“What has been will be again”: A Machine Learning Approach to the Analysis of Natural Language

Thesis submitted for the degree “Doctor of Philosophy”

Yoram Singer

Submitted to the Senate of the Hebrew University in the year 1995

This work was carried out under the supervision of
Prof. Naftali Tishby

Acknowledgments

I am deeply grateful for the guidance and support of my advisor, Prof. Naftali Tishby. I am grateful to Tali for giving me a start on research, for his generous financial support, for encouraging me throughout my studies, and for his friendship.

Thanks to Dana Ron for being such a great collaborator and for the many things I learned during our work together.

I wish to give special thanks to Manfred Warmuth, Dave Helmbold and David Haussler for their friendship and hospitality during my stays at the University of California at Santa Cruz.

Thanks to Hinrich Schütze for a fruitful collaboration and for introducing me to computational linguistics.

I would also like to thank Ido Dagan, Peter Dayan, Shlomo Dubnov, Shai Fine, Yoav Freund, Gil Fucs, Itay Gat, Mike Kearns, Scott Kirkpatrick, Fernando Pereira, Ronitt Rubinfeld, Rob Schapire, Andrew Senior, and Daphna Weinshall, for being valuable friends and colleagues.

Finally, I am very grateful for the generous financial support provided by the Clore foundation.

Contents

Abstract

'Understanding' human communication, whether printed, written, spoken or even gestured, is one of the long standing goals of artificial intelligence. The broad and multi-disciplinary research on language analysis has demonstrated that human language is far too complex to be captured by a fixed set of prescribed rules, and that major efforts should be devoted to computational models and algorithms for automatic machine learning from past experience. This thesis focuses on new directions in analysis of natural language from the standpoint of computational learning. The emphasis of the thesis is on *practical* methods that automatically acquire or approximate the structure of natural language. A substantial part of the work is devoted to the theoretical aspects of the proposed models and their learning algorithms.

We start with a model-based approach to on-line cursive handwriting analysis. In this model, on-line handwriting is considered to be a modulation of a simple cycloidal pen motion, described by two coupled oscillations with a constant linear drift along the writing line. A general pen trajectory is efficiently encoded by slow modulations of amplitudes and phase lags of the two oscillators. The motion parameters are then quantized into a small number of values without altering the intelligibility of writing. A general procedure for the estimation and quantization of these cycloidal motion parameters for arbitrary handwriting is presented. The result is a discrete *motor control representation* of continuous pen motion, via the quantized levels of the model parameters.

The next chapters explore the issue of modeling complex temporal sequences such as the motor control commands. Two subclasses of probabilistic automata are investigated: acyclic state-distinguishable probabilistic automata, and variable memory Markov models. Whereas general probabilistic automata are hard to infer, we show that these two subclasses are efficiently learnable. Several natural language analysis problems are presented and the proposed learning algorithm is used to automatically acquire the structure of the temporal sequences that arise in language analysis problems. In particular, we show how to approximate the distribution of the possible pronunciations of spoken words, acquire the structure of noun phrases in natural printed text, build a model for the English language and use the model to correct corrupted text. We also design a model for E.*coli* DNA which can be used to parse DNA strands. We end this part with a description and evaluation of a complete system, based on a variable memory Markov model, that assigns the proper part-of-speech tag for words in an English text.

Chapter 5 combines the various models and algorithms to build a complete cursive handwriting recognition system. The approach to recognizing cursive scripts consists of several stages. The first is the dynamical encoding of the writing trajectory into a sequence of discrete motor control symbols, as presented at the beginning of the thesis. In the second stage, a set of acyclic probabilistic finite automata which model the distribution of the different cursive letters are used to calculate the probabilities of subsequences of motor control commands. Lastly, a language model, based on a Markov model with variable memory length, is used for selecting the most likely transcription of a written script. The learning algorithms presented and analyzed in this thesis are used for training the system. Our experiments show that about 90% of the letters are correctly identified. Moreover, the training (learning) and recognition algorithms are very efficient and the online versions of the automata learning algorithms can be used to adapt to new writers with new writing styles, yielding a robust start-up recognition scheme.

Chapter 1

Introduction

As humans, we have day to day experience as fast and maleable learners of language. Language serves us primarily as a means of communication and exists in different forms. The first goal of automatic methods that acquire and analyze the structure of natural language is to build machines that will aid us in everyday tasks. Dictation machines, printed text readers, speech synthesizers, natural interfaces to databases, etc., are just some of the more striking examples of such machines. Besides the practical benefits of designing machines that learn, the study of machine learning may help us understand many aspects of human intelligence better, in particular language learning, acquisition and adaptation.

This thesis focuses on models and algorithms that use past experience, in other words, ones that *learn from examples*. An alternative approach to the analysis of human language is to build specialized systems. This involves designing and implementing fully determined systems that include predefined rules for each possible input sequence that represent one of the possible forms of human communication. This approach is also referred to as the knowledge-based or knowledge-engineering approach. There are situations where a fixed prescribed set of rules suffice to perform a limited task. For example, industrial machines such as robots that assemble cars or bar-code readers, usually employ a knowledge engineering approach. The same paradigm was also applied to language analysis problems, with the assumption that if a human can interpret language, it should be possible to find the invariant parameters and mechanisms used in the language understanding process. For instance, several speech recognition systems employ a knowledge engineering approach by associating each phoneme with a set of rules extracted by experts who can read spectrograms. A spectrogram is a color-coded display of the power spectrum of the acoustic signal. There are people who can 'decode' these displays and recognize what was said by looking at a spectrogram without actually hearing the acoustic signal. The rules these experts use to read spectrograms can be quantitatively defined and incorporated into a speech recognition system. Similar ideas have been developed by several researchers in the field of handwriting analysis and recognition. For examples of different implementations and reviews of the knowledge-based approach, see [29, 30, 41, 59, 143, 46, 57, 128]. Although such systems achieve moderate performance on limited tasks, the recent research of language has demonstrated that natural language is far too complex to be captured by a fixed set of prescribed rules. Thus, major efforts should be devoted to computational models and algorithms for automatic machine learning from past experience.

In the machine learning approach, rules or mappings are inferred from examples. The first question that arises is the definition of 'examples'. Natural language takes on different forms, such as the acoustic signal for spoken language and drawings of letters on paper for written text. Therefore, we need to decide how to represent such signals prior to the design and implementation of a learning algorithm. Once a representation of the input has been chosen, we can find a rule that maps the chosen representation to a yet more compact representation, such as the phonetic transcription in the case of speech, and the ascii format for written text. The inferred rule may be chosen from a predefined, yet arbitrarily large and possibly infinite, set of rules. The rule is then applied to new examples to perform tasks such as prediction, classification and identification. The second question that immediately arises is what classes of mappings/rules can be used. For example, is it at all possible to consider all functions that map an acoustic signal to the sequence of words which were uttered ? If not, what classes of rules are 'reasonable' ? If we choose a rich class of rules, we might find one that performs well. However, searching for a good rule from a huge complex set might be intractable. Therefore, a primary goal is to identify classes of rules that are rich enough to capture the complex structure of natural language, on the one hand, and are simple enough to be learned efficiently by machines, on the other. The overview below presents the existing approaches and methodologies dealing with the problems of representation and learning.

Machine Representation of Human Generated Signals

There is a large variety of forms and ways to represent human generated signals, as demonstrated in Figure 1.1. This section reviews machine representations of spoken, written and printed language, that are used for language analysis. We also discuss some of the advantages and disadvantages of current representation methods.

Figure 1.1: Graphical representations of the word `information`.

Both spoken and written language can be viewed as a sequence of signals generated by a physical dynamical system controlled by the brain in order to transmit and carry the relevant information. In the case of speech, the dynamics is that of the articulatory system, whereas in handwriting the controlled system is another motor system, namely, the human arm. Similarly, limb movements generate signals in sign language that are interpreted by the visual system as in the case of handwriting. A major difficulty in the analysis of such temporal structures is the need to separate the intrinsic dynamics of the system from the more relevant information of the (unknown) control signals. A common practice is to first *preprocess* the input signal and transform it to a more compact representation. In most if not all speech and handwriting recognition systems this preprocessing is not reversible and finding an inverse transformation, from the more compact representation back to the input signal, is useless if not impossible.

In normal speech production, the chest cavity expands and contracts to force air from the lungs out through the trachea past the glottis. If the vocal cords are tensed, as for voiced speech such as vowels, they will vibrate in a relaxation oscillator mode, modulating the air into discrete puffs or pulses. If the vocal cords are spread apart, the air stream passes unaffected through the glottis yielding unvoiced sounds. Most of the preprocessing techniques employ a linear model as an approximation of the vocal tract [87]. First, the signal is sampled at a fixed rate. The waveform is then blocked into frames of fixed length. A common practice is to pre-filter the speech signal prior to linear modeling and perform a nonlinear logarithmic transformation on the resulting filter coefficients [105]. This *fixed* transformation is performed regardless of the speed of articulation, gender of the speaker, the quality of the recording, and many other factors. These transformations however may result in a loss of important information about the signal, such as its phase. Furthermore, the fixed rate analysis smoothes out rapid changes that frequently occur in unvoiced phones and carry important information about the uttered context.

Handwritten text can be captured and analyzed in two modes: off-line and on-line. In an off-line mode, handwritten text is captured by an optical scanner that converts the image written on paper into digitized bit patterns (pixels). Image processing techniques are then used to find the location of the written text and extract spatial features that are later used for tasks such as classification, identification and recognition [32]. In an on-line mode, a transducer device that continuously captures the location of the writing device is used. The temporal signal of pen locations is usually sampled at a fixed rate and then quantized, yielding a discrete sequence that represents the pen motion. Many capturing devices provide additional information, such as the instantaneous pressure applied and the proximity of the pen to the writing plane (when the pen is lifted from the paper). The resulting sequence is usually filtered, re-sampled, and smoothed. Then, features such as the local curvature and speed of the pen are extracted [128]. The purpose of the various transformations and feature extraction is to enforce invariances under distortions. However, the transformations may distort the signal and lose information that is relevant to recognition. Furthermore, the extracted features are fixed and *manually* determined by the designer who naturally cannot predict all the possible variations of handwritten text.

Higher levels of representations, such as parts-of-speech, are used for written text (*cf.* [22]). In large recognition systems, intermediate representations, such as phonemes and phones of speech, are used to categorize partially classified signals (*cf.* [107]). Such representations are discrete and usually constructed by system designers who incorporate some form of linguistic knowledge. The final representation level is usually a standard form of machine stored text such as the ascii format. The goal of research in machine learning is to design and analyze learning algorithms that infer rules that form mappings between the different representations, level to level up to the most abstract one. The next section briefly overviews the mathematical framework that has been developed within the computer science community to analyze and evaluate learning algorithms that infer such rules.

A Formal Framework for Learning from Examples

The study of models and mechanisms for learning has attracted researchers from different branches of science, including philosophy, linguistics, biology, neuroscience, physics, computer science, and electrical engineering. The approaches applied to the problem of learning vary immensely. An in-depth overview of approaches is clearly beyond the scope of this short introduction. For a comprehensive overview, see for instance the survey papers and books on learning and its applications, by Anderson and Rosenfeld [3], Rumelhart and McClelland [117] (connectionist approaches to learning), Holland [65] (genetic learning algorithms), Charniak [22] (statistical language acquisition), Dietterich [36], Devroye [35] (experimental machine learning), Duda and Hart [37] (pattern recognition), and collections of articles in [122].

In order to formally analyze learning algorithms, a mathematical model of learning must be defined first. The notion of mathematical study of learning is by no means new. It has roots in several research disciplines such as inductive inference [6], pattern recognition [37], information theory [31], probability theory [103, 38], and statistical mechanics [56, 120]. The model we mostly use in this thesis, known as the model of *probably approximately correct (PAC)* learning, was introduced by L.G. Valiant [133] in 1984. Valiant's paper has promoted research on formal models of learning known as *computational learning theory*. Computational learning theory stems from several different sources but the most influential study is probably the seminal work of Vapnik, dated in the seventies [134, 135]. The formal framework of computational learning is different from older works in inductive inference and pattern recognition in its emphasis on efficiency and robustness. The aim in building learning machines is to find algorithms that are efficient in their running time, memory consumption, and the amount of collected data required for efficient learning to take place. Robustness implies that the learning algorithms will perform well against *any* probability distribution of the data, and that the inferred rule need not be an exact mapping but rather a good approximation. Due to its relevance to the theoretical results presented in this thesis, we continue with a brief introduction to the PAC learning model.

In his paper, Valiant defined the notion of concept learning as follows: A concept is a rule that divides a domain of instances into a negative part and a positive part. Each instance in the domain is therefore assigned a label denoted by a plus or minus sign. The role of a learning algorithm is to find a good approximation of the concept. The learning algorithm has access to *labeled examples* and knowledge about the class of possible concepts. The output of a learning algorithm is a prediction rule, formally termed hypothesis, from the class of possible concepts. The examples are chosen from a fixed, yet *unknown* distribution. The *error* of a learning algorithm is the probability it will misclassify a new instance when the instance was picked in random according to the (unknown) target distribution. The PAC model requires that the prediction error of a learning algorithm could be made arbitrarily small: for each positive number ϵ the algorithm should be able to find a hypothesis with error less than ϵ. However, the algorithm is allowed to completely fail with a small probability which should be less than δ ($\delta > 0$). We will also refer to δ as a confidence value. In order to meet the efficiency demands, the running time of the algorithm and the number of examples provided should be polynomial in $1/\epsilon$ and $1/\delta$.

Since the publication of Valiant's paper, several extensions and modification to the PAC model have been suggested. For instance, models and algorithms for online learning, noisy examples, membership and equivalent queries have been suggested and analyzed (*cf.* [73]). The extension most relevant to this work is the notion of learning distributions [72]. In the distribution learning model, the learning algorithm receives *unlabeled* instances generated according to an unknown target distribution and its goal is to approximate this target distribution.

A hypothesis $\widehat{H}$ is an ϵ-*good* hypothesis with respect to a probabilistic distribution H if

$$D_{KL}[P_H \| P_{\widehat{H}}] \leq \epsilon \ ,$$

where P_H and $P_{\widehat{H}}$ are the distributions that H and $\widehat{H}$ generate, respectively. D_{KL} is the *Kullback-Leibler* divergence between the two distributions,

$$D_{KL}[P_H \| P_{\widehat{H}}] \stackrel{\text{def}}{=} \sum_{x \in \mathcal{X}} P_H(x) \log \frac{P_H(x)}{P_{\widehat{H}}(x)} \ ,$$

where $\mathcal{X}$ is the domain of instances. Although the *Kullback-Leibler* (KL) divergence was chosen as a distance measure between distributions, similar definitions can be considered for other distance measures such as the variation and the quadratic distance. The KL-divergence is also termed the cross-entropy and is motivated by information theoretic problems of efficient source coding as follows. The KL-divergence between H and $\widehat{H}$ corresponds to the average number of bits needed to encode instances drawn from the $\mathcal{X}$ using the probabilistic model $\widehat{H}$, when the actual distribution generating the examples is H. The KL-divergence bounds the variation (or L_1) distance as follows [31],

$$D_{KL}(P_1 \| P_2) \geq \frac{1}{2 \log 2} \| P_1 - P_2 \|_1^2 \ .$$

Since the L_1 norm bounds the L_2 norm, the last bound holds for the quadratic distance as well. We require that for every given $\delta > 0$ and $\epsilon > 0$, the learning algorithm outputs a hypothesis, $\widehat{H}$, such that with a probability of at least $1 - \delta$, $\widehat{H}$ is a ϵ-good hypothesis with respect to the target distribution H. The learning algorithm is *efficient* if it runs in time polynomial in $1/\epsilon$ and $1/\delta$ and the number of examples needed is polynomial in $1/\epsilon$ and $1/\delta$ as well.

Deterministic and Probabilistic Models for Temporal Sequences

One of the major goals of this work is to find classes of concepts that approximate the distribution of *temporal* sequences, and to design, analyze, and implement efficient learning algorithms for these classes while taking into account the complex nature of human generated sequences. We now give a brief overview of the more popular temporal models and the learning results concerning these models. The formal definitions of the models used in this thesis are deferred to later chapters.

Deterministic Automata

A Deterministic Finite Automaton (DFA) is a state machine in which each state is associated with a transition function and an output function. The transition function defines the next state to move to, depending on the current input symbol which belongs to a set called the input alphabet. The output function labels each state with a symbol from a finite set, termed the output alphabet. We may assume that the output alphabet is binary and each state is assigned a label, denoted by a + or a − sign. The results discussed here simply generalize for larger alphabets. A DFA has a single starting state. Thus, each input string is associated with a string of + and − signs that were output by the states while reading the string, starting from the start state. We say that a DFA accepted a string if the last symbol that was output by the automaton is +. Hence a state labeled by + is also referred to as an accepting state.

Deterministic finite automata are perhaps the simplest class among the classes of temporal models. This leads to the assumption that a general scheme for learning automata should exist. However, there are several intractability results which show that if the learning algorithm only has access to labeled examples, then the inference problem is hard. Gold [51] and Angluin [4] showed that the problem of finding the smallest automaton consistent with a set of positive and negative examples is NP-complete. Furthermore, in [100] Pitt and Warmuth showed that even finding a good approximation to the minimal consistent DFA is NP-hard and in [83] Li and Vazirani showed that finding an automaton 9/8 larger than the smallest consistent automaton is still NP-complete. Fortunately, there are situations where a DFA can be efficiently learned. Specifically, if the learning algorithm is allowed to choose its examples, then deterministic automata are learnable in polynomial time [50, 5]. Moreover, in [132, 45] it was shown that typical[1] deterministic automata can be learned efficiently. The performance of the learning algorithm for DFAs presented by Trakhtenbrot and Brazdin' in [132] was experimentally tested by Lang [80]. Although deterministic automata are too simple to capture the complex structure of natural sequences, these theoretical and experimental results had influence on the design and analysis of learning algorithms for probabilistic automata.

Probabilistic Automata

In its most general form, a Probabilistic Finite Automaton (PFA) is a probabilistic state machine known as a Hidden Markov Model (HMM). A separate section is devoted to HMMs and the focus here is on a more restricted class of PFAs which are sometimes termed unifilar HMMs. For brevity, we will refer to this subclass simply as PFAs. In a similar way to a DFA, a PFA is associated with a transition function. Each transition is associated with a symbol from the input alphabet and with a (nonzero) probability, such that the sum of probabilities of the transitions outgoing from a state is 1. The number of transitions is restricted such that at most one outgoing edge is labeled by each symbol from the alphabet. Such PFAs are probabilistic generators of strings. Alternatively, PFAs can be viewed as a measure over strings from the input alphabet. A PFA can have a single start state or an initial probability distribution over its states. In the latter case, the probability of a string is the sum of the probabilities of the state sequences that can generate the string, each weighted by the initial probability value of its first state.

The problem of learning PFAs from an *infinite* stream of strings was studied in [115, 34]. The analyses presented in those papers have the spirit of inductive inference techniques in the sense that the learner is required to output a sequence of hypotheses which converges to the target PFA *in the limit* of an arbitrarily large sample size. In [20], Carrasco and Oncina discuss an alternative algorithm for learning in the limit when the algorithm has access to a source of independently generated sample strings. As discussed previously, this type of analysis is not suitable for more realistic finite sample size scenarios. An important intractability result for learning PFAs, which is relevant to this work, was presented by Kearns *et. al.* in [72]. They show that PFAs are not

[1] DFAs in which the underlying graph is arbitrary, but the accept/reject labels on the states are chosen randomly.

efficiently learnable under the widely acceptable assumption that there is no efficient algorithm for learning noisy parity functions in the PAC model. Furthermore, the subclass of PFAs, which they show are hard to learn, are (width two) acyclic PFAs in which the distance in the L_1 norm (and hence also the KL-divergence) between the distributions generated starting from every pair of states is large.

An even simpler class of PFAs that has been studied extensively is the class of order L Markov chains. This model was first examined by Shannon [121] for modeling statistical dependencies in the English language. Markov models, also known as n-gram models, have been the prime tool for language modeling in speech recognition (*cf.* [69, 28]). While it has always been clear that natural texts are not Markov processes of any finite order [52], because of very long range correlations between words in a text such as those arising from subject matter, low-order alphabetic n-gram models have been used very effectively for such tasks as statistical language identification and spelling correction. Höffgen [63] also studied related families of Markov chains, where his algorithms depend exponentially and not polynomially on the order, or memory length, of the distributions.

Hidden Markov Models

Hidden Markov models are probably the most popular type of probabilistic automata, because of their general structure. HMMs have been applied to a wide variety of problems, such as speech recognition [82, 104], handwriting recognition [12, 47, 93], natural text processing [71] and biological sequence analysis [76, 48]. Each state of a hidden Markov model is associated with a probabilistic transition *and* output functions. In its most general form, the transition function at a state defines the probability of moving from that state to any other state of the model. At each state, the output probability function defines the probability of observing a symbol from the output alphabet. Thus, the states themselves are not directly observable. There are no known efficient learning algorithms for HMMs, although several ad-hoc learning procedures have been suggested lately (*cf.* [127]). A common practice is to estimate the parameters of a given model so as to maximize the probability of the training data by the model. This technique, called the *Baum-Welch* method or the *forward-backward* algorithm [9, 10, 11], is a special case of the EM (Expectation-Maximization) algorithm [33]. Although in practice the EM algorithm provides a powerful framework that yields good solutions in many real-world problems, this algorithm is only guaranteed to converge to a *local* maximum [142]. Thus, there is some doubt whether the hypothesis it outputs can serve as a good approximation for the target distribution. Alternative maximum likelihood parameter estimation techniques are based on nonlinear optimization techniques such as the steepest descent. However, these techniques, as well, guarantee convergence only to a local maximum of the parameters surface. Although there are hopes that the problem can be overcome by improving the algorithm used or by finding a new approach, there is strong evidence that the problem cannot be solved efficiently.

Abe and Warmuth [2] studied the problem of *training* HMMs. The HMM training involves approximating an arbitrary, unknown source distribution by distributions generated by HMMs. They show that HMMs are *not* trainable in time polynomial in the alphabet size, unless $RP = NP$. Gillman and Sipser [49] examined the problem of *exactly inferring* an (ergodic) HMM over a binary alphabet when the inference algorithm can query a *probability oracle* for the long-term probability of any binary string. They show that inference is hard: any algorithm for inference must make exponentially many oracle calls. Their method is information theoretic and does not depend on separation assumptions for any complexity classes.

Even if the algorithm is allowed to run in time exponential in the alphabet size, then there are no known algorithms which run in time polynomial in the number of states of the target HMM.

In addition, the successful applications of the HMM approach are mostly found in cases where its full power is not utilized. Namely, there is one, highly probable state sequence (the Viterbi sequence) whose probability is much higher than all the other state sequences. Thus, the states are actually not hidden [88, 89]. Therefore, in many real-world applications HMMs are used with the most likely state-sequence, which essentially restricts their distributions to those generated by PFAs [108, 104].

Despite these discouraging results, the EM-based estimation procedure for HMMs has in practice proved itself to be a powerful tool when combined with careful implementation, e.g., using cross-validation to prevent overfitting of the estimated parameters. An interesting and unresolved question is therefore to determine what is common to many of the learning problems that makes a hill-climbing algorithms such as EM work well.

Temporal Connectionist Models

A great deal of interest today has been sparked for connectionist models (*cf.* [117]) which are motivated and inspired by biological learning mechanisms. Generally (and informally) speaking, temporal connectionist models are characterized by a state vector from an arbitrary vector space, a state mapping function from the state space to itself, and output functions. The mapping and the output functions can be either deterministic or probabilistic. The mapping can be defined explicitly using parametric vector functions or implicitly via, for instance, a set of (stochastic) differential equations. Examples of such models are the Hopfield model, the Boltzman and Helmholtz machines, recurrent neural networks, and time (tapped) delay neural networks [61]. Extensive research of learning such models has been carried out in the last decade yielding genuine learning algorithms. Most of the learning algorithms search for 'good' parameters for a predefined model and roughly fall into two categories: gradient based search and exhaustive search methods (e.g., Monte-Carlo methods). Therefore, algorithms such as back propagation and the wake and sleep algorithm, although sophisticated, cannot guarantee a good approximation of the source from which the examples were drawn. Moreover, recent work (*cf.* [123]) shows that certain connectionist models are equivalent to a Turing machine. Therefore, the intractability results of learning deterministic and probabilistic automata clearly hold for temporal connectionist models as well. However, as in the case of HMMs, connectionist models have performed exceptionally well in real world applications (see for instance [1]). Therefore, the design of constructive algorithms for connectionist models and the analysis of the error of the models on real data is one of the more challenging and interesting research goals of theoretical and experimental machine learning.

Thesis Overview

Chapter 2 presents a new approach to discrete machine representation of cursively written text. As opposed to traditional approaches described in previous sections, we devise an adaptive estimation procedure (rather than a fixed transformation). Specifically, we describe and evaluate a model-based approach to on-line cursive handwriting analysis and recognition. In this model, on-line handwriting is considered to be a modulation of a simple cycloidal pen motion, described by two coupled oscillations with a constant linear drift along the line of the writing. By slow modulations of the amplitudes and phase lags of the two oscillators, a general pen trajectory can be efficiently encoded. These parameters are then quantized into a small number of values without altering the intelligibility of writing. A general procedure for the estimation and quantization of these cycloidal motion parameters for arbitrary handwriting is presented. The result is a discrete *motor control representation* of the continuous pen motion, via the quantized levels of the model parameters. This motor control representation enables successful recognition of cursive scripts as will be described in later chapters. Moreover, the discrete motor control representation greatly reduces the variability of different writing styles and writer specific effects. The potential of this representation for cursive script recognition is explored in detail in later chapters.

Chapter 3 proposes and analyzes a distribution learning algorithm for a subclass of *Acyclic Probabilistic Finite Automata* (APFA). This subclass is characterized by a certain distinguishability property of states of the automata. Here, we are interested in modeling short sequences, rather than long sequences, that can be characterized by the stationary distributions of their subsequences. This problem is conventionally addressed by using Hidden Markov Models (HMMs) or string matching algorithms. We prove that our algorithm can efficiently learn distributions generated by the subclass of APFAs we investigate. In particular, we show that the KL-divergence between the distribution generated by the target source and the distribution generated by our hypothesis can be made small with high confidence in polynomial time and polynomial sample complexity. We present two applications of our algorithm. In the first, we demonstrate how APFAs can be used to build multiple-pronunciation models for spoken words. We evaluate the APFA-based pronunciation models on labeled speech data. The good performance (in terms of the log-likelihood obtained on test data) achieved by the APFAs and the incredibly small amount of time needed for learning suggests that the learning algorithm of APFAs might be a powerful alternative to commonly used probabilistic models. In the second application, we show how the model combined with a dynamic programming scheme can be used to acquire the structure of noun phrases in natural text.

We continue to investigate practical learning algorithms for probabilistic automata in Chapter 4. In this chapter we propose and analyze a distribution learning algorithm for variable memory length Markov processes. These processes can be described by a subclass of probabilistic finite automata which we term *Probabilistic Suffix Automata*. Though results for learning distributions generated by sources with similar structure show that this problem is hard, the algorithm here is shown to efficiently learn distributions generated by the more restricted sources generated by probabilistic suffix automata. Here, as well, the KL-divergence between the distribution generated by the target source and the distribution generated by the hypothesis output by the learning algorithm can be made small with high confidence in polynomial time and sample complexity. We discuss and evaluate several applications based on the proposed model. First, we apply the algorithm to construct a model of the English language, and use this model to correct corrupted text. In the second application we construct a simple stochastic model for E.*coli* DNA. Lastly, we describe, analyze, and discuss an implementation of a part-of-speech tagging system based on a variable memory length Markov model. While the resulting system is much simpler than state-of-the-art tagging systems, its performance is comparable to any of the published systems.

In Chapter 5 we describe how the various models and learning algorithms presented in the previous chapters can be combined to build a complete system that recognizes cursive scripts. Our approach to cursive script recognition involves several stages. The first is the dynamical encoding of the writing trajectory into the sequence of discrete motor control symbols presented in Chapter 2. In the second stage, a set of acyclic probabilistic finite automata, which model the distribution of the different cursive letters, are used to calculate the probabilities of subsequences of motor control commands. Finally, a language model, based on a Markov model with variable memory length, is used to select the most likely transcription of a written script. The learning algorithms presented and analyzed in Chapters 3 and 4 are used to train the system. Our experiments show that about 90% of the letters are correctly identified. Moreover, the training (learning) and recognition algorithms are very efficient and the online versions of the automata learning algorithms are used to adapt to new writers with new writing styles, enabling a robust startup recognition scheme.

We give conclusions, mention some important open problems, and suggest directions for future research in Chapter 6.

Chapter 2

Dynamical Encoding of Cursive Handwriting

2.1 Introduction

Cursive handwriting is a complex graphic realization of natural human communication. Its production and recognition involve a large number of highly cognitive functions including vision, motor control, and natural language understanding. Yet the traditional approach to handwriting recognition has focused so far mostly on computer vision and computational geometric techniques. The recent emergence of pen computers with high resolution tablets has made available dynamic (temporal) information as well and created the need for robust on-line handwriting recognition algorithms. Considerable effort has been spent in the past years on on-line cursive handwriting recognition (for general reviews see [101, 102, 128]), but there are no robust, low error rate recognition schemes available yet.

Research of the motor aspects of handwriting has suggested that the pen movements produced during cursive handwriting are the result of 'motor programs' controlling the writing apparatus. This view was used for natural synthesis of cursive handwriting (see e.g., E. Doojies, pp. 119–130 in [102]). There have been several attempts to construct dynamical models of handwriting for recognition. Some of these works are based on a similar approach to ours (e.g., D.E. Rumelhart in [116]). None of the previous works, however, have actually solved the inverse dynamics problem of 'revealing' the 'motor code' used for the production of cursive handwriting.

Motivated by the oscillatory motion model of handwriting, as introduced by, e.g., Hollerbach [66], we develop a robust parameter estimation and regularization scheme which serves for the analysis, synthesis, and coding of cursive handwriting. In Hollerbach's model, cursive handwriting is described by two independent oscillatory motions superimposed on a constant linear drift along the line of writing. When the parameters are fixed, the result of these dynamics is a cycloidal motion along the line of the drift (see Figure 2.1). By modulations of the cycloidal motion parameters, arbitrary handwriting can be generated. The difficulty, however, is to generate writing by a *low rate* modulation, much lower than the original rate of the oscillatory signals. In this work, we propose an efficient low rate encoding of the cycloidal motion modulation and demonstrate its utility for robust synthesis and analysis of the process.

