvon's algorithms. The figures are graphical representation of possible directed graphs of Dedina (top figure) and Yvon (bottom figure). Notice that Yvon allows orthographic strings that overlap by more than one symbol to be connected. The corresponding baseforms were not included in order to simplify the graphs. The best paths are denoted by thick lines.

3.3 Hidden Alignment Model

Yvon's scoring function (line 10 of best-path) ignores potentially important information that may be deduced from the alignment dictionary. For example, if paths p_1 and p_2 differ by only one partial alignment, path p_1 containing alignment y_1 and path p_2 containing alignment y_2, Yvon favors path p_1 if $\mathsf{length}(y_1) > \mathsf{length}(y_2)$ regardless of the number of times y_1 occurs relative to y_2. If y_2 occurred many more times than y_1, it is possible that the model should favor y_2 instead of y_1. This is particularly likely if the baseform diversity of y_2 is less than the baseform diversity of y_1. The baseform diversity of an orthographic string w is the number of possible baseforms for w in the aligned dictionary. Yvon's algorithm does not exploit the diversity statistic.

3.3.1 Model Definition

The deficiencies in Yvon's model motivated the development of the hidden alignment model. Rather than assuming *a priori* that one partial alignment is better than another by some arbitrary alignment characteristic, e.g. string length, the hidden alignment model tries to learn which partial alignments are favorable by generating baseforms for the alignment dictionary.

A hidden alignment model $\phi = \langle A, B, \delta, \lambda \rangle$ consists of a finite grapheme alphabet A, a finite phoneme alphabet B, a function $\delta : \{A \times B^\star\}^\star \rightarrow [0, 1]$ that represents the probability that a orthographic string will have a particular aligned baseform, and a mixture parameter function $\lambda : A^\star \rightarrow [0, 1]$ that determines the probability that a given orthographic string is utilized to condition the baseform prediction.

The hidden alignment model uses the λ parameters to determine the degree to which it should blend the baseform prediction of a particular orthographic string, w, with other possible orthographic strings, $\lfloor w \rfloor$ and $\lceil w \rceil$. $\lfloor w \rfloor$ designates the set of orthographic strings considered after w and $\lceil w \rceil$ designates the set of orthographic strings considered prior to w. We delay the discussion of how $\lfloor w \rfloor$ and $\lceil w \rceil$ are constructed until section 3.3.2. As $\lambda(w)$ approaches unity, the model is increasingly confident that it should use w instead of a orthographic string $w' \in \lfloor w \rfloor$ as the basis for making the baseform prediction. Conversely, as $\lambda(w)$ approaches zero, the model is increasingly confident that it should use a orthographic string in $\lfloor w \rfloor$. The probability of using a particular orthographic string w given this utilization of the mixture parameters is:

$$P(w|\phi, \lceil w \rceil) \doteq \lambda(w) \prod_{w' \in \lceil w \rceil} (1 - \lambda(w')) \tag{37}$$

The probability that the model uses a partial alignment y for w is $\delta(y|w, \phi)$. The δ function is estimated from the alignment dictionary as:

$$\delta(y|w, \phi) \doteq \frac{c(y.v|w)}{c(w)} \tag{38}$$

where $c(\sigma)$ is the number of times σ occurs in the alignment dictionary D.

The following subsections define and discuss the set of functions necessary to support the hidden alignment model. We begin by considering a generative interpretation of the hidden alignment model that demonstrates how the model parameters can be used to generate an aligned baseform for an arbitrary string. The set of functions necessary to evaluate and improve model performance are considered after, followed by a discussion of optimal baseform generation.

3.3.2 Hidden Alignment Model Generation Algorithm

An algorithm defining the generative process of the hidden alignment model is now formulated. The choice of alignments is stochastically determined by the current model parameters in algorithm 23. The first substantive choice made by the model is which orthographic string to use for predicting the baseform (line 4 of ha-model-generate). The length of the string is $i + j$. i is the number of symbols the current partial alignment overlaps with the previous partial alignment and j determines the number of graphemes for which the model must generate a baseform. Given the choice of i and j, the model has implicitly chosen how much of the already generated phoneme string will be used to condition the baseform prediction (line 5 of ha-model-generate). Given these choices, a baseform must be generated for the last j symbols of o.

Algorithm 23 *Hidden Alignment Model Baseform Generation Algorithm*

ha-model-generate(w, ϕ)
$v = \epsilon$
while $|v| < |w|$
  pick i and j according to alignment-prob$(w, |v|, i, j, \phi)$
    $o = w_{|v|-i}^{|v|+j}$
    $v_o = v_{|v|+1-i}^{|v|}$
  pick v_e according to $\delta(\langle o, v_o \cdot v_e \rangle \, | w, j, \phi)$
  $v \mathrel{\cdot}= v_e$
return v

alignment-prob(w, t, i, j, ϕ)
overlap $= t$; extension $= |w| - t$
do
  if(overlap $= i \wedge$ extension $= j$)
    return $P(w_i^j | \phi)$
while $\langle$overlap,future$\rangle$ = next-indices(t,overlap,future)

alignment-prob calculates the probability of choosing a particular i and j at position t of w. The next-indices function has at least four reasonable implementations. next-indices defines an ordering over the set of orthographic strings considered at symbol t in orthographic string w. The four possible enumeration algorithms we consider are:

1. long context (lc)
2. long future (lf)
3. long string, favoring context (lsc)
4. long string, favoring future (lsf)

As stated earlier, these enumeration algorithms provide the order in which the orthographic strings are considered. The lc algorithm favors orthographic strings that have the greatest overlap. The lf algorithm favors orthographic strings that extend maximally into the future. Both the lsc and lsf algorithms favor longer strings – they differ as to whether they prefer to first consider orthographic strings that overlap heavily with the past of that extend far into the future.

Consider the string *hello*. Suppose the model has generated a baseform for the first two symbol *he*. In order to continue the generation process the model must determine whether it will generate a baseform for *l*, *ll*, or *llo*. In addition it must determine how many symbols in the past will be used to condition the baseform prediction. Here is the order in which the four different enumeration algorithms will consider the substrings:

LC - hello,hell,hel,ello,ell,el,llo,ll,l
LF - hello,ello,llo,hell,ell,ll,l,hel,el
LSC - hello,hell,ello,hel,ell,llo,el,ll,l
LSF - hello,ello,hell,llo,ell,hel,ll,el,l

One may wonder why it matters which enumeration method is used since the same set of strings is considered. Recall that the probability with which a particular orthographic string is used is dependent on $\lambda(w)$ and $\lambda(w')$, $\forall w' \in \lceil w \rceil$. Consider a particular orthographic string $w' \in \lceil w \rceil$. If $\lambda(w') \approx 1$, that decreases the probability with which w will be used. The extreme case, $\lambda(w') = 1$, ensures that w will not be used. The enumeration algorithms can thus be used as a way to bias the model – in other words, one can take advantage of *a priori* knowledge regarding which orthographic strings are more likely to provide good results.

While the ha-model-generate will generate a baseform for w, the quality of the baseform is dictated by the model parameters. If λ and δ do not mirror reality, ha-model-generate will probably not produce an accurate baseform. Furthermore, even if λ and δ model reality well, there is no guarantee that ha-model-generate will produce the most likely baseform according to ϕ.

3.3.3 Hidden Alignment Model Evaluation Algorithm

The goal of this subsection is to provide an efficient algorithm for computing the probability of generating a complete alignment, z. Such an algorithm could be used

Algorithm 24 *Hidden Alignment Model Evaluation Algorithm*

ha-model-evaluate(z, ϕ)
1. for $t = 0$ to $|z.w|$
2. $\quad p[t] = 0$
3. $p[0] = 1$
4. for $t = 0$ to $|z.w| - 1$
5. $\quad i = t;\ j = |z.w|$ - t
6. $\quad$ do
7. $\quad\quad o = z.w_{t+1-i}^{t+j};\ v_o = z.v_{t+1-i}^{t};\ v_e = z.v_{t+1}^{t+j}$
8. $\quad\quad p[t+j] = p[t] \cdot P(o|\phi) \cdot \delta(\langle o, v_o \cdot v_e \rangle\, |\phi)$
9. $\quad$ while $\langle i, j \rangle =$ next-indices(t, i, j)
10. return $p[t]$

as the means for determining the most likely baseform for w ignoring questions of efficiency. One could simply enumerate all possible baseforms for w and pick the one with the highest probability.

Calculating the probability of a complete alignment is non-trivial because of the fact that it is possible for many generation paths to result in the same complete alignment. A generation path is a selection of overlapping partial alignments that result in z. Thus, in order to compute $P(z|w, \phi)$, it is necessary to marginalize the joint probability function, $P(z, g|w, \phi)$ over all possible generation paths, g:

$$P(z|w, \phi) = \sum_g P(z, g|w, \phi) \tag{39}$$

Calculating $P(z, g|w, \phi)$ is relatively straightforward given the hidden alignment model.

To make the summation efficient, we introduce a dynamic programming algorithm (algorithm 24) that calculates the probability of generating the baseform in $O(|w|^3)$ time and $O(|w|)$ space. Lines 1-3 of ha-model-evaluate initialize a table that stores the probability of generating the first k symbols of the complete alignment in $p[k]$. Lines 4-9 enumerate all possible ways of overlapping partial alignments that result in the complete alignment. Line 8 reflects the impact of choosing to generate a baseform for the next j symbols of w conditioned upon the length i partial alignment. Finally, $p[|w|]$ stores the probability of generating the complete alignment z for w.