The pen trajectory is discretized in time by considering only the zero vertical velocity points. In between these points, the handwriting is approximated by an unconstrained cycloidal motion using the values of the parameters estimated at the zero vertical velocity points. Further, we show that the amplitude modulation can be quantized to a small number of levels (five for the vertical amplitude modulation and three for the horizontal amplitude modulation), and the results are robust. The vertical oscillation is described as an almost synchronous process, i.e. the angular velocity is transformed to be constant. The horizontal oscillation is then described in terms of its phase lag to the vertical oscillation and thus becomes synchronous as well. The modeling and estimation processes can be viewed as a many-to-one mapping, from the continuous *pen motion* to a discrete set of *motor control symbols*. While this dramatically reduces the coding bit rate, we show that the relevant recognition information is regularized and preserved.

This chapter is organized as follows. In Section 2.2, we discuss Hollerbach's model and demonstrate its advantages in representing handwriting over standard geometric techniques. In Section 2.3, we describe our analysis-by-synthesis methodology and define the goal to be an efficient motor encoding of the process. In Section 2.4 we introduce two global transformations: correction of the writing orientation and slant equalization. We show that such preprocessing further assists in regularizing the process, which simplifies the parameter estimation phase. In Section 2.5 we discuss the model's parameter estimation. Sects. 2.6 through 2.8 introduce a series of quantizations and discretizations of the dynamic parameters, which both lower the encoding bit rate and improve the readability of the writing. Section 2.9 summarizes the discrete representation of the cursive handwriting process and shows that this representation is stable in the sense that similar words result in similar *motor control symbols*. Finally, in Section 2.10 we briefly discuss the usage of the motor control symbols for cursive scripts recognition and other related tasks.

2.2 The Cycloidal Model

Handwriting is generated by the human motor system, which can be described by a spring muscle model near equilibrium. This model assumes that the muscle operates in the linear small deviation regions. Movements are excited by selecting a pair of agonist-antagonist muscles, modeled by a spring pair. If we further assume that the friction is balanced by an equal muscular force, then the process of handwriting can be approximated by a system of two orthogonal opposing pairs of ideal springs. In a general form, the spring muscle system can be described by the following differential equation

$$M \begin{bmatrix} \ddot{x} \\ \ddot{y} \end{bmatrix} = -K \begin{bmatrix} x \\ y \end{bmatrix} \quad , \tag{1.1}$$

where M and K are 2×2 matrices that can be diagonalized simultaneously. This system can be transformed to a diagonalized system described by the following decoupled equations set

$$\begin{cases} M_x \ddot{x} = K_{1,x}(x_1 - x) - K_{2,x}(x - x_2) \\ M_y \ddot{y} = K_{1,y}(y_1 - y) - K_{2,y}(y - y_2) \end{cases} , \tag{1.2}$$

where $K_{1,x}, K_{2,x}, K_{1,y}, K_{2,y}$ are the spring constants, and x_1, x_2, y_1, y_2 are the spring equilibrium positions. Solving these equations with the initial condition that the system has a constant velocity (drift) in the horizontal direction yields the following parametric form

$$\begin{cases} x(t) = A\cos(\omega_x(t - t_0) + \phi_x) + C(t - t_0) \\ y(t) = B\cos(\omega_y(t - t_0) + \phi_y) \end{cases} . \tag{1.3}$$

The angular velocities ω_x and ω_y are determined by the ratios between the spring constants and masses. $A, B, C, \phi_x, \phi_y, t_0$ are the integration parameters determined by the initial conditions. This set describes two independent oscillatory motions, superimposed on a linear constant drift along the line of writing, generating cycloids. Different cycloidal trajectories can be achieved by changing the spring constants and zero settings at the appropriate time. The relationship between the horizontal amplitude modulation $A_x(t)$, the horizontal drift C, and the phase lag, $\phi(t) = \phi_x(t) - \phi_y(t)$, controls the letter corner shape (cusp), as demonstrated in Figure 2.1.

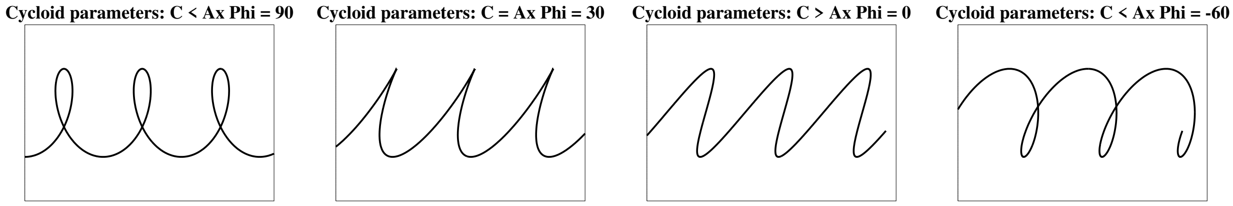


Figure 2.1: Various cycloidal writing curves.

We further restrict the model by assuming that the angular velocities are tied, i.e. $\omega_x(t) \approx \omega_y(t) \stackrel{\Delta}{=} \omega(t)$, and that $\phi_y(t) = 0$. These assumptions are not too restrictive, as will be shown later. With these assumptions, the equations governing the oscillations in the velocity domain can be written as

$$\begin{cases} V_x(t) = A_x(t)\sin\left(\omega(t)(t - t_0) + \phi(t)\right) + C \\ V_y(t) = A_y(t)\sin\left(\omega(t)(t - t_0)\right) \end{cases}, \tag{1.4}$$

where t_0 is the writing onset time, $A_x(t)$ and $A_y(t)$ are the horizontal and the vertical instantaneous amplitude modulations, $\omega(t)$ is the instantaneous angular velocity, $\phi(t)$ is the horizontal phase lag, and C is the horizontal drift velocity. By definition, the oscillation phase $\theta(t) = \int_0^t \omega(t)dt$ is monotonic in time. Hence, the time parameterization of the velocity equations can be changed, using the chain rule, $\frac{dX}{d\theta} = \frac{dX}{dt}\frac{dt}{d\theta}$, to phase parameterization of the following form

$$\begin{cases} V_x(\theta) = A_x(\theta)\sin(\theta + \phi(\theta)) + C\left[\frac{dt}{d\theta}\right] \\ V_y(\theta) = A_y(\theta)\sin(\theta) \end{cases}. \tag{1.5}$$

As already demonstrated, different cycloid parameters yield different letter forms. The transition from one letter to another can be achieved by a gradual change in the parameter space. A smooth pen trajectory can be obtained in this way. Standard differential geometry parameterizations (e.g., curvature versus arc-length), however, have difficulties expressing infinite curvature (corners), which are handled naturally in our model. This problem is demonstrated in Figure 2.2. In this simple example, a cycloid trajectory was produced by setting the parameters $A_x(t) = A_y(t) = C = 1$ and gradually changing $\phi(t)$ from $0°$ to $+180°$. The resulting trajectory after integration of the velocities is a smooth curve which has the form of the letter `w`. However, the curvature diverges at the middle cusp.

2.3 Methodology

Using the velocity equation presented in the previous section, handwriting can be represented as a slowly varying dynamical system whose control parameters are the cycloidal parameters $A_x(t), A_y(t)$, and $\phi(t)$. In this work, it is shown that these dynamical parameters have an efficient discrete coding that can be represented by a discretely controlled, dynamical system. The

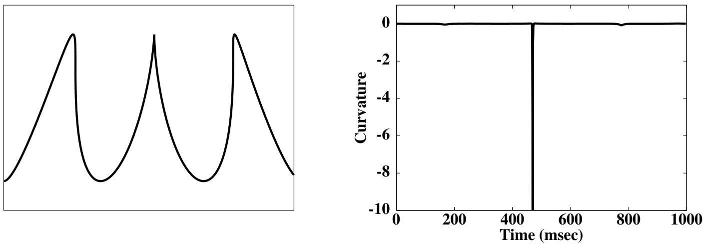


Figure 2.2: A synthetic cycloid and its curvature.

inputs to this system are *motor control symbols* which define the instantaneous cycloidal parameters. These parameters change only at restricted times. Our motor system 'translates' these motor control symbols to continuous arm movements. An illustration of this system is given in Figure 2.3 where the system is denoted by H and the control symbols by (x_i, y_i).

Decoding and recognition, as implied by this model, are done by solving an *inverse dynamics* problem. The following sections describe our solution to this inverse problem. A series of parameter estimation schemes that reveal the discrete control symbols is presented. Each stage in the process is verified via an *analysis-by-synthesis* technique. This technique uses the estimated parameters and the underlying model to reconstruct the trajectory. At every stage the synthesized curve is examined to determine whether the relevant recognition information is preserved. The result is a mechanism that maps the continuous pen trajectories to the discrete motor control symbols. A more systematic approach which uses control theoretical schemes is being developed.

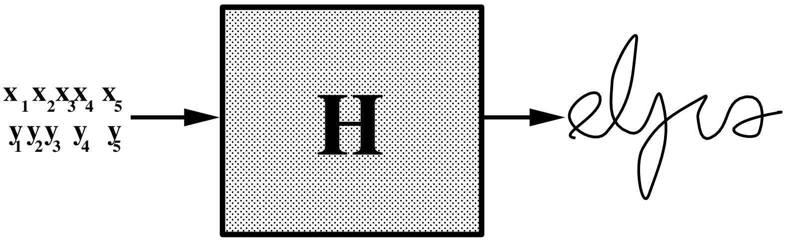


Figure 2.3: A discrete controlled system that maps *motor control symbols* to pen trajectories.

2.4 Global Transformations

On-line handwriting need not be oriented horizontally, and usually the handwriting is slanted. In this section normalization processes that eliminate different writing orientations and writing slants are described. These transformations are performed prior to any modeling to make the input scheme more robust. In this process, we do not estimate any of the dynamic parameters but use the general form of the dynamic equations.

2.4.1 Correction of the Writing Orientation

On-line handwriting is sampled in a general unconstrained position. This results in a non-horizontal direction of writing. Even when the writing direction is horizontal, there are position variations due to the oscillations; thus, the general orientation is defined as the average slope of the trajectory. Robust statistic estimation [138] is used to estimate the general orientation, rather than a simple linear regression, since there are measurement errors both in the vertical and the horizontal pen positions. The sampled points $(X(i), Y(i))$ are randomly divided into pairs $\{(X(2i_k), Y(2i_k)), (X(2i_k+1), Y(2i_k+1))\}$, such that $X(2i_k+1) \geq X(2i_k)$. The estimated writing orientation is $\hat{W} = \frac{\sum_k \{Y(2i_k+1) - Y(2i_k)\}}{\sum_k \{X(2i_k+1) - X(2i_k)\}}$. The angle of the writing direction α is $\tan^{-1}\hat{W}$, and the velocity vectors are rotated as follows

$$\begin{cases} V'_x(t) = & V_x(t)\cos(\alpha) & + & V_y(t)\sin(\alpha) \\ V'_y(t) = & -V_x(t)\sin(\alpha) & + & V_y(t)\cos(\alpha) \end{cases} \quad . \tag{1.6}$$

2.4.2 Slant Equalization

Handwriting is normally slanted. In the spring muscle model, this implies that the spring pairs are not orthogonal and only the general (1.1) is valid. The amount of coupling can be estimated by measuring the correlation between the horizontal and vertical velocities. Removing the slant is equivalent to decoupling the oscillation equations. This decoupling is desired since it is a writer-dependent property which does not contain any context information. The decoupling enables an independent estimation of the oscillation parameters for the *phase-lag regularization* stage, described in Section 2.7, and simplifies the estimation scheme.

The decoupling can be viewed as a transformation from a nonorthogonal to an orthogonal coordinate system in which one of the axes is the direction of writing. The horizontal velocity after slant equalization (denoted by $\tilde{V}_x$) is statistically uncorrelated with the vertical velocity V_y ($\tilde{V}_x \perp V_y$). Therefore, the original velocity can be written as $V_x(t) = \tilde{V}_x + A(t)V_y(t)$. Assuming stationarity, this requirement means that $E(\tilde{V}_x V_y) = 0$ and $A(t) = A$. If we assume that the slant is almost constant, then the stationarity assumption holds. The *maximum likelihood* estimator for A, assuming that the measurement noise is Gaussian, is

$$\hat{A} = \frac{E(V_x V_y)}{E(V_y V_y)} = \frac{\sum_{t=1}^{N} V_x(t)V_y(t)}{\sum_{t=1}^{N} V_y(t)V_y(t)} \quad . \tag{1.7}$$

There are writers whose writing slant changes significantly even within a single word. For those writers, the *projection coefficient* $A(t)$ is estimated locally. We assume though that along a short interval the slant is constant (local stationarity assumption). In order to estimate $A(t_0)$ we compute the short time correlation between $V_x(t)$ and $V_y(t)$ after multiplying them by a window centered at t_0

$$\hat{A}(t_0) = \frac{\sum_{t=1}^{N} V_x(t)V_y(t)W(t_0 - t)}{\sum_{t=1}^{N} V_y(t)V_y(t)W(t_0 - t)} \quad , \tag{1.8}$$

where W is a *Hanning window*[1], frequently used in *short time Fourier analysis* applications [96]. We empirically set the width of the window to contain about 5 cycles of V_y. After finding $\hat{A}(t)$ (or $\hat{A}$ if we assume a constant slant) the horizontal velocity after slant equalization is $\tilde{V}_x(t) = V_x(t) - \hat{A}(t)V_y(t)$. The slant equalization process is depicted in Figure 2.4, where the original handwriting is shown with the handwriting after slant equalization with a stationary slant assumption.

[1] Hanning window of length N is defined as $W_{Hanning}(n) = \frac{1}{2}\left(1 - \cos(\frac{2\pi n}{N-1})\right)$.

Figure 2.4: The result of the slant equalization process.

2.5 Estimating the Model Parameters

The cycloidal Equation (1.4) is too general. The problem of estimating its continuous parameters is ill-defined since there are more parameters than observations. Therefore, we would like to constrain the values of the parameters while preserving the intelligibility of the handwriting. It is shown in this section that by restricting the values of the parameters, a compact coding of the dynamics is achieved while preserving intelligibility.

Assuming that the model is a good approximation of the true dynamics, then the horizontal drift, C, can be estimated as follows, $\hat{C} = \frac{1}{N}\sum_{i=1}^{N} V_x(n)$, where N is the number of digitized points. Under the model assumptions, $\hat{C}$ converges to C and is an unbiased estimator. In order to check the assumption that C is really constant we calculated it for different words and locally within a word using a sliding window. The small variations in the estimator $\hat{C}$ indicate that our assumption is correct. At this point we perform one more normalization by dividing the velocities $V_x(t)$ and $V_y(t)$ by $\hat{C}$. The result is a set of normalized equations with $C = 1$. Henceforth, the constant drift is subtracted from the horizontal velocity and it is added whenever the spatial signal is reconstructed. Integration of the normalized set results in a fixed height handwriting, independent of its original size. The normalizations and transformations presented so far are supported by physiological experiments [64, 79] that show evidence of spatial and temporal invariance of the motor system.

We assume that the cycloidal trajectory describes the natural pen motion between the velocity zero-crossings and changes in the dynamical parameters occur at the zero-crossings only, to keep the continuity. This assumption implies that the angular velocities $\omega_x(t), \omega_y(t)$ and the amplitude modulation $A_x(t), A_y(t)$ are almost constant between consecutive zero-crossings. A good approximation can be achieved by identifying the velocity zero-crossings, setting the local angular velocities to match the time between two consecutive zero-crossings, and setting the velocities to values such that the total pen displacement between two zero-crossings is preserved. Denote by t_i^x and t_i^y the ith zero-crossing of the horizontal and vertical locations, and by L_i^x and L_i^y the horizontal and vertical progression during the ith interval (after subtracting the horizontal drift), respectively. The estimated amplitudes are

$$
\begin{array}{l}
\int_{t_i^x}^{t_{i+1}^x} \hat{A}_i^x \sin(\frac{\pi}{t_{i+1}^x - t_i^x}(t - t_i^x))dt = L_i^x \Rightarrow \hat{A}_i^x = \frac{L_i^x \pi}{2(t_{i+1}^x - t_i^x)} \\
\int_{t_i^y}^{t_{i+1}^y} \hat{A}_i^y \sin(\frac{\pi}{t_{i+1}^y - t_i^y}(t - t_i^y))dt = L_i^y \Rightarrow \hat{A}_i^y = \frac{L_i^y \pi}{2(t_{i+1}^y - t_i^y)}
\end{array} .
$$

The angular velocities are set independently and the phase lag, $\phi(t)$, is currently set to 0. The result of this process is a compact representation of the writing process, demonstrated by the resynthesized curve which is similar to the original, as shown in Figure 2.5.

At this stage we can represent the writing process as two statistically independent, single-dimensional, oscillatory movements. Free oscillatory movement is assumed between consecutive zero-crossings, while switching of the dynamic parameters occurs only at these points.

Each of the original sampled points, denoted by (x, y), is quantized to 8 bits. Quantizing the amplitudes and the zero-crossings indices to 8 bits reduces the number of bits needed to represent the curve, as shown in Figure 2.9. The original code length is indexed as stage 1. Stage 2 is the velocity approximation described in this section. The total description length of the trajectory at this point is reduced by a factor of 7.

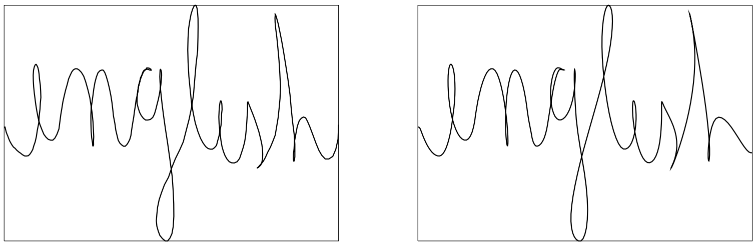


Figure 2.5: The original and the reconstructed handwriting after amplitudes coding.

2.6 Amplitude Modulation Discretization

The amplitudes $A_x(t), A_y(t)$ define the vertical and horizontal scale of the letters. From measurements of written words, the possible values of these amplitudes appear to be limited to a few typical values with small variations. We assume statistical independence of the amplitude values and perform discretization separately for the horizontal and vertical velocities. Nevertheless strong correlations remain between the velocities, which can be reduced in later stages.

2.6.1 Vertical Amplitude Discretization

Examination of the vertical velocity dynamics reveals the following:

- There is a virtual center of the vertical movements. The pen trajectory is approximately symmetric around this center.
- The vertical velocity zero-crossings occur while the pen is at almost fixed vertical levels, which correspond to high, normal and small modulation values.

These observations are presented in Figure 2.6, where the vertical position is plotted as a function of time. Using this apparent quantization we allow five possible pen positions, denoted by $H_1, \cdots, H_5$, which satisfy the symmetry constraints, $\frac{1}{2}(H_1 + H_5) = \frac{1}{2}(H_2 + H_4) = H_3$. Let, $\alpha = H_2 - H_1 = H_5 - H_4$ and $\beta = H_3 - H_2 = H_4 - H_3$ (Figure 2.6). Then, the possible curve lengths are, 0 , α , β , $\alpha + \beta$, $\alpha + 2\beta$, 2β , $2(\alpha + \beta)$.

The five-level description is a qualitative view. The levels achieved at the vertical velocity zero-crossings vary around $H_1, \ldots, H_5$. The variation around each level is approximated by a normal

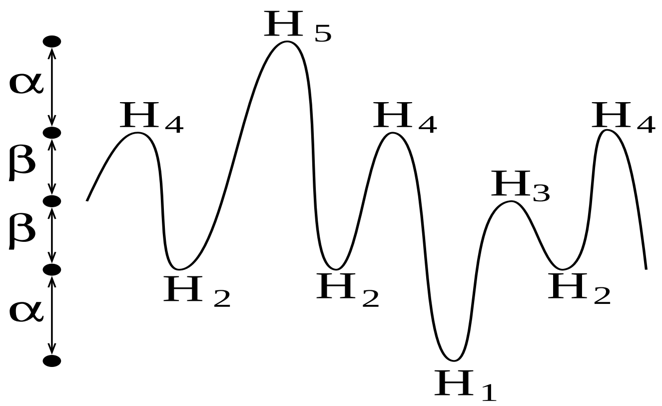


Figure 2.6: Illustration of the vertical positions as a function of time.

distribution with an unknown common variance. The distributions around the levels are assumed to be fixed and characteristic for each writer. Let I_t ($I_t \in 1, \ldots, 5$) be the level indicator, i.e., the index of the level obtained at the tth zero-crossing. We need to estimate concurrently the five mean levels $H_1, \ldots, H_5$, their common variance σ, and the indicators I_t. Yet the observed data are just the actual levels, $L(t)$, which are composed of the 'true' levels, H_{I_t}, and an observation gaussian noise ξ, $L(t) = H_{I_t} + \xi$ ($\xi \sim N(0, \sigma)$). Therefore, the *complete data* consist of the sequence of levels and indicators $\{I_t, L(t)\}$, while the observed data (also termed *incomplete data*) are just the sequence of levels, $L(t)$. The task of estimating the parameter $\{H_i, \sigma\}$ is a classical situation of *maximum likelihood* parameters estimation from *incomplete data*, commonly solved by the *EM* algorithm [33]. A full description of the use of *EM* in our case is given in Appendix A. The handwriting synthesized from the quantized amplitudes is depicted in Figure 2.7.

2.6.2 Horizontal Amplitude Discretization

The quantization of the horizontal progression between two consecutive velocity zero-crossings is simpler. In general, there are three types of letters, thin (like `i`), normal (`n`), and fat (`o`). These typical levels can be found using a standard scalar quantization technique.

2.7 Horizontal Phase Lag Regularization

After performing slant equalization, the velocities $V_x(t)$ and $V_y(t)$ are approximately statistically uncorrelated. Since $\omega_x \approx \omega_y$, the two velocities can be statistically uncorrelated if the phase lag between V_x and V_y is $\pm 90°$ on the average. Thus, the horizontal velocity, V_x, is close to its local extrema, while V_y is near zero, and vice versa. Since the phase lag changes continuously, a change from a positive phase lag to a negative one (or vice versa) must pass through $0°$. There are places of local halt in both velocities, so a zero phase lag is also common. When the phase lag is $0°$, the vertical and horizontal oscillations become coherent, and their zero-crossings occur at about the same time. These observations are supported by empirical evidence, as shown in Figure 2.8, where the horizontal and the vertical velocities of the word shown in Figure 2.4 are plotted. Note that the phase lag is likely to be $\pm 90°$ or $0°$. This phenomenon supports our discrete dynamical approach, and the phase lag between the oscillations is discretized to $\pm 90°$ or $0°$. We now describe how the best discrete phase-lag trajectory is found.

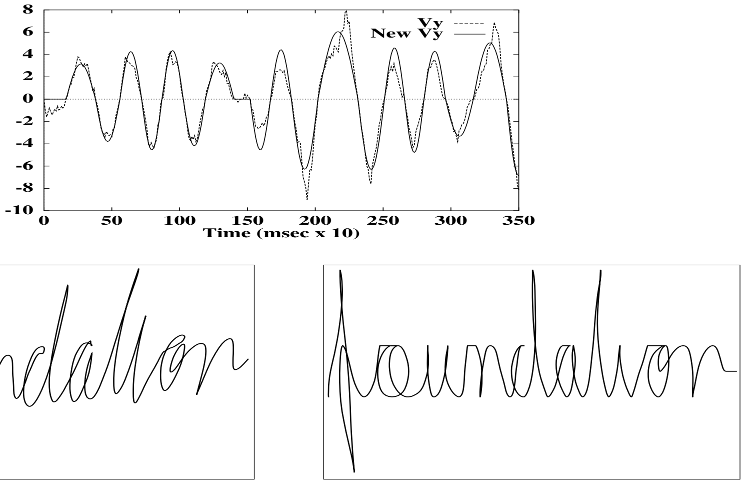


Figure 2.7: The original and the quantized vertical velocity (*top*), the original handwriting (*bottom left*), and the reconstructed handwriting after quantization of the horizontal and vertical amplitudes (*bottom right*).

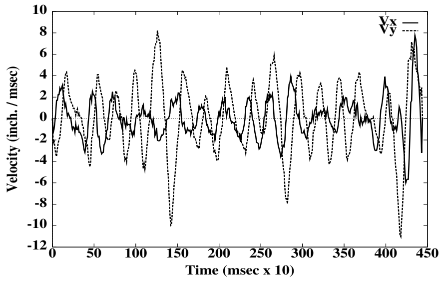


Figure 2.8: The horizontal and the vertical velocities of the word shown in Figure 1.4 (after removing the slant).

Examining the cycloidal model for each Roman cursive letter reveals that the horizontal to vertical angular velocity ratio is at most 2, i.e., $\max\left\{\frac{\omega_x}{\omega_y}, \frac{\omega_y}{\omega_x}\right\} \leq 2$. Thus, for English cursive handwriting the ratio $\frac{\omega_x}{\omega_y}$ is restricted to the range $[\frac{1}{2}, 2]$. Combining the angular velocity ratio limitations with the discrete set of possible phase-lags implies that the possible angular velocity ratios are: 1:1, 1:2, 2:1, 2:3, and 3:2. Four of these cases are plotted in Figure 2.9 with the corresponding spatial curves, assuming that the horizontal drift is zero. The vertical velocity V_y is plotted with a solid line and the horizontal velocity V_x with a dotted line.

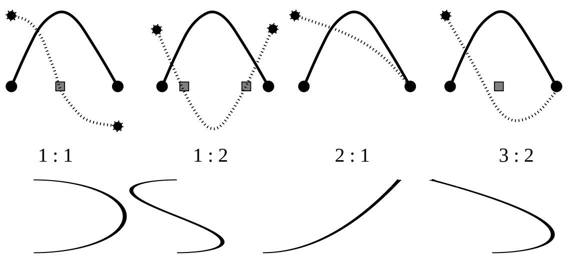


Figure 2.9: The possible phase-lag relations and the corresponding spatial curves.

We view the vertical velocity V_y as a 'master clock', where the zero-crossings are the clock onset times. V_x is viewed as a 'slave clock' whose pace varies around the 'master clock'. The rate ratio between the clocks is limited to at most 2. Thus, V_y induces a grid for V_x zero-crossings.

The grid is composed of V_y zero-crossings and multiples of quarters of the zero-crossings (the bold circles and the grey rectangles in Figure 2.10). V_x zero-crossings occur on a subset of the grid. The phase trajectory is defined over a subset of this grid, which is consistent with the discrete phase constraints and the angular velocities ratio limit. The allowed transitions for one grid point are plotted by dashed lines in Figure 2.10. For each two allowed grid points the phase trajectory is calculated. For example, if t_i and t_j are two grid points and there is a V_y zero-crossing at t_k where $t_i < t_k < t_j$, then the horizontal velocity phase along the time interval $[t_i, t_j]$ should meet the following conditions: $\theta_x(t_i) = 2\pi n$, $\theta_x(t_k) = 2\pi(n + \frac{1}{4})$, $\theta_x(t_j) = 2\pi(n + \frac{1}{2})$. The phase trajectory is linearly interpolated between the induced grid points. Hence, the phase along the time interval $[t_i, t_j]$ is

$$\theta_x(t) = \begin{cases} 2\pi n + \frac{\pi}{2}\frac{t-t_i}{t_k-t_i} & t_i \le t < t_k \\ 2\pi n + \frac{\pi}{2} + \frac{\pi}{2}\frac{t-t_k}{t_j-t_k} & t_k \le t < t_j \end{cases} .$$

If there is no V_y zero-crossing between the grid points or there are two V_y zero-crossings, the V_x phase lag changes linearly between the zero-crossings. In those cases, the phase trajectory along the grid points is

$$\theta_x(t) = 2\pi n + \pi \frac{t - t_i}{t_j - t_i} .$$

Given the horizontal phase lag and assuming that the amplitude modulation is constant along one grid interval, the amplitudes that will preserve the horizontal progression are calculated. Denoting by L the horizontal progression, the approximated horizontal amplitude modulation along the time interval $[t_i, t_j]$ is

$$A'_{i,j} = \frac{L}{\int_{t_i}^{t_j} \sin(\theta_x(t))\, dt} ,$$

and the approximation error along this interval is

$$Error_{Approx}([t_i, t_j]) = \int_{t_i}^{t_j} \Big(V(t) - A'_{i,j} \sin(\theta_x(t))\Big)^2 dt .$$

Formally, let the set of possible grid points be $\bar{T} = \{t_1, t_2, \ldots, t_N\}$. We are looking for a subset $\tilde{T} = \{t_{i_1}, t_{i_2}, \ldots, t_{i_K}\} \subset \bar{T}$ such that all the pairs $t_{i_j}, t_{i_{j+1}}$ are allowed, with the minimal induced approximation error

$$\tilde{T} = \arg\min_{T' \subset T} \sum_{i_j \in T'} Error_{Approx}(\left[t_{i_j}, t_{i_{j+1}}\right]) .$$

For each grid point t_i a set of allowed previous grid points S_{t_i} is defined. The accumulated error at the grid point t_i can be calculated by *dynamic programming* using the following local minimization,

$$Error(t_j) = \min_{t_i \in S_{t_j}} \{Error(t_i) + Error_{Approx}([t_i, t_j])\} .$$

An illustration of the optimization process is depicted in Figure 2.10. The best phase trajectory is found by back-tracking from the best grid point of the last V_x zero-crossing. The result of this process is plotted in Figure 2.11. This process 'ties' the two oscillations and represents the horizontal oscillations in terms of the vertical oscillations. Therefore, only the vertical velocity

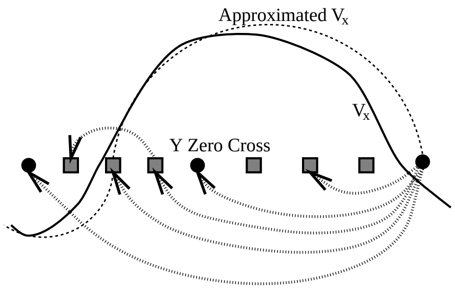


Figure 2.10: Phase lag trajectory optimization by *dynamic programming*. V_x is approximated by limiting its zero-crossings to a grid which is denoted in the figure by *bold circles* (V_y zero-crossings) and *grey rectangles*.

zero-crossings have to be located in the estimation process. This further reduces the number of bits needed to code the handwriting trajectory as indicated by stage 5 in Figure 2.9. Since the horizontal oscillations are less stable and more noisy, this scheme avoids many of the problems encountered when estimating the horizontal parameters directly.

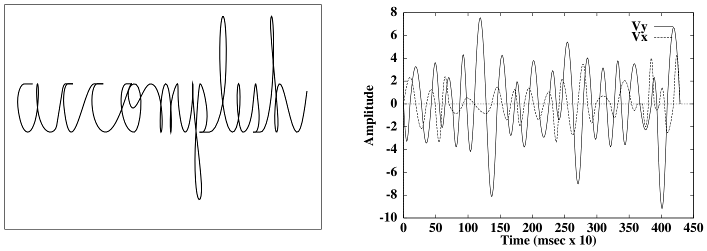


Figure 2.11: The horizontal and vertical velocities and the reconstructed handwriting after phase-lag regularization.

2.8 Angular Velocity Regularization

Until now the original angular velocities of the vertical oscillations were preserved. Hence, in order to reconstruct the velocities, the exact timing of the zero-crossings is kept. Our experiments reveal that all writers have their own typical angular velocity for the oscillations. These findings seem to be in contradiction to previous experiments, where it has been shown that a tendency exists for spatial characteristics to be more invariant than the temporal characteristics [129, 130] and to Hollerbach's claim that both the amplitudes and the angular velocity are scaled during the writing of tall letters like `l`. For the purpose of representing handwriting as the output of a discrete, controlled, oscillatory system, fixing the angular velocity does not incur difficulties, and the approximated velocities preserve the context as shown in Figure 2.12. Fixing the angular velocity can also be seen as a basic writing rhythm which may actually be supported by neurobiological evidence [16]. Since the horizontal oscillations are derived from the vertical oscillations by changing the phase lag, fixing the vertical angular velocity implies that the angular velocities of both the vertical and the horizontal oscillations are fixed. The angular velocity variations for each writer are small except in short intervals, where the writer hesitates or stops. The total halt intervals can be omitted or used for natural segmentation. The angular velocity is fixed to its typical value, and the time between two consecutive zero-crossings becomes constant.

The amplitudes are modified so that the total vertical and horizontal progressions are preserved. Since the horizontal and vertical progressions are quantized discrete values, the time scaling implies that the possible amplitudes are discrete as well. The time scaling can be viewed as a change in parameterization of the oscillation equations from time to phase, as denoted by (1.5). Assuming that the angular velocity, ω, is almost constant implies that $\frac{dt}{d\theta}$ is almost constant as well. The normalized dynamic equations which describe the handwriting become

$$\begin{cases} V_x(\theta) = A_x(\theta)\sin(\theta + \phi(\theta)) + 1 \\ V_y(\theta) = A_y(\theta)\sin(\theta) \end{cases} , \tag{1.9}$$

where $\phi(\theta) \in \{-90°, 0°, 90°\}$, $A_x(\theta) \in \{A_x^1, A_x^2, A_x^3\}$, and $A_y(\theta) \in \left\{A_y^1, A_y^2, A_y^3, A_y^4, A_y^5\right\}$. The result of this process is shown in Figure 2.12 where the original script and reconstructed script (after all stages including angular regularization) are plotted together with the synchronized vertical velocity. Note that the vertical velocity attains only a few discrete values at the maximal points of the oscillations. The number of bits needed to encode the writing curves is reduced after this final stage by a factor of about *100* compared with the original encoding of the writing curves (stage 6 in Figure 2.9).

The synthesized velocities are not 'natural' due to the switching scheme of the velocity parameters, which results in very large accelerations at the zero-crossings. Our simple synthesis scheme was used in order to verify our assumption that cursive handwriting can be represented as the output of a discrete, controlled system. Other synthesis schemes can be applied to yield more 'natural' velocities. For example the principle of minimal jerk by Hogan and Flash [64] can be used for synthesis.

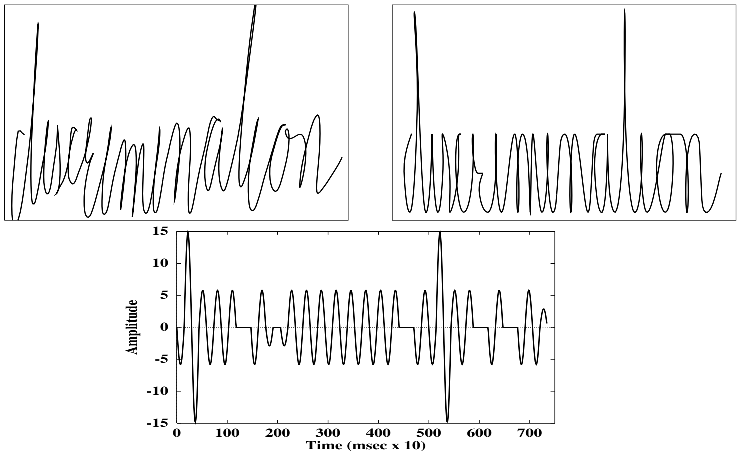


Figure 2.12: The original and the reconstructed handwriting after angular velocity regularization (*top figures*) and the final vertical velocity (*bottom figure*).

2.9 The Discrete Control Representation

So far, we have introduced a dynamic model which describes the velocities of a cursive writing process as a constrained modulation of underlying oscillatory processes. The imposed limitations on the dynamical control parameters result in a good approximation which is similar to the original. We then introduced a series of transformations which led to synchronous oscillations. As a result, a many-to-one mapping from the continuous velocities $V_x(t), V_y(t)$ to a discrete symbol set was generated. This set is composed of a cartesian product of the discrete vertical and horizontal amplitude modulation values and the phase-lag orientation between the horizontal and vertical velocities.

Tracking the number of bits that are needed to encode the velocities (Figure 2.9) reveals that the discretization and regularization processes gradually reduce the bit rate. This indicates that our discrete controlled system representation is well suited for compression and recognition applications. The transformation closes part of the gap between different writing styles and different writers. Keeping track of the transformations themselves can be used for writer identification. Here we introduce one possible discrete representation of the resulting discrete control. Our representation does not correspond directly to the original dynamic parameters but rather involves a one-to-one transformation of them.

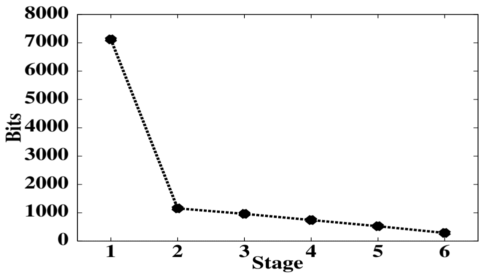


Figure 2.13: The number of bits needed to encode cursive handwriting along the various stages.

Further, we describe the two discrete control processes as the output of two synchronized stochastic processes. The output can be written in two rows. The first row describes the appropriate vertical level (which can be one of 5 values) each time $V_y(t) = 0$. Whenever there is a vertical velocity zero-crossing, the corresponding automaton outputs a symbol which is the index of the level obtained at the zero-crossing. Similarly, the second automaton outputs a symbol when a horizontal velocity zero-crossing occurs. This symbol corresponds to the horizontal amplitude modulation for the next interval. Special care is taken when tracking the discrete control of the horizontal oscillations, since the phase is not explicit but changes its state implicitly. Yet if the initial horizontal oscillation phase is known, then the total phase trajectory can be reconstructed from this information. The first output symbol of the horizontal automaton is the initial phase denoted by $\pm$. Since the oscillation processes are synchronized by the angular velocity regularization, we only need to record the order of the automata output. When the two automata output symbols at the same time, it means that the oscillation phases became coherent; otherwise, there is a $\pm 90°$ phase lag. The angular velocity ratio limitation implies that each of the automata can output at most two consecutive symbols, while the other automaton is silent. The following is an example for such a representation for the same word ('toccata') written twice.

```
20502004004004020404044402040400402040444020402050240040040204022
43040333330210501050341050332022050303104020402050333021040205 03
```

```
205020040400402040404440204440204004402040205020044040040204022
330503333321040105033105033105033331040205020443330202104020503
```

Note that the sequences of motor control commands are similar, and that simple rules may be found to match the two sequences. In fact, in this example, if we omit the horizontal (lower) output and squeeze the gaps for the vertical (upper) one, then the upper sequences for the two words are identical. This implies that much of the information is embedded in the vertical oscillations.

Finally, in order to encode short strokes such as the dots above the letters `i` and `j`, bars for `t` and crosses for `x`, a third encoding row is defined. Let the symbol `1` be the code for crosses, `2` for bars, and `3` for dots. A value `0` in this row represents no activity. Since the purpose of such short strokes is to add additional information that disambiguates letters (e.g., a `t` and a `l` can be distinguished mostly due to the bar drawn over the letter `t`), the third row is *spatially* aligned to the first two rows. That is, each symbol in this row is encoded in correspondence to the *location* of its appearance and not the *time* of it appearance. An example of the result of a full encoding, together with its synthesized cursive handwriting, is depicted in Figure 2.14. The complete code in this example is,

```
204020402420510300204020403040340240020420400244044020402050204024044020403424002004
420303320502413044010401040204024033204203333423310402040204020403310401050250333533
003000000000000000000000000000000000000000000000000000000020003000000000000000000000
```

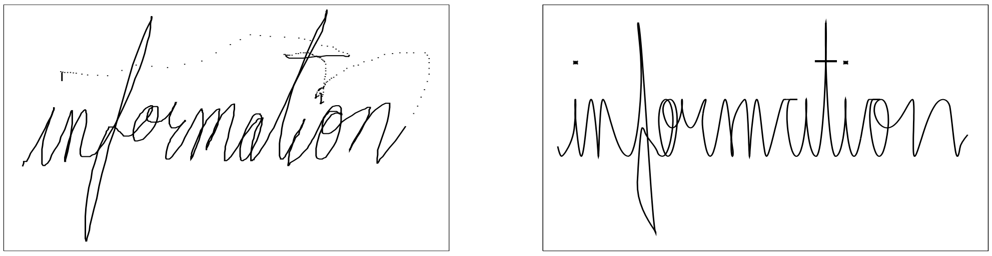


Figure 2.14: Example of the full dynamical encoding of cursive handwriting. The original pen trajectory is depicted on the left. The trajectory is composed of the pen movements on the paper as well as an approximation of the projection of pen movements onto the writing plane when the pen does not touch the writing plane. The reconstructed handwriting is plotted on the right. The encoding is composed of the temporal motor control commands for the continuous on-paper pen movements and spatial encoding of short strokes such as dots above `i`'s and bars over `t`'s.

2.10 Discussion

Although the idea that the pen movements in the production of cursive script are the result of a simple 'motor program' is quite old, revealing this 'motor code' is a difficult inverse-dynamic problem. In this chapter, we presented a robust scheme which transforms the continuous pen movements into discrete *motor control symbols*. These symbols can be interpreted as a possible high level coding of the motor system. The relationship between this representation and the actual cognitive representation of handwriting remains open, though there is some psychophysical experimental evidence linking the recognition time to the writing time for handwriting [44]. The *discrete motor control* representation largely reduces the variability in different writing styles and writer specific effects. We later show, in Chapter 5, how to use the discrete motor control representation for cursive scripts recognition. Since different writing styles are transformed to the same representation, the transformation itself can be used for *text independent* writer identification and verification tasks.

Chapter 3

Short But Useful

3.1 Introduction

An important class of problems that arise in machine learning applications is that of modeling classes of short sequences, such as the motor control commands introduced in Chapter 2, with their possibly complex variations. As we will see later, such sequence models are essential and useful, for instance, in handwriting and speech recognition, natural language processing, and biochemical sequence analysis. Our interest here is specifically in modeling short sequences, that correspond to objects such as "words" in a language or short protein sequences and not in the asymptotic statistical properties of very long sequences.

The common approaches to the modeling and recognition of such sequences are string matching algorithms (e.g., Dynamic Time Warping [118]) on the one hand, and Hidden Markov Models (in particular 'left-to-right' HMMs) on the other hand [104, 106]. The string matching approach usually assumes the existence of a sequence prototype (reference template) together with a local noise model, from which the probabilities of deletions, insertions, and substitutions, can be deduced. The main weakness of this approach is that it does not treat any context dependent, or nonlocal variations, without making the noise model much more complex. This property is unrealistic for many of the above applications due to phenomena such as "coarticulation" in speech and handwriting, or long range chemical interactions (due to geometric effects) in biochemistry. Some of the weaknesses of HMMs were discussed in Chapter 1. Another drawback of HMMs is that the current HMM training algorithms are neither online nor adaptive in the model's topology. The weak aspects of the string machining techniques and of hidden Markov models motivate our modeling technique presented in this chapter.

The alternative we consider here is using *Acyclic Probabilistic Finite Automata* (APFA) for modeling distributions on short sequences such as those mentioned above. These automata seem to capture well the context dependent variability of such sequences. We present and analyze an efficient and easily implementable learning algorithm for a subclass of APFAs that have a certain *distinguishability* property which is defined subsequently. Our result should be contrasted with the intractability result for learning PFAs described by Kearns *et. al.* [72]. They show that PFAs are not efficiently learnable under the widely acceptable assumption that there is no efficient algorithm for learning noisy parity functions in the PAC model. Furthermore, the subclass of PFAs which they show are hard to learn, are (width two) APFAs in which the distance in the L_1 norm (and hence also the KL-divergence) between the distributions generated starting from every pair of states is large.

More formally, we present an algorithm for efficiently learning distributions on strings generated by a subclass of APFAs which have the following property. For every pair of states in an automaton

M belonging to this class, the distance in the L_∞ norm between the distributions generated starting from these two states is non-negligible. Namely, this distance is an inverse polynomial in the size of M. We call the minimal distance between the distributions generated by the states a *distinguishability parameter* and denote it by μ. Our algorithm runs in time polynomial in the size of the target PFA M and in μ. The learning algorithm has also an online mode whose performance is comparable to that of the *batch* mode.

One of the key techniques applied in this chapter is that of using some form of *signatures* of states in order to distinguish between the states of the target automaton. This technique was presented in the pioneering work of Trakhtenbrot and Brazdin' [132] in the context of learning deterministic finite automata (DFAs). The same idea was later applied by Freund *et. al.* [45] in their work on learning typical DFAs. In the same work they proposed to apply the notion of *statistical signatures* to learning typical PFAs.

The outline of our learning algorithm is roughly the following. In the course of the algorithm we maintain a sequence of directed edge-labeled acyclic graphs. The first graph in this sequence, named the *sample tree*, is constructed based on the a sample generated by the target APFA, while the last graph in the sequence is the underlying graph of our hypothesis APFA. Each graph in this sequence is transformed into the next graph by a *folding operation* in which a pair of nodes that have passed a certain *similarity test* are merged into a single node (and so are the pairs of their respective successors).

The structure of this chapter is as follows. We end this section with a short overview on related algorithms and applications. In Sections 3.2 and 3.3 we give several definitions related to APFAs, and define our learning model. In Section 3.4 we present our learning algorithm. In Section 3.5 we state and prove our main theorem concerning the correctness of the learning algorithm. In Section 3.6, we conclude the analysis with an online version of the learning algorithm. In the second part, which includes Sections 3.7 and 3.8, we describe and evaluate two applications of the model. First, we demonstrate how APFAs can be used to build multiple-pronunciation models for spoken words. We also show and discuss the usage of APFAs to the identification of noun phrases in natural English text.

A similar technique of merging states was also applied by Carrasco and Oncina [20], and by Stolcke and Omohundro [127]. Carrasco and Oncina give an algorithm which identifies *in the limit* distributions generated by PFAs. Stolcke and Omohundro describe a learning algorithm for HMMs which merges states based on a Bayesian approach, and apply their algorithm to build pronunciation models for spoken words. Examples and reviews of practical models and algorithms for multiple-pronunciation can be found in [24, 109], and on syntactic structure acquisition in [18, 53].

3.2 Preliminaries

We start with a formal definition of a Probabilistic Finite Automaton. The definition we use is slightly nonstandard in the sense that we assume a final symbol and a final state. A *Probabilistic Finite Automaton (PFA)* M is a 7-tuple $(Q, q_0, q_f, \Sigma, \zeta, \tau, \gamma)$ where:

- Q is a finite set of *states*;
- $q_0 \in Q$ is the *starting state*;

- $q_f \notin Q$ is the *final state*;
- Σ is a finite *alphabet*;
- $\zeta \notin \Sigma$ is the *final symbol*;
- $\tau : Q \times \Sigma \bigcup \{\zeta\} \rightarrow Q \bigcup \{q_f\}$ is the *transition function*;
- $\gamma : Q \times \Sigma \bigcup \{\zeta\} \rightarrow [0, 1]$ is the *next symbol probability function*.

The function γ must satisfy the following requirement: for every $q \in Q$, $\sum_{\sigma \in \Sigma} \gamma(q, \sigma) = 1$. We allow the transition function τ to be undefined only on states q and symbols σ, for which $\gamma(q, \sigma) = 0$. We require that for every $q \in Q$ such that $\gamma(q, \zeta) > 0$, $\tau(q, \zeta) = q_f$. We also require that q_f can be reached (i.e., with nonzero probability) from *every* state q which can be reached from the starting state, q_0. τ can be extended to be defined on $Q \times \Sigma^*$ in the following recursive manner: $\tau(q, s_1 s_2 \ldots s_l) = \tau(\tau(q, s_1 \ldots s_{l-1}), s_l)$, and $\tau(q, \mathsf{e}) = q$ where e is the empty string.

A PFA M generates strings of finite length ending with the symbol ζ, in the following sequential manner. Starting from q_0, until q_f is reached, if q_i is the current state, then the next symbol is chosen (probabilistically) according to $\gamma(q_i, \cdot)$. If $\sigma \in \Sigma$ is the symbol generated, then the next state, q_{i+1}, is $\tau(q_i, \sigma)$. Thus, the probability M generates a string $s = s_1 \ldots s_{l-1} s_l$, where $s_l = \zeta$, denoted by $P^M(s)$ is

$$P^M(s) \stackrel{\text{def}}{=} \prod_{i=0}^{l-1} \gamma(q_i, s_{i+1}) \ . \tag{3.1}$$

This definition implies that $P^M(\cdot)$ is in fact a probability distributions over strings ending with the symbol ζ, i.e.,

$$\sum_{s \in \Sigma^* \zeta} P^M(s) = 1 \ .$$

For a string $s = s_1 \ldots s_l$ where $s_l \neq \zeta$ we choose to use the same notation $P^M(s)$ to denote the probability that s is a *prefix* of some generated string $s' = ss''\zeta$. Namely,

$$P^M(s) = \prod_{i=0}^{l-1} \gamma(q_i, s_{i+1}) \ .$$

Given a state q in Q, and a string $s = s_1 \ldots s_l$ (that does not necessarily end with ζ), let $P^M_q(s)$ denote the probability that s is (a prefix of a string) generated starting from q. More formally

$$P^M_q(s) \stackrel{\text{def}}{=} \prod_{i=0}^{l-1} \gamma(\tau(s_1, \ldots, s_i), s_{i+1}) \ .$$

The following definition is central to this chapter.

DEFINITION 3.2.1 *For $0 \leq \mu \leq 1$, we say that two states, q_1 and q_2 in Q are* **μ-distinguishable**, *if there exists a string s for which $|P^M_{q_1}(s) - P^M_{q_2}(s)| \geq \mu$. We say that a PFA M is* **μ-distinguishable**, *if every pair of states in M are* **μ-distinguishable**.[1]

[1] As noted in the analysis of our algorithm in Section 3.5, we can use a slightly weaker version of the above definition, in which we require that only pairs of states with non-negligible weight be distinguishable.

We shall restrict our attention to a subclass of PFAs which have the following property: the underlying graph of every PFA in this subclass is *acyclic*. The *depth* of an acyclic PFA is defined to be the length of the longest path from q_0 to q_f. In particular, we consider *leveled* acyclic PFAs. In such a PFA, each state belongs to a single level d, where the starting state, q_0 is the only state in level 0, and the final state, q_f, is the only state in level D, where D is the depth of the PFA. All transitions from a state in level d must be to states in level $d+1$, except for transitions labeled by the final symbol, ζ, which need not be restricted in this way. We denote the set of states belonging to level d, by Q_d. The following claim can easily be verified.

Lemma 3.2.1 *For every acyclic PFA M having n states and depth D, there exists an equivalent leveled acyclic PFA, $\tilde{M}$, with at most $n(D-1)$ states.*

Proof: We define $\bar{M} = (\bar{Q}, \Sigma, \zeta, \bar{\tau}, \bar{\gamma}, \bar{q_0}, \bar{q_f})$ as follows. For every state $q \in Q - \{q_f\}$, and for each level d such that there exists a string s of length d for which $\tau(q_0, s) = q$, we have a state $\bar{q}_d \in \bar{Q}$. For $q = q_0$, $(\bar{q_0})_0$ is simply the starting state of $\bar{M}$, $\bar{q}_0$. For every level d and for every $\sigma \in \Sigma \bigcup \{\zeta\}$, $\bar{\gamma}(\bar{q}_d, \sigma) = \gamma(q, \sigma)$. For $\sigma \in \Sigma$, $\bar{\tau}(\bar{q}_d, \sigma) = q'_{d+1}$, where $q' = \tau(q, \sigma)$, and $\bar{\tau}(\bar{q}_d, \zeta) = \bar{q}_f$. Every state is copied at most $D-1$ times hence the total number of states in $\bar{M}$ is at most $n(D-1)$. ■

3.3 The Learning Model

In this section we describe our learning model which is a slightly modified version of the definition of *ϵ-good* hypothesis introduced in Chapter 1.

DEFINITION 3.3.1 *Let M be the target PFA and let $\widehat{M}$ be a hypothesis PFA. Let P^M and $P^{\widehat{M}}$ be the two probability distributions they generate respectively. We say that $\widehat{M}$ is an* **ϵ-good** *hypothesis with respect to M, for $\epsilon \geq 0$, if*

$$\mathcal{D}_{KL}[P^M || P^{\widehat{M}}] \leq \epsilon \quad ,$$

where $\mathcal{D}_{KL}[P^M || P^{\widehat{M}}]$ is the **Kullback Liebler (KL) divergence** *(also known as the cross-entropy) between the distributions and is defined as follows:*

$$\mathcal{D}_{KL}[P^M || P^{\widehat{M}}] \stackrel{\text{def}}{=} \sum_{s \in \Sigma^* \zeta} P^M(s) \log \frac{P^M(s)}{P^{\widehat{M}}(s)} \quad .$$

Our learning algorithm for PFAs is given a *confidence* parameter $0 < \delta \leq 1$, and an *approximation* parameter $\epsilon > 0$. The algorithm is also given an upper bound n on the number of states in M, and a *distinguishability* parameter $0 < \mu \leq 1$, indicating that the target automaton is μ-distinguishable.[2] The algorithm has access to strings generated by the target PFA, and we ask that it output with probability at least $1-\delta$ an ϵ-good hypothesis with respect to the target PFA. We also require that the learning algorithm be *efficient*, i.e., that it runs in time polynomial in $\frac{1}{\epsilon}$, $\log \frac{1}{\delta}$, $|\Sigma|$, and in the bounds on $\frac{1}{\mu}$ and n.

[2]These last two assumption can be removed by *searching* for an upper bound on n and a lower bound on μ. This search is performed by testing the hypotheses the algorithm outputs when it runs with growing values of n, and decreasing values of μ. Such a test can be done by comparing the log-likelihood of the hypotheses on additional test data.

3.4 The Learning Algorithm

In this section we describe our algorithm for learning acyclic PFAs. An *online* version of this algorithm is described in Section 3.6.

Let S be a given multiset of sample strings generated by the target PFA M. In the course of the algorithm we maintain a series of directed leveled acyclic graphs $G_0, G_1, \ldots, G_{N+1}$, where the final graph, G_{N+1}, is the underlying graph of the hypothesis automaton. In each of these graphs, there is one node, v_0, which we refer to as the *starting node*. Every directed edge in a graph G_i is labeled by a symbol $\sigma \in \Sigma \bigcup \{\zeta\}$. There may be more than one directed edge between a pair of nodes, but for every node, there is at most one outgoing edge labeled by each symbol. If there is an edge labeled by σ connecting a node v to a node u, then we denote it by $v \xrightarrow{\sigma} u$. If there is a labeled (directed) path from v to u corresponding to a string s, then we denote it similarly by $v \overset{s}{\Rightarrow} u$.

Each node v is *virtually* associated with a *multiset* of strings $S(v) \subseteq S$. These are the strings in the sample which correspond to the (directed) paths in the graph that *pass* through v when starting from v_0, i.e.,

$$S(v) \stackrel{\text{def}}{=} \{s : s = s's'' \in S,\ v_0 \overset{s'}{\Rightarrow} v\}_{multi} .$$

We define an additional, related, multiset, $S_{gen}(v)$, that includes the substrings in the sample which can be seen as *generated* from v. Namely,

$$S_{gen}(v) \stackrel{\text{def}}{=} \{s'' : \exists s' \text{ s.t. } s's'' \in S \text{ and } v_0 \overset{s'}{\Rightarrow} v\}_{multi} .$$

For each node v, and each symbol σ, we associate a count, $m_v(\sigma)$, with v's outgoing edge labeled by σ. If v does not have any outgoing edges labeled by σ, then we define $m_v(\sigma)$ to be 0. We denote $\sum_\sigma m_v(\sigma)$ by m_v, and it always holds by construction that $m_v = |S(v)|$ $(= |S_{gen}(v)|)$, and $m_v(\sigma)$ equals the number of strings in $S_{gen}(v)$ whose first symbol is σ.

The initial graph G_0 is the *sample tree*, T_S. Each node in T_S is associated with a *single* string which is a prefix of a string in S. The root of T_S, v_0, corresponds to the empty string, and every other node, v, is associated with the prefix corresponding to the labeled path from v_0 to v.

We now describe our learning algorithm. For a more detailed description see the pseudo-code that follows. We would like to stress that the multisets of strings, $S(v)$, are maintained only virtually, thus the data structure used along the run of the algorithm is only the current graph G_i, together with the counts on the edges. For $i = 0, \ldots, N-1$, we associate with G_i a level, $d(i)$, where $d(0) = 1$, and $d(i) \geq d(i-1)$. This is the level in G_i we plan to operate on in the transformation from G_i to G_{i+1}. We transform G_i into G_{i+1} by what we call a *folding* operation. In this operation we choose a pair of nodes u and v, both belonging to $d(i)$, which have the following properties: for a predefined threshold m_0 (that is set in the analysis of the algorithm) both $m_u \geq m_0$ and $m_v \geq m_0$, and the nodes are *similar* in a sense defined subsequently. We then merge u and v, and all pairs of nodes they reach, respectively. If u and v are merged into a new node, w, then for every σ, we let $m_w(\sigma) = m_u(\sigma) + m_v(\sigma)$. The virtual multiset of strings corresponding to w, $S(w)$, is simply the union of $S(u)$ with $S(v)$. An illustration of the folding operation is depicted in Figure 3.1.

Let G_N be the last graph in this series for which there does not exist such a pair of nodes. We transform G_N into G_{N+1}, by performing the following operations. First, we merge all leaves in

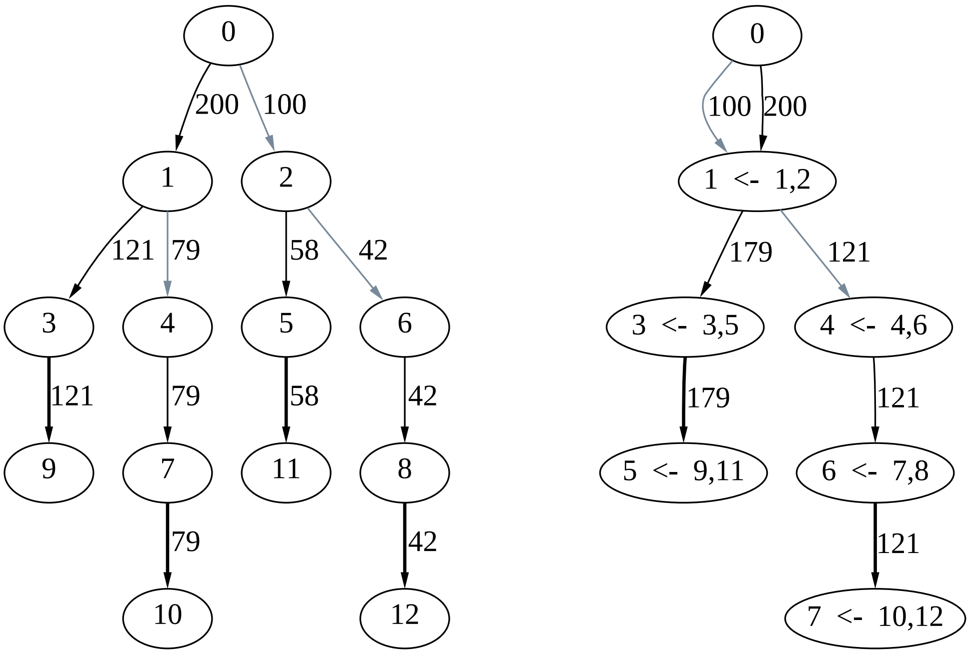


Figure 3.1: An illustration of the folding operation. The graph on the right is constructed from the graph on the left by merging the nodes v_1 and v_2. The different edges represent different output symbols: gray is 0, black is 1 and bold black edge is ζ.

G_N into a single node v_f. Next, for each level d in G_N, we merge all nodes u in level d for which $m_u < m_0$. Let this node be denoted by $small(d)$. Lastly, for each node u, and for each symbol σ such that $m_u(\sigma) = 0$, if $\sigma = \zeta$, then we add an edge labeled by ζ from u to v_f, and if $\sigma \in \Sigma$, then we add an edge labeled by σ from u to $small(d+1)$ where d is the level u belongs to.

Finally, we define our hypothesis PFA $\widehat{M}$ based on G_{N+1}. We let G_{N+1} be the underlying graph of $\widehat{M}$, where v_0 corresponds to q_0, and v_f corresponds to q_f. For every state q in level d that corresponds to a node u, and for every symbol $\sigma \in \Sigma \bigcup \{\zeta\}$, we define

$$\gamma(q,\sigma) = (m_u(\sigma)/m_u)(1 - (|\Sigma| + 1)\gamma_{min}) + \gamma_{min} \ , \tag{3.2}$$

where γ_{min} is set in the analysis of the algorithm.

It remains to define the notion of *similar* nodes used in the algorithm. Roughly speaking, two nodes are considered similar if the statistics according to the sample, of the strings which can be seen as generated from these nodes, is similar. More formally, for a given node v and a string s, let $m_v(s) \stackrel{\text{def}}{=} |\{t : t \in S_{gen}(v),\ t = st'\}_{multi}|$. We say that a given pair of nodes u and v, are *similar* if for *every* string s,

$$|m_v(s)/m_v - m_u(s)/m_u| \le \mu/2 \ .$$

As noted before, the algorithm does not maintain the multisets of strings $S_{gen}(v)$. However, the values $m_v(s)/m_v$ and $m_u(s)/m_u$ can be computed efficiently using the counts on the edges of the graphs, as described in the function **Similar** presented below.

For sake of simplicity of the pseudo-code below, we associate with each node in a graph G_i, a number in $\{1, \ldots, |G_i|\}$. The algorithm proceeds level by level. At each level, it searches for pairs of nodes, belonging to that same level, which can be folded. It does so by calling the function **Similar** on every pair of nodes u and v, whose counts, m_u and m_v, are above the threshold m_0. If the function returns *similar*, then the algorithm merges u and v using the routine **Fold**. Each call to **Fold** creates a new (smaller) graph. When level D is reached, the last graph, G_N, is transformed into G_{N+1} as described below in the routine **AddSlack**. The final graph, G_{N+1} is then transformed into a PFA while smoothing the transition probabilities (Procedure **GraphToPFA**).