3.3.4 Hidden Alignment Model Estimation Algorithm

In this section an expectation maximization (EM) algorithm is formulated for the hidden alignment model. The central goal is to provide a set of functions that allow the modification of the λ parameters such that the probability of generating a given complete alignment increases, or remains the same, after each EM application.

Since the choices of overlapping orthographic strings is hidden, the expected number of times each of these events occurs in the training data must be recorded. Given that knowledge, it is possible to adjust the λ parameters in the maximization step of the algorithm.

Hidden Alignment Model Forward Algorithm

In order to calculate the expected number of times a given event occurs, both a forward and a backward pass through the training data is necessary. In the forward pass, we calculate:

$$\alpha[t,i,j] \doteq p(h = \left\langle w_1^t, v_1^t \right\rangle, o = w_{t+1-i}^{t+j}, v_o = v_{t+1-i}^{t}, v_e = v_{t+1}^{j} | \phi, w, v) \tag{40}$$

The variable $\alpha[t,i,j]$ represents the probability of generating the first t symbols of the partial alignment $\langle w_1^t, v_1^t \rangle$, picking the overlap length, i, and the extension length, j. The overlap and extension length choices implicitly define the orthographic string o, the overlap baseform v_o, and the extension baseform v_e. Algorithm 25 calculates the α table in $O(|w|^3)$ time and space. Lines 1-6 of hidden alignment model-forward initialize the alpha table. Lines 10-11 choose the appropriate orthographic string, overlapping baseform, and extension baseform based on the values of t, i, and j and lines 11-12 carry forward the impact of these choices on the rest of the table.

Hidden Alignment Model Backward Algorithm

The backward pass calculates the backward variable:

$$\begin{aligned} \beta[t,i,j] &\doteq p(\left\langle w_{t+1}^{|w|}, v_{t+1}^{|w|} \right\rangle | h = \left\langle w_1^t, v_1^t \right\rangle, o = w_{t+1-i}^{t+j}, v_o = v_{t+1-i}^{t}, v_e = v_{t+1}^{t+j} \phi, w) && (41) \\ &= p(\left\langle w_{t+j+1}^{|w|}, v_{t+j+1}^{|w|} \right\rangle | h = \left\langle w_1^t, v_1^t \right\rangle, \phi, w) && (42) \end{aligned}$$

Algorithm 25 *Hidden Alignment Model Forward Algorithm*

ha-model-forward(z, ϕ)
for $t = 0$ to $|z.w|$
    $p[t] = 0$
    for $i = 0$ to $|z.w|$
        for $j = 0$ to $|z.w|$
            $\alpha[t, i, j] = 0$
$p[0] = 1$
for $t = 0$ to $|z.w|$-1
    $i = t$; $j = |z.w| - t$
    do
        $o = z.w^{t+j}_{t+1-i}$; $v_o = z.v^{t}_{t+1-i}$; $v_e = z.v^{t+j}_{t+1}$
        $\alpha[t, i, j] = p[t] \cdot P(o|\phi) \cdot \delta(\langle o, v_o \cdot v_e\rangle |o, \phi)$
        $p[t + j]$ += $\alpha[t, i, j]$
    while $(i, j) =$ next-indices(t, i, j)

$\beta[t, i, j]$ is the probability of generating the final $|w|$-t symbols of the aligned baseform given that the model generated the first t symbols in the alignment, that the model decided to use a partial alignment that overlaps with the past i symbols, and that the baseform was extended by j symbols. Algorithm 26 calculates the β values in $O(|w|^3)$ time and $O(|w|)$ space.

Hidden Alignment Model Expectation Algorithm

With the forward and backward values calculated, the $\gamma[i, j]$ variable is introduced:

$$\gamma[t, i, j] \doteq p(o = w^{t+j}_{t-i}, v_o = v^{t}_{t-i} v_e = v^{j}_{t+1} | \langle w, v\rangle, \phi) \tag{43}$$

$$= \frac{\alpha[t, i, j] \cdot \beta[t, i, j]}{P(\langle w, v\rangle |w, \phi)} \tag{44}$$

$$= \frac{\alpha[t, i, j] \cdot \beta[t + j]}{P(\langle w, v\rangle |w, \phi)} \tag{45}$$

$\gamma[t, i, j]$ represents the probability of choosing, at time t, to use a orthographic string of length $i + j$ and a baseform of length i to condition the prediction of a baseform of length j.

Algorithm 26 *Hidden Alignment Model Backward Algorithm*

ha-model-backward(z, ϕ)
for $t = 0$ to $|z.w|$
    $\beta[0] = 0$
$\beta[|z.w|] = 1$
for $t = |z.w|$-1 downto 0
    $i = t$; $j = |z.w| - t$
    do
        $o = z.w_{t+1-i}^{t+j}$; $v_o = z.v_{t+1-i}^{t}$; $v_e = z.v_{t+1}^{t+j}$
        $\beta[t]$ += $\beta[t+j] \cdot P(o|\phi) \cdot \delta(\langle o, v_o \cdot v_e\rangle \, |o, \phi)$
    while $(i, j) =$ next-indices(t, i, j)

Given the γ table, it is possible to calculate the expected number of times a particular orthographic string, w, is used as opposed to the number of times w could have been utilized but some $w' \in \lfloor w \rfloor$ was used instead. Algorithm 27 performs this accumulation in $O(|w|^5)$ time and $O(|w|^2)$ space.

Hidden Alignment Model Maximization Algorithm

With the λ^+ and λ^- values calculated, it is trivial to perform the maximization step (algorithm 28). The model estimates $\lambda(x) as \frac{\lambda^+(x)}{\lambda^+(x)+\lambda^-(x)}$.

Hidden Alignment Model Estimation Algorithm

Given this set of functions an explicit parameter estimation algorithm for the hidden alignment model is now defined (algorithm 29). The estimation algorithm uses the current model to calculate the expected number of times each substring of the aligned dictionary is used to generate a baseform for each complete alignment z in D. After these expectations are accumulated, the λ parameters are updated via the ha-model-maximization-step. ha-model-expectation-maximization can be used in an iterative fashion to improve model parameters.

Algorithm 27 *Hidden Alignment Model Expectation Algorithm*

ha-model-expectation-step(z,λ^+,λ^-,ϕ)
for $t = 0$ to $|z.w|$
passed-string-set $= \{\}$
$i = t$; $j = |z.w| - t$
do
$o = z.w_{t+1-i}^{t+j}$
$\lambda^+(\mathrm{o})$ += $\gamma[t, i, j]$
$\forall$passed-string $\in$ passed-string-set
λ^-(passed-string) += $\gamma[t, i, j]$
passed-string-set $\cup$= o
while $(i, j) =$ next-indices(t, i, j)

Algorithm 28 *Hidden Alignment Model Maximization Algorithm*

ha-model-maximization-step(λ^+, λ^-, ϕ)
$\forall z \in D$
$\lambda(z.w) = \frac{\lambda^+(z.w)}{\lambda^+(z.w)+\lambda^-(z.w)}$

Algorithm 29 *Hidden Alignment Model Parameter Estimation Algorithm*

ha-model-expectation-maximization(D, ϕ)
$\forall z \in D$
$\lambda^+(z.w) = 0$
$\lambda^-(z.w) = 0$
$\forall z \in D$
ha-model-forward(z, ϕ)
ha-model-backward(z, ϕ)
ha-model-expectation-step$(z, \lambda^+, \lambda^-, \phi)$
ha-model-maximization-step$(\lambda^+, \lambda^-, \phi)$

3.3.5 Hidden Alignment Model Baseform Generation

Our central goal is to generate the best baseform for an arbitrary orthographic string. A sensible definition of best baseform for the hidden alignment model is the most likely baseform according to the model parameters. Intuitively, we want to find v such that:

$$v =_{v' \in V(w)}^{argmax} p(\langle w, v' \rangle, \phi) \tag{46}$$

where $V(w)$ is set the set of all possible baseforms for the orthographic string w. Enumerating the set of possible baseforms is infeasible as the cardinality of the set grows exponentially in the length of the orthographic string, $O(k^{|w|})$ (k is the number of possible phonemes per grapheme). Worse yet, the time to evaluate $p(\langle w, v' \rangle | w, \phi)$ is $O(|w|^3)$, for a total running time of $O(|w|^3 k^{|w|})$.

It's fortunate that typically certain baseforms are much more likely than others. This knowledge can be translated into a simple heuristic: assuming the model has found a baseform, v for w with probability $P(\langle w, v \rangle | w, \phi)$, there is no need to consider baseforms in the set $V(w)$ such that $\forall v' \in V(w)$, $P(\langle w, v' \rangle | w, \phi) < P(\langle w, v \rangle | w, \phi)$. This heuristic allows a branch and bound search algorithm to be introduced that builds up the baseform v that maximizes $P(\langle w, v \rangle | w, \phi)$ by repeatedly extending the current best partial baseform.

Algorithm 30 runs in $O(w^4 k^{|w|})$ time and $O(k^{|w|})$ space. The extra w factor in the time bound comes from the fact that ha-model-evaluate must be called at each level in the tree. Of course, these are worst-case bounds and one should expect the average case to be significantly better.

An additional branch and bound algorithm will now be considered that can result in generating baseforms that are non-optimal according to ha-model-evaluate. Instead, ha-model-viterbi-branch-and-bound finds the most-likely complete alignment whereas ha-model-branch-and-bound finds the most likely baseform by summing over all possible complete alignments. ha-model-viterbi-branch-and-bound will closely approximate ha-model-branch-and-bound if the sum over all possible complete alignments is dominated by one term.

Algorithm 30 *Hidden Alignment Model Branch and Bound Algorithm*

```
ha-model-full-branch-and-bound(w, φ)
h = heap-create()
heap-insert(h, 0, “”, 1.0)
best-v = “”
best-prob = 0.0
while !heap-is-empty(h)
    item = heap-remove(h)
    t = item.t; v = item.v
    if (p < best-prob)
        return best-v
    else if (t = |w| + 1 and p > best-prob)
        best-v = v; best-prob = p
    else
        ∀k ∈ LegalPhones(w_t)
            p' = ha-model-evaluate(w_0^t, v · k, φ)
            heap-insert(h, t + 1, v · k, p')
```

Algorithm 31 *Hidden Alignment Model Viterbi Branch and Bound Algorithm*

ha-model-viterbi-branch-and-bound(w, ϕ)

h = heap-create()

heap-insert(h, 0, “”, 1.0)

best-v = “”

best-prob = 0.0

while !heap-is-empty(h)

    item = heap-remove(h)

    t = item.t; v = item.v

    if ($p <$ best-prob)

        return best-v

    else if $t = |w| + 1$ and $p >$ best-prob

        best-v = v; best-prob = p

    else

        i = t; $j = |w| - t$

        do

            $o = w_{t-i+1}^{t+j}$; $v_o = v_{t-i+1}^{t}$; $v_e = v_{t+1}^{t+j}$

            $p' = p \cdot P(o|\phi) \cdot \delta(\langle o, v_o \cdot v_e \rangle \, | o, \phi)$

            heap-insert(h, $t + j$, $v \cdot v_e$, p')

        while (i, j) = next-indices(t, i, j)

3.4 Theoretical Results

Any baseform generation model can be modeled as a function that maps orthographic strings to phonetic strings: $\psi : a^\star \rightarrow b^\star$. One can use this perspective to investigate the relative strength of two baseform generation model classes. For example, if a model class $\mathcal{X}$ is able to represent a superset of the functions that the model class $\mathcal{Y}$ can represent, the model class $\mathcal{X}$ is strictly more powerful than the model class $\mathcal{Y}$.

We now compare the theoretical strength of the two baseform generation models. Lemma 9 demonstrates that there exists a hidden alignment model whose representative function ψ can not be emulated by any function for a model in Yvon's model class (for some fixed dictionary D).

Lemma 9 $\exists D\ \exists \phi\ \forall \phi'\ \exists x \in A^\star[\psi(x|\phi, D) \neq \psi(x|\phi', D)]$

The idea of the proof is to force the hidden alignment model ϕ to use a substring with a different partial baseform than would be available for any longer substring that Yvon's model ϕ' may use.

Proof:

Suppose

$$\begin{aligned} D = \quad & \{\langle a_1a_2a_3a_4, b_1b_2b_3b_4\rangle\,, \\ & \langle a_2a_3a_4a_5, b_2b_3b_4b_5\rangle\,, \\ & \langle a_1a_2a_3, b_1'b_2'b_3'\rangle\,, \\ & \langle a_3a_4a_5, b_3'b_4'b_5'\rangle\} \end{aligned}$$

where $b_i \neq b_i'$ $0 > i \geq 5$. Let $x = a_1a_2a_3a_4a_5$. Given D, $\psi(x|\phi') = b_1b_2b_3b_4b_5$ because Yvon's model will use $\langle a_1a_2a_3a_4, b_1b_2b_3b_4\rangle$ *and* $\langle a_2a_3a_4a_5, b_2b_3b_4b_5\rangle$ to generate the baseform (there is an overlap at $\langle a_2a_3a_4, b_2b_3b_4\rangle$). Now let $\lambda(a_1a_2a_3a_4) = 0$, $\lambda(a_2a_3a_4a_5) = 0$, $\lambda(a_1a_2a_3) = 1$, $\lambda(a_3a_4a_5) = 1$, $\delta(b_1b_2b_3|a_1a_2a_3, \phi) = 0$, $\delta(b_2b_3b_4|a_2a_3a_4, \phi) = 0$, $\delta(b_1'b_2'b_3'|a_1a_2a_3, \phi) = 1$, and $\delta(b_2'b_3'b_4'|a_2a_3a_4, \phi)$. These parameter settings force the hidden alignment model to ignore the strings, $a_1a_2a_3a_4$ and $a_2a_3a_4a_5$. In addition, the parameter settings force the hidden alignment model to use the partial baseforms

$b'_1 b'_2 b'_3$ and $b'_3 b'_4 b'_5$ for $a_1 a_2 a_3$ and $a_3 a_4 a_5$ respectively. Thus, $\psi(x|\phi, D) = b'_1 b'_2 b'_3 b'_4 b'_5$ and therefore $\exists D\ \exists \phi\ \forall \phi' \exists x \in A^\star[\psi(x|\phi, D) \neq \psi(x|\phi', D)]$. □

This result would be greatly strengthened by an analogous lemma defining the relationship in the other direction, i.e. establishing whether every model in Yvon's class can be transformed into an equivalent hidden alignment model.

3.5 Empirical Results

We now present empirical results for experiments performed on a 28,695 entry aligned dictionary of primarily proper names. The dictionary was constructed by Ken Davies and Lalit Bahl at IBM TJ Watson Research Center. The dictionary contained 24,808 unique orthographic strings with each orthographic string having approximately 1.16 alignments.

3.5.1 Evaluation Paradigm

We randomly divided the dictionary into two sections: a training dictionary containing 25,862 entries (22,328 unique orthographic strings) and a testing dictionary containing 2,833 entries (2,400 unique orthographic strings). All alignments for a particular orthographic string were either entirely in the training dictionary or entirely in the testing dictionary. The λ parameters of the hidden alignment model were initialized to a small constant value, 0.1. The λ parameters were calculated by performing 10 expectation-maximization iterations on the entire training dictionary. After each model was constructed and trained on the training dictionary, a baseform was generated for every orthographic string in the testing dictionary.

We measured model performance by calculating word accuracy and phoneme accuracy. Each baseform generated was compared to the correct baseform in the testing dictionary. We then computed the word and phoneme accuracy using the following formula:

$$\text{word accuracy} = \frac{\#\ \text{words correct}}{\#\ \text{words total}} \tag{47}$$

Model Type	Word Accuracy		Phoneme Accuracy	
Yvon	68.1		91.8	
Hidden Alignment Model branch-and-bound	Before Training	After Training	Before Training	After Training
LC	60.2	47.1	89.0	83.8
LF	60.4	46.5	89.1	84.5
LSC	60.0	46.9	89.0	83.6
LSF	60.0	47.2	89.0	83.6
Hidden Alignment Model viterbi-branch-and-bound	Before Training	After Training	Before Training	After Training
LC	59.8	50.7	89.4	85.5
LF	61.4	52.1	89.8	85.9
LSC	61.0	51.9	89.7	85.9
LSF	61.3	52.4	89.7	85.7

Table 6: Word and phoneme accuracy results for the IBM proper name dictionary. Four hidden alignment model's are constructed for each branch-and-bound algorithm: one for each string enumeration method. The hidden alignment models were trained for 10 expectation-maximization iterations. The results clearly demonstrate that Yvon's model outperforms the hidden alignment model significantly in both word and phoneme accuracy measures. The hidden alignment model's performance is actually better without training. This suggests that either (1) the maximization algorithm is poorly suited for this problem, or (2) the model class is improper.

$$\text{phone accuracy} = \frac{\#\text{ phones correct}}{\#\text{ phones total}} \tag{48}$$

If there were two or more alignments present in the test dictionary, the one the most-closely matched the generated baseform was used for purposes of the accuracy statistics. The number of phonemes correct was determined by calculating the minimum edit distance between the generated baseform and the correct baseform.

As can be seen in table 6, Yvon's generation algorithm substantially out-performs the hidden alignment model regardless of which branch-and-bound algorithm or string enumeration method was employed. Even more surprising was that applying the expectation-maximization procedure actually caused a degradation in word and phoneme accuracy. In addition, the viterbi branch-and-bound algorithm gave slightly better results than the full summation branch-and-bound algorithm regardless of enumeration technique or number of training iterations (except for LC enumeration method with no training). Finally, the four enumeration methods had similar results.

3.6 Discussion

The disparity in results between the hidden alignment model and Yvon's model could be due to multiple factors but we believe there are three fundamental reasons. First, the maximization process that is performed to improve model accuracy only guarantees an increase in the probability the model assigns to complete alignments. This maximization has an undefined relationship to the maximization of model accuracy. Second, we believe the model class can be improved. The mixture parameters can be redefined over partial alignments instead of orthographic strings. Finally, the limited amount of training data can adversely impact mixture parameter estimation.