Algorithm Learn-Acyclic-PFA

1. Initialize: $i := 0$, $G_0 := T_S$, $d(0) := 1$, $D :=$ depth of T_S ;
2. While $d(i) < D$ do:
   (a) Look for nodes j and j' from level $d(i)$ in G_i which have the following properties:
      i. $m_j \geq m_0$ and $m_{j'} \geq m_0$;
      ii. **Similar**$(j, 1, j', 1) =$ *similar* ;
   (b) If such a pair is not found let $d(i) := d(i) + 1$; /* return to while statement */
   (c) Else: /* Such a pair is found: transform G_i into G_{i+1} */
      i. $G_{i+1} := G_i$;
      ii. Call **Fold**(j, j', G_{i+1}) ;
      iii. Renumber the states of G_{i+1} to be consecutive numbers in the range $1, \ldots, |G_{i+1}|$;
      iv. $d(i+1) := d(i)$, $i := i + 1$;
3. Set $N := i$; Call **AddSlack**(G_N,G_{N+1},D) ;
4. Call **GraphToPFA**(G_{N+1},$\widehat{M}$) .

Function Similar(u, p_u, v, p_v)

1. If $|p_u - p_v| \geq \mu/2$ Return *non-similar* ;
2. Else-If $p_u < \mu/2$ and $p_v < \mu/2$ Return *similar* ;
3. Else $\forall \sigma \in \Sigma \bigcup \zeta$ do:
   (a) $p'_u = p_u m_u(\sigma)/m_u$; $p'_v = p_v m_v(\sigma)/m_v$;
   (b) If $m_u(\sigma) = 0$ $u' :=$ *undefined* else $u' := \tau(u, \sigma)$;
   (c) If $m_v(\sigma) = 0$ $v' :=$ *undefined* else $v' := \tau(v, \sigma)$;
   (d) If Similar$(u', p'_u, v', p'_v) ==$ *non-similar*
   Return *non-similar* ;
4. Return *similar*. /* Recursive calls ended and found similar */

Subroutine Fold(j, j', G)

1. For all the nodes k in G and $\forall \sigma \in \Sigma$ such that $k \xrightarrow{\sigma} j'$, change the corresponding edge to end at j, namely set $k \xrightarrow{\sigma} j$;
2. $\forall \sigma \in \Sigma \bigcup \zeta$:
   (a) If $m_j(\sigma) = 0$ and $m'_j(\sigma) > 0$, let k be such that $j' \xrightarrow{\sigma} k$; set $j \xrightarrow{\sigma} k$;
   (b) If $m_j(\sigma) > 0$ and $m_{j'}(\sigma) > 0$, let k and k' be the indices of the states such that $j \xrightarrow{\sigma} k$, $j' \xrightarrow{\sigma} k'$;
   Recursively fold k, k': call **Fold**$(k, k'$,**G**$)$;
   (c) $m_j(\sigma) := m_{j'}(\sigma) + m_j(\sigma)$;
3. $G := G - \{j'\}$.

Subroutine AddSlack(G, G', D)

1. Initialize: $G' := G$;
2. Merge all nodes in G' which have no outgoing edges, into v_f (which is defined to belong to level D);
3. For $d := 1, \ldots, D-1$ do:
   Merge all nodes j in level d for which $m_j < m_0$ into $small(d)$;
4. For $d := 0, \ldots, D-1$ and for every j in level d do:
   (a) $\forall \sigma \in \Sigma$: If $m_j(\sigma) = 0$ then add an edge labeled σ from j to $small(d)$;
   (b) If $m_j(\zeta) = 0$ then add an edge labeled σ from j to v_f (set $j \xrightarrow{\zeta} v_f$);

Subroutine GraphToPFA$(G, \widehat{M})$

1. Let G be the underlying graph of $\widehat{M}$;
2. Let $\hat{q}_0$ be the state corresponding to v_0, and let $\hat{q}_f$ be the state corresponding to v_f;
3. For every state $\hat{q}$ in $\widehat{M}$ and for every $\sigma \in \Sigma \bigcup \{\zeta\}$:

$$\hat{\gamma}(\hat{q}, \sigma) := (m_v(\sigma)/m_v)(1 - (|\Sigma| + 1)\gamma_{min}) + \gamma_{min} \quad ,$$

   where v is the node corresponding to $\hat{q}$ in G.

3.5 Analysis of the Learning Algorithm

In this section we state and prove our main theorem regarding the correctness and efficiency of the learning algorithm Learn-Acyclic-PFA, described in Section 3.4.

Theorem 1 *For every given distinguishability parameter $0 < \mu \leq 1$, for every μ-distinguishable target acyclic PFA M, and for every given confidence parameter $0 < \delta \leq 1$, and approximation parameter $\epsilon > 0$, Algorithm* Learn-Acyclic-PFA *outputs a hypothesis PFA, $\widehat{M}$, such that with probability at least $1-\delta$, $\widehat{M}$ is an ϵ-good hypothesis with respect to M. The running time of the algorithm is polynomial in $\frac{1}{\epsilon}$, $\log \frac{1}{\delta}$, $\frac{1}{\mu}$, n, D, and $|\Sigma|$.*

We would like to note that for a given approximation parameter ϵ, we may slightly weaken the requirement that M be μ-distinguishable. It suffices that we require that every pair of states q_1 and q_2 in M such that both $P^M(q_1)$ and $P^M(q_2)$ are greater than some ϵ' (which is a function of ϵ, μ and n), q_1 and q_2 be μ-distinguishable. For sake of simplicity, we give our analysis under the slightly stronger assumption.

Without loss of generality, (based on Lemma 3.2.1) we may assume that M is a leveled acyclic PFA with at most n state in each of its D levels. We add the following notations.

- For a state $q \in Q_d$,
  - $W(q)$ denotes the set of *all* strings in Σ^d which *reach* q; $P^M(q) \stackrel{\text{def}}{=} \sum_{s \in W(q)} P^M(s)$.
  - m_q denotes the number of strings *in the sample* (including repetitions) which *pass through* q, and for a string s, $m_q(s)$ denotes the number of strings in the sample which pass through q and continue with s. More formally,

$$m_q(s) = \left|\{t \,:\, t \in S, t = t_1 s t_2, \text{ where } \tau(q_0, t_1) = q\}_{multi}\right| .$$

- For a state $\hat{q} \in \hat{Q}_d$, $W(\hat{q})$ $m_{\hat{q}}$, $m_{\hat{q}}(s)$, and $P^{\widehat{M}}(\hat{q})$ are defined similarly. For a node v in a graph G_i constructed by the learning algorithm, $W(v)$ is defined analogously. (Note that m_v and $m_v(s)$ were already defined in Section 3.4).
- For a state $q \in Q_d$ and a node v in G_i, we say that v *corresponds* to q, if $W(v) \subseteq W(q)$.

In order to prove Theorem 1, we first need to define the notion of a *good* sample with respect to a given target (leveled) PFA. We prove that with high probability a sample generated by the target PFA is good. We then show that if a sample is good then our algorithm constructs a hypothesis PFA which has the properties stated in the theorem.

A *Good* Sample

In order to define when a sample is good in the sense that it has the statistical properties required by our algorithm, we introduce a class of PFAs $\mathcal{M}$, which is defined below. The reason for introducing this class is roughly the following. The heart of our algorithm is in the folding operation, and the similarity test that precedes it. We want to show that, on one hand, we do not fold pairs of nodes which correspond to two different states, and on the other hand, we fold most pairs of nodes that do correspond to the same state. By "most" we essentially mean that in our final hypothesis, the weight of the *small* states (which correspond to the unfolded nodes whose counts are small) is in fact small.

Whenever we perform the *similarity* test between two nodes u and v, we compare the statistical properties of the corresponding multisets of strings $S_{gen}(u)$ and $S_{gen}(v)$, which "originate" from the two nodes, respectively. Thus, we would like to ensure that if both sets are of substantial size, then each will be in some sense *typical* to the state it was generated from (assuming there exists one such single state for each node). Namely, we ask that the relative weight of any prefix of a string in each of the sets will not deviate much from the probability it was generated starting from the corresponding state.

For a given level d, let G_{i_d} be the first graph in which we start folding nodes in level d. Consider some specific state q in level d of the target automaton. Let $S(q) \subseteq S$ be the subset of sample strings which pass through q. Let $v_1, \ldots, v_k$ be the nodes in G which correspond to q, in the sense that each string in $S(q)$ passes through one of the v_i's. Hence, these nodes induce a partition of $S(q)$ into the sets $S(v_1), \ldots, S(v_k)$. It is clear that if $S(q)$ is large enough, then, since the strings were generated independently, we can apply Chernoff bounds (see Appendix C) to get that with high probability $S(q)$ is typical to q. But we want to know that *each* of the $S(v_i)$'s is typical to q. It is clearly not true that *every* partition of $S(q)$ preserves the statistical properties of q. However, the graphs constructed by the algorithm do not induce arbitrary partitions, and we are able to characterize the possible partitions in terms of the automata in $\mathcal{M}$. This characterization also helps us bound the weight of the small states in our hypothesis.

Given a target PFA M let $\mathcal{M}$ be the set of PFAs $\{M' = (Q', q'_0, \{q'_f\}, \Sigma, \tau', \gamma', \zeta)\}$ which satisfy the following conditions:

1. For each state q in M there exist several *copies* of q in M', each uniquely labeled. q'_0 is the only copy of q_0, and we allow there to be a set of final states $\{q'_f\}$, all copies of q_f. If q' is a copy of q then for every $\sigma \in \Sigma \bigcup \{\zeta\}$,

   (a) $\gamma'(q', \sigma) = \gamma(q, \sigma)$;

   (b) if $\tau(q, \sigma) = t$, then $\tau'(q', \sigma) = t'$, where t' is a copy of t.

   Note that the above restrictions on γ' and τ' ensure that $M' \equiv M$, i.e.,

   $$\forall s \in \Sigma^* \zeta \, , \; P^{M'}(s) = P^{M}(s) \ .$$

2. A copy of a state q may be either *major* or *minor*. A major copy is either *dominant* or *non-dominant*. Minor copies are always non-dominant.

3. For each state q, and for every symbol σ and state r such that $\tau(r, \sigma) = q$, there exists a unique major copy of q labeled by (q, r, σ). There are no other major copies of q. Each minor copy of q is labeled by (q, r', σ), where r' is a non-dominant (either major or minor) copy of r (and as before $\tau(r, \sigma) = q$). A state may have no minor copies, and its major copies may be all dominant or all non-dominant.

4. For each dominant major copy q' of q and for every $\sigma \in \Sigma \bigcup \{\zeta\}$, if $\tau(q,\sigma) = t$, then $\tau'(q',\sigma) = (t,q,\sigma)$. Thus, for each symbol σ, *all* σ transitions from the dominant major copies of q are to the *same* major copy of t. The starting state q_0' is always dominant.

5. For each non-dominant (either major or minor) copy q' of q, and for every symbol σ, if $\tau(q,\sigma) = t$ then $\tau'(q',\sigma) = (t,q',\sigma)$, where, as defined in item (2) above, (t,q',σ) is a minor copy of t. Thus, each non-dominant major copy of q is the root of a $|\Sigma|$-*ary* tree, and all it's descendants are (non-dominant) minor copies.

An illustrative example of the types of copies of states is depicted in Figure 3.2.

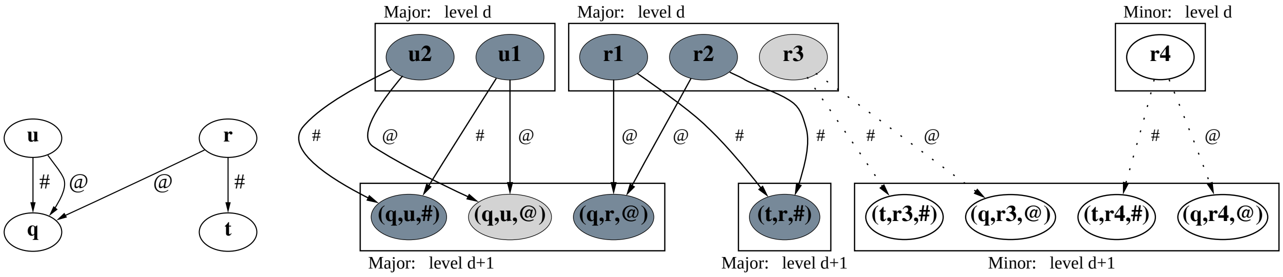


Figure 3.2: Left: Part of the original automaton, M, that corresponds to the copies on the right part of the figure. Right: The different types of copies of M's states: copies of a state are of two types *major* and *minor*. A subset of the major copies of every state is chosen to be *dominant* (dark-gray nodes). The major copies of a state in the next level are the next states of the dominant states in the current level.

By the definition above, each PFA in $\mathcal{M}$ is fully characterized by the choices of the sets of dominant copies among the major copies of each state. Since the number of major copies of a state q is exactly equal to the number of transitions going into q in M, and is thus bounded by $n|\Sigma|$, there are at most $2^{n|\Sigma|}$ such possible choices for every state. There are at most n states in each level, and hence the size of $\mathcal{M}$ is bounded by $((2^{|\Sigma|n})^n)^D = 2^{|\Sigma|n^2 D}$. As we show in Lemma 3.5.3, if the sample is good, then there exists a correspondence between some PFA in $\mathcal{M}$ and the graphs our algorithm constructs. We use this correspondence to prove Theorem 1.

Definition 3.5.1 *A sample S of size m is (ϵ_0, ϵ_1)-*good *with respect to M if for every $M' \in \mathcal{M}$ and for every state $q' \in Q'$:*

1. *$P^{M'}(q') \geq 2\epsilon_0$, then $m_{q'} \geq m_0$, where*

$$m_0 = \frac{|\Sigma|\, n^2\, D^2 \,+\, 2D \ln\left(8(|\Sigma|+1)\right) \,+\, \ln \frac{1}{\delta}}{\epsilon_1^2} \;;$$

2. *If $m_{q'} \geq m_0$, then for every string s,*

$$|m_{q',s}/m_{q'} - P_{q'}^{M'}(s)| \leq \epsilon_1 \;;$$

Lemma 3.5.1 *With probability at least $1-\delta$, a sample of size*

$$m \geq \max\left(\frac{|\Sigma|\, n^2\, D \,+\, \ln \frac{2D}{\epsilon_0 \delta}}{\epsilon_0^2}, \frac{m_0}{\epsilon_0}\right) \;,$$

is (ϵ_0, ϵ_1)-good with respect to M.

Proof: In order to prove that the sample has the first property with probability at least $1-\delta/2$, we show that for every $M' \in \mathcal{M}$, and for every state $q' \in M'$, $m_{q'}/m \geq P^{M'}(q') - \epsilon_0$. In particular, if follows that for every state q' in any given PFA M', if $P^{M}(q') \geq 2\epsilon_0$, then $m_{q'}/m \geq \epsilon_0$, and thus $m_{q'} \geq \epsilon_0 m \geq m_0$. For a given $M' \in \mathcal{M}$, and a state $q' \in M'$, if $P^{M'}(q') \leq \epsilon_0$, then necessarily $m_{q'}/m \geq P^{M'}(q') - \epsilon_0$. There are at most $1/\epsilon_0$ states for which $P^{M'}(q') \geq \epsilon_0$ in each level, and hence, using Hoeffding's inequality (see Appendix C), with probability at least $1-\delta 2^{-(|\Sigma|n^2D+1)}$, for each such q', $m_{q'}/m \geq P^{M'}(q') - \epsilon_0$. Since the size of $\mathcal{M}$ is bounded by $2^{|\Sigma|n^2D}$, the above holds with probability at least $1-\delta/2$ for every M'.

And now for the second property. Since

$$
\begin{aligned}
m_0 &= \frac{|\Sigma|\, n^2\, D \;+\; 2D \ln\left(8(|\Sigma|+1)\right) + \ln\frac{1}{\delta}}{\epsilon_1^2} && (3.3)\\
&> \frac{1}{\epsilon_1^2} \ln \frac{8(|\Sigma+1)^{2D}\, 2^{|\Sigma|n^2D}}{\delta} \;, && (3.4)
\end{aligned}
$$

for a given M', and a given q', if $m_{q'} \geq m_0$ then using Hoeffding's inequality, and since there are less than $2(|\Sigma|+1)^D$ strings that can be generated starting from q', with probability larger than

$$
1 - \frac{\delta}{4(|\Sigma|+1)^D\, 2^{|\Sigma|n^2D}} \;,
$$

for every s, $|m_{q',s}/m_{q'} - P^{M'}_{q'}(s)| \leq \epsilon_1$. Since there are at most $2(|\Sigma|+1)^D$ states in M' (a bound on the size of the full tree of degree $|\Sigma+1|$), and using our bound on $|\mathcal{M}|$, we have the second property with probability at least $1-\delta/2$, as well. ■

Proof of Theorem 1

The proof of Theorem 1 is based on the following lemma in which we show that for every state q in M there exists a "representative" state $\hat{q}$ in $\widehat{M}$, which has significant weight, and for which $\hat{\gamma}(\hat{q},\cdot) \approx \gamma(q,\cdot)$.

Lemma 3.5.2 *If the sample is* (ϵ_0,ϵ_1)*-good for*

$$
\epsilon_1 < \min(\mu/4, \epsilon^2/8(|\Sigma|+1)) \;,
$$

then for $\epsilon_3 \leq 1/(2D)$*, and for* $\epsilon_2 \geq 2n|\Sigma|\epsilon_0/\epsilon_3$*, we have the following. For every level* d *and for every state* $q \in Q_d$*, if* $P^M(q) \geq \epsilon_2$ *then there exists a state* $\hat{q} \in \hat{Q}_d$ *such that:*

1. $P^M(W(q) \bigcap W(\hat{q})) \geq (1 - d\epsilon_3) P^M(q)$,
2. *for every symbol* σ, $\gamma(q,\sigma)/\hat{\gamma}(\hat{q},\sigma) \leq 1 + \epsilon/2$.

The proof of Lemma 3.5.2 is derived based on the following lemma in which we show a relationship between the graphs constructed by the algorithm and a PFA in $\mathcal{M}$.

Lemma 3.5.3 *If the sample is (ϵ_0, ϵ_1)-good, for $\epsilon_1 < \mu/4$, then there exists a PFA $M' \in \mathcal{M}$, $M' = (Q', q_0', \{q_f'\}, \Sigma, \tau', \gamma', \zeta)$, for which the following holds. Let G_{i_d} denote the first graph in which we consider folding nodes in level d. Then, for every level d, there exists a* one-to-one *mapping Φ_d from the nodes in the d'th level of G_{i_d},* into *Q_d', such that for every v in the d'th level of G_{i_d}, $W(v) = W(\Phi_d(v))$. Furthermore, $q' \in M'$ is a dominant major copy* iff $m_{q'} \geq m_0$.

Proof: We prove the claim by induction on d. M' is constructed in the course of the induction, where for each d we choose the dominant copies of the states in Q_d.

For $d = 1$, G_{i_1} is G_0. Based on the definition of $\mathcal{M}$, for every $M' \in \mathcal{M}$, for every $q \in Q_1$, and for every σ such that $\tau(q_0, \sigma) = q$, there exists a copy of q, $(q, \{q_0'\}, \sigma)$ in Q_1'. Thus for every v in the first level of G_0, all symbols that reach v reach the same state $q' \in M'$, and we let $\Phi_1(v) = q'$. Clearly, no two vertices are mapped to the same state in M'. Since all states in Q_1' are major copies by definition, we can choose the dominant copies of each state $q \in Q_1$ to be all copies q' for which there exists a node v such that $\Phi_1(v) = q'$, and m_v $(= m_{\Phi_1(v)}) \geq m_0$.

Assume the claim is true for $1 \leq d' < d$, we prove it for d. Though M' is only partially defined, we allow ourselves to use the notation $W(q')$ for states q' which belong to the levels of M' that are already constructed. Let $q \in Q_{d-1}$, let $\{q_i'\} \subset Q_{d-1}'$ be its copies, and for each i such that $\Phi_{d-1}^{-1}(q_i')$ is defined, let $u_i = \Phi_{d-1}^{-1}(q_i')$. Based on the *goodness* of the sample and our requirement on ϵ_1, for each u_i such that $m_{u_i} \geq m_0$, and for every string s, the difference between $P_{q_i'}^{M'}(s)$ and $m_{u_i}(s)/m_{u_i}$ is less than $\mu/4$. Hence, if a pair of nodes, u_i and u_j, mapped to q_i' and q_j' respectively, are tested for similarity by the algorithm, than the procedure Similar returns *similar*, and they are folded into one node v. Clearly, for every s, since

$$m_v(s)/m_v = (m_{u_i}(s) + m_{u_j}(s))/(m_{u_i} + m_{u_j}) ,$$

then $|m_v(s)/m_v - P_q^M(s)| < \mu/4$, and the same is true for any possible node that is the result of folding some subset of the u_i's that satisfy $m_{u_i} \geq m_0$. Since the target automaton is μ-distinguishable, none of these nodes are folded with any node w such that $\Phi_{d-1}(w) \notin \{q_i'\}$. Note that by the induction hypothesis, for every u_i such that $m_{q_i'} = m_{u_i} \geq m_0$, q_i' is a dominant copy of q.

Let v be a node in the d'th level of G_{i_d}. We first consider the case where v is a result of folding nodes in level $d-1$ of $G_{i_{d-1}}$. Let these nodes be $\{u_1, \ldots, u_\ell\}$. By the induction hypothesis they are mapped to states in Q_{d-1}' which are all dominant major copies of some state $r \in Q_{d-1}$. Let σ be the label of the edge entering v. Then

$$
\begin{aligned}
W(v) &= \bigcup_{j=1}^{\ell} W(u_j) \circ \sigma && (3.5)\\
&= \bigcup_{j=1}^{\ell} W(\Phi_{d-1}(u_j)) \circ \sigma && (3.6)\\
&= W((q, r, \sigma)) , && (3.7)
\end{aligned}
$$

where $q = \tau(r, \sigma)$. We thus set $\Phi_d(v) = q'$, where $q' = (q, r, \sigma)$ is a major copy of q in Q_d'. If $m_v \geq m_0$, we choose q' to be a dominant copy of q. If v is not a cause of any such merging in the previous level, then let $u \in G_{i_d}$ be such that $u \xrightarrow{\sigma} v$. Then

$$\begin{aligned} W(v) &= W(u) \circ \sigma & (3.8)\\ &= W(\Phi_{d-1}(u)) \circ \sigma & (3.9)\\ &= W(\tau'(\Phi_{d-1}(u), \sigma)) \,, & (3.10)\end{aligned}$$

and we simply set

$$\Phi_d(v) = \tau'(\Phi_{d-1}(u), \sigma) \ .$$

If $m_u \geq m_0$, then $\Phi_{d-1}(u)$ is a (single) dominant copy of some state $r \in Q_{d-1}$, and $q' = \Phi_d(v)$ is a major copy. If $m_v \geq m_0$, we choose q' to be a dominant copy of q. ∎

Proof of Lemma 3.5.2 : For both claims we rely on the relation that is shown in Lemma 3.5.3, between the graphs constructed by the algorithm and some PFA M' in $\mathcal{M}$. We show that the weight in M' of the dominant copies of every state $q \in Q_d$ for which $P^M(q) \geq \epsilon_2$ is at least $1 - d\epsilon_3$ of the weight of q. The first claim directly follows, and for the second claim we apply the goodness of the sample. We prove this by induction on d.

For $d = 1$: The number of copies of each state in Q^1_d is at most $|\Sigma|$. By the goodness of the sample, each copy whose weight is greater than $2\epsilon_0$, is chosen to be dominant, and hence the total weight of the dominant copies is at least $\epsilon_2 - 2|\Sigma|\epsilon_0$ which based on our choice of ϵ_2 and ϵ_3, is at least $(1 - \epsilon_3)\epsilon_2$.

For $d > 1$: By the induction hypothesis, the total weight of the dominant major copies of a state r in Q_{d-1} is at least $(1 - (d-1)\epsilon_3)P^M(r)$. For $q \in Q_d$, The total weight of the major copies of q is thus at least

$$\sum_{r,\sigma:\, r \xrightarrow{\sigma} q} (1 - (d-1)\epsilon_3)P^M(r) \cdot \gamma(r, \sigma) \;=\; (1 - (d-1)\epsilon_3)P^M(q) \ . \tag{3.11}$$

There are at most $n|\Sigma|$ major copies of q, and hence the weight of the non-dominant ones is at most $2n|\Sigma|\epsilon_0 < \epsilon_3\epsilon_2$ and the claim follows.

And now for the second claim. We break the analysis into two cases. If $\gamma(q,\sigma) \leq \gamma_{min} + \epsilon_1$, then since $\hat{\gamma}(\hat{q}, \sigma) \geq \gamma_{min}$ by definition, and $\epsilon_1 \leq \epsilon^2/(8(|\Sigma|+1))$, if we choose $\gamma_{min} = \epsilon/(4|(\Sigma|+1))$, then $\gamma(q,\sigma)/\hat{\gamma}(\hat{q},\sigma) \leq 1 + \epsilon/2$, as required.

If $\gamma(q,\sigma) > \gamma_{min} + \epsilon_1$, then let $\gamma(q,\sigma) = \gamma_{min} + \epsilon_1 + x$, where $x > 0$. Based on our choice of ϵ_2 and ϵ_3, for every $d \leq D$, $\epsilon_2(1 - d\epsilon_3) \geq 2\epsilon_0$. By the goodness of the sample, and the definition of $\hat{\gamma}(\cdot,\cdot)$, we have that

$$\begin{aligned} \hat{\gamma}(\hat{q},\sigma) &\geq (\gamma(q,\sigma) - \epsilon_1)(1 - (|\Sigma|+1)\gamma_{min}) + \gamma_{min} & (3.12)\\ &= (x + \gamma_{min})(1 - \epsilon/4) + \gamma_{min} & (3.13)\\ &\geq \frac{x + \gamma_{min}(1+\epsilon/2)}{1+\epsilon/2} \geq \frac{\gamma(q,\sigma)}{1+\epsilon/2} \ . & (3.14)\end{aligned}$$

∎

Proof of Theorem 1: We prove the theorem based on Lemma 3.5.2. For brevity of the following computation, we assume that M and $\widehat{M}$ generate strings of length exactly D. This can be assumed without loss of generality, since we can require that both PFAs "pad" each shorter string they generate, with a sequence of ζ's, with no change to the KL-divergence between the PFAs.

$$
\begin{aligned}
\mathcal{D}_{KL}(\, P^M \| P^{\widehat{M}}\,) \quad &= \quad \sum_{\sigma_1 \ldots \sigma_D} P^M(\sigma_1 \ldots \sigma_D) \log \frac{P^M(\sigma_1 \ldots \sigma_D)}{P^{\widehat{M}}(\sigma_1 \ldots \sigma_D)} \\
&= \quad \sum_{\sigma_1} \sum_{\sigma_2 \ldots \sigma_D} P^M(\sigma_1) P^M(\sigma_2 \ldots \sigma_D | \sigma_1) \left[\log \frac{P^M(\sigma_1)}{P^{\widehat{M}}(\sigma_1)} + \log \frac{P^M(\sigma_2 \ldots \sigma_D | \sigma_1)}{P^{\widehat{M}}(\sigma_2 \ldots \sigma_D | \sigma_1)} \right] \\
&= \quad \sum_{\sigma_1} P^M(\sigma_1) \log \frac{P^M(\sigma_1)}{P^{\widehat{M}}(\sigma_1)} \\
&+ \quad \sum_{\sigma_1} P^M(\sigma_1) \cdot \mathcal{D}_{KL}\left(P^M(\sigma_2 \ldots \sigma_D | \sigma_1) \| P^{\widehat{M}}(\sigma_2 \ldots \sigma_D | \sigma_1) \right) \\
&= \quad \sum_{\sigma_1} P^M(\sigma_1) \log \frac{P^M(\sigma_1)}{P^{\widehat{M}}(\sigma_1)} + \sum_{\sigma_1} P^M(\sigma_1) \sum_{\sigma_2} P^M(\sigma_2|\sigma_1) \log \frac{P^M(\sigma_2|\sigma_1)}{P^{\widehat{M}}(\sigma_2|\sigma_1)} \\
&+ \quad \ldots \\
&+ \quad \sum_{\sigma_1 \ldots \sigma_d} P^M(\sigma_1 \ldots \sigma_d) \sum_{\sigma_{d+1}} P^M(\sigma_{d+1}|\sigma_1 \ldots \sigma_d) \log \frac{P^M(\sigma_{d+1}|\sigma_1 \ldots \sigma_d)}{P^{\widehat{M}}(\sigma_{d+1}|\sigma_1 \ldots \sigma_d)} \\
&+ \quad \ldots \\
&+ \quad \sum_{\sigma_1 \ldots \sigma_{D-1}} P^M(\sigma_1 \ldots \sigma_{D-1}) \sum_{\sigma_D} P^M(\sigma_D|\sigma_1 \ldots \sigma_{D-1}) \log \frac{P^M(\sigma_D|\sigma_1 \ldots \sigma_{D-1})}{P^{\widehat{M}}(\sigma_D|\sigma_1 \ldots \sigma_{D-1})} \\
&= \quad \sum_{d=0}^{D-1} \sum_{q \in Q_d} \sum_{\hat{q} \in \hat{Q}_d} P^M\left(W(q) \bigcap W(\hat{q}) \right) \sum_{\sigma} P_q^M(\sigma) \log \frac{P_q^M(\sigma)}{P_{\hat{q}}^{\widehat{M}}(\sigma)} \\
&= \quad \sum_{d=0}^{D-1} \sum_{q \in Q_d} P^M(q) \sum_{\hat{q} \in \hat{Q}_d} P^M\left(W(q) \bigcap W(\hat{q}) \right) / P^M(q) \sum_{\sigma} P_q^M(\sigma) \log \frac{P_q^M(\sigma)}{P_{\hat{q}}^{\widehat{M}}(\sigma)} \\
&\leq \quad \sum_{d=0}^{D-1} \sum_{q \in Q_d,\ P^M(q) < \epsilon_2} P^M(q) \log(1/\gamma_{min}) \\
&+ \quad \sum_{d=0}^{D-1} \sum_{q \in Q_d,\ P^M(q) \geq \epsilon_2} P^M(q)[(1 - d\epsilon_3)\log(1 + \epsilon/2) + d\epsilon_3 \log(1/\gamma_{min})] \\
&\leq \quad (nD\epsilon_2 + D^2\epsilon_3)\log(1/\gamma_{min}) + \epsilon/2
\end{aligned}
$$

If we choose ϵ_2 and ϵ_3 so that

$$\epsilon_2 \leq \epsilon/(4nD\log(1/\gamma_{min})) \quad \text{and} \quad \epsilon_3 \leq \epsilon/(4D^2\log(1/\gamma_{min})) \ ,$$

then the expression above is bounded by ϵ, as required. Adding the requirements on ϵ_2 and ϵ_3 from Lemma 3.5.2, we get the following requirement on ϵ_0:

$$\epsilon_0 \leq \epsilon^2/(32n^2|\Sigma|D^3\log^2(4(|\Sigma|+1)/\epsilon)) \ ,$$

from which we can derive a lower bound on m by applying Lemma 3.5.1. ∎

3.6 An Online Version of the Algorithm

In this section we describe an *online* version of our learning algorithm. The online algorithm is used in our cursive handwriting recognition system described in Chapter 5. We start by defining our notion of *online learning* in the context of learning distributions on strings.