3.6.1 Parameter Estimation Weakness

The parameter estimation scheme considered in this chapter is based on the expectation-maximization algorithm. The fundamental goal of this algorithm is to increase the likelihood that a given alignment is generated for a given orthographic string. Unfortunately, the ultimate measure of success is quite separate from the probability of a particular alignment. We are solely concerned with the number of phoneme errors when generating the alignment. The expectation maximization algorithm does not explicitly attempt to minimize the number of phoneme errors. Figure 28 provides clear evidence that maximizing the probability that an alignment is generated for a particular error actually increases the number of phoneme errors! We believe this relationship is partially due to the fact that the maximization process is not explicitly trying to maximize model accuracy.

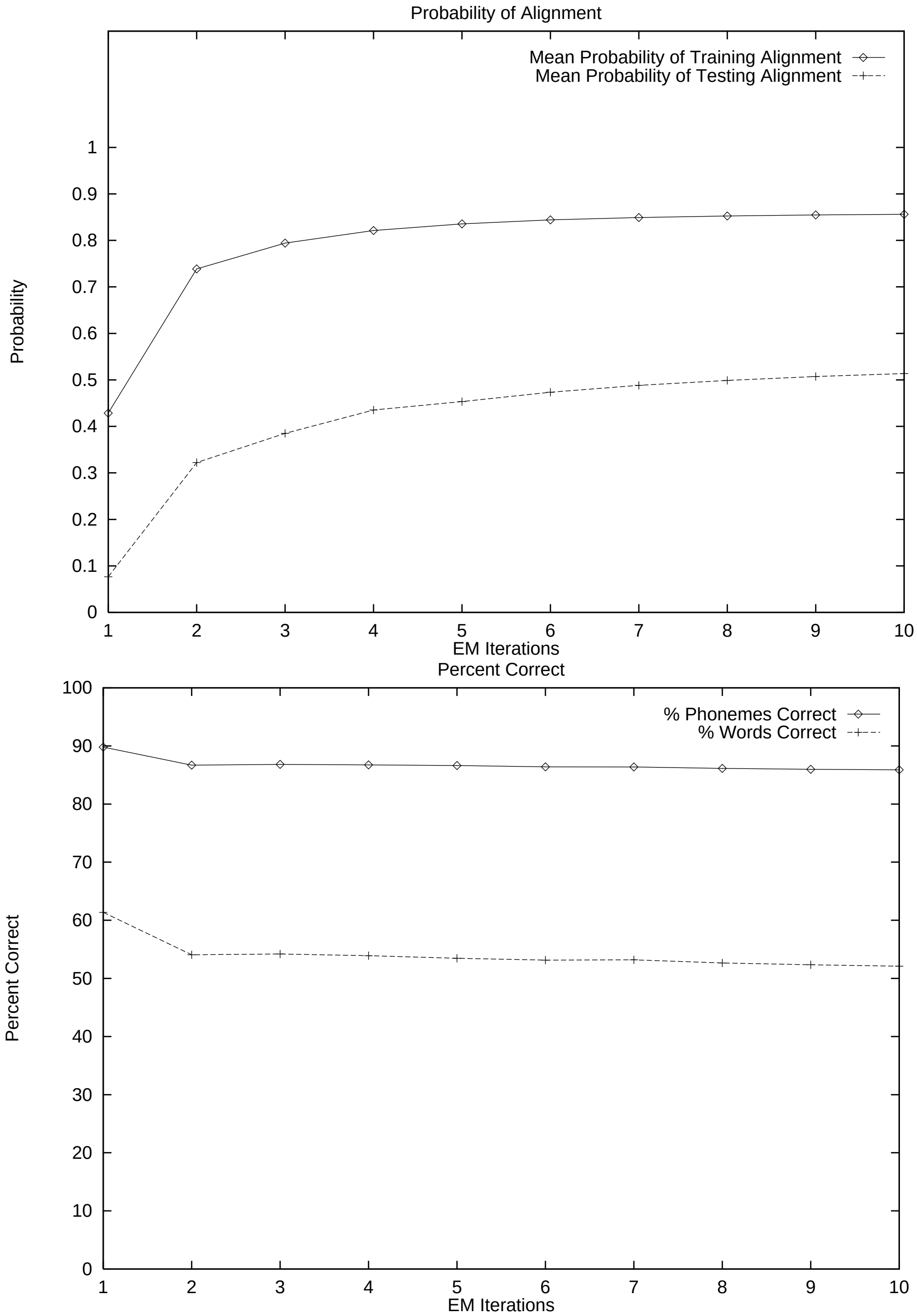

Figure 28: These graphs demonstrate the relationship between mean probability of alignments drawn from training and testing set and the percentage of phonemes and word that were correctly generated for the test set. The inverse relationship is very disturbing. This graph corresponds to LF string enumeration experiment for ha-model-branch-and-bound-alignment. The λ parameters were initialized with values of 0.1. The graph is representative of all the experiments.

3.6.2 Model Class Weaknesses

Mixture Parameter Definition

A possible weakness in the hidden alignment model centers around the mixture parameters. The domain over which the λ parameters are defined could be extended to be over partial alignments as opposed to just orthographic strings. This would allow the λ parameters to be more precise. For example, a orthographic string w with two possible baseforms v_1 and v_2 may be an excellent conditioning factor if aligned with v_1 but not with v_2. The current scheme calculates the mixture parameter for w based on w's average benefit with v_1 and v_2, thereby understating the benefit of using w with v_1 and overstating the benefit of using w with v_2.

A re-definition of the λ parameters would increase the number of model parameters (by a factor detailing the number of baseforms per orthographic string). Thus, it's not clear that the current amount of training data would allow for a naive re-definition. It's likely that the λ parameters would need to be tied under such a scheme.

3.7 Future Work

We believe future work should concentrate on two major areas: reformulation of the parameter estimation algorithm and the exploration of alternative model classes. In the previous section we commented on a possible modification to the hidden alignment model class, we now concentrate on outlining an alternative maximization procedure.

3.7.1 Alternative Maximization Procedure

The expectation-maximization procedure that is currently employed to estimate model parameters could be scrapped in favor of a more direct scheme that seeks to explicitly maximize model accuracy.

One such approach is motivated by the directed graphs produced by Yvon's algorithm (lines 1-8 of `best-path` in algorithm 22). Such a graph represents all possible overlapping paths given the model. In essence, it implicitly represents all possible baseforms that the model could generate for a given orthographic string. Assuming that each partial alignment y has a $\lambda(y) \subset [0, 1]$ associated with it that defines the degree to which the model should use y in the solution, a scoring function can be defined over the paths. More formally, the scoring function $\mathcal{S} : \pi \rightarrow \Re^+$ maps each possible path to a real number by taking the product of the λ values on a given path. The path π that maximizes $\mathcal{S}(\pi)$ should be chosen.

To train the model, the current λ settings could be used to determine the best baseform for each orthographic string from a withheld set of such strings. These baseforms could then be compared to the correct baseform and adjustments could be made to the λ parameters so as to increase the likelihood that a correct baseform is generated. The question of which λ values to adjust and by how much is important and needs additional research. We offer a high-level view of a novel parameter estimation scheme in algorithm 32 that adjusts the λ parameters by hill-climbing. It increases the respective mixture parameters by a constant value r on the highest-value path that results in the correct baseform for w and decreases the mixture parameters along the path currently chosen by the model. A more sophisticated algorithm can be envisioned that adjusts the step-size r contingent upon the situation, e.g. increase r if the current path chosen by the model has a significantly higher score than the highest scoring path that corresponds to the correct baseform.

Clearly, the parameter estimation scheme proposed requires a new model class. Although the λ values can be considered, in a loose sense, probabilities, the model itself is not a probability model because it does not adhere to the axioms of probability, e.g. the sum over all possible paths is not necessarily one.

Algorithm 32 *Explicit Accuracy Maximization Training Algorithm*

eam-train(λ, D, r)
Let $\langle T, W \rangle =$ divide-data-set(D)
foreach $w \in W$
  $\text{dgraph} =$ create-yvon-graph(w, T)
  $\Pi = \{p : p = p_1 \ldots p_k \text{ s.t. } p_1 = \texttt{source} \wedge p_k = \texttt{sink}\}$
  $\pi = \underset{\pi' \in \Pi}{argmax}$ path-score(λ, π')
  if (path2baseform$(\pi) \neq$ correct-baseform(w, W))
    adjust-model$(\lambda,$ correct-baseform$(w, W),\ \Pi, \pi, r)$

adjust-model$(\lambda, v, \Pi, \pi, r)$
$C = \{c : c \in \Pi \wedge$ path2baseform$(c) = v\}$
$c = \underset{\pi' \in C}{argmax}$ path-score(λ, π')
for $i = 1$ to $|c|$
  $\lambda(c_i) = \min(\lambda(c_i) + r, 1\text{-}\epsilon)$
for $i = 1$ to $|\pi|$
  $\lambda(\pi_i) = \max(\lambda(\pi_i) - r, \epsilon)$

path-score(λ, π)
$score = 1$
for $i = 1$ to $|\pi|$
  $score \mathrel{\star}= \lambda(\pi_i)$

Chapter 4

Conclusion

In this thesis, we investigate the use of mixture models for two important natural language processing applications: natural language text modeling and phonetic baseform generation. We propose five novel mixture model classes for these two problems and analyze the empirical and theoretical characteristics of these model classes. Overall, it's our opinion that the results highlight the successes, and future potential, that mixture models have in modeling natural language phenomena. Although many linguists would correctly assert that these mixture model classes are incapable of modeling certain linguistic structures, the models, nonetheless, are a useful engineering tool that adequately model many real-world situations.