3.6.1 An Online Learning Model

In the online setting, the algorithm is presented with an infinite sequence of *trials*. At each time step, t, the algorithm receives a trial string $s^t = s_1 \ldots s_{\ell-1}\zeta$ generated by the target machine, M, and it should output the probability assigned by its current hypothesis, H_t, to s^t. The algorithm then transforms H_t into H_{t+1}. The hypothesis at each trial need not be a PFA, but may be any data structure which can be used in order to define a probability distribution on strings. In the transformation from H_t into H_{t+1}, the algorithm uses only H_t itself, and the new string s^t. Let the *error* of the algorithm on s^t, denoted by $err_t(s^t)$, be defined as $\log(P^M(s^t)/P_t(s^t))$. We shall be interested in the *average cumulative* error $Err_t \stackrel{\rm def}{=} \frac{1}{t}\sum_{t'\leq t} err_t(s^t)$.

We allow the algorithm to err an *unrecoverable error* at some stage t, with total probability that is bounded by Δ. We ask that there exist functions $\delta(t,\mu,n,D,|\Sigma|)$, and $\epsilon(t,\mu,n,D,|\Sigma|)$, such that the following hold. $\delta(t,\mu,n,D,|\Sigma|)$ is of the form $\beta_1(\mu,n,D,|\Sigma|)2^{t^{-\alpha_1}}$, where β_1 is a polynomial in μ, n, D, and $|\Sigma|$, and $0<\alpha_1<1$, and $\epsilon(t)$ is of the form $\beta_2(\mu,n,D,|\Sigma|)t^{-\alpha_2}$, where β_2 is a polynomial in μ, n, D, and $|\Sigma|$, and $0<\alpha_2<1$. Since we are mainly interested in the dependence of the functions on t, let them be denoted for short by $\delta(t)$, and $\epsilon(t)$. For every trial t, if the algorithm has not erred an unrecoverable error prior to that trial, then with probability at least $1-\delta(t)$, the average cumulative error is small, namely $Err_t \leq \epsilon(t)$. Furthermore, we require that the size of the hypothesis H_t be a *sublinear* function of t. This last requirement implies that an algorithm which simply remembers *all* trial strings, and each time constructs a new PFA "from scratch" is not considered an online algorithm.

3.6.2 An Online Learning Algorithm

We now describe how to modify the *batch* algorithm Learn-Acyclic-PFA, presented in Section 3.4, to become an online algorithm. The pseudo-code for the algorithm is presented at the end of the section. At each time t, our hypothesis is a graph $G(t)$, which has the same form as the graphs used by the batch algorithm. $G(1)$, the initial hypothesis, consists of a single root node v_0 where for every $\sigma \in \Sigma \bigcup \{\zeta\}$, $m_{v_0}(\sigma) = 0$ (and hence, by definition, $m_{v_0} = 0$). Given a new trial string s^t, the algorithm checks if there exists a path corresponding to s^t, in $G(t)$. If there are missing nodes and edges on the path, then they are added. The counts corresponding to the new edges and nodes are all set to 0. The algorithm then outputs the probability that a PFA defined based on $G(t)$ would have assigned to s^t. More precisely, let $s^t = s_1 \ldots s_\ell$, and let $v_0 \ldots v_\ell$ be the nodes on the path corresponding to s^t. Then the algorithm outputs the following product:

$$P_t(s^t) = \Pi_{i=0}^{\ell-1}\left(\frac{m_{v_i}(s_{i+1})}{m_{v_i}}(1-(|\Sigma|+1)\gamma(t)) + \gamma_{min}(t)\right),$$

where $\gamma_{min}(t)$ is a decreasing function of t.

The algorithm adds s^t to $G(t)$, and increases by one the counts associated with the edges on the path corresponding to s^t in the updated $G(t)$. If for some node v on the path, $m_v \geq m_0$, then we execute stage (2) in the batch algorithm, starting from $G_0 = G(t)$, and letting $d(0)$ be the depth of v, and D be the depth of $G(t)$. We let $G(t+1)$ be the final graph constructed by stage (2) of the batch algorithm.

In the algorithm described above, as in the batch algorithm, a decision to fold two nodes in a graph $G(t)$, which do not correspond to the same state in M, is an *unrecoverable* error. Since the algorithm does not backtrack and "unfold" nodes, the algorithm has no way of recovering from such a decision, and the probability assigned to strings passing through the folded nodes, may be erroneous from that point on. Similarly to the analysis in the batch algorithm, it can be shown that for an appropriate choice of m_0, the probability that we perform such a merge at any time in the algorithm, is bounded by Δ. If we never perform such merges, we expect that as t increases, we both encounter nodes that correspond to states with decreasing weights, and our predictions become "more reliable" in the sense that $m_v(\sigma)/m_v$ gets closer to its expectation (and the probability of a large error decreases). A more detailed analysis can give precise bounds on $\delta(t)$ and $\epsilon(t)$.

What about the size of our hypotheses? Let a node v be called *reliable* if $m_v \geq m_0$. Using the same argument needed for showing that with probability at least $1 - \Delta$ we never merge nodes that correspond to different states, we get that with the same probability we merge every pair of reliable nodes which correspond to the same state. Thus, the number of reliable nodes is never larger than $D\,n$. From every reliable node there are edges going to at most $|\Sigma|$ unreliable nodes. Each unreliable node is a root of a tree in which there are at most $D\,m_0$ additional unreliable nodes. We thus get a bound on the number of nodes in $G(t)$ which is *independent* of t. Since for every v and σ in $G(t)$, $m_v(\sigma) \leq t$, the counts on the edges contribute a factor of $\log t$ to the total size the hypothesis.

Algorithm Online-Learn-Acyclic-PFA

1. Initialize: $t := 1$, $G(1)$ is a graph with a single node v_0, $\forall \sigma \in \Sigma \bigcup \{\zeta\}$, $m_{v_0}(\sigma) = 0$;
2. Repeat:
   (a) Receive the new string s^t;
   (b) If there does not exist a path in $G(t)$ corresponding to s^t, then add missing edges and nodes to $G(t)$, and set their corresponding counts to 1.
   (c) Let $v_0 \ldots v_\ell$ be the nodes on the path corresponding to s^t in $G(t)$;
   (d) Output: $P_t(s^t) = \Pi_{i=0}^{\ell-1} \left(\frac{m_{v_i}(s_{i+1})}{m_{v_i}} (1 - (|\Sigma| + 1)\gamma_{min}(t)) + \gamma_{min}(t) \right)$;
   (e) Add 1 to the count of each edge on the path corresponding to s^t in $G(t)$;
   (f) If for some node v_i on the path $m_{v_i} = m_0$ then do:
      i. $i := 0$, $G_0 = G(t)$, $d(0) =$ depth of v_i, $D =$ depth of $G(t)$;
      ii. Execute step (2) in **Learn-Acyclic-PFA**;
      iii. $G(t+1) := G_i$, $t := t + 1$.

3.7 Building Pronunciation Models for Spoken Words

We slightly modified our algorithm in order to get a more compact APFA. We chose to fold nodes with small counts into the graph itself (instead of adding the extra nodes, $small(d)$). We also allowed folding states from different levels, thus the resulting hypothesis is more compact. For the online mode we simply left edges with zero count 'floating', that is the out degree of each node is at most $|\Sigma|$.

In natural speech, a word might be pronounced differently by different speakers. For example, the phoneme `t` in `often` is often omitted, the phoneme `d` in the word `muddy` might be flapped, *etc.*. One possible approach to model such pronunciation variations is to construct stochastic models that capture the distributions of the possible pronunciations for words in a given database. The models should reflect not only the alternative pronunciations but also the apriori probability of a given phonetic transcription of the word. This probability depends on the distribution of the different speakers that uttered the words in the training set. Such models can be used as a component in a speech recognition system. The same problem was studied in [127]. Here, we briefly discuss how our algorithm for learning APFAs can be used to efficiently build probabilistic pronunciation models for words.

We used the TIMIT (Texas Instruments-MIT) database. This database contains the acoustic waveforms of continuous speech with phone labels from an alphabet of 62 phones, that constitute a temporally aligned phonetic transcription to the uttered words. For the purpose of building pronunciation models, the acoustic data was ignored and we partitioned the phonetic labels according to the words that appeared in the data. We then built an APFA for each word in the data set. Examples of the resulting APFAs for the words `have, had` and `often` are shown in Figure 3.3. The symbol labeling each edge is one of the possible 62 phones or the final symbol, ζ, represented in the figure by the string `End`. The number on each edge is the count associated with the edge, i.e., the number of times the edge was traversed in the training data. The figure shows that the resulting models indeed capture the different pronunciation styles. For instance, all the possible pronunciations of the word `often` contain the phone `f` and there are paths that share the optional `t` (the phones `tcl t`) and paths that omit it. Similar phenomena are captured by the models for the words `have` and `had` (the optional semivowels `hh` and `hv` and the different pronunciations for `d` in `had` and for `v` in `have`).

In order to quantitatively check the performance of the models, we filtered and partitioned the data in the same way as in [127]. That is, words occurring between 20 and 100 times in the data set were used for evaluation. Of these, 75% of the occurrences of each word were used as training data for the learning algorithm and the remaining 25% were used for evaluation. The models were evaluated by calculating the log probability (likelihood) of the proper model on the phonetic transcription of each word in the test set. The results are summarized in Table 3.1. The performance of the resulting APFAs is surprisingly good, compared to the performance of the Hidden Markov Model reported in [127]. To be cautious, we note that it is not certain whether the better performance (in the sense that the likelihood of the APFAs on the test data is higher) indeed indicates better performance in terms of recognition error rate. Yet, the much smaller time needed for the learning suggests that our algorithm might be the method of choice for this problem when large amounts of training data are presented.

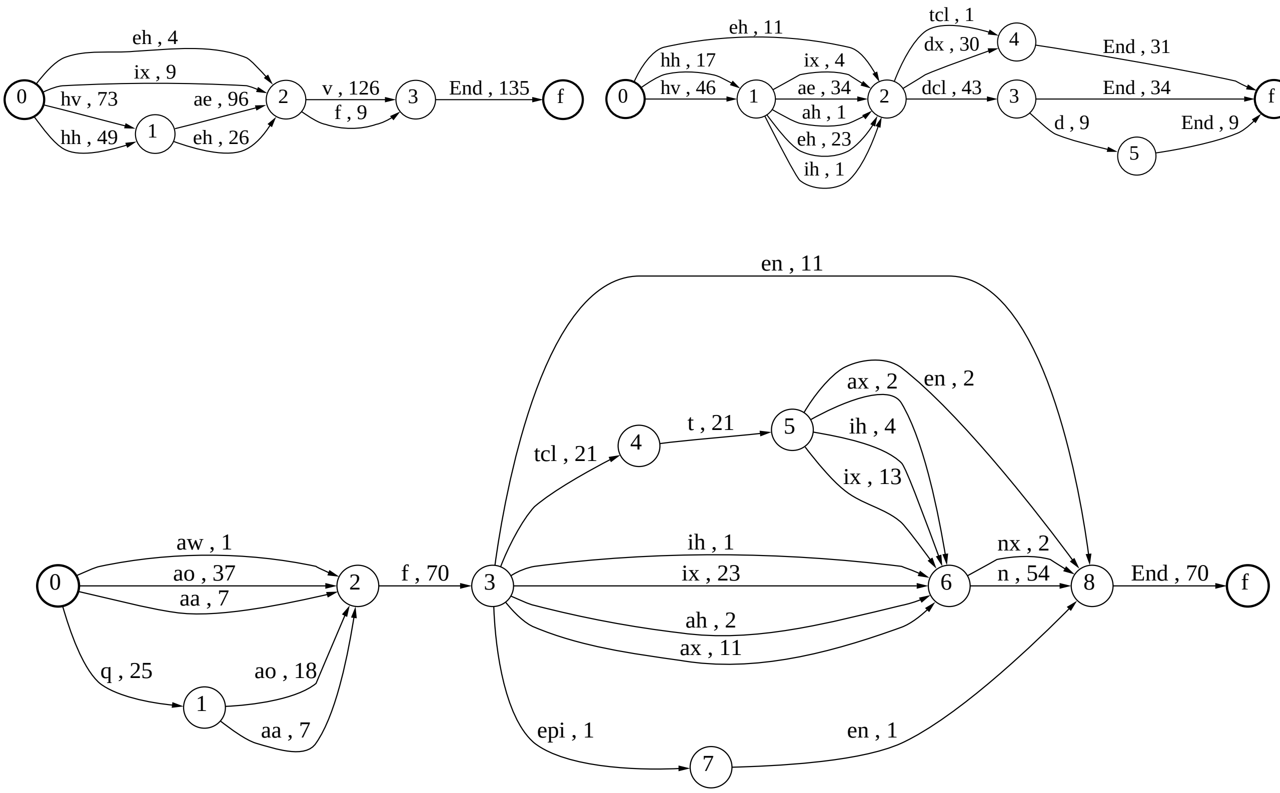


Figure 3.3: Examples of pronunciation models based on APFAs for the words `have`, `had` and `often` trained from the TIMIT database.

Model	*Log-Likelihood*	*Perplexity*	*States*	*Transitions*	*Training Time*
APFA	-2142.8	1.563	1398	2197	23 **seconds**
HMM [127]	-2343.0	1.849	1204	1542	29:49 **minutes**

Table 3.1: The performance of APFAs compared to Hidden Markov Models (HMM) as reported by Stolcke and Omohundro. *Log-Likelihood* is the logarithm of the probability induced by the two classes of models on the test data, *Perplexity* is the average number of phones that can follow in any given context within a word. Although the HMM has fewer states and transitions than the APFA, it has more parameters than the APFA since an additional output probability distribution vector is associated with each state of the HMM.

3.8 Identification of Noun Phrases in Natural Text

In this section we describe and evaluate an English noun phrase recognizer based on competing APFAs. Recognizing noun phrases is an important task in automatic text processing, for applications such as information retrieval, translation tools and data extraction from texts. A common practice is to recognize noun phrases by first analyzing the text with a part of speech tagger, which assigns the appropriate part of speech (verb, noun, adjective etc.) for each word in the text. Then, noun phrases are identified by manually defined regular expressions that are matched against the part of speech sequences (*cf.* [18, 53]). In the next chapter we describe a part of speech tagging system. In this section, we use a *tagged* data set from the UPENN tree-bank corpus and concentrate merely on identifying noun phrases using the tagged corpus.

In addition to the tagging information, the corpus is segmented into sentences and the noun phrases appearing in each sentence are marked. We used the marked and tagged corpus to build two models based on APFAs: the first was built from the noun phrases segments and the second from all the 'fillers', i.e., the consecutive tagged words that do not belong to any noun phrase. Therefore, each 'filler' is enclosed by either a noun phrase or by a begin/end of sentence marker. The advantage of such an approach is in its flexibility. We can construct models for other syntactic structures, such as verb phrases, by finer segmentation of the 'fillers'. Therefore, we can keep the noun phrase APFA unaltered, while more APFAs for other syntactic structures are built. We used over $250,000$ marked tags and tested the performance on more than $37,000$ tags. The segmentation scheme presented subsequently is a variation on a dynamic programming technique. Since we extensively use this technique in the next chapters, we defer an elaborate description of the technique to the coming chapters.

Without loss of generality, we assume that there are two APFAs: a filler APFA and a noun phrase APFA, denoted by M^{np} and M^{f}, respectively. The noun phrase identification procedure presented here simply generalize for the case of several different syntactic structures. Identifying or locating noun phrases in a tagged sentence is done by dividing the sentence into non-overlapping segments, each is either a noun phrase or a filler. The segmentation is done via a competition between the two APFAs as follows. Denote the tags that constitute a sentence by $t_1, t_2, \ldots, t_L$. A segmentation $\mathcal{S}$ is a sequence of $K+1$ monotonically increasing indices, $\mathcal{S} = s_0, s_1, \ldots, s_K$, such that $s_0 = 1$, $s_K = L+1$. Each segment is also associated with an indicator from the set $\{np, f\}$. Let the sequence of indicators be denoted by $\mathcal{I} = i_1, i_2, \ldots, i_K\, (i_j \in \{np, f\})$. A pair of a segmentation sequence and a sequence of indicators is termed the syntactic parsing of the sentence. The likelihood of a tagged sentence given a possible syntactic parsing is,

$$P(t_1, t_2, \ldots, t_L | \mathcal{S}, \mathcal{I}) = \prod_{k=1}^{K} P^{i_k}(t_{s_{k-1}}, \ldots, t_{s_k - 1}, \zeta) \ ,$$

where ζ is the final symbol which we add to the set of possible part-of-speech tags. If apriori all possible parsings of a sentence are of equal probability then the above is proportional to the probability of a syntactic parsing given the tagged sentence. The most likely parsing of a sentence is found using a dynamic programming scheme. Using the same scheme, we can also calculate the probability that a tagged word belongs to a noun phrase as follows,

$$P(t_j \text{ belongs to a noun phrase} | t_1, t_2, \ldots, t_L) = \sum_{S,I \text{ s.t. } \exists j:\, s_k \le j < s_{k+1},\, i_j = np} P(t_1, t_2, \ldots, t_L | \mathcal{S}, \mathcal{I}) \ \Big/ \ \sum_{S,I} P(t_1, t_2, \ldots, t_L | \mathcal{S}, \mathcal{I}) \ .$$

We classify t_j as a part of a noun phrase if the above probability is greater than $1/2$. We tested the performance of our APFA based identification scheme by comparing the classification of the system to the actual markers of noun phrases. A typical result is given in Table 3.2. Less than 2.5% of the words were misclassified by the system.

While the results obtained are comparable to the performance of systems that employ manually designed regular expressions, our approach does not require any intervention of an expert. Rather, it is based on the APFA learning algorithm combined with a dynamic programming based parsing scheme. Furthermore, as discussed above, identifying other syntactic structures can be achieved using the same approach, without any changes to the noun phrase APFA.

`Sentence`	Tom	Smith	,	group	chief	executive	of	U.K.	metals
`POS tag`	PNP	PNP	,	NN	NN	NN	IN	PNP	NNS
`Classification`	1	1	0	1	1	1	0	1	1
`Prediction`	0.99	0.99	0.01	0.98	0.98	0.98	0.02	0.99	0.99
`Sentence`	and	industrial	materials	maker	,	will	become	chairman	.
`POS tag`	CC	JJ	NNS	NN	,	MD	VB	NN	.
`Classification`	1	1	1	1	0	0	0	1	0
`Prediction`	0.67	0.96	0.99	0.96	0.03	0.03	0.01	0.87	0.01

Table 3.2: Identification of noun phrases using competing APFAs. In this typical example, a long noun phrase is identified correctly with high confidence. The table contains for each word in the sentence its part-of-speech tag, a classification bit set to 1 iff the word is a part of a noun phrase, and the probability of belonging to a noun phrase as assigned by the APFAs based system.

Chapter 4

The Power of Amnesia

4.1 Introduction

In this chapter we study a different subclass of probabilistic automata. Here we are interested in the *stationary* properties of sequences that are used in the analysis of language [71, 92] and also of biological sequences such as DNA and proteins [76]. These kinds of complex sequences clearly do not have any simple underlying statistical source since they are generated by natural sources. However, they typically exhibit the following statistical property, which we refer to as the *short memory* property. If we consider the (empirical) probability distribution on the next symbol given the preceding subsequence of some given length, then there exists a length L (the *memory length*) such that the conditional probability distribution does not change substantially if we condition it on preceding subsequences of length greater than L.

This observation lead Shannon, in his seminal paper [121], to suggest modeling such sequences by Markov chains of order $L > 1$, where the order is the memory length of the model. Alternatively, such sequences may be modeled by Hidden Markov Models (HMMs) which are more complex distribution generators and hence may capture additional properties of natural sequences. These statistical models define rich families of sequence distributions and moreover, they give efficient procedures both for generating sequences and for computing their probabilities. However, both models have severe drawbacks. The size of Markov chains grows exponentially with their order, and hence only very low order Markov chains can be considered in practical applications. Such low order Markov chains might be very poor approximators of the relevant sequences. In the case of HMMs, there are known hardness results concerning their learnability as well as practical disadvantages as we discussed in Chapter 1.

In this chapter we propose and analyze a simple stochastic model based on the following motivation. It has been observed that in many natural sequences, the memory length depends on the context and is *not fixed*. The model we suggest is hence a variant of order L Markov chains, in which the order, or equivalently, the memory, is variable. We describe this model using a subclass of Probabilistic Finite Automata (PFA), which we name *Probabilistic Suffix Automata* (PSA).

Each state in a PSA is labeled by a string over an alphabet Σ. The transition function between the states is defined based on these string labels, so that a walk on the underlying graph of the automaton, related to a given sequence, always ends in a state labeled by a suffix of the sequence. The lengths of the strings labeling the states are bounded by some upper bound L, but different states may be labeled by strings of different length, and are viewed as having *varying* memory length. When a PSA generates a sequence, the probability distribution on the next symbol generated is completely defined given the previously generated subsequence of length at most L. Hence, as mentioned above, the probability distributions these automata generate can be equivalently generated by Markov chains of order L, but the description using a PSA may be much more succinct. Since the size of order L markov chains is exponential in L, their estimation requires data length and time exponential in L.

In our learning model we assume that the learning algorithm is given a sample (consisting either of several sample sequences or of a single sample sequence) generated by an unknown target PSA M of some bounded size. The algorithm is required to output a hypothesis machine $\hat{M}$, which is not necessarily a PSA but which has the following properties. $\hat{M}$ can be used both to efficiently generate a distribution which is similar to the one generated by M, and given any sequence s, it can efficiently compute the probability assigned to s by this distribution.

Several measures of the quality of a hypothesis can be considered. Since we are mainly interested in models for statistical classification and pattern recognition, the most natural measure is the Kullback-Leibler (KL) divergence. Our results hold equally well for the variation (L_1) distance and other norms, which are upper bounded by the KL-divergence. Since the KL-divergence between Markov sources grows linearly with the length of the sequence, the appropriate measure is the KL-divergence per symbol. Therefore, we use a goodness measure slightly different from the one used in the previous chapter: we define an ϵ-good hypothesis to be an hypothesis which has at most ϵ KL-divergence *per symbol* to the target source. The hypothesis our algorithm outputs, belongs to a class of probabilistic machines named Probabilistic Suffix Trees (PST). The learning algorithm grows such a suffix tree starting from a single root node, and adaptively adds nodes (strings) for which there is strong evidence in the sample that they significantly affect the prediction properties of the tree.

We show that every distribution generated by a PSA can equivalently be generated by a PST which is not much larger. The converse is not true in general. We can however characterize the family of PSTs for which the converse claim holds, and in general, it is always the case that for every PST there exists a not much larger PFA that generates an equivalent distribution. There are some contexts in which PSAs are preferable, and some in which PSTs are preferable, and therefore we use both representation. For example, PSAs are more efficient generators of distributions, and since they are probabilistic automata, their well defined state space and transition function can be exploited by dynamic programming algorithms which are used for solving many practical problems. In addition, there is a natural notion of the *stationary distribution* on the states of a PSA which PSTs lack. On the other hand, PSTs sometimes have more succinct representations than the equivalent PSAs, and there is a natural notion of growing them.

Stated formally, our main theoretical result is the following. If both a bound L, on the memory length of the target PSA, and a bound n, on the number of states the target PSA has, are known, then for every given $0 < \epsilon < 1$ and $0 < \delta < 1$, our learning algorithm outputs an ϵ-good hypothesis PST, with confidence $1 - \delta$, in time polynomial in L, n, $|\Sigma|$, $\frac{1}{\epsilon}$ and $\frac{1}{\delta}$. Furthermore, such a hypothesis can be obtained from a *single* sample sequence if the sequence length is also polynomial in a parameter related to the rate in which the target machine converges to its stationary distribution. Despite an intractability result concerning the learnability of distributions generated by Probabilistic Finite Automata [72] (discussed in Chapter 1), our restricted model can be learned in a PAC-like sense efficiently. This has not been shown so far for any of the more popular sequence modeling algorithms.

The machines used as our hypothesis representation, namely Probabilistic Suffix Trees (PSTs), were introduced (in a slightly different form) in [110] and have been used for other tasks such as universal data compression [110, 111, 139, 141]. Perhaps the strongest among these results and which is most tightly related to the results presented in this chapter was presented by Willems *et. al.* in [141]. This paper describes an efficient sequential procedure for universal data compression for PSTs by using a larger model class. This algorithm can be viewed as a distribution learning algorithm but the hypothesis it produces is not a PST or a PSA and hence cannot be used for many applications. Willems *et. al.* show that their algorithm can be modified to give the minimum description length PST. However, in case the source generating the examples is a PST, they are able to show that this PST convergence only *in the limit* of infinite sequence length to that source.

The model we propose is used very effectively for such tasks as correcting corrupted text and in fact even simpler models have been the tool of choice for language modeling in speech recognition systems. However, we would like to emphasize that any finite state model cannot capture the recursive nature of natural language. For instance, very long range correlations between words in a text such as those arising from subject matter or even relatively local dependencies created by very long but frequent compound names or technical terms, cannot be captured by our model. In the last chapter we speculate about possible extensions that may cope with such long range correlations.

This chapter has two parts. In the first part we describe and analyze our model and its learning algorithm while the second part is devoted to applications of the model. We start the first part with Section 4.2 in which we give basic definitions and notation and describe the families of distributions studied in this chapter, namely those generated by PSAs and those generated by PSTs. In Section 4.4 we discuss the relation between the above two families of distributions and present some equivalence results. In Section 4.5 the learning algorithm is described. Most of the proofs regarding the correctness of the learning algorithm are given in Section 4.6. The second part begins with a demonstration of the power of our learning algorithm. In Section 4.7 we use our algorithm to learn the 'low-order' alphabetic structure of natural English text, and use the resulting hypothesis for correcting corrupted text. In Section 4.8 we use our algorithm to build a simple stochastic model for E.*coli* DNA. Finally, in Section 4.9 we describe and evaluate a complete part-of-speech tagging system based on the proposed model and its learning algorithm. The more technical lemmas regarding the correctness of the learning algorithm are given in Appendix B.

4.2 Preliminaries

In this section we describe the family of distributions studied in this chapter. We start with some basic notation that we use throughout the chapter.

4.2.1 Basic Definitions and Notations

Let Σ be a finite alphabet. By Σ^* we denote the set of all possible strings over Σ. For any integer N, Σ^N denotes all strings of length N, and $\Sigma^{\leq N}$ denotes the set of all strings with length *at most* N. The empty string is denoted by $\mathtt{e}$. For any string $s = s_1 \ldots s_l$, $s_i \in \Sigma$, we use the following notations:

- The longest prefix of s different from s is denoted by $\mathit{prefix}(s) \stackrel{\mathrm{def}}{=} s_1 s_2 \ldots s_{l-1}$.

- The longest suffix of s different from s is denoted by $\mathit{suffix}(s) \stackrel{\rm def}{=} s_2 \ldots s_{l-1}s_l$.
- The set of all suffixes of s is denoted by $\mathit{Suffix}^*(s) \stackrel{\rm def}{=} \{s_i \ldots s_l \mid 1 \le i \le l\} \cup \{\mathsf{e}\}$. A string s' is a *proper* suffix of s, if it a suffix of s but is not s itself.
- Let s^1 and s^2 be two strings in Σ^*. If s^1 is a suffix of s^2 then we shall say that s^2 is a *suffix extension* of s^1.
- A set of strings S is called a *suffix free* set if $\forall s \in S$, $\mathit{Suffix}^*(s) \cap S = \{s\}$.

4.2.2 Probabilistic Finite Automata and Prediction Suffix Trees

Probabilistic Finite Automata

In this chapter we use the standard definition of probabilistic finite automata. We therefore repeat the definition of a PFA A *Probabilistic Finite Automaton (PFA)* M is a 5-tuple $(Q, \Sigma, \tau, \gamma, \pi)$, where Q is a finite set of *states*, Σ is a finite *alphabet*, $\tau : Q \times \Sigma \rightarrow Q$ is the *transition function*, $\gamma : Q \times \Sigma \rightarrow [0,1]$ is the *next symbol probability function*, and $\pi : Q \rightarrow [0,1]$ is the *initial probability distribution* over the starting states. The functions γ and π must satisfy the following conditions: for every $q \in Q$, $\sum_{\sigma\in\Sigma} \gamma(q,\sigma) = 1$, and $\sum_{q\in Q} \pi(q) = 1$. We assume that the transition function τ is defined on all states q and symbols σ for which $\gamma(q,\sigma) > 0$, and on no other state-symbol pairs. τ can be extended to be defined on $Q \times \Sigma^*$ as follows:

$$\tau(q, s_1s_2 \ldots s_l) = \tau(\tau(q, s_1 \ldots s_{l-1}), s_l) = \tau(\tau(q, \mathit{prefix}(s)), s_l) \ .$$

This standard form of a PFA generates strings of *infinite* length, but we shall always discuss probability distributions induced on prefixes of these strings which have some specified finite length. If P_M is the probability distribution M defines on infinitely long strings, then P_M^N, for any $N \ge 0$, will denote the probability induced on strings of length N. We shall sometimes drop the superscript N, assuming that it is understood from the context. The probability that M generates a string $r = r_1r_2 \ldots r_N$ in Σ^N is

$$P_M^N(r) = \sum_{q^0 \in Q} \pi(q^0) \prod_{i=1}^{N} \gamma(q^{i-1}, r_i) \ , \tag{4.1}$$

where $q^{i+1} = \tau(q^i, r_i)$.

Probabilistic Suffix Automata

We are interested in learning a subclass of PFAs which we name *Probabilistic Suffix Automata* (PSA). These automata have the following property. Each state in a PSA M is labeled by a string of finite length in Σ^*. The set of strings labeling the states is suffix free. For every two states $q^1, q^2 \in Q$ and for every symbol $\sigma \in \Sigma$, if $\tau(q^1, \sigma) = q^2$ and q^1 is labeled by a string s^1, then q^2 is labeled by a string s^2 which is a suffix of $s^1 \cdot \sigma$. In order that τ be well defined on a given set of strings S, not only must the set be suffix free, but it must also have the following property. For every string s in S labeling some state q, and every symbol σ for which $\gamma(q,\sigma) > 0$, there exists a string in S which is a suffix of $s\sigma$. For our convenience, from this point on, if q is a state in Q then q will also denote the string labeling that state.

We assume that the underlying graph of M, defined by Q and $\tau(\cdot,\cdot)$, is *strongly connected*, i.e., for every pair of states q and q' there is a directed path from q to q'. Note that in our definition of PFAs we assumed that the probability associated with each transition (edge in the underlying graph) is non-zero, and hence strong connectivity implies that every state can be reached from every other state with non-zero probability. For simplicity we assume M is *aperiodic*, i.e., that the greatest common divisor of the lengths of the cycles in its underlying graph is 1. These two assumptions ensure us that M is ergodic. Namely, there exists a distribution Π_M on the states such that for every state we may start at, the probability distribution on the state reached after time t as t grows to infinity, converges to Π_M. The probability distribution Π_M is the unique distribution satisfying

$$\Pi_M(q) = \sum_{q' \ s.t. \ \tau(q',\sigma)=q} \Pi_M(q')\gamma(q',\sigma) \ , \tag{4.2}$$

and is named the *stationary distribution* of M. We ask that for every state q in Q, the initial probability of q, $\pi(q)$, be the stationary probability of q, $\Pi_M(q)$. It should be noted that the assumptions above are needed only when learning from a single sample string and not when learning from many sample strings. However, for sake of brevity we make these requirements in both cases.

For any given $L \geq 0$, the subclass of PSAs in which each state is labeled by a string of length at most L is denoted by L-PSA. An example 2-PSA is depicted in Figure 4.1. A special case of these automata is the case in which Q includes *all* strings in Σ^L. An example of such a 2-PSA is depicted in Figure 4.1 as well. These automata can be described as *Markov chains of order L*. The states of the Markov chain are the symbols of the alphabet Σ, and the next state transition probability depends on the last L states (symbols) traversed. Since every L-PSA can be extended to a (possibly much larger) equivalent L-PSA whose states are labeled by all strings in Σ^L, it can always be described as a Markov chain of order L. Alternatively, since the states of an L-PSA might be labeled by only a small subset of $\Sigma^{\leq L}$, and many of the suffixes labeling the states may be much shorter than L, it can be viewed as a Markov chain with *variable order*, or *variable memory*.

Learning Markov chains of order L, i.e., L-PSAs whose states are labeled by *all* Σ^L strings, is straightforward (though it takes time exponential in L). Since the 'identity' of the states (i.e., the strings labeling the states) is known, and since the transition function τ is uniquely defined, learning such automata reduces to approximating the next symbol probability function γ. For the more general case of L-PSAs in which the states are labeled by strings of variable length, the task of an efficient learning algorithm is much more involved since it must reveal the identity of the states as well.

Prediction Suffix Trees

Though we are interested in learning PSAs, we choose as our hypothesis class the class of *prediction suffix trees* (PST) defined in this section. We later show (Section 4.4) that for every PSA there exists an equivalent PST of roughly the same size.

A PST T, over an alphabet Σ, is a tree of degree $|\Sigma|$. Each edge in the tree is labeled by a single symbol in Σ, such that from every internal node there is exactly one edge labeled by each symbol. The nodes of the tree are labeled by pairs (s,γ_s) where s is the string associated with the walk starting from that node and ending in the root of the tree, and $\gamma_s : \Sigma \rightarrow [0,1]$ is the *next symbol probability function* related with s. We require that for every string s labeling a node in the tree, $\sum_{\sigma \in \Sigma} \gamma_s(\sigma) = 1$.

As in the case of PFAs, a PST T generates strings of infinite length, but we consider the probability distributions induced on finite length prefixes of these strings. The probability that T generates a string $r = r_1 r_2 \ldots r_N$ in Σ^N is

$$P_T^N(r) = \Pi_{i=1}^{N} \gamma_{s^{i-1}}(r_i) \ , \tag{4.3}$$

where $s^0 = \mathtt{e}$, and for $1 \leq j \leq N-1$, s^j is the string labeling the deepest node reached by taking the walk corresponding to $r_i r_{i-1} \ldots r_1$ starting at the root of T. For example, using the PST depicted in Figure 4.1, the probability of the string 00101, is $0.5 \times 0.5 \times 0.25 \times 0.5 \times 0.75$, and the labels of the nodes that are used for the prediction are $s^0 = \mathtt{e}$, $s^1 = 0$, $s^2 = 00$, $s^3 = 1$, $s^4 = 10$. In view of this definition, the requirement that every internal node have *exactly* $|\Sigma|$ sons may be loosened, by allowing the omission of nodes labeled by substrings which are generated by the tree with probability 0.

PSTs therefore generate probability distributions in a similar fashion to PSAs. As in the case of PSAs, symbols are generated sequentially and the probability of generating a symbol depends only on the previously generated substring of some bounded length. In both cases there is a simple procedure for determining this substring, as well as for determining the probability distribution on the next symbol conditioned on the substring. However, there are two (related) differences between PSAs and PSTs. The first is that PSAs generate each symbol simply by traversing a single edge from the current state to the next state, while for each symbol generated by a PST, one must walk down from the root of the tree, possibly traversing L edges. This implies that PSAs are more efficient generators. The second difference is that while in PSAs for each substring (state) and symbol, the next state is well defined, in PSTs this property does not necessarily hold. Namely, given the current generating node of a PST, and the next symbol generated, the next node is not necessarily uniquely defined, but might depend on previously generated symbols which are not included in the string associated with the current node. For example, assume we have a tree whose leaves are: `1,00,010,110` (see Figure 4.2). If `1` is the current generating leaf and it generates `0`, then the next generating leaf is either `010` or `110` depending on the symbol generated just prior to `1`.

PSTs, like PSAs, can always be described as Markov chains of (fixed) finite order, but as in the case of PSAs this description might be exponentially large.

We shall sometimes want to discuss only the structure of a PST and ignore its prediction property. In other words, we will be interested only in the string labels of the nodes and not in the values of $\gamma_s(\cdot)$. We refer to such trees as *suffix trees*. We now introduce two more notations. The set of leaves of a suffix tree T is denoted by $\mathcal{L}(T)$, and for a given string s labeling a node v in T, $T(s)$ denotes the subtree rooted at v.

4.3 The Learning Model

The main features of the learning model under which we present our results in this chapter where presented in the introduction and further developed in the previous chapter. Here we describe

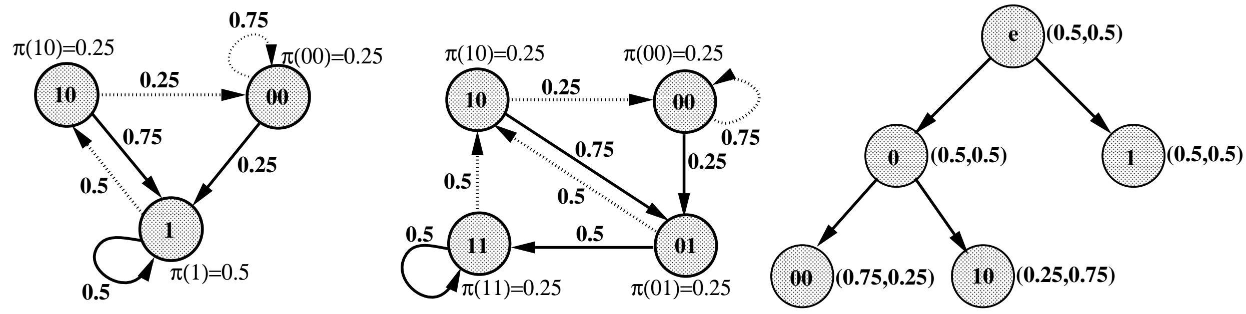


Figure 4.1: Left: A 2-PSA. The strings labeling the states are the suffixes corresponding to them. Bold edges denote transitions with the symbol '1', and dashed edges denote transitions with '0'. The transition probabilities are depicted on the edges. Middle: A 2-PSA whose states are labeled by *all* strings in $\{0,1\}^2$. The strings labeling the states are the last two observed symbols before the state was reached, and hence it can be viewed as a representation of a Markov chain of order 2. Right: A prediction suffix tree. The prediction probabilities of the symbols '0' and '1', respectively, are depicted beside the nodes, in parentheses. The three models are equivalent in the sense that they induce the same probability distribution on strings from $\{0,1\}^\star$.

some additional details which were not presented previously. As before, we measure the goodness of a model using its KL-divergence to the target. As discussed in Section 4.1, the KL-divergence between sources that induce probabilities over arbitrarily long sequences, grows linearly with the length of the sequences. Therefore, the appropriate measure is the KL-divergence *per symbol*. We therefore say that a PST T is an ϵ-good hypothesis with respect to a PSA M, if for every $N > 0$, $\frac{1}{N} D_{KL}[P_M^N || P_T^N] \leq \epsilon$. In addition to the parameters δ and ϵ, we assume that the learning algorithm for PSAs is given the maximum length L of the strings labeling the states of the target PSA M, and an upper bound n on the number of states in M. The second assumption can be easily removed by *searching* for an upper bound. This search is performed by testing the hypotheses the algorithm outputs when it runs with growing values of n. We analyze the following two learning scenarios. In the first scenario the algorithm has access to a source of sample strings of minimal length $L+1$, independently generated by M. In the second scenario it is given only a *single* (long) sample string generated by M. In both cases we require that it output a hypothesis PST $\hat{T}$, which with probability at least $1-\delta$ is an ϵ-good hypothesis with respect to M.

The only drawback to having a PST as our hypothesis instead of a PSA (or more generally a PFA), is that the prediction procedure using a tree is somewhat less efficient (by at most a factor of L). Since no transition function is defined, in order to predict/generate each symbol, we must walk from the root until a leaf is reached. As mentioned earlier, we show in Section 4.4 that every PST can be transformed into an equivalent PFA which is not much larger. This PFA differs from a PSA only in the way it generates the first L symbols. We also show that if the PST has a certain property (defined in Section 4.4), then it can be transformed into an equivalent PSA.

In order to measure the efficiency of the learning algorithm, we separate the case in which the algorithm is given a sample consisting of independently generated sample strings, from the case in which it is given a single sample string. In the first case we say that the learning algorithm is *efficient* if it runs in time polynomial in L, n, $|\Sigma|$, $\frac{1}{\epsilon}$ and $\frac{1}{\delta}$. In order to define efficiency in the latter case we need to take into account an additional property of the model – its mixing or convergence rate. To do this we next discuss another parameter of PSAs (actually, of PFAs in general).

For a given PSA, M, let R_M denote the $n \times n$ stochastic transition matrix defined by $\tau(\cdot,\cdot)$ and

$\gamma(\cdot,\cdot)$ when ignoring the transition labels. That is, if s^i and s^j are states in M and the last symbol in s^j is σ, then $R_M(s^i, s^j)$ is $\gamma(s^i, \sigma)$ if $\tau(s^i, \sigma) = s^j$, and 0 otherwise. Hence, R_M is the transition matrix of an ergodic Markov chain.

Let $\tilde{R}_M$ denote the *time reversal* of R_M. That is,

$$\tilde{R}_M(s^i, s^j) = \frac{\Pi_M(s^j) R_M(s^j, s^i)}{\Pi_M(s^i)} \quad ,$$

where Π_M is the stationary probability vector of R_M as defined in Equation (4.2). Define the *multiplicative reversiblization* U_M of M by $U_M = R_M \tilde{R}_M$. Denote the second largest eigenvalue of U_M by $\lambda_2(U_M)$.

If the learning algorithm receives a single sample string, we allow the length of the string (and hence the running time of the algorithm) to be polynomial not only in L, n, $|\Sigma|$, $\frac{1}{\epsilon}$, and $\frac{1}{\delta}$, but also in $1/(1 - \lambda_2(U_M))$. The rationale behind this is roughly the following. In order to succeed in learning a given PSA, we must observe each state whose stationary probability is non-negligible enough times so that the algorithm can identify that the state is significant, and so that the algorithm can compute (approximately) the next symbol probability function. When given several independently generated sample strings, we can easily bound the size of the sample needed by a polynomial in L, n, $|\Sigma|$, $\frac{1}{\epsilon}$, and $\frac{1}{\delta}$, using Chernoff bounds (see Appendix C). When given one sample string, the given string must be long enough so as to ensure convergence of the probability of visiting a state to the stationary probability. We show that this convergence rate can be bounded using the expansion properties of a weighted graph related to U_M [90] or more generally, using algebraic properties of U_M, namely, its second largest eigenvalue [40].

4.4 On The Relations Between PSTs and PSAs

In this section we show that for every PSA there exists an equivalent PST which is not much larger. This allows us to consider the PST equivalent to our target PSA, whenever it is convenient. We also show that for every PST there exists an equivalent PFA which is not much larger and which is a slight variant of a PSA. Furthermore, if the PST has a certain property, defined subsequently, and denoted by Property$*$, then it can be emulated by a PSA. This equivalent representation is exploited by dynamic programming algorithms as shown later in this chapter for tasks such as correcting corrupted text and part of speech tagging.

Emulation of PSAs by PSTs

Theorem 2 *For every L-PSA, $M = (Q, \Sigma, \tau, \gamma, \pi)$, there exists an equivalent PST T_M, of maximal depth L and at most $L \cdot |Q|$ nodes.*

Proof: Let T_M be the tree whose leaves correspond to the strings in Q (the states of M). For each leaf s, and for every symbol σ, let $\gamma_s(\sigma) = \gamma(s, \sigma)$. This ensures that for every string which is a suffix extension of some leaf in T_M, both M and T_M generate the next symbol with the same probability. The remainder of this proof is hence dedicated to defining the next symbol probability functions for the internal nodes of T_M. These functions must be defined so that T_M generates all strings related to nodes in T_M, with the same probability as M.

For each node s in the tree, let the *weight* of s, denoted by w_s, be defined as follows

$$w_s \stackrel{\text{def}}{=} \sum_{s' \in Q \ \text{s.t.}\ s \in \mathit{Suffix}^*(s')} \pi(s') \tag{4.4}$$

In other words, the weight of a leaf in T_M is the stationary probability of the corresponding state in M; and the weight of an internal node labeled by a string s, equals the sum of the stationary probabilities over all states of which s is a suffix. Note that the weight of any internal node is the sum of the weights of all the leaves in its subtree, and in particular $w_{\mathsf{e}} = 1$. Using the weights of the nodes we assign values to the γ_s's of the internal nodes s in the tree in the following manner. For every symbol σ let

$$\gamma_s(\sigma) = \sum_{s' \in Q \ \text{s.t.}\ s \in \mathit{Suffix}^*(s')} \frac{w_{s'}}{w_s} \gamma(s', \sigma) \ . \tag{4.5}$$

According to the definition of the weights of the nodes, it is clear that for every node s, $\gamma_s(\cdot)$ is in fact a probability function on the next output symbol as required in the definition of prediction suffix trees.

What is the probability that M generates a string s which is a node in T_M (a suffix of a state in Q)? By definition of the transition function of M, for every $s^0 \in Q$, if $s' = \tau(s^0, s)$, then s' must be a suffix extension of s. Thus $P_M(s)$ is the sum over all such s' of the probability of reaching s', when s^0 is chosen according to the initial distribution $\pi(\cdot)$ on the starting states. But if the initial distribution is stationary then at any point the probability of being at state s' is just $\pi(s')$, and

$$P_M(s) \ = \sum_{s' \in Q \ \text{s.t.}\ s \in \mathit{Suffix}^*(s')} \pi(s') \ = \ w_s \ . \tag{4.6}$$

We next prove that $P_{T_M}(s)$ equals w_s as well. We do this by showing that for every $s = s_1 \ldots s_l$ in the tree, where $|s| \geq 1$, $w_s = w_{\mathit{prefix}(s)} \gamma_{\mathit{prefix}(s)}(s_l)$. Since $w_{\mathsf{e}} = 1$, it follows from a simple inductive argument that $P_{T_M}(s) = w_s$.

By our definition of PSAs, $\pi(\cdot)$ is such that for every $s \in Q$, $s = s_1 \ldots s_l$,

$$\pi(s) = \sum_{s' \ \text{s.t.}\ \tau(s', s_l) = s} \pi(s') \gamma(s', s_l) \ . \tag{4.7}$$

Hence, if s is a leaf in T_M then

$$\begin{aligned} w_s \ = \ \pi(s) &\stackrel{(a)}{=} \sum_{s' \in \mathcal{L}(T_M) \ \text{s.t.}\ s \in \mathit{Suffix}^*(s' s_l)} w_{s'} \gamma_{s'}(s_l) \\ &\stackrel{(b)}{=} \sum_{s' \in \mathcal{L}(T_M(\mathit{prefix}(s)))} w_{s'} \gamma_{s'}(s_l) \\ &\stackrel{(c)}{=} w_{\mathit{prefix}(s)} \gamma_{\mathit{prefix}(s)}(s_l) \ , \end{aligned} \tag{4.8}$$

where (a) follows by substituting $w_{s'}$ for $\pi(s')$ and $\gamma_{s'}(s_l)$ for $\gamma(s', s_l)$ in Equation (4.7), and by the definition of $\tau(\cdot, \cdot)$; (b) follows from our definition of the structure of prediction suffix trees;

and (c) follows from our definition of the weights of internal nodes. Hence, if s is a leaf, $w_s = w_{prefix(s)}\gamma_{prefix(s)}(s_l)$ as required.

If s is an internal node then using the result above and Equation (4.5) we get that

$$\begin{aligned} w_s &= \sum_{s' \in \mathcal{L}(T_M(s))} w_{s'} \\ &= \sum_{s' \in \mathcal{L}(T_M(s))} w_{prefix(s')}\gamma_{prefix(s')}(s_l) \\ &= w_{prefix(s)}\gamma_{prefix(s)}(s_l) \ . \end{aligned} \tag{4.9}$$

It is left to show that the resulting tree is not bigger than L times the number of states in M. The number of leaves in T_M equals the number of states in M, i.e. $|\mathcal{L}(T)| = |Q|$. If every internal node in T_M is of full degree (i.e. the probability T_M generates any string labeling a leaf in the tree is strictly greater than 0) then the number of internal nodes is bounded by $|Q|$ and the total number of nodes is at most $2|Q|$. In particular, the above is true when for every state s in M, and every symbol σ, $\gamma(s,\sigma) > 0$. If this is not the case then we can simply bound the total number of nodes by $L \cdot |Q|$. ∎

An example of the construction described in the proof of Theorem 2 is illustrated in Figure 4.1. The PST on the right was constructed based on the PSA on the left, and is equivalent to it. Note that the next symbol probabilities related with the leaves and the internal nodes of the tree are as defined in the proof of the theorem.

Emulation of PSTs by PFAs

Property$*$ For every string s labeling a node in the tree, T,

$$P_T(s) = \sum_{\sigma\in\Sigma} P_T(\sigma s) \ .$$

Before we state our theorem, we observe that Property$*$ implies that for *every* string r,

$$P_T(r) = \sum_{\sigma\in\Sigma} P_T(\sigma r) \tag{4.10}$$

This is true for the following simple reasoning. If r is a node in T, then Equality (4.10) is equivalent to Property$*$. Otherwise let $r = r_1 r_2$, where r_1 is the longest prefix of r which is a leaf in T.

$$P_T(r) = P_T(r_1) \cdot P_T(r_2|r_1) \tag{4.11}$$
$$= \sum_\sigma P_T(\sigma r_1) \cdot P_T(r_2|r_1) \tag{4.12}$$
$$= \sum_\sigma P_T(\sigma r_1) \cdot P_T(r_2|\sigma r_1) \tag{4.13}$$
$$= \sum_\sigma P_T(\sigma r) \ , \tag{4.14}$$

where Equality (4.13) follows from the definition of PST's.

Theorem 3 *For every PST, T, of depth L over Σ there exists an equivalent PFA, M_T, with at most $L \cdot |\mathcal{L}(T)|$ states. Furthermore, if Property$*$ holds for T, then it has an equivalent PSA.*

Proof: In the proof of Theorem 2, we were given a PSA M and we defined the equivalent suffix tree T_M to be the tree whose leaves correspond to the states of the automaton. Thus, given a suffix tree T, the natural dual procedure would be to construct a PSA M_T whose states correspond to the leaves of T. The first problem with this construction is that we might not be able to define the transition function τ on all pairs of states and symbols. That is, there might exist a state s and a symbol σ such that there is no state s' which is a suffix of $s\sigma$. The solution is to extend T to a larger tree T' (of which T is a subtree) such that τ is well defined on the leaves of T'. It can easily be verified that the following is an equivalent requirement on T': for each symbol σ, and for every leaf s in T', $s\sigma$ is either a leaf in the subtree $T'(\sigma)$ rooted at σ, or is a suffix extension of a leaf in $T'(\sigma)$. In this case we shall say that T' *covers* each of its children's subtrees. Viewing this in another way, for every leaf s, the longest *prefix* of s must be either a leaf or an internal node in T'. We thus obtain T' by adding nodes to T until the above property holds.

The next symbol probability functions of the nodes in T' are defined as follows. For every node s in $T \cap T'$ and for every $\sigma \in \Sigma$, let $\gamma'_s(\sigma) = \gamma_s(\sigma)$. For each new node $s' = s'_1 \ldots s'_l$ in $T' - T$, let $\gamma'_{s'}(\sigma) = \gamma_s(\sigma)$, where s is the longest suffix of s' in T (i.e. the deepest ancestor of s' in T). The probability distribution generated by T' is hence equivalent to that generated by T. From Equality (4.10) it directly follows that if Property$*$ holds for T, then it holds for T' as well.

Based on T' we now define $M_T = (Q, \Sigma, \tau, \gamma, \pi)$. If Property$*$ holds for T, then we define M_T as follows. Let the states of M_T be the leaves of T' and let the transition function be defined as usual for PSAs (i.e. for every state s and symbol σ, $\tau(s,\sigma)$ is the unique suffix of $s\sigma$.) Note that the number of states in M_T is at most L times the number of leaves in T, as required. This is true since for each original leaf in the tree T, at most $L-1$ prefixes might be added to T'. For each $s \in Q$ and for every $\sigma \in \Sigma$, let $\gamma(s,\sigma) = \gamma'_s(\sigma)$, and let $\pi(s) = P_T(s)$. It should be noted that M_T is not necessarily ergodic. It follows from this construction that for every string r which is a suffix extension of a leaf in T', and every symbol σ, $P_{M_T}(\sigma|r) = P_T(\sigma|r)$. It remains to show that for every string r which is a node in T', $P_{M_T}(r) = P_{T'}(r)$ $(= P_T(r))$. For a state $s \in Q$, let $P^s_{M_T}(r)$ denote the probability that r is generated assuming we start at state s. Then,

$$\begin{aligned}
P_{M_T}(r) &= \sum_{s \in Q} \pi(s) P^s_{M_T}(r) && (4.15)\\
&= \sum_{s \in Q} \pi(s) P_{M_T}(r|s) && (4.16)\\
&= \sum_{s \in \mathcal{L}(T')} P_{T'}(s) P_{T'}(r|s) && (4.17)\\
&= \sum_{s \in \mathcal{L}(T')} P_{T'}(sr) && (4.18)\\
&= P_{T'}(r) \ , && (4.19)
\end{aligned}$$

where Equality (4.16) follows from the definition of PSAs, Equality (4.17) follows from our definition of $\pi(\cdot)$, and Equality (4.19) follows from a series of applications of Equality (4.10).

If T does not have Property$*$, then we may not be able to define an initial distribution on the states of the PSA M_T such that for every string r which is a node in T', $P_{M_T}(r) = P_{T'}(r)$. We thus define a slight variant of M_T as follows. Let the states of M_T be the leaves of T' and *all their prefixes*, and let $\tau(\cdot,\cdot)$ be defined as follows: for every state s and symbol σ, $\tau(s,\sigma)$ is the *longest* suffix of $s\sigma$. Thus, M_T has the structure of a *prefix* tree combined with a PSA. If we define $\gamma(\cdot,\cdot)$ as above, and let the empty string, $\mathtt{e}$, be the single starting state (i.e., $\pi(\mathtt{e}) = 1$), then, by definition, M_T is equivalent to T.

An illustration of the constructions described above is given in Figure 4.2. ■

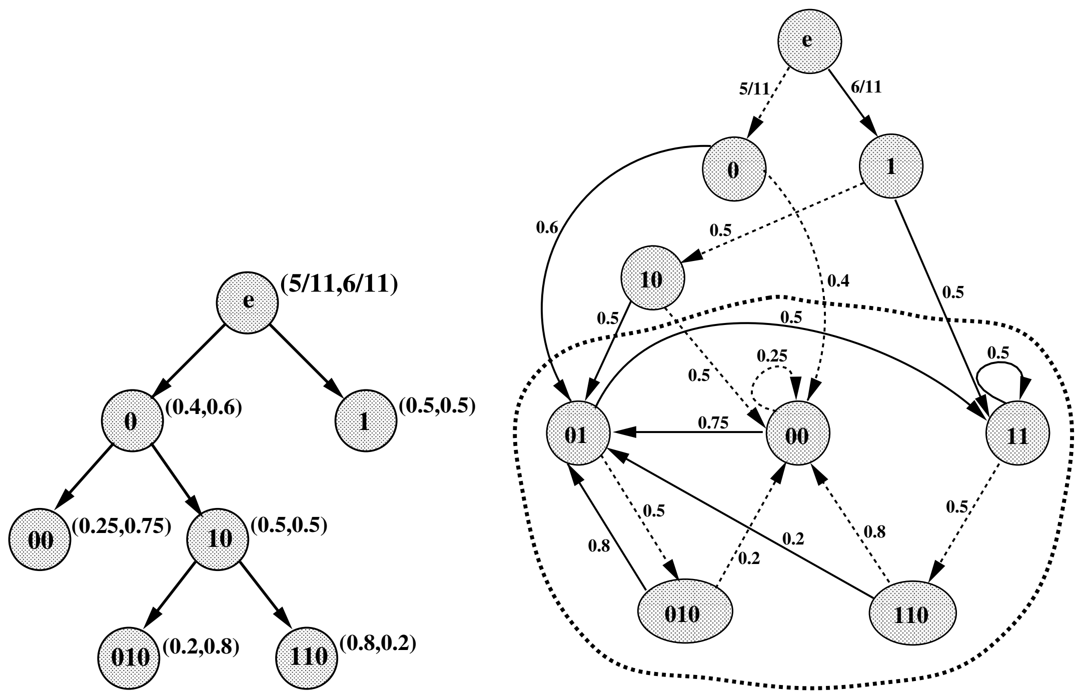


Figure 4.2: Left: A Prediction suffix tree. The prediction probabilities of the symbols '0' and '1', respectively, are depicted beside the nodes, in parentheses. Right: The PFA that is equivalent to the PST on the left. Bold edges denote transitions with the symbol '1' and dashed edges denote transitions with '0'. Since Property$*$ holds for the PST, then it actually has an equivalent PSA which is defined by the circled part of the PFA. The initial probability distribution of this PSA is: $\pi(01) = 3/11$, $\pi(00) = 2/11$, $\pi(11) = 3/11$, $\pi(010) = 3/22$, $\pi(110) = 3/22$. Note that states '11' and '01' in the PSA replaced the node '1' in the tree.

4.5 The Learning Algorithm

We start with an overview of the algorithm. Let $M = (Q, \Sigma, \tau, \gamma, \pi)$ be the target L-PSA we would like to learn, and let $|Q| \le n$. According to Theorem 2, there exists a PST T, of size bounded by $L \cdot |Q|$, which is equivalent to M. We use the sample statistics to define the *empirical probability function*, $\tilde{P}(\cdot)$, and using $\tilde{P}$, we construct a suffix tree, $\bar{T}$, which with high probability is a subtree of T. We define our hypothesis PST, $\hat{T}$, based on $\bar{T}$ and $\tilde{P}$.

The construction of $\bar{T}$ is done as follows. We start with a tree consisting of a single node (labeled by the empty string $\mathtt{e}$) and add nodes which we have reason to believe should be in the tree. A node v labeled by a string s is added as a leaf to $\bar{T}$ if the following holds. The empirical probability of s,

$\tilde{P}(s)$, is non-negligible, and for some symbol σ, the empirical probability of observing σ following s, namely $\tilde{P}(\sigma|s)$, differs substantially from the empirical probability of observing σ following (s), namely $\tilde{P}(\sigma|(\mathsf{s}))$. Note that (s) is the string labeling the parent node of v. Our decision rule for adding v, is thus dependent on the ratio between $\tilde{P}(\sigma|s)$ and $\tilde{P}(\sigma|(\mathsf{s}))$. We add a given node only when this ratio is substantially *greater* than 1. This suffices for our analysis (due to properties of the KL-divergence), and we need not add a node if the ratio is smaller than 1.

Thus, we would like to grow the tree level by level, adding the sons of a given leaf in the tree, only if they exhibit such a behavior in the sample, and stop growing the tree when the above is not true for any leaf. The problem is that the node might belong to the tree even though its next symbol probability function is equivalent to that of its parent node. The leaves of a PST must differ from their parents (or they are redundant) but internal nodes might not have this property. The PST depicted in Figure 4.1 illustrates this phenomena. In this example, $\gamma_0(\cdot) \equiv \gamma_{\mathsf{e}}(\cdot)$, but both $\gamma_{00}(\cdot)$ and $\gamma_{10}(\cdot)$ differ from $\gamma_0(\cdot)$. Therefore, we must continue testing further potential descendants of the leaves in the tree up to depth L.

As mentioned before, we do not test strings which belong to branches whose empirical count in the sample is small. This way we avoid exponential grow-up in the number of strings tested. A similar type of *branch-and-bound* technique (with various bounding criteria) is applied in many algorithms which use trees as data structures (*cf.* [78]). The set of strings tested at each step, denoted by $\bar{S}$, can be viewed as a kind of potential *frontier* of the growing tree $\bar{T}$, which is of bounded size. After the construction of $\bar{T}$ is completed, we define $\hat{T}$ by adding nodes so that all internal nodes have full degree, and defining the next symbol probability function for each node based on $\tilde{P}$. These probability functions are defined so that for every string s in the tree and for every symbol σ, $\gamma_s(\sigma)$ is bounded from below by γ_{min} which is a parameter that is set subsequently. This is done by using a conventional smoothing technique. Such a bound on $\gamma_s(\sigma)$ is needed in order to bound the KL-divergence between the target distribution and the distribution our hypothesis generates.

The above scheme follows a *top-down* approach since we start with a tree consisting of a single root node and a frontier consisting only of its children, and incrementally grow the suffix tree $\bar{T}$ and the frontier $\bar{S}$. Alternatively, a *bottom-up* procedure can be devised. In a bottom-up procedure we start by putting in $\bar{S}$ *all* strings of length at most L which have significant counts, and setting $\bar{T}$ to be the tree whose nodes correspond to the strings in $\bar{S}$. We then trim $\bar{T}$ starting from its leaves and proceeding up the tree by comparing the prediction probabilities of each node to its parent node as done in the top-down procedure. The two schemes are equivalent and yield the same prediction suffix tree. However, we find the incremental top-down approach somewhat more intuitive, and simpler to implement. Moreover, our top-down procedure can be easily adapted to an online setting which is useful in some practical applications.