Our results demonstrate that the classic natural language text model, proposed by Jelinek and Mercer [27], can be improved upon by utilizing alternative interpretations of the mixture parameters. For example, both the hierarchical nonemitting and collapsed nonuniform model classes consistently outperformed the classic model in empirical evaluations and Ristad and Thomas have proven that the hierarchical nonemitting model is strictly more powerful than the classic model from a theoretical point of view [46]. Our empirical results also elucidate the impact of important model decisions such as parameter tying scheme, withholding algorithm, model order, and training corpus.

The hidden alignment model was proposed in chapter three as a model class for generating phonetic baseforms. Although the model class is probabilistic in nature, a simpler model proposed by Yvon [61] outperforms the hidden alignment model. These results are disappointing and we identify several veins of future research in this area.

4.1 Natural Language Text Modeling

Several important contributions were introduced in the natural language text modeling chapter of the thesis. We now summarize these contributions.

4.1.1 Novel Model Classes

Four new model classes were introduced in chapter two. These novel model classes were motivated by alternative interpretations of the model's mixture parameters which manifest themselves in the string generation, string probability assignment, and parameter estimation algorithms. We also demonstrate that these algorithms need polynomial space and time.

4.1.2 Empirical Evaluation

Over 900 experiments were performed to assess the relative empirical quality of the model classes considered in this thesis. In addition, these experiments helped to investigate the questions of how to tie parameters, which withholding algorithm to use, and which model order is best?

Model Class

The empirical evaluation of the model classes clearly show that the popular classic model [27] can be improved upon by utilizing alternative model classes. Although the hierarchical nonemitting model and the collapsed nonuniform model need more time and space to function, they consistently outperformed the classic model. Overall, the results strongly motivate the use of the hierarchical nonemitting model in all experimental situations.

Parameter Tying

The parameter tying scheme can significantly impact model quality, especially if the forward estimation withholding algorithm is used. The frequency based tying schemes (this includes codelength) performed very well over diverse experimental conditions. Other tying schemes such as divergence and entropy performed poorly. Finally, building models with no parameter tying resulted in poor performance unless k-fold cross-estimation withholding was used.

Withholding Algorithm

The forward and k-fold cross-estimation withholding algorithms gave similar results assuming some form of parameter tying was utilized. Nonetheless, model whose parameters were estimated using k-fold cross-estimation always outperformed those models whose parameters were estimated using forward estimation. This suggests that those who seek to minimize cross perplexity should use k-fold cross-estimation. In addition, if no parameter tying scheme is used, k-fold cross-estimation must be used in order to obtain good results.

Model Order

Although many in the NLP community use bigram or trigram word models, the results suggest that quadgram, and even higher order, word models can reduce perplexity even further. Thus, our finding is that if (1) the proper model class is chosen and (2) the estimation process is carried out well, researchers should use the highest order model they can afford from a computational point of view. Our results for the classic model suggest why the opposite view, i.e. trigrams are very good, may be en vogue as the classic model is plagued by the overfitting problem. Other model classes, such as the hierarchical nonemitting model, do not fall prey to this pitfall.

4.1.3 Theoretical Evaluation

The theoretical survey provided in the natural language text modeling chapter establishes a partial hierarchy over the model classes. This hierarchy defines the model classes' relative strength. Our results prove that the classic model is equivalent to the basic Markov model, a model with no interpolation, and they also show that all of the novel model classes are strictly more powerful than the classic model.

4.2 Phonetic Baseform Generation

In chapter three, we discuss the problem of automatic phonetic baseform generation. That is, building models that generate a phonetic representation for arbitrary orthographic strings. The contributions made in that chapter are now discussed.

4.2.1 Novel Model Class

We provide a new model class, the hidden alignment model, that couches the generative process of baseforms in probabilistic terms, Partially motivated by a non-stochastic model proposed by Yvon[61], the hidden alignment model can be used to generate a baseform for orthographic strings. In chapter three, we provide string generation, parameter estimation, and probability evaluation algorithms for the hidden alignment model. Although resource demands grow exponentially for the string generation procedure, a branch-and-bound algorithm is provided that works well in practice. The worst-case analysis of this algorithm, however, is still exponential with respect to the length of the orthographic string. The other algorithms require polynomial time and space.

4.2.2 Empirical Evaluation

The empirical evaluation examines the relative performance of the hidden alignment model class and Yvon's model class. Yvon's model substantially outperforms the hidden alignment model on a proper name database provided by IBM. These results are disappointing as intuition suggests that a probabilistic model should outperform the simpler model proposed by Yvon.

4.2.3 Theoretical Evaluation

In chapter three, we also investigate the theoretical strength of Yvon's model class compared to the hidden alignment model class. We view each model as defining a function that maps orthographic strings to phonetic ones. A model class $\mathcal{X}$ is stronger than another model class $\mathcal{Y}$ if there exists a fixed aligned dictionary for which no model in $\mathcal{Y}$ can properly emulate some model in $\mathcal{X}$ but for every fixed aligned dictionary, any model in $\mathcal{Y}$ can be emulated by some model in $\mathcal{X}$. Our results show that there does exist a hidden alignment model constructed from a fixed aligned dictionary for which no Yvon model can emulate. The reverse relation is left as an open question. Thus, from a theoretical point of view, it's either the case that (1) the hidden alignment model is strictly more powerful than Yvon's model, or (2) neither model is strictly more powerful than the other.

4.3 Concluding Remarks

The problem of natural language processing must be tackled if computers are to become ubiquitous in all facets of our lives. Although computers play an integral role at our jobs and to a lesser degree in our homes, there are many potential applications and benefits for natural language processing systems. First, speech-driven user interfaces have the potential to greatly increase worker productivity. While many individuals can't type fifty words a minute, nearly all fluent speakers can dictate English at twice this rate. If speech-driven interfaces are integrated seamlessly into existing applications, there is significant reason to believe that such productivity gains could be realized. Second, natural language processing systems can increase convenience and decrease cost of service. For example, being able to make plane reservations via an automatic system would allow the airline industry to greatly reduce their customer service costs while simultaneously being able to better serve their customers, e.g. being able to handle more call throughput. Third, natural language processing applications and interfaces could reduce the number of injuries currently experienced by frequent computer users. Many users currently suffer from carpal-tunnel syndrome

– a direct result of unnatural human-machine interfaces.

While there is still much to be done to realize the vision, this thesis adds a link to a long chain of applied research in the natural language processing community. We believe that some of the novel model classes proposed herein have demonstrated theoretical and empirical qualities that strongly motivate their use in real-world applications.

Appendix A

Appendix

Tables 7-16 contain the unabridged, unfiltered results for the classic, hierarchical nonemitting, and collapsed nonuniform models.

WSJ 1987-1989 – k-fold Cross-Estimation									
N	Ord	Divrg	Cdlen	FrOrd	FrDvOrd	FrDv	Freq	Entrp	None
Classic Model									
1	155.8	153.2	151.0	151.2	**150.7**	**150.7**	151.2	153.0	**150.7**
2	101.0	99.9	94.3	94.4	**93.9**	**93.9**	94.6	100.5	94.0
3	92.2	93.8	86.4	**86.0**	88.5	88.6	86.4	98.1	89.2
Hierarchical Nonemitting Model									
1	155.8	153.2	151.0	151.2	**150.7**	**150.7**	151.2	153.0	**150.7**
2	100.3	98.8	94.0	94.1	**92.0**	92.1	94.3	97.7	92.1
3	91.1	91.5	85.6	85.2	**83.0**	83.2	85.7	89.4	83.2
Collapsed Nonuniform Model									
1	158.3	155.0	152.1	152.2	**151.7**	**151.7**	152.2	154.8	**151.7**
2	103.7	101.8	95.4	95.5	**93.2**	93.3	95.7	100.3	93.3
3	93.7	94.4	86.7	86.3	**84.0**	**84.0**	86.8	91.4	84.8

Table 7: k-fold Cross Estimation Experiments on WSJ 1987-1989 corpus (in Perplexity)

WSJ 1987-1989 – k-fold Cross-Estimation									
N	Ord	Divrg	Cdlen	FrOrd	FrDvOrd	FrDv	Freq	Entrp	None
Classic Model									
1	7.284	7.259	7.238	7.240	**7.236**	**7.236**	7.240	7.257	**7.236**
2	6.658	6.642	6.559	6.561	**6.553**	**6.553**	6.564	6.651	6.555
3	6.527	6.552	6.433	**6.426**	6.468	6.469	6.433	6.616	6.479
Hierarchical Nonemitting Model									
1	7.284	7.259	7.238	7.240	**7.236**	**7.236**	7.240	7.257	**7.236**
2	6.648	6.626	6.555	6.556	**6.524**	6.525	6.559	6.610	6.525
3	6.509	6.516	6.420	6.413	**6.375**	6.379	6.421	6.482	6.379
Collapsed Nonuniform Model									
1	7.307	7.276	7.249	7.250	**7.245**	**7.245**	7.250	7.274	**7.245**
2	6.696	6.670	6.576	6.577	**6.542**	6.544	6.580	6.648	6.544
3	6.550	6.561	6.438	6.431	**6.392**	**6.392**	6.440	6.514	6.406

Table 8: k-fold Cross Estimation Experiments on WSJ 1987-1989 corpus (in Entropy)

WSJ 1987-1989 – Forward Estimation									
N	Ord	Divrg	Cdlen	FrOrd	FrDvOrd	FrDv	Freq	Entrp	None
Classic Model									
1	155.7	153.3	**151.2**	151.3	151.7	151.7	151.3	153.0	152.6
2	101.1	100.0	**94.4**	94.5	95.0	94.7	94.6	100.4	109.3
3	92.3	94.0	86.5	**86.1**	89.6	89.2	86.5	98.0	114.6
Hierarchical Nonemitting Model									
1	155.7	153.3	**151.2**	151.3	151.7	151.7	151.3	153.0	152.6
2	100.3	98.9	94.1	94.2	93.0	**92.7**	94.3	97.6	100.8
3	91.2	91.7	85.8	85.4	84.1	**83.7**	85.8	89.3	95.2
Collapsed Nonuniform Model									
1	158.2	155.1	**152.3**	**152.3**	152.8	152.8	**152.3**	154.8	153.8
2	103.7	101.9	95.5	95.6	94.2	**93.8**	95.7	100.2	102.7
3	93.8	94.6	86.9	86.4	84.8	**84.2**	86.9	91.0	97.7

Table 9: Forward Estimation Experiments on WSJ 1987-1989 corpus (in Perplexity)

WSJ 1987-1989 – Forward Estimation									
N	Ord	Divrg	Cdlen	FrOrd	FrDvOrd	FrDv	Freq	Entrp	None
Classic Model									
1	7.283	7.260	**7.240**	7.241	7.245	7.245	7.241	7.257	7.254
2	6.660	6.644	**6.561**	6.562	6.570	6.565	6.564	6.650	6.772
3	6.528	6.555	6.435	**6.428**	6.485	6.479	6.435	6.615	6.840
Hierarchical Nonemitting Model									
1	7.283	7.260	**7.240**	7.241	7.245	7.245	7.241	7.257	7.254
2	6.648	6.628	6.556	6.558	6.539	**6.534**	6.559	6.609	6.655
3	6.511	6.519	6.423	6.416	6.394	**6.387**	6.423	6.481	6.573
Collapsed Nonuniform Model									
1	7.306	7.277	**7.251**	**7.251**	7.256	7.256	**7.251**	7.274	7.265
2	6.696	6.671	6.577	6.579	6.558	**6.552**	6.580	6.647	6.682
3	6.552	6.564	6.441	6.433	6.406	**6.396**	6.441	6.508	6.610

Table 10: Forward Estimation Experiments on WSJ 1987-1989 corpus (in Entropy)

WSJ 1989 – k-fold Cross-Estimation									
N	Ord	Divrg	Cdlen	FrOrd	FrDvOrd	FrDv	Freq	Entrp	None
Classic Model									
1	184.6	179.0	175.6	175.9	**174.9**	**174.9**	175.9	179.2	175.2
2	131.8	128.9	**122.0**	**122.0**	122.7	122.8	122.4	131.8	123.7
3	123.1	123.9	115.3	**114.4**	118.8	119.0	115.4	131.9	121.3
4	120.5	124.0	113.9	**112.4**	118.4	117.2	113.8	136.8	123.0
5	119.6	124.3	**111.4**	111.7	118.6	116.3	113.4	141.2	124.5
Hierarchical Nonemitting Model									
1	184.6	179.0	175.6	175.9	**174.9**	**174.9**	175.9	179.2	175.2
2	130.7	127.3	121.4	121.4	**118.9**	119.0	121.8	126.2	119.6
3	121.5	120.6	114.1	113.2	**111.0**	111.1	114.2	118.4	111.9
4	119.2	119.4	112.6	111.3	**109.4**	109.5	112.6	117.1	110.6
5	118.5	119.0	110.5	110.8	109.1	**109.0**	112.1	116.9	110.4
Collapsed Nonuniform Model									
1	191.4	183.3	178.1	178.5	**177.4**	**177.4**	178.5	183.5	177.7
2	136.3	131.9	123.7	123.7	**121.1**	121.3	124.1	130.3	121.8
3	125.9	124.8	115.9	115.0	111.9	**111.8**	116.0	120.5	114.6
4	122.9	124.0	114.0	112.7	**109.3**	**109.3**	114.0	117.9	113.5
5	121.9	124.0	111.7	111.9	**108.4**	108.5	113.4	117.1	113.4

Table 11: k-fold Cross Estimation Experiments on WSJ 1989 corpus (in Perplexity)

WSJ 1989 – k-fold Cross-Estimation									
N	Ord	Divrg	Cdlen	FrOrd	FrDvOrd	FrDv	Freq	Entrp	None
Classic Model									
1	7.528	7.484	7.456	7.459	**7.450**	**7.450**	7.459	7.485	7.453
2	7.042	7.010	**6.931**	**6.931**	6.939	6.940	6.935	7.042	6.951
3	6.944	6.953	6.849	**6.838**	6.892	6.895	6.850	7.043	6.922
4	6.913	6.954	6.832	**6.812**	6.888	6.873	6.830	7.096	6.943
5	6.902	6.958	**6.800**	6.803	6.890	6.862	6.825	7.142	6.960
Hierarchical Nonemitting Model									
1	7.528	7.484	7.456	7.459	**7.450**	**7.450**	7.459	7.485	7.453
2	7.030	6.992	6.924	6.924	**6.894**	6.895	6.928	6.980	6.902
3	6.925	6.914	6.834	6.823	**6.794**	6.796	6.835	6.888	6.806
4	6.897	6.900	6.815	6.798	**6.773**	6.775	6.815	6.872	6.789
5	6.889	6.895	6.788	6.792	6.770	**6.768**	6.809	6.869	6.787
Collapsed Nonuniform Model									
1	7.580	7.518	7.477	7.480	**7.471**	**7.471**	7.480	7.520	7.473
2	7.091	7.043	6.951	6.951	**6.920**	6.922	6.955	7.026	6.928
3	6.976	6.963	6.857	6.845	6.806	**6.805**	6.858	6.913	6.840
4	6.941	6.954	6.833	6.816	**6.772**	**6.772**	6.833	6.881	6.827
5	6.930	6.954	6.803	6.806	**6.760**	6.762	6.825	6.872	6.825

Table 12: k-fold Cross Estimation Experiments on WSJ 1989 corpus (in Entropy)

WSJ 1989 – Forward Estimation									
N	Ord	Divrg	Cdlen	FrOrd	FrDvOrd	FrDv	Freq	Entrp	None
Classic Model									
1	184.6	180.1	176.5	**176.2**	178.4	178.4	**176.2**	179.7	191.4
2	131.9	129.5	122.7	**122.4**	127.9	126.5	122.7	132.0	164.6
3	123.2	124.5	115.9	**114.8**	124.9	122.9	115.7	132.0	174.9
4	120.6	124.6	114.5	**112.9**	124.9	120.5	114.2	136.9	181.3
5	119.7	125.0	114.2	**112.2**	125.3	119.7	113.6	141.4	184.4
Hierarchical Nonemitting Model									
1	184.6	180.1	176.5	**176.2**	178.4	178.4	**176.2**	179.7	191.4
2	130.8	127.8	122.0	121.8	122.8	**121.5**	122.1	126.3	146.1
3	121.6	121.1	114.7	113.6	115.0	**113.4**	114.5	118.5	142.4
4	119.3	120.0	113.2	**111.7**	113.6	**111.7**	112.9	117.1	142.7
5	118.6	119.5	112.7	**111.2**	113.3	111.3	112.3	117.0	143.1
Collapsed Nonuniform Model									
1	191.4	184.6	179.3	**178.8**	181.4	181.4	**178.8**	184.2	194.4
2	136.4	132.4	124.4	124.1	125.1	**123.9**	124.4	130.6	149.6
3	126.1	125.5	116.6	115.5	116.0	**113.9**	116.3	120.4	146.2
4	123.1	124.7	114.7	113.2	113.3	**111.3**	114.3	117.7	146.3
5	122.1	124.7	114.0	112.5	112.4	**110.4**	113.7	116.9	146.6

Table 13: Forward Estimation Experiments on WSJ 1989 corpus (in Perplexity)