Let P denote the probability distribution generated by M. We now formally define the *empirical probability function* $\tilde{P}$, based on a given sample generated by M. For a given string s, $\tilde{P}(s)$ is roughly the relative number of times s appears in the sample, and for any symbol σ, $\tilde{P}(\sigma|s)$ is roughly the relative number of times σ appears after s. We give a more precise definition below.

If the sample consists of one sample string r of length m, then for any string s of length at most

L, define $\chi_j(s)$ to be 1 if $r_{j-|s|+1} \ldots r_j = s$ and 0 otherwise. Let

$$\tilde{P}(s) = \frac{1}{m-L} \sum_{j=L}^{m-1} \chi_j(s) \ , \tag{4.20}$$

and for any symbol σ, let

$$\tilde{P}(\sigma|s) = \frac{\sum_{j=L}^{m-1} \chi_{j+1}(s\sigma)}{\sum_{j=L}^{m-1} \chi_j(s)} \ . \tag{4.21}$$

If the sample consists of m' sample strings $r^1, \ldots, r^{m'}$, each of length $\ell \geq L+1$, then for any string s of length at most L, define $\chi_j^i(s)$ to be 1 if $r^i_{j-|s|+1} \ldots r^i_j = s$, and 0 otherwise. Let

$$\tilde{P}(s) = \frac{1}{m'(\ell - L)} \sum_{i=1}^{m'} \sum_{j=L}^{\ell-1} \chi_j^i(s) \ , \tag{4.22}$$

and for any symbol σ, let

$$\tilde{P}(\sigma|s) = \frac{\sum_{i=1}^{m'} \sum_{j=L}^{l-1} \chi_{j+1}(s\sigma)}{\sum_{i=1}^{m'} \sum_{j=L}^{l-1} \chi_j(s)} \ . \tag{4.23}$$

For simplicity we assume that all the sample strings have the same length and that this length is polynomial in n, L, and Σ. The case in which the sample strings are of different lengths can be treated similarly, and if the strings are too long then we can ignore parts of them.

In the course of the algorithm and in its analysis we refer to several parameters which are all simple functions of ϵ, n, L and $|\Sigma|$, and are set as follows:

$$\begin{aligned}
\epsilon_2 &= \frac{\epsilon}{48L} \ , \\
\gamma_{min} &= \frac{\epsilon_2}{|\Sigma|} = \frac{\epsilon}{48L|\Sigma|} \ , \\
\epsilon_0 &= \frac{\epsilon}{2nL\log(1/\gamma_{min})} = \frac{\epsilon}{2nL\log(48L|\Sigma|/\epsilon)} \ , \\
\epsilon_1 &= \frac{\epsilon_2}{8n\epsilon_0\gamma_{min}} = \frac{L\log(48L|\Sigma|/\epsilon)|\Sigma|}{4\epsilon} \ .
\end{aligned}$$

The size of the sample is set in the analysis of the algorithm.

A pseudo code describing the learning algorithm is given in Figure 4.3 and an illustrative run of the algorithm is depicted in Figure 4.4.

Algorithm Learn-PSA

1. Initialize $\bar{T}$ and $\bar{S}$: let $\bar{T}$ consist of a single root node (corresponding to e), and let
$$\bar{S} \leftarrow \{\sigma \mid \sigma \in \Sigma \text{ and } \tilde{P}(\sigma) \geq (1-\epsilon_1)\epsilon_0\} \ .$$
2. While $\bar{S} \neq \emptyset$, pick any $s \in \bar{S}$ and do:
   (a) Remove s from $\bar{S}$;
   (b) If there exists a symbol $\sigma \in \Sigma$ such that
   $$\tilde{P}(\sigma|s) \geq (1+\epsilon_2)\gamma_{min} \ \text{ and } \ \tilde{P}(\sigma|s)/\tilde{P}(\sigma|(\mathsf{s})) > 1 + 3\epsilon_2 \ ,$$
   then add to $\bar{T}$ the node corresponding to s and all the nodes on the path from the deepest node in $\bar{T}$ that is a suffix of s, to S;
   (c) If $|s| < L$ then for every $\sigma' \in \Sigma$, if
   $$\tilde{P}(\sigma' \cdot s) \geq (1-\epsilon_1)\epsilon_0 \ ,$$
   then add $\sigma' \cdot s$ to $\bar{S}$.
3. Initialize $\hat{T}$ to be $\bar{T}$.
4. Extend $\hat{T}$ by adding all missing sons of internal nodes.
5. For each s labeling a node in $\hat{T}$, let
$$\hat{\gamma}_s(\sigma) = \tilde{P}(\sigma|s')(1 - |\Sigma|\gamma_{min}) + \gamma_{min} \ ,$$
where s' is the longest suffix of s in $\bar{T}$.

Figure 4.3: Algorithm **Learn-PSA**

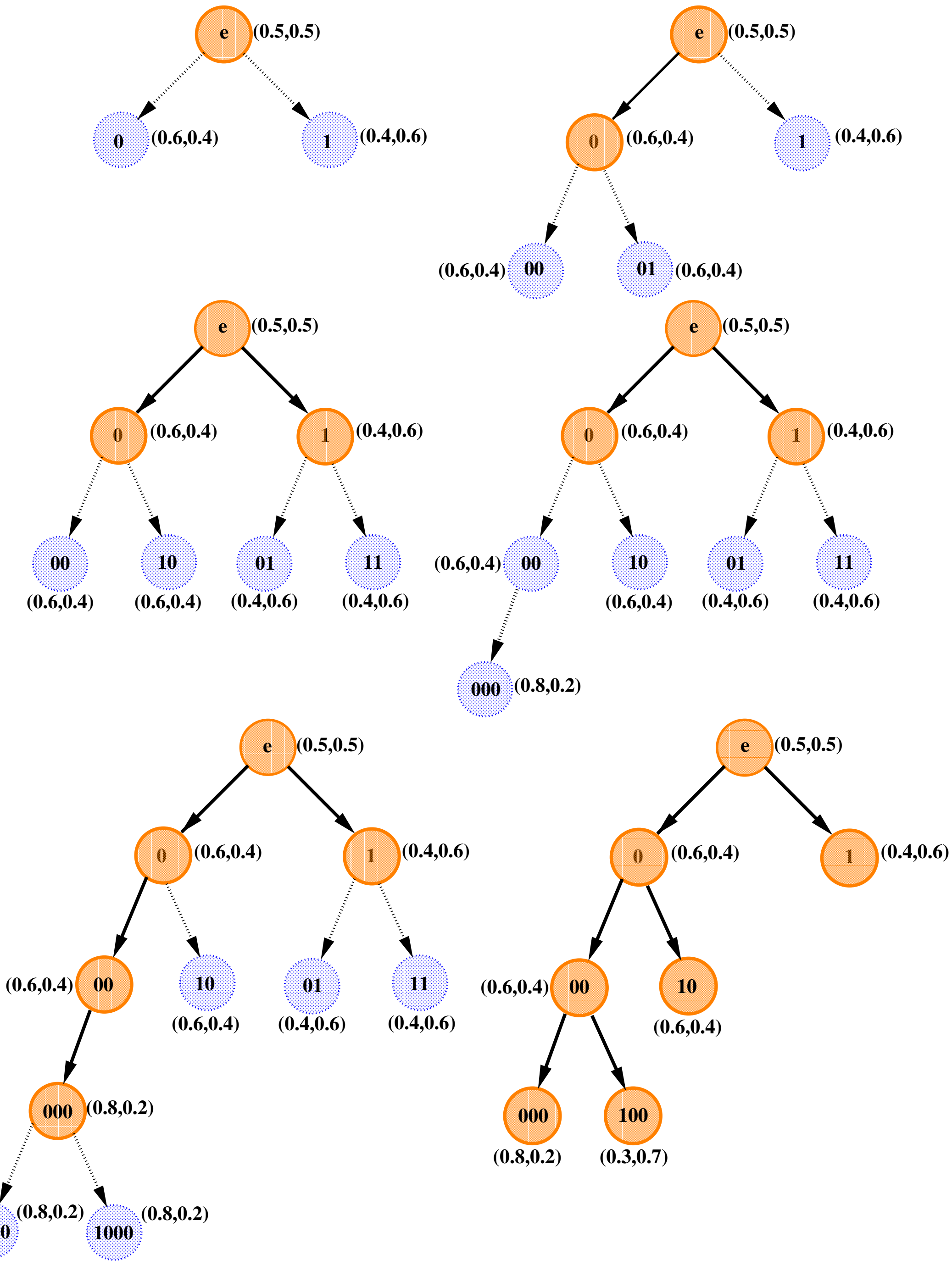


Figure 4.4: An illustrative run of the learning algorithm. The prediction suffix trees created along the run of the algorithm are depicted from left to right and top to bottom. At each stage of the run, the nodes from $\bar{T}$ are plotted in dark grey while the nodes from $\bar{S}$ are plotted in light grey. The alphabet is binary and the predictions of the next bit are depicted in parenthesis beside each node. The final tree is plotted on the bottom right part and was built by adding to $\bar{T}$ (bottom left) all missing children. Note that the node labeled by 100 was added to the final tree but is not part of any of the intermediate trees. This can happen when the probability of the string 100 is small.

4.6 Analysis of the Learning Algorithm

In this section we state and prove our main theorem regarding the correctness and efficiency of the learning algorithm *Learn-PSA*, described in Section 4.5.

Theorem 4 *For every target PSA M, and for every given confidence parameter $0 < \delta < 1$, and approximation parameter $0 < \epsilon < 1$, Algorithm* Learn-PSA *outputs a hypothesis PST, $\hat{T}$, such that with probability at least $1 - \delta$:*

1. *$\hat{T}$ is an ϵ-good hypothesis with respect to M.*
2. *The number of nodes in $\hat{T}$ is at most $|\Sigma| \cdot L$ times the number of states in M.*

If the algorithm has access to a source of independently generated sample strings, then its running time is polynomial in $L, n, |\Sigma|, \frac{1}{\epsilon}$ and $\frac{1}{\delta}$. If the algorithm has access to only one sample string, then its running time is polynomial in the same parameters and in $1/(1 - \lambda_2(U_M))$.

In order to prove the theorem above we first show that with probability $1 - \delta$, a large enough sample generated according to M is *typical* to M, where *typical* is defined subsequently. We then assume that our algorithm in fact receives a typical sample and prove Theorem 4 based on this assumption. Roughly speaking, a sample is typical if for every substring generated with non-negligible probability by M, the empirical counts of this substring and of the next symbol given this substring, are not far from the corresponding probabilities defined by M.

Definition 4.6.1 *A sample generated according to M is* **typical** *if for every string $s \in \Sigma^{\leq L}$ the following two properties hold:*

1. *If $s \in Q$ then $|\tilde{P}(s) - \pi(s)| \leq \epsilon_1 \epsilon_0$;*
2. *If $\tilde{P}(s) \geq (1 - \epsilon_1)\epsilon_0$ then for every $\sigma \in \Sigma$, $|\tilde{P}(\sigma|s) - P(\sigma|s)| \leq \epsilon_2 \gamma_{min}$;*

Where ϵ_0, ϵ_1, ϵ_2, and γ_{min} were defined in Section 4.5.

Lemma 4.6.1

1. *There exists a polynomial m_0' in L, n, $|\Sigma|$, $\frac{1}{\epsilon}$, and $\frac{1}{\delta}$, such that the probability that a sample of $m' \geq m_0'(L, n, |\Sigma|, \frac{1}{\epsilon}, \frac{1}{\delta})$ strings each of length at least $L + 1$ generated according to M is typical is at least $1 - \delta$.*
2. *There exists a polynomial m_0 in L, n, $|\Sigma|$, $\frac{1}{\epsilon}$, $\frac{1}{\delta}$, and $1/(1-\lambda_2(U_M))$, such that the probability that a single sample string of length $m \geq m_0(L, n, |\Sigma|, \frac{1}{\epsilon}, \frac{1}{\delta}, 1/(1 - \lambda_2(U_M)))$ generated according to M is typical is at least $1 - \delta$.*

The proof of Lemma 4.6.1 is provided in Appendix B.

Let T be the PST equivalent to the target PSA M, as defined in Theorem 2. In the next lemma we prove two claims. In the first claim we show that the prediction properties of our hypothesis

PST $\hat{T}$, and of T, are similar. We use this in the proof of the first claim in Theorem 4, when showing that the KL-divergence per symbol between $\hat{T}$ and M is small. In the second claim we give a bound on the size of $\hat{T}$ in terms of T, which implies a similar relation between $\hat{T}$ and M (second claim in Theorem 4).

Lemma 4.6.2 *If Learn-PSA is given a typical sample then:*

1. *For every string s in T, if $P(s) \geq \epsilon_0$ then $\dfrac{\gamma_s(\sigma)}{\hat{\gamma}_{s'}(\sigma)} \leq 1 + \epsilon/2$, where s' is the longest suffix of s corresponding to a node in $\hat{T}$.*
2. *$|\hat{T}| \leq (|\Sigma| - 1) \cdot |T|$.*

Proof: (Sketch, the complete proofs of both claims are provided in Appendix B.)

In order to prove the first claim, we argue that if the sample is typical, then there cannot exist such strings s and s' which falsify the claim. We prove this by assuming that there exists such a pair, and reaching contradiction. Based on our setting of the parameters ϵ_2 and γ_{min}, we show that for such a pair, s and s', the ratio between $\gamma_s(\sigma)$ and $\gamma_{s'}(\sigma)$ must be bounded from below by $1 + \epsilon/4$. If $s = s'$, then we have already reached a contradiction. If $s \neq s'$, then we can show that the algorithm must add some longer suffix of s to $\bar{T}$, contradicting the assumption that s' is the longest suffix of s corresponding to a node in $\hat{T}$. In order to bound the size of $\hat{T}$, we show that $\bar{T}$ is a subtree of T. This suffices to prove the second claim, since when transforming $\bar{T}$ into $\hat{T}$, we add at most all $|\Sigma| - 1$ siblings of every node in $\bar{T}$. We prove that $\bar{T}$ is a subtree of T, by arguing that in its construction, we did not add any string which does not correspond to a node in T. This follows from the decision rule according to which we add nodes to $\bar{T}$. $\Box$

Proof of Theorem 4: According to Lemma 4.6.1, with probability at least $1 - \delta$ our algorithm receives a typical sample. Thus according to the second claim in Lemma 4.6.2, $|\hat{T}| \leq (|\Sigma| - 1) \cdot |T|$ and since $|T| \leq L \cdot |Q|$, then $|\hat{T}| \leq |\Sigma| \cdot L \cdot |Q|$ and the second claim in the theorem is valid.

Let $r = r_1 r_2 \ldots r_N$, where $r_i \in \Sigma$, and for any prefix $r^{(i)}$ of r, where $r^{(i)} = r_1 \ldots r_i$, let $s[r^{(i)}]$ and $\hat{s}[r^{(i)}]$ denote the strings corresponding to the deepest nodes reached upon taking the walk $r_i \ldots r_1$ on T and $\hat{T}$ respectively. In particular, $s[r^{(0)}] = \hat{s}[r^{(0)}] = \mathsf{e}$. Let $\hat{P}$ denote the probability distribution generated by $\hat{T}$. Then

$$\text{a}\frac{1}{N} \sum_{r \in \Sigma^N} P(r) \log \frac{P(r)}{\hat{P}(r)} = \tag{4.24}$$

$$= \frac{1}{N} \sum_{r \in \Sigma^N} P(r) \cdot \log \frac{\prod_{i=1}^{N} \gamma_{s[r^{(i-1)}]}(r_i)}{\prod_{i=1}^{N} \hat{\gamma}_{\hat{s}[r^{(i-1)}]}(r_i)} \text{b} \tag{4.25}$$

$$= \frac{1}{N} \sum_{r \in \Sigma^N} P(r) \cdot \sum_{i=1}^{N} \log \frac{\gamma_{s[r^{(i-1)}]}(r_i)}{\hat{\gamma}_{\hat{s}[r^{(i-1)}]}(r_i)} \text{c} \tag{4.26}$$

$$= \frac{1}{N} \sum_{i=1}^{N} [\sum_{\substack{r \in \Sigma^N \text{ s.t.} \\ P(s[r^{(i-1)}]) < \epsilon_0}} P(r) \cdot \log \frac{\gamma_{s[r^{(i-1)}]}(r_i)}{\hat{\gamma}_{\hat{s}[r^{(i-1)}]}(r_i)}$$

$$+ \sum_{\substack{r\in\Sigma^N \text{ s.t.} \\ P(s[r^{(i-1)}])\geq\epsilon_0}} P(r)\cdot \log \frac{\gamma_{s[r^{(i-1)}]}(r_i)}{\hat{\gamma}_{\hat{s}[r^{(i-1)}]}(r_i)} \quad] \ .\mathrm{d} \tag{4.27}$$

For every $1 \leq i \leq N$, the first term in the parenthesis in Equation (4.27) can be bounded as follows. For each string r, the worst possible ratio between $\gamma_{s[r^{(i-1)}]}(r_i)$ and $\hat{\gamma}_{\hat{s}[r^{(i-1)}]}(r_i)$, is $1/\gamma_{min}$. The total weight of all strings in the first term equals the total weight of all the nodes in T whose weight is at most ϵ_0, which is at most $nL\epsilon_0$. The first term is thus bounded by $nL\epsilon_0 \log(1/\gamma_{min})$. Based on Lemma 4.6.2, the ratio between $\gamma_{s[r^{(i-1)}]}(r_i)$ and $\hat{\gamma}_{\hat{s}[r^{(i-1)}]}(r_i)$ for every string r in the second term in the parenthesis, is at most $1 + \epsilon/2$. Since the total weight of all these strings is bounded by 1, the second term is bounded by $\log(1 + \epsilon/2)$. Combining the above with the value of ϵ_0 (that was set in Section 4.5 to be $\epsilon/\left(2nL\log(1/\gamma_{min})\right)$), we get that,

$$\frac{1}{N} D_{KL}[P^N||\hat{P}^N] \ \leq \ \frac{1}{N}\cdot N\left[n\, L\, \epsilon_0 \log\frac{1}{\gamma_{min}} \ + \log(1+\epsilon/2)\right] \ \leq \ \epsilon \ . \tag{4.28}$$

Using a straightforward implementation of the algorithm, we can get a (very rough) upper bound on the running time of the algorithm which is of the order of the square of the size of the sample times L. In this implementation, each time we add a string s to $\bar{S}$ or to $\bar{T}$, we perform a complete pass over the given sample to count the number of occurrences of s in the sample and its next symbol statistics. According to Lemma 4.6.1, this bound is polynomial in the relevant parameters, as required in the theorem statement. Using the following more time-efficient, but less space-efficient implementation, we can bound the running time of the algorithm by the size of the sample times L. For each string in $\bar{S}$, and each leaf in $\bar{T}$ we keep a set of pointers to all the occurrences of the string in the sample. For such a string s, if we want to test which of its extensions, σs should we add to $\bar{S}$ or to $\bar{T}$, we need only consider all occurrences of s in the sample (and then distribute them accordingly among the strings added). For each symbol in the sample there is a single pointer, and each pointer corresponds to a single string of length i for every $1 \leq i \leq L$. Thus the running time of the algorithm is of the order of the size of the sample times L. ■

4.7 Correcting Corrupted Text

In many machine recognition systems such as speech or handwriting recognizers, the recognition scheme is divided into two almost independent stages. In the first stage a low-level model is used to perform a (stochastic) mapping from the observed data (e.g., the acoustic signal in speech recognition applications) into a high level alphabet. If the mapping is accurate then we get a correct sequence over the high level alphabet, which we assume belongs to a corresponding high level language. However, it is very common that errors in the mapping occur, and sequences in the high level language are corrupted. Much of the effort in building recognition systems is devoted to correct the corrupted sequences. In particular, in many optical and handwriting character recognition systems, the last stage employs natural-language analysis techniques to correct the corrupted sequences. This can be done after a good model of the high level language is learned from uncorrupted examples of sequences in the language. We now show how to use PSAs in order to perform such a task.

We have performed experiments with different texts such as, Brown corpus, the Gutenberg Bible, moderns stories (e.g., Milton's Paradise Lost), and the bible. We have also carried out evaluations on large data sets such as the ARPA North-American Business News (NAB) corpus. We now describe the results when the learning algorithm was applied to the bible. The alphabet we used consists of the english letters and the blank character. We removed Jenesis and it served as a test set. The algorithm was applied to the rest of the books with $L = 30$, and the accuracy parameters (ϵ_i) were of order $O(\sqrt{N})$, where N is the length of the training data. We used a slightly modified criterion to build the suffix tree $\bar{T}$ from $\bar{S}$: we compared the KL-divergence between the probability function of a node and the probability functions of its predecessors (instead of comparing ratios of probabilities). The resulting PST has less than 3000 nodes. This PST was transformed into a PSA in order to apply an efficient text correction scheme which is described subsequently. The final automaton constitutes both of states that are of length 2, like `'qu'` and `'xe'`, and of states which are 8 and 9 symbols long, like `'shall be'` and `'there was'`. This indicates that the algorithm really captures the notion of variable memory that is needed in order to have accurate predictions. Building a Markov chain of order L in this case is clearly not practical since it requires $|\Sigma|^L = 27^9 = 7625597484987$ states!

Let $\bar{r} = (r_1, r_2, \ldots, r_t)$ be the observed (corrupted) text. If an estimation of the corrupting noise probability is given, then we can calculate for each state sequence $\bar{q} = (q_0, q_1, q_2, \ldots, q_t)$, $q_i \in Q$, the probability that $\bar{r}$ was created by a walk over the PSA which constitutes of the states $\bar{q}$. For $0 \le i \le t$, let X_i be a random variable over Q, where $X_i = q$ denotes the event that the ith state passed was q. For $1 \le i \le t$ let Y_i be a random variable over Σ, where $Y_i = \sigma$ denotes the event that the ith symbol observed was σ. For $\bar{q} \in Q^{t+1}$, let $\bar{X} = \bar{q}$ denote the joint event that $X_i = q_i$ for every $0 \le i \le t$, and for $\bar{r} \in \Sigma^t$, let $\bar{Y} = \bar{r}$ denote the joint event that $Y_i = r_i$ for every $1 \le i \le t$. If we assume that the corrupting noise is i.i.d and is independent of the states that constitute the walk, then the most likely state sequence, $\bar{q}_{ML}$, is

$$
\begin{aligned}
\bar{q}_{ML} &= \arg\max_{\bar{q} \in Q^{t+1}} P\left(\bar{X} = \bar{q} | \bar{Y} = \bar{r}\right) = \arg\max_{\bar{q} \in Q^{t+1}} P\left(\bar{Y} = \bar{r} | \bar{X} = \bar{q}\right) P(\bar{X} = \bar{q}) && (4.29) \\
&= \arg\max_{\bar{q} \in Q^{t+1}} \left\{ \left(\prod_{i=1}^{t} P\left(Y_i = r_i | \bar{X} = \bar{q}\right) \right) \times \right. \\
&\qquad \left. \left(\pi(q_0) \prod_{i=1}^{t} P\left(X_i = q_i | X_{i-1} = q_{i-1}\right) \right) \right\} && (4.30) \\
&= \arg\max_{\bar{q} \in Q^{t}} \left\{ \sum_{i=1}^{t} \log\left(P\left(Y_i = r_i | X_i = q_i\right) + \log(\pi(q_0)) + \right. \right. \\
&\qquad \left. \sum_{i=1}^{t} \log\left(P\left(X_i = q_i | X_{i-1} = q_{i-1}\right)\right) \right\} \quad , && (4.31)
\end{aligned}
$$

where for deriving the last Equality (4.31) we used the monotonicity of the log function and the fact that the corruption noise is independent of the states. Let the string labeling q_i be $s_1, \ldots, s_l$. Then $P(Y_i = r_i | X_i = q_i)$ is the probability that r_i is an uncorrupted symbol if $r_i = s_l$, and is the probability that the noise process flipped s_l to be r_i otherwise. Note that the sum (4.31) can be computed efficiently in a recursive manner. Moreover, the maximization of Equation (4.29) can be performed efficiently by using a *dynamic programming* (DP) scheme [13]. This scheme requires

$O(|Q| \times t)$ operations. If $|Q|$ is large, then approximation schemes to the optimal DP, such as the *stack decoding* algorithm [68] can be employed. Using similar methods it is also possible to correct errors when insertions and deletions of symbols occur as well.

We tested the algorithm by taking a text from Jenesis and corrupting it in two ways. First, we altered every letter (including blanks) with probability 0.2. In the second test we altered every letter with probability 0.1 and we also changed each blank character, in order to test whether the resulting model is powerful enough to cope with non-uniform noise. The result of the correction algorithm for both cases as well as the original and corrupted texts are depicted in Figure 4.5.

Original Text:
and god called the dry land earth and the gathering together of the waters called he seas and god saw that it was good and god said let the earth bring forth grass the herb yielding seed and the fruit tree yielding fruit after his kind

Corrupted text (1):
and god cavsed the drxjland earth ibd shg gathervng together oj the waters cflled re seas aed god saw thctpit was good ann god said let tae earth bring forth gjasb tse hemb yielpinl peed and thesfruit tree sielxing fzuitnafter his kind

Corrected text (1):
and god caused the dry land earth and she gathering together of the waters called he sees and god saw that it was good and god said let the earth bring forth grass the memb yielding peed and the fruit tree fielding fruit after his kind

Corrupted text (2):
andhgodpcilledjthesdryjlandbeasthcandmthelgatceringhlogetherjfytrezaatersoczlled xherseasaknddgodbsawwthathitqwasoqoohanwzgodcsaidhletdtheuejrthriringmforth hbgrasstthexherbyieldingzseedmazdctcybfruitttreeayieldinglfruztbafherihiskind

Corrected text (2):
and god called the dry land earth and the gathering together of the altars called he seasaked god saw that it was took and god said let the earthriring forth grass the herb yielding seed and thy fruit treescielding fruit after his kind

Figure 4.5: Correcting corrupted text (example taken from the bible).

We would like to point out that states labeled by strings of length greater than 6 appear when the algorithm is trained on texts of length much shorter than the bible. Hence, the notion of variable memory that is needed in order to have accurate predictions is a general phenomena in natural texts. Moreover, the results presented in Figure 4.5 are by no means exotic. We obtained similar results, although somewhat less impressive, for much smaller data sets. For instance, see Figure 4.6 for results of correcting a text corrupted by the same noise (uniform case) using a PSA trained on "Alice in the Wonderland".

Original Text:
alice opened the door and found that it led into a small passage not much larger than a rat hole she knelt down and looked along the passage into the loveliest garden you ever saw how she longed to get out of that dark hall and wander about among those beds of bright flowers and those cool fountains but she could not even get her head through the doorway and even if my head would go through thought poor alice it would be of very little use without my shoulders

Corrupted Text:
alice opsneg fhe daor and fpund erat id led into umsnkll passabe not mxch lcrger rhjn fvrac holeesheskneIt down and looked alotg tve passagh into thc ltvbliest gardemx-thuriverhsfw how snn longey towget out of that ark hall and wgnderaaboux amoig ghosewbeds of bridht faowers nnd xhhsefcoolrfeuntains but shh cozld not fjen gk-tnherqaevx whrougx kte dootwayzatd evzo if my heaf wouwd uo throqgh tzought poor alice it wjcwd bq of vlry litkle ust withaut my shoulberu

Corrected Text:
alice opened the door and found that it led into his all passage not much larger then forat hole she knelt down and looked along the passigh into the stabliest garded thuriver she how and longey to get out of that dark hall and wonder about along those beds of bright frowers and those cool feentains but she could not feen got her neve through the doo way and ever if my head would to through thought poor alice it would be of very little use without my shoulders

Figure 4.6: Cleaning corrupted text (example taken from "Alice in the Wonderland").

We compared the performance of the PSA we constructed to the performance of Markov chains of order $0-3$. The performance is measured by the negative log-likelihood obtained by the various models on the (uncorrupted) test data, normalized per observation symbol. The negative log-likelihood measures the amount of 'statistical surprise' induced by the model. The results are summarized in Table 4.1. The first four entries correspond to the Markov chains of order $0-3$, and the last entry corresponds to the PSA. The order of the PSA is defined to be $\log_{|\Sigma|}(|Q|)$. These empirical results imply that using a PSA of reasonable size, we get a better model of the data than if we had used a much larger full order Markov chain.

	Fixed Order Markov				PSA
Model Order	0	1	2	3	1.84
Number of States	1	27	729	19683	432
Negative Log-Likelihood	0.853	0.681	0.560	0.555	0.456

Table 4.1: Comparison of full order Markov chains versus a PSA (a Markov model with variable memory).

4.8 Building A Simple Model for E.*coli* DNA

The DNA alphabet is composed of four nucleotides denoted by: `A,C,T,G`. DNA strands are composed of sequences of protein coding genes and fillers between those regions named intergenic regions. Locating the coding genes is necessary, prior to any further DNA analysis. Using manually segmented data of E. *coli* [114] we built two different PSAs, one for the coding regions and one for the intergenic regions. We disregarded the internal (triplet) structure of the coding genes and the existence of start and stop codons at the beginning and the end of those regions. The models were constructed based on 250 different DNA strands from each type, their lengths ranging from 20 bases to several thousands. The PSAs built are rather small compared to the HMM model described in [76]: the PSA that models the coding regions has 65 states and the PSA that models the intergenic regions has 81 states.

We tested the performance of the models by calculating the log-likelihood of the two models obtained on test data drawn from intergenic regions. In 90% of the cases the log-likelihood obtained by the PSA trained on intergenic regions was higher than the log-likelihood of the PSA trained on the coding regions. Misclassifications (when the log-likelihood obtained by the second model was higher) occurred only for sequences shorter than 100 bases. Moreover, the log-likelihood difference between the models scales linearly with the sequence length where the slope is close to the KL-divergence between the Markov models (which can be computed from the parameters of the two PSAs), as depicted in Figure 4.7. The main advantage of PSA models is in their simplicity. Moreover, the log-likelihood of a set of substrings of a given strand can be computed in time linear in the number of substrings. The latter property combined with the results mentioned above indicate that the PSA model might be used when performing tasks such as DNA gene locating. However, we should stress that we have done only a preliminary step in this direction and the results obtained in [76] as part of a complete parsing system are better.