WSJ 1989 – Forward Estimation									
N	Ord	Divrg	Cdlen	FrOrd	FrDvOrd	FrDv	Freq	Entrp	None
Classic Model									
1	7.528	7.493	7.464	**7.461**	7.479	7.479	**7.461**	7.489	7.580
2	7.043	7.017	6.939	**6.935**	6.999	6.983	6.939	7.044	7.363
3	6.945	6.960	6.857	**6.843**	6.965	6.941	6.854	7.044	7.450
4	6.914	6.961	6.839	**6.819**	6.965	6.913	6.835	7.097	7.502
5	6.903	6.966	6.835	**6.810**	6.969	6.903	6.828	7.144	7.527
Hierarchical Nonemitting Model									
1	7.528	7.493	7.464	**7.461**	7.479	7.479	**7.461**	7.489	7.580
2	7.031	6.998	6.931	6.928	6.940	**6.925**	6.932	6.981	7.191
3	6.926	6.920	6.842	6.828	6.845	**6.825**	6.839	6.889	7.154
4	6.898	6.907	6.823	**6.803**	6.828	**6.803**	6.819	6.872	7.157
5	6.890	6.901	6.816	**6.797**	6.824	6.798	6.811	6.870	7.161
Collapsed Nonuniform Model									
1	7.580	7.528	7.486	**7.482**	7.503	7.503	**7.482**	7.525	7.603
2	7.092	7.049	6.959	6.955	6.967	**6.953**	6.959	7.029	7.225
3	6.978	6.972	6.865	6.852	6.858	**6.832**	6.862	6.912	7.192
4	6.944	6.962	6.842	6.823	6.824	**6.798**	6.837	6.879	7.193
5	6.932	6.962	6.833	6.814	6.812	**6.787**	6.829	6.869	7.196

Table 14: Forward Estimation Experiments on WSJ 1989 corpus (in Entropy)

Brown – k-fold Cross Estimation									
N	Ord	Divrg	Cdlen	FrOrd	FrDvOrd	FrDv	Freq	Entrp	None
Classic Model									
1	3.602	3.602	3.602	3.602	3.602	3.602	3.602	**3.601**	3.602
2	2.950	2.951	**2.949**	**2.949**	2.950	2.950	**2.949**	2.951	2.950
3	2.495	2.495	**2.483**	**2.483**	2.486	2.486	2.484	2.496	2.490
4	2.236	2.237	**2.212**	**2.212**	2.217	2.218	2.217	2.238	2.231
5	2.133	2.133	**2.100**	2.101	2.115	2.112	2.105	2.132	2.149
6	2.092	2.094	**2.055**	2.058	2.090	2.082	2.060	2.100	2.164
7	2.074	2.088	**2.037**	2.041	2.093	2.077	2.044	2.103	2.212
8	2.066	2.098	**2.030**	2.034	2.108	2.084	2.040	2.124	2.277
9	2.062	2.116	**2.028**	2.031	2.123	2.093	2.042	2.151	2.334
Hierarchical Nonemitting Model									
1	3.602	3.602	3.602	3.602	3.602	3.602	3.602	**3.601**	3.602
2	2.949	2.948	**2.946**	**2.946**	**2.946**	**2.946**	**2.946**	2.947	**2.946**
3	2.486	2.486	2.474	2.474	**2.473**	**2.473**	2.475	2.481	**2.473**
4	2.223	2.223	2.201	2.201	**2.192**	**2.192**	2.206	2.212	2.193
5	2.122	2.116	2.090	2.090	**2.075**	**2.075**	2.095	2.102	2.076
6	2.082	2.078	2.045	2.047	**2.027**	**2.027**	2.051	2.062	2.031
7	2.066	2.067	2.027	2.030	**2.008**	**2.008**	2.035	2.051	2.015
8	2.058	2.067	2.021	2.023	**1.999**	**2.000**	2.030	2.051	2.010
9	2.055	2.068	2.018	2.021	**1.996**	**1.996**	2.029	2.053	2.009
Collapsed Nonuniform Model									
1	**3.607**	**3.607**	**3.607**	**3.607**	**3.607**	**3.607**	**3.607**	**3.607**	**3.607**
2	2.954	2.953	**2.951**	**2.951**	**2.951**	**2.951**	**2.951**	2.952	**2.951**
3	2.495	2.494	2.482	2.482	**2.481**	**2.481**	2.483	2.489	**2.481**
4	2.236	2.235	2.211	2.210	**2.201**	**2.201**	2.216	2.223	2.203
5	2.134	2.131	2.097	2.098	2.082	**2.081**	2.103	2.112	2.087
6	2.092	2.091	2.049	2.053	2.032	**2.029**	2.056	2.067	2.045
7	2.074	2.084	2.030	2.035	2.010	**2.007**	2.038	2.051	2.034
8	2.066	2.094	2.022	2.028	2.001	**1.996**	2.033	2.048	2.037
9	2.062	2.111	2.019	2.026	1.995	**1.992**	2.034	2.051	2.045

Table 15: k-fold Cross-Estimation Experiments on Brown corpus

Brown – Forward Estimation									
N	Ord	Divrg	Cdlen	FrOrd	FrDvOrd	FrDv	Freq	Entrp	None
Classic Model									
1	3.604	**3.603**	**3.603**	**3.603**	**3.603**	**3.603**	**3.603**	**3.603**	**3.603**
2	**2.950**	2.954	**2.950**	**2.950**	2.952	2.952	2.951	**2.950**	2.952
3	2.496	2.501	**2.485**	2.486	2.498	2.497	**2.485**	2.497	2.507
4	2.238	2.245	**2.217**	2.219	2.238	2.228	2.219	2.236	2.285
5	2.135	2.141	**2.105**	2.111	2.140	2.123	2.108	2.132	2.238
6	2.093	2.102	**2.060**	2.067	2.118	2.092	2.063	2.103	2.269
7	2.075	2.095	**2.042**	2.050	2.123	2.089	2.048	2.111	2.327
8	2.067	2.106	**2.035**	2.043	2.139	2.097	2.044	2.137	2.394
9	2.063	2.123	**2.033**	2.040	2.156	2.107	2.045	2.167	2.448
Hierarchical Nonemitting Model									
1	3.604	**3.603**	**3.603**	**3.603**	**3.603**	**3.603**	**3.603**	**3.603**	**3.603**
2	2.949	2.948	**2.946**	**2.946**	2.947	2.947	2.947	**2.946**	2.947
3	2.488	2.488	**2.476**	**2.476**	2.480	2.479	**2.476**	2.481	2.486
4	2.227	2.226	2.204	2.206	2.207	**2.202**	2.208	2.212	2.237
5	2.123	2.119	2.093	2.097	2.092	**2.084**	2.097	2.102	2.150
6	2.083	2.081	2.048	2.053	2.047	**2.037**	2.054	2.062	2.130
7	2.067	2.070	2.031	2.037	2.030	**2.018**	2.038	2.051	2.135
8	2.060	2.070	2.024	2.030	2.022	**2.010**	2.033	2.051	2.144
9	2.058	2.070	2.022	2.028	2.019	**2.006**	2.032	2.052	2.154
Collapsed Nonuniform Model									
1	3.610	3.610	**3.609**	**3.609**	**3.609**	**3.609**	**3.609**	**3.609**	**3.609**
2	2.953	2.954	**2.951**	**2.951**	2.952	2.952	2.952	**2.951**	2.952
3	2.498	2.496	**2.483**	**2.483**	2.487	2.486	**2.483**	2.490	2.491
4	2.239	2.238	2.214	2.215	2.215	**2.210**	2.217	2.224	2.241
5	2.136	2.134	2.099	2.104	2.096	**2.089**	2.106	2.113	2.151
6	2.094	2.095	2.051	2.058	2.048	**2.037**	2.059	2.068	2.128
7	2.076	2.088	2.033	2.041	2.028	**2.014**	2.042	2.053	2.134
8	2.070	2.100	2.026	2.035	2.020	**2.003**	2.038	2.051	2.147
9	2.067	2.118	2.024	2.033	2.015	**1.999**	2.040	2.054	2.161

Table 16: Forward Estimation Experiments on Brown corpus

Bibliography

[1] N. Abramson. *Information Theory and Coding.* McGraw-Hill, New York, 1963.

[2] Lalit R. Bahl, Frederick Jelinek, and Robert L. Mercer. A Maximum Likelihood Approach to Continuous Speech Recognition. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, PAMI-5(2):179-190, March 1983.

[3] L. Baum and J. Eagon. An Inequality with Applications to Statistical Estimation for Probabilistic Functions of a Markov Process and to Models for Ecology. *Bull. AMS 73* 1967, 360-363.

[4] L. Baum, T. Petrie, G. Soules and N. Weiss. A Maximization Technique Occurring in the Statistical Analysis of Proababilistic Functions of Markov Chains. *Ann. Math. Stat. 41* 1970, 164-171.

[5] Timothy C. Bell, John G. Cleary and Ian H. Witten. *Text Compression.* Prentice Hall, NJ, 1990.

[6] Edward A. Bender and S. Gill Williamson. *Foundations of Applied Combinatorics.* Addison-Wesley, CA, 1991.

[7] Adam L. Berger, Stephen A. Della Peitra and Vincent J. Della Pietra. A Maximum Entropy Approach to Natural Language Processing. *Computational Linguistics*, vol. 22, num 1, 1996.

[8] Peter F. Brown, Stephen A. Della Pietra, Vincent J. Della Pietra, Jennifer C. Lai and Robert L. Mercer. An Estimate of an Upper Bound for the Entropy of English. *Computational Linguistics*, vol. 18, num 1, 1992.

[9] Roy J. Byrd and Martin S. Chodorow. Using an On-line Dictionary to Find Rhyming Words and Pronunciations for Unknown Words. *Proc. 23rd Annual Meeting of the Association for Computational Linguistics*, 1985.

[10] Stanley Chen and Joshua Goodman. An Empirical Study of Smoothing Techniques for Language Modeling. *Proc. of 34th ACL*, 1996.

[11] Kenneth W. Church and William A. Gale. A Comparision of the Enhanced Good-Turing and Deleted Estimation Methods for Estimating Probabilities of English Bigrams. *Computer Speech and Language*, vol 5, 1991.

[12] J. Cleary and I. Witten. Data Compression using Adaptive Coding and Partial String Matching. *IEEE Trans. Comm.*, COM-32, 4, 1984.

[13] Cecil H. Coker, Kenneth W. Church, and Mark Y. Liberman. Morphology and Rhyming: Two Powerful Alternatives to Letter-to-Sound Rules. *Conference on Speech Synthesis*. Autrans, France, 1990.

[14] Thomas M. Cover and Roger C. King. A Convergent Gambling Estimate of the Entropy of English. *IEEE Trans. on Information Theory*, Vol. IT-24, No. 4, July 1978.

[15] J.N. Darroch and D. Ratcliff. Generalized Iterative Scaling for Log-Linear Models. *The Annals of Mathematical Statistics*, 43(5): 1470-1480,1972.

[16] Carl de Marcken. *Unsupervise Language Acquisition*. Ph.D. Dissertation, MIT, 1996.

[17] Michael J. Dedina and Howard C. Nusbaum. PRONOUNCE: a Program for Pronunciation by Analogy. *Computer Speech and Language*, 5:55-64, 1991.

[18] Sabine Deligne, Frédéric Bimbot and François Yvon. Phonetic Transcription by Variable Length Sequences: Joint Multigrams. *Proc. European Conference on Speech Communication and Technology (Eurospeech)*. Madrid, Sep 1995.

[19] A.P. Dempster, N.M. Laird, and D.B. Rubin. Maximum-likelihood from Incompleted Data via the EM Algorithm. *J. Royal Statistical Society Ser. B (methodological)*, 39:1-38, 1977.

[20] W.N. Francis and H. Kucera. *Frequency Analysis of English Usage: Lexicon and Grammar*. Houghton Mifflin, MA.

[21] I.J. Good. The Population Frequencies of Species and the Estimation of Population Parameters. *Biometrika*, vol. 40, num. 3 and 4, 1953.

[22] John E. Hopcroft and Jeffrey D. Ullman. *Introduction to Automata Theory, Languages, and Computation.* Addison-Wesley, 1979.

[23] X.D. Huang, Y. Ariki, and M.A. Jack. *Hidden Markov Models for Speech Recognition.* Edinburgh University Press, 1990.

[24] F. Jelinek. *Probabilistic Information Theory.* McGraw-Hill, New York, 1968.

[25] F. Jelinek, J.D. Lafferty and R.L. Mercer. Basic Methods of Probabilistic Context Free Grammars. *Speech Recognition and Understanding: Recent Advances*, NATO ASI Series, Vol. F 75, Edited by P. Laface and R. De Mori. Springer-Verlag, 1992.

[26] Frederick Jelinek, Robert L. Mercer and Salim Roukos. Principles of Lexical Language Modeling for Speech Recognition.

[27] F. Jelinek and R.L. Mercer. Interpolated Estimation of Markov Source Parameters from Sparse Data. *Pattern Recognition in Practice* (Amsterdam, 1970), E. S. Gelsema and L. N. Kanal, Eds., North Holland, 381-397.

[28] Philip N. Johnson-Laird. *The Computer and the Mind: An Introduction to Cognitive Science.* Harvard University Press, MA, 1988.

[29] Philip N. Johnson-Laird. *Mental Models: Towards a Cognitive Science of Language, Inference, and Consciousness* Harvard University Press, MA, 1983.

[30] S. Katz. Estimation of Probabilities from Sparse Data for the Language Model Component of a Speech Recognizer. *IEEE Trans. ASSP 35* 1987, 400-401.

[31] T. Kohonen. Dynamically Expanding Context, with Application to the Correction of Symbol Strings in the Recognition of Continuous Speech. *Proc. 8th International Conference on Pattern Recognition*, 1986.

[32] P.S. Laplace. *Philosophical Essay on Probabilities.* Springer-Verlag, 1995. Translated by A.I. Dale from 5th French edition of 1825.

[33] D.J. MacKay and L.C. Peto. A Hierarchical Dirichlet Language Model. *Natural Language Engineering 1*,1994.

[34] A. Moffat. A Note on the PPM Data Compression Algorithm. Research Report 88/7, Departmnet of Computer Science, University of Melbourne, Parkville, Victoria, Australia, 1988.

[35] R. Pasco. *Source Coding Algorithms for Fast Data Compression.* Ph.D. Thesis, Department of Electrical Engineering, Stanford University, 1976.

[36] G. Potamianos and F. Jelinek. The Study of N-gram and Decision Tree Letter Language Modeling Methods. *Technical Report, Center for Language and Speech Processing, The Johns Hopkins University*, 1997.

[37] T. Reinhardt. *Anaphora and Semantic Interpretation.* The University of Chicago Press, Chicago, 1983.

[38] G. Riccardi, R. Pieraccini, and E. Bocchieri. Stochastic Automata for Language Modeling. *Computer Speech and Language 10*, 1996.

[39] J.J. Rissanen. Generalized Kraft Inequality and Arithmetic Coding. *IBM J. Research and Development*, 20, 198-203, May 1976.

[40] J. Rissanen. Probability Estimation for Symbols - Observed or Not. *International Symposium on Information Theory*, (Quebec, CN), 1983.

[41] J. Rissanen. A Universal Data Compression System. *IEEE Trans. Inform. Theory IT-29*, 5, 1983.

[42] J. Rissanen. Complexity of Strings in the Class of Markov Sources. *IEEE Trans. Inform. Theory IT-32*, 4, 1986.

[43] Eric Sven Ristad. *The Language Complexity Game*. MIT Press, MA, 1993.

[44] Eric Sven Ristad and Peter N. Yianilos. Library of Practical Abstractions.

[45] Eric Sven Ristad. A Natural Law of Succession. *Technical Report CS-TR-495-95*, Princeton University, 1995.

[46] Eric Sven Ristad and Robert G. Thomas. Hierarchical Non-emitting Models. *Proc. of 35th ACL*, 1997.

[47] Eric Sven Ristad and Robert G. Thomas. Hierarchical Non-emitting Models. *Technical Report CS-TR-544-97*, 1997.

[48] Eric Sven Ristad and Robert G. Thomas. Nonmonotonic Extension Models. *Technical Report CS-TR-486-95*, Princeton University, 1995.

[49] Eric Sven Ristad and Robert G. Thomas. Nonuniform Markov Models. *Technical Report CS-TR-536-96*, Princeton University, 1995.

[50] Eric Sven Ristad and Robert G. Thomas. Nonuniform Markov Models. *IEEE ICAASP Conference*, 1997.

[51] Eric Sven Ristad and Robert G. Thomas. New Techniques for Context Modeling. *Proc. of 33rd ACL*, 1995.

[52] Eric Sven Ristad and Robert G. Thomas. Context Models in the MDL Framework. *IEEE Data Compression Conference*, 1995.

[53] Eric Sven Ristad and Peter N. Yianilos. Finite Growth Models. *Technical Report CS-TR-533-96*, Princeton University, 1996.

[54] Roni Rosenfeld. Adaptive Statistical Language Modeling: A Maximum Entropy Approach. *Ph.D. Thesis, Computer Science Department, Carnegie Mellon University*, CMU-CS-94-138, 1994.

[55] T. Sejnowski and C.R. Rosenberg. Parallel Networks that Learn to Pronounce English Text. *Complex Systems*, 1:145-168, 1987.

[56] C.E. Shannon. A Mathematical Theory of Communication. *Bell System Technical Journal*, 27, 398-403, July, 1948.

[57] C.E. Shannon. Prediction and Entropy of Printed English. *Bell System Technical Journal*, 50-64, 1951.

[58] Daniel D.K. Sleator and Davy Temperley. Parsing English with a Link Grammar. *Technical Report CMU-CS-91-196*, Carnegie Mellon University, 1991.

[59] Kari Torkkola. An Efficient Way to Learn English Grapheme-to-Phoneme Rules Automatically. *Proc. International Conference on Acoustics, Speech, and Signal Processing*, vol. 2, 1993.

[60] Peter N. Yianilos. *Topics in Computational Hidden State Modeling*. Ph.D. Dissertation. Princeton University Computer Science Department, June 1997.

[61] François Yvon. Grapheme-to-Phoneme Conversion Using Multiple Unbounded Overlapping Chunks. *Proc. Conference on New Methods in Natural Language Processing*, 1996.

[62] François Yvon. *Prononcer Par Analogie: Motivations, Formalisations, et Évaluations* Thèse de Doctorat de l'Ecole Nationale Supérieure des Télécommunications, ENST E 015, Paris (France), 1996.

[63] François Yvon. Paradigmatic Cascades: a Linguistically Sound Model of Pronunciation by Analogy. *Proc. 35th annual meeting of the Association for Computational Linguistic*, Madrid, July 97.

[64] J. Ziv and A. Lempel. A Universal Algorithm for Sequential Data Compression. *IEEE Trans. Information Theory*, IT-23 (3), 337-343, 1977.