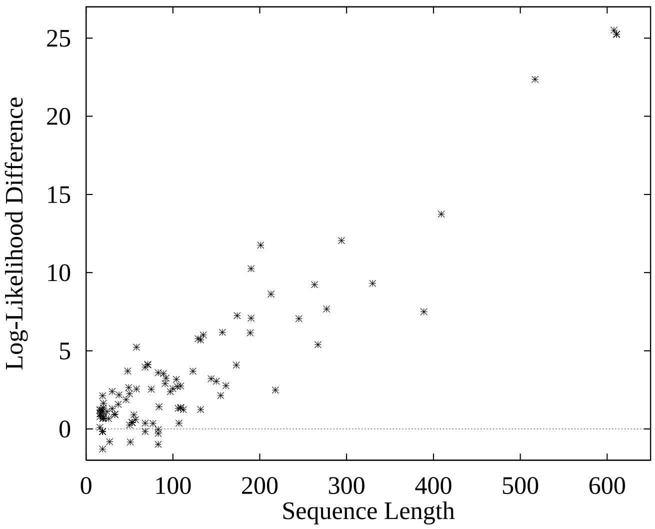


Figure 4.7: The difference between the log-likelihood induced by a PSA trained on data taken from intergenic regions and a PSA trained on data taken from coding regions. The test data was taken from intergenic regions. In 90% of the cases the likelihood of the first PSA was higher.

4.9 A Part-Of-Speech Tagging System

In this section we present a new approach to disambiguating syntactically ambiguous words in context, based on a Markov model with variable memory. Our approach has several advantages over existing methods: It is easy to implement; Classification of new tags using our system is simple and efficient; The results achieved, using simplified assumptions for the static tag probabilities, are encouraging. In a test of out system tagger on the Brown corpus, 95.81% of tokens are correctly classified.

4.9.1 Problem Description

Many words in English have several parts of speech (*POS*). For example "book" is used as a noun in "She read a book." and as a verb in "She didn't book a trip." Part-of-speech tagging is the problem of determining the syntactic part of speech of an occurrence of a word in context. In any given English text, most tokens are syntactically ambiguous since most of the high-frequency English words have several parts of speech. Therefore, a correct syntactic classification of words in context is important for most syntactic and other higher-level processing of natural language text such as the noun phrase identification scheme presented in the previous chapter.

Two probabilistic models have been widely used for part-of-speech tagging: fixed order Markov models and Hidden Markov models. Examples for such POS tagging systems are given in [27, 23]. When a fixed order Markov model is employed for tagging, a short memory (small order) is typically used, since the number of possible combinations grows exponentially. For example, assuming there are 184 different tags, as in the Brown corpus, there are $184^3 = 6,229,504$ different order 3 combinations of tags (of course not all of these will actually occur as shown in [140]). Because of the large number of parameters higher-order fixed length models are hard to estimate and several heuristics have been devised (see [19] for a rule-based approach to incorporating higher-order information). In a *Hidden Markov Model* (HMM) [77, 70], a different state is defined for each POS tag and the transition and output probabilities are estimated using the EM [33] algorithm, which as discussed previously, guarantees convergence to a local minimum [142]. The advantage of an HMM is that its parameters can be estimated using untagged text. On the other hand, the estimation procedure is time consuming, and a fixed model (topology) is assumed. Another disadvantage is due to the local convergence properties of the EM algorithm. The solution obtained depends on the initial setting of the model's parameters, and different solutions are obtained for different parameter initialization schemes. This phenomenon discourages linguistic analysis based on the output of the model.

4.9.2 Using a PSA for Part-Of-Speech Tagging

The heart of our system is a PSA built from the tagging information while ignoring the actual words. Thus, the PSA approximates the distribution of sequences of the part-of-speech tags. On top of the PSA we added a simple probabilistic model that estimates the probability of observing a word w when the corresponding POS tag is t. We estimate the syntactic information, that is the probability of a specific word belonging to a tag class, using a modified *maximum likelihood* estimation scheme from the individual word counts. The whole structure of the system, for two states, is depicted in Figure 4.8. In the figure, $s^1 = t_1, t_2, \ldots, t_n$ and $s^2 = t_i, \ldots, t_{n+1}$ $(i \geq 1)$ are the strings (sequence of POS tags) labeling the nodes. Each transition of the PSA is associated also with a probability distribution vector, denoted by $P(w|t_{n+1})$, which, as described above, is the probability that the word w belongs to the tag class t_{n+1}. $P(t_{n+1}|s^1) = P(t_{n+1}|t_1, t_2, \ldots, t_n)$ is therefore the transition probability from state s^1 to state s^2.

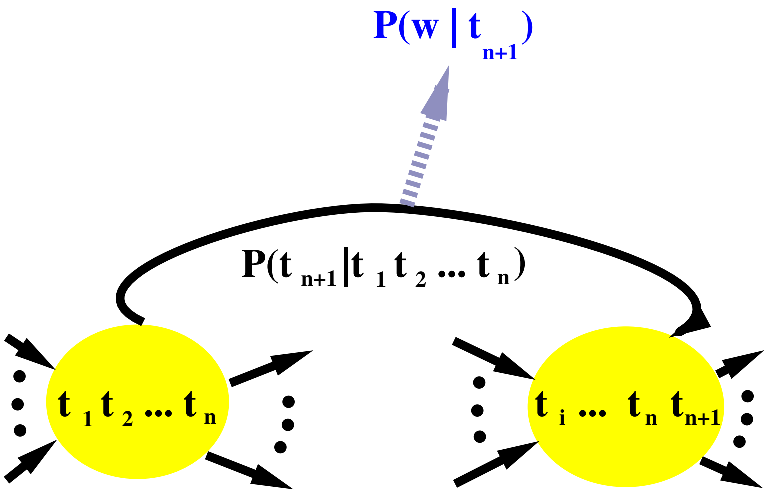


Figure 4.8: The structure of a PSA based Part-Of-Speech taggings system.

When tagging an unlabeled sequence of words $w_{1,n}$, we want to find the tag sequence $t_{1,n}$ that is most likely for $w_{1,n}$. We can maximize the joint probability of $w_{1,n}$ and $t_{1,n}$ to find this sequence:[1]

$$\begin{aligned}
\mathcal{T}(w_{1,n}) &= \arg\max_{t_{1,n}} P(t_{1,n}|w_{1,n}) \\
&= \arg\max_{t_{1,n}} \frac{P(t_{1,n}, w_{1,n})}{P(w_{1,n})} \\
&= \arg\max_{t_{1,n}} P(t_{1,n}, w_{1,n}) \ .
\end{aligned}$$

The joint probability $P(t_{1,n}, w_{1,n})$ can be expressed as a product of conditional probabilities as follows,

$$\begin{aligned}
&P(t_{1,n}, w_{1,n}) = \\
&\quad P(t_1)P(w_1|t_1)P(t_2|t_1, w_1)P(w_2|t_{1,2}, w_1) \\
&\quad \ldots P(t_n|t_{1,n-1}, w_{1,n-1})P(w_n|t_{1,n}, w_{1,n-1}) \\
&\quad = \prod_{i=1}^{n} P(t_i|t_{1,i-1}, w_{1,i-1})P(w_i|t_{1,i}, w_{1,i-1}) \ .
\end{aligned}$$

With the simplifying assumption that the probability of a tag only depends on previous tags and that the probability of a word only depends on its tags, we get that

$$P(t_{1,n}, w_{1,n}) = \prod_{i=1}^{n} P(t_i|t_{1,i-1})P(w_i|t_i) \ .$$

[1] Part of the following notation is adapted from [23].

Since we use a PSA to approximate the distribution of sequences of part-of-speech tags, $P(t_i|t_{1,i-1})$ equals to $\gamma(q^{i-1}, t_i)$ where $q^{i-1} = \tau(q^0, t_{1,i-1})$ and q^0 is the starting state of the PSA.[2] The most likely tags $t_{1,n}$ for a sequence of words $w_{1,n}$ are found using the Viterbi algorithm according to the following equation,

$$\mathcal{T}_{\mathrm{M}}(w_{1,n}) = \arg\max_{t_{1,n}} \prod_{i=1}^{n} \gamma(q^{i-1}, t_i) P(w_i|t_i) \ .$$

We estimate $P(w_i|t_i)$ indirectly from $P(t_i|w_i)$ using Bayes' Theorem,

$$P(w_i|t_i) = \frac{P(w_i)P(t_i|w_i)}{P(t_i)} \ .$$

The terms $P(w_i)$ are constant for a given sequence w_i and can therefore be omitted from the maximization. We find the maximum likelihood estimation of $P(t_i)$ by calculating the relative frequency of t_i in the training corpus. The estimation of the static parameters $P(t_i|w_i)$ is described in the next section.

We built a PST from the part of speech tags in the Brown corpus [43], with every tenth sentence removed (a total of 1,022,462 tags). The four stylistic tag modifiers "FW" (foreign word), "TL" (title), "NC" (cited word), and "HL" (headline) were ignored reducing the complete set of 471 tags to 184 different tags.

The resulting automaton has 49 states: the null state (denoted by ϵ), 43 first order states (one symbol long) and 5 second order states (two symbols long). This means that 184-43=141 states were not (statistically) different enough to be included as separate states in the automaton. An analysis reveals two possible reasons. Frequent symbols such as "ABN" ("half", "all", "many" used as pre-quantifiers, e.g. in "many a younger man") and "DTI" (determiners that can be singular or plural, "any" and "some") were not included because they occur in a variety of diverse contexts or often precede unambiguous words. For example, when tagged as "ABN" "half", "all", and "many" tend to occur before the unambiguous determiners "a", "an" and "the".

Some tags were not included because they were too rare. For example, "HVZ*" ("hasn't") is not a state although a following "- ed" form is always disambiguated as belonging to class "VBN" (past participle). But since this is a rare event, the state "HVZ*" is not a state of the automaton. We in fact lost some accuracy in tagging because of the suffix tree growing criterion as several "-ed" forms after forms of "have" were mistagged as "VBD" (past tense).

The two-symbol states were "AT JJ", "AT NN", "AT VBN", "JJ CC", and "MD RB" (article adjective, article noun, article past participle, adjective conjunction, modal adverb). Table 4.2 lists two of the largest differences in transition probabilities for each state. The varying transition probabilities are based on differences between the syntactic constructions in which the two competing states occur. For example, adjectives after articles ("AT JJ") are almost always used attributively which makes a following preposition impossible and a following noun highly probable, whereas a predicative use favors modifying prepositional phrases. Similarly, an adverb preceded by a modal ("MD RB") is followed by an infinitive ("VB") half the time, whereas other adverbs occur less often in pre-infinitival position. On the other hand, a past participle is virtually impossible after "MD RB" whereas adverbs that are not preceded by modals modify past participles quite often.

[2] A PSA does not a starting state but rather an initial probability distribution over all states. We use instead a PFA constructed from the PST output by the learning algorithm (see Section 4.4). This PFA has a single starting state and its ergodic subgraph is a PSA.

transition to	one-symbol state	two-symbol state
NN	JJ: 0.45	AT JJ: 0.69
IN	JJ: 0.06	AT JJ: 0.004
IN	NN: 0.27	AT NN: 0.35
.	NN: 0.14	AT NN: 0.10
NN	VBN: 0.08	AT VBN: 0.48
IN	VBN: 0.35	AT VBN: 0.003
NN	CC: 0.12	JJ CC: 0.04
JJ	CC: 0.09	JJ CC: 0.58
VB	RB: 0.05	MD RB: 0.48
VBN	RB: 0.08	MD RB: 0.0009

Table 4.2: States for which the statistical prediction is significantly different when using a longer suffix for prediction. These states are identified automatically by the learning algorithm. A better prediction and classification of POS-tags is achieved by adding these states with only a small increase in the computation time.

4.9.3 Estimation of the Static Parameters

In order to compute the static parameters $P(w^j|t^i)$ used in the tagging equations described above, we need to estimate the conditional probabilities $P(t^i|w^j)$ (the probability that a given word w^j will appear with tag t^i). A possible approximation would be to use the maximum likelihood estimator,

$$P(t^i|w^j) = \frac{C(t^i, w^j)}{C(w^j)} \quad ,$$

where $C(t^i, w^j)$ is the number of times t^i is tagged as w^j in the training text and $C(w^j)$ is the number of times w^j occurs in the training text. However, some form of smoothing is necessary, since any new text will contain new words, for which $C(w^j)$ is zero. Also, words that are rare will only occur with some of their possible parts of speech in the training text. A common solution to this problem is to use the *add*-1 estimator,

$$P(t^i|w^j) = \frac{C(t^i, w^j) + 1}{C(w^j) + I} \quad ,$$

where I is the number of tags, 184 in our case. It turns out that such smoothing is not appropriate for our problem. The reason is the distinction between closed-class and open-class words. Some syntactic classes like verbs and nouns are productive, others like articles are not. As a consequence, the probability that a new word is an article is zero, whereas it is high for verbs and nouns. We therefore need a smoothing scheme that takes this fact into account.

Extending an idea in [23], we estimate the probability of *tag conversion* to find an adequate smoothing scheme. Open and closed classes differ in that words often add a tag from an open class, but rarely from a closed class. For example, a word that is first used as a noun will often be used as a verb subsequently, but closed classes such as possessive pronouns ("my", "her", "his") are rarely used with new syntactic categories after the first few thousand words of the Brown corpus. We only have to take stock of these "tag conversions" to make informed predictions on new tags when confronted with unseen text. Formally, let $W_l^{i,\neg k}$ be the set of words that have been seen with t^i, but not with t^k in the training text up to word w_l. Then, we can estimate the probability that a word with tag t^i will later be seen with tag t^k as the proportion of words allowing tag t^i but not t^k

that later add t^k,

$$P_{lm}(i \to k) = \frac{|\{n|l < n \leq m \ \wedge \ w_n \in W_l^{i,\neg k} \cap W_{n-1}^{i,\neg k} \ \wedge \ t_n = t^k\}|}{|W_l^{i,\neg k}|} \ .$$

This formula also applies to words we haven't seen so far, if we regard such words as having occurred with a special tag "U" for "unseen". In this case, $W_l^{U,\neg k}$ is the set of words that haven't occurred up to l. $P_{lm}(U \to k)$ then estimates the probability that an unseen word has tag t^k. Table 4.3 shows the estimates of tag conversion we derived from our training text for $l = 1022462 - 100000$, $m = 1022462$, where 1022462 is the number of words in the training text. To avoid sparse data problems we assumed zero probability for types of tag conversion with less than 100 instances in the training set.

tag conversion	estimated probability
U → NN	0.29
U → JJ	0.13
U → NNS	0.12
U → NP	0.08
U → VBD	0.07
U → VBG	0.07
U → VBN	0.06
U → VB	0.05
U → RB	0.05
U → VBZ	0.01
U → NP$	0.01
VBD → VBN	0.09
VBN → VBD	0.05
VB → NN	0.05
NN → VB	0.01

Table 4.3: Estimates for tag conversion.

Our smoothing scheme is therefore the following heuristic modification of the *add*-1 technique,

$$P(t^i|w^j) = \frac{C(t^i, w^j) + \sum_{k_1 \in T_j} P_{lm}(k_1 \to i)}{C(w^j) + \sum_{k_1 \in T_j, k_2 \in T} P_{lm}(k_1 \to k_2)} \ ,$$

where T_j is the set of tags that w^j has in the training set and T is the set of all tags. This scheme has the following desirable properties:

- As with the *add*-1 technique, smoothing has a small effect on estimates that are based on large counts.
- The difference between closed-class and open-class words is respected. The probability for conversion to a closed class is zero and is not affected by smoothing.
- Prior knowledge about the probabilities of conversion to different tag classes is incorporated. For example, an unseen word w^j is five times as likely to be a noun than an adverb. Our estimate for $P(t^i|w^j)$ is correspondingly five times higher for "NN" than for "RB".

4.9.4 Analysis of Results

Our result on the test set of 114392 words (the tenth of the Brown corpus not used for training) was 95.81%. Table 4.4 shows the 20 most frequent errors.

PSA: Correct Tag:	JJ	VBN	NN	VBD	IN	CS	NP	RP	QL	RB	VB	VBG
NN	259						102	100			69	66
VBD		228										
NNS			227									
VBN				219								
JJ			165							71		
VB			142									
CS					112							
NP	110		194									
IN						103						
VBG			94									
RB	63				63				76			
QL										64		

Table 4.4: Most common errors.

Three typical examples for the most common error (tagging nouns as adjectives) are "Communist", "public" and "homerun" in the following sentences.

- the Cuban fiasco and the Communist military victories in Laos
- to increase public awareness of the movement
- the best homerun hitter

The words "public" and "communist" can be used as adjectives or nouns. Since in the above sentences an adjective is syntactically more likely, this was the tagging chosen by the system. The noun "homerun" didn't occur in the training set, therefore the priors for unknown words biased the tagging towards adjectives, again because the position is more typical of an adjective than of a noun.

Two examples of the second most common error (tagging past tense forms ("VBD") as past participles ("VBN")) are "called" and "elected" in the following sentences:

- the party called for government operation of all utilities
- When I come back here after the November election you'll think, you're my man – elected.

Most of the VBD/VBN errors were caused by words that have a higher prior for "VBN" so that in a situation in which both forms are possible according to local syntactic context, "VBN" is chosen. More global syntactic context is necessary to find the right tag "VBD" in the first sentence. The second sentence is an example of one of the tagging *mistakes* in the Brown corpus, "elected" is clearly used as a past participle, not as a past tense form.

4.9.5 Comparative Discussion

Charniak et al.'s result of 95.97% [23] is slightly better than ours. This difference is probably due to the omission of rare tags that permit reliable prediction of the following tag (the case of "HVZ*" for "hasn't").

Kupiec achieves up to 96.36% correctness [77], without using a tagged corpus for training as we do. But the results are not easily comparable with ours since a lexicon is used that lists only possible tags. This can result in increasing the error rate when tags are listed in the lexicon that do not occur in the corpus. But it can also decrease the error rate when errors due to bad tags for rare words are avoided by looking them up in the lexicon. Our error rate on words that do not occur in the training text is 57%, since only the general priors are used for these words in decoding. This error rate could probably be reduced substantially by incorporating outside lexical information.

While the learning algorithm of a PSA is efficient and the resulting tagging system is very simple and efficient, the accuracy achieved is rather moderate. This is due to several reasons. As mentioned at the beginning of the chapter, any finite memory Markov model cannot capture the recursive nature of natural language. A PSA can accommodate longer statistical dependencies than a traditional full-order Markov model, but due to its Markovian nature long-distance statistical correlations are neglected. Therefore, a PSA based tagger can be used for pruning many of the tagging alternatives using its prediction probability, but not as a complete tagging system. Furthermore, the PSA is better utilized in low level language processing tasks such as correcting corrupted text as demonstrated in Section 4.7.

Another drawback of the current tagging scheme is the independence assumption of the underlying tags and the observed words, and the ad-hoc estimation of the static probabilities. A possible more systematic estimation scheme would be to estimate these probabilities using Bayesian statistics, by assigning a discrete probability distribution, such as the Dirichlet distribution [15] or a mixture of Dirichlet distributions [7], to each tag class.

Chapter 5

Putting It All Together

5.1 Introduction

While the fast emerging technology of pen-computing is already available on the world's markets, there is still a gap between the state of the hardware and the quality of the available handwriting recognition algorithms. This gap is due to the absence of reliable and robust cursive handwriting recognition methods. Surprisingly, only recently the close relation between cursive handwriting and speech recognition has been fully appreciated, and a large number of researchers are now working in this direction (see for example [12, 47, 93]). Yet there are some important differences between the analysis of speech and handwriting, which are essential to the successful transfer of speech recognition algorithms to online handwriting.

Though both types of signals can be viewed as temporal sequences used for human communication, the physical mechanisms underlying handwriting are entirely different from those of speech. Whereas speech is both acoustically generated and perceived, handwriting is generated by our hand motor system and is visually perceived. Just as it was impossible to make any progress in speech recognition without a good physically based model of the signal, it is probably as difficult to do so for cursive handwriting. In the case of speech, such models are usually based on spectral analysis, either through linear predictive coding (LPC) [87] or directly in the frequency domain. These models utilize the understanding of the acoustic production of the signal to obtain efficient encoding of the relevant information. Such encodings reduce the amount of redundant information and enforce invariances under distortions which are not useful for the recognition process.

We believe that a similar physical model is required also in the case of handwriting in order for an analogous approach to be effective. Therefore, the dynamical encoding of cursive handwriting, described in Chapter 2, is used as the front-end to our cursive handwriting recognition system. The result of the encoding process is *discrete* sequences of motor control commands. The motor control representation enables efficient application of the learning algorithms presented in Chapter 3 and 4. We use a combination of probabilistic automata to build a (probabilistic) mapping from the low level motoric representation to a higher level of representation, namely, the characters that constitute the written text.

The accumulated experience in speech recognition in the past 30 years has yielded some important lessons that are also relevant to handwriting. The first is that one cannot completely predefine the basic 'units' to be modeled due to the strong co-articulation effects. Therefore, any model must allow some variability of the basic units in different contexts and by different speakers. A second important ingredient of a good stochastic model of speech, as well as handwriting, is adaptability. Most, if not all, currently used models in speech and handwriting recognition are difficult to adapt (for example see [104, 137]), and require vast amounts of training data to show some robustness.

The alternative that we use is acyclic probabilistic finite automata (APFA). Although simpler than HMMs, these automata seem to capture well the context dependent variability of short motor control commands. Moreover, the online learning algorithm for APFAs enables a simple yet powerful scheme that adapts the models' topology as well as their parameters to new sequences of motor control commands.

Another important lesson from speech recognition is that there is no clear separation between the low level models of the basic "units" of speech and the higher level language models, and that the two should be addressed together, on the same statistical basis. To apply this principle to cursive handwriting we need to consider a hierarchy of probabilistic models, in which the lower level deals directly with the discrete motor control commands using a set of APFAs, while the higher level operates on results of the APFAs incorporating linguistic knowledge. We use the Markov model with variable memory length, described and analyzed in Chapter 4, to automatically acquire and approximate the structure of language by building a model from natural English texts. There are several advantages to this approach. First, no explicit language knowledge, such as a predefined dictionary, is required. Therefore, a time consuming dictionary search is avoided. Second, the Markovian language model can be easily swapped or adapted. Moreover, an online adaptation to new syntactic styles can be achieved by updating the language model structure and parameters 'on-the-fly' without any further changes to the system itself. Moreover, our recognition scheme is not limited to isolated words. Our language model naturally incorporates the notion of word boundaries by treating the blank character in the same way as all the other English characters. Therefore, word boundaries are automatically identified while searching for the most likely transcription of a cursively written text.

Our approach to online recognition of cursive scripts has a only a little overlap with the current and past methods which are the result of over 30 years of research in this area. Reviews on the different recognition approaches are given in [85, 128, 17]. The more recent approaches are based on local and *redundant* feature extraction (*cf.* [17]) that are fed into a probabilistic model such as an HMM [12, 47, 93], a neural network [55], a self-organizing map [119, 91], or an hybrid structure, which recently became popular, that combines a neural network with an HMM [14, 42]. In most systems, the features extraction stage is a fixed irreversible transformation that maps pen trajectories to sets of local features such as the local curvature and absolute speed. The learning algorithms for the probabilistic models are mostly based on a gradient descent search or the EM algorithm whose weaknesses were discussed in previous chapters. We believe that our algorithmic based approach may provide a better cursive handwriting analysis tool than the existing approaches.

The structure of this chapter is as follows. In Section 5.2 we describe how APFAs are used to approximate the distribution of the possible motor control commands that represent the cursive letters. In Section 5.3 we discuss an automatic scheme to segment and train the letter APFAs given a training set of transcribed words. Then, in Section 5.4 we describe a scheme that assigns probabilities to sequences outside the training set. This scheme can tolerate noise that may substitute, insert and delete symbols from the motor control sequences. In Section 5.5 we describe the usage of a Markov model with a variable memory as a language model in our cursive handwriting recognition system. Finally, in Section 5.6 the system is described, evaluated, and a complete run of the system is demonstrated.

5.2 Building Stochastic Models for Cursive Letters

In this section we show how the learning algorithm for APFAs is used to build stochastic models that approximate the distribution of the motor control commands. We assume that each cursive word was segmented to non-overlapping segments which correspond to the letter constituents of the word. We later show how an automatic segmentation scheme can be devised.

In order to build stochastic models for the different cursive letters, we combine the 3 different channels that constitute the motor control commands (see Chapter 2) by taking the cartesian product of the three channels at each time point. The result is triplets of the form $\mathcal{Y} \times \mathcal{X} \times \mathcal{C}$ where $\mathcal{X}, \mathcal{Y} \in \{0, 1, 2, 3, 4, 5\}$ and $\mathcal{C} \in \{0, 1, 2, 3\}$. Hence, the alphabet consists of 144 different symbols. These symbols represent quantized horizontal and vertical amplitude modulations, the phase-lag between the horizontal and vertical oscillations and delayed strokes such as dots and bars. The symbol $0 \times 0 \times 0$ represents zero modulation and it is used to denote pen lifts and end of writing activity. This symbol serves as the final symbol (ζ) for building the APFAs for cursive letters.

Different Roman letters map to different sequences of motor control commands. Moreover, since there are different writing styles and due to the existence of noise in the human motor system, the same cursive letter can be written in many different ways. This results in different sequences representing the same letter. We used the modified version of the APFA learning on several hundreds of examples of segmented cursive letters to build 26 APFAs, one for each lower-case cursive English letter. In order to verify that the resulting APFAs have indeed learned the distributions of the sequences that represent the cursive letters, we performed a simple sanity check. Random walks using each of the 26 APFAs were used to generate synthetic motor control commands. The forward dynamic model was then used to transform these synthetic strings into pen trajectories. This process, known as *analysis-by-synthesis*, is widely used for testing the quality of speech models. A typical result of such random walks on the corresponding APFAs is given in Figure 5.1. All the synthesized letters are intelligible. The distortions are partly due to the compact representation of the dynamic model and not necessarily a failure of the learning algorithm.

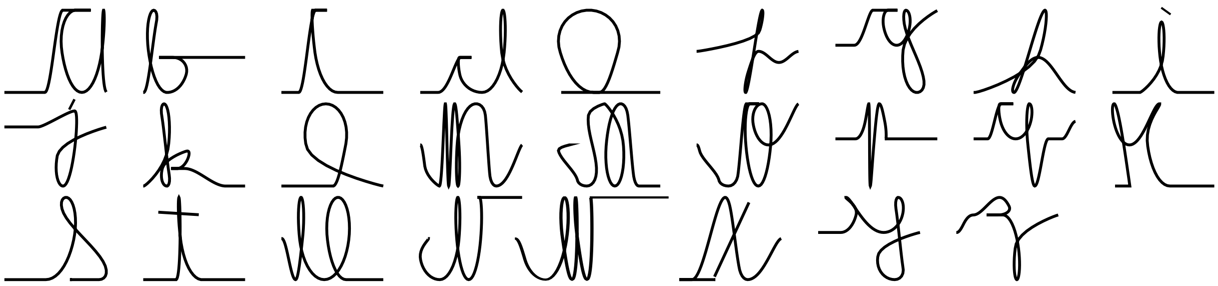


Figure 5.1: Synthetic cursive letters created by random walks using the 26 letter APFAs.

We also performed a test that checks whether different random walks using the same APFA are consistent in the sense that letter drawings, generated from different random walks, are intelligible. Typical results are shown in Figure 5.2, where several synthetic letters, created using the APFA that represents the cursive letter `k` (which has a rather complex spatial structure) are depicted. All the random walks created intelligible drawings. Moreover, the letters start and end in several different ways. This indicates that the APFAs also capture effects of neighboring letters. These effects are similar to the co-articulation effects between phonemes in speech. Thus, the APFAs indeed capture some of the variability of written letters due to the different contexts.

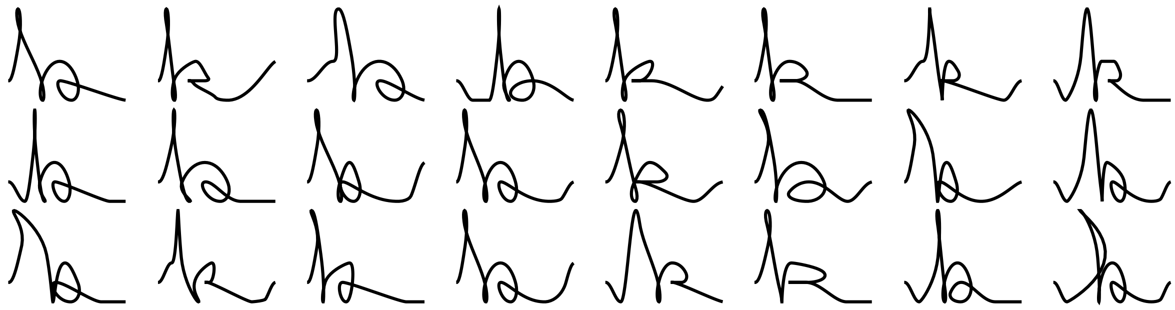


Figure 5.2: Synthetic cursive letters created by random walks using the APFA that represents the letter `k`.

It is also interesting to look at the intermediate automata built along the run of the APFA learning algorithm. Several of the intermediate automata that were built when the algorithm was trained on segmented data that represent the cursive letter `l` are shown on Figure 5.3. The number of training sequences in this examples is 195, and the initial automaton has 209 states. In order to represent such large automata we ignored the third channel that encodes the delayed strokes. Hence, for representation purposes only, the alphabet is of the form $\mathcal{X} \times \mathcal{Y}$ and its size is 36. Thus, the symbol labeling each edge in the figure is one of the possible 36 motor control commands and the final symbol ζ is 0×0. The number on each edge is the count associated with the edge, that is, the number of times the edge was traversed in the training data. The top left automaton in the figure is the initial sample tree, hence all of its leaves are connected to the final state with an edge labeled with the final symbol. The intermediate automata are drawn at every tenth iteration, left to right and top to bottom. The final automaton, which was output by the learning algorithm after 41 merging iterations, is drawn at the bottom part of the figure. The intermediate automata at the start of the merging process are very 'bushy' with no apparent structure. After 20 iterations, when more merges have been performed, a compact structure starts to appear. Finally, the resulting automaton has only 12 states with an interesting structure. All the outgoing edges from state 4 and the incoming edges into state 5 are labeled by symbols of the form $5 \times x,\ x \in \{0, 1, 2, 3, 4, 5\}$. Since all the paths from the start state to the final state must pass through either state 4 or 5, it implies that a symbol of the form $5 \times x$ must be generated by *any* random walk using one of the existing paths in the automaton. This symbol corresponds to high vertical modulation value (the top part of the letter `l`). Therefore, states 4 and 5 'encode' the fact that the letter `l` is characterized by a high vertical modulation value.

5.3 An Automatic Segmentation and Training Scheme

In the previous section we described how to build a set of probabilistic models from *segmented* motor control commands. However, such data is usually not available since it requires vast amounts of manual work. Moreover, in cursive handwriting, as in continuous speech, there is no clear notion of letter boundaries. Therefore, one of the intermediate tasks in building a cursive recognition system is devising an automatic scheme that segments a cursively written word into its letters constituents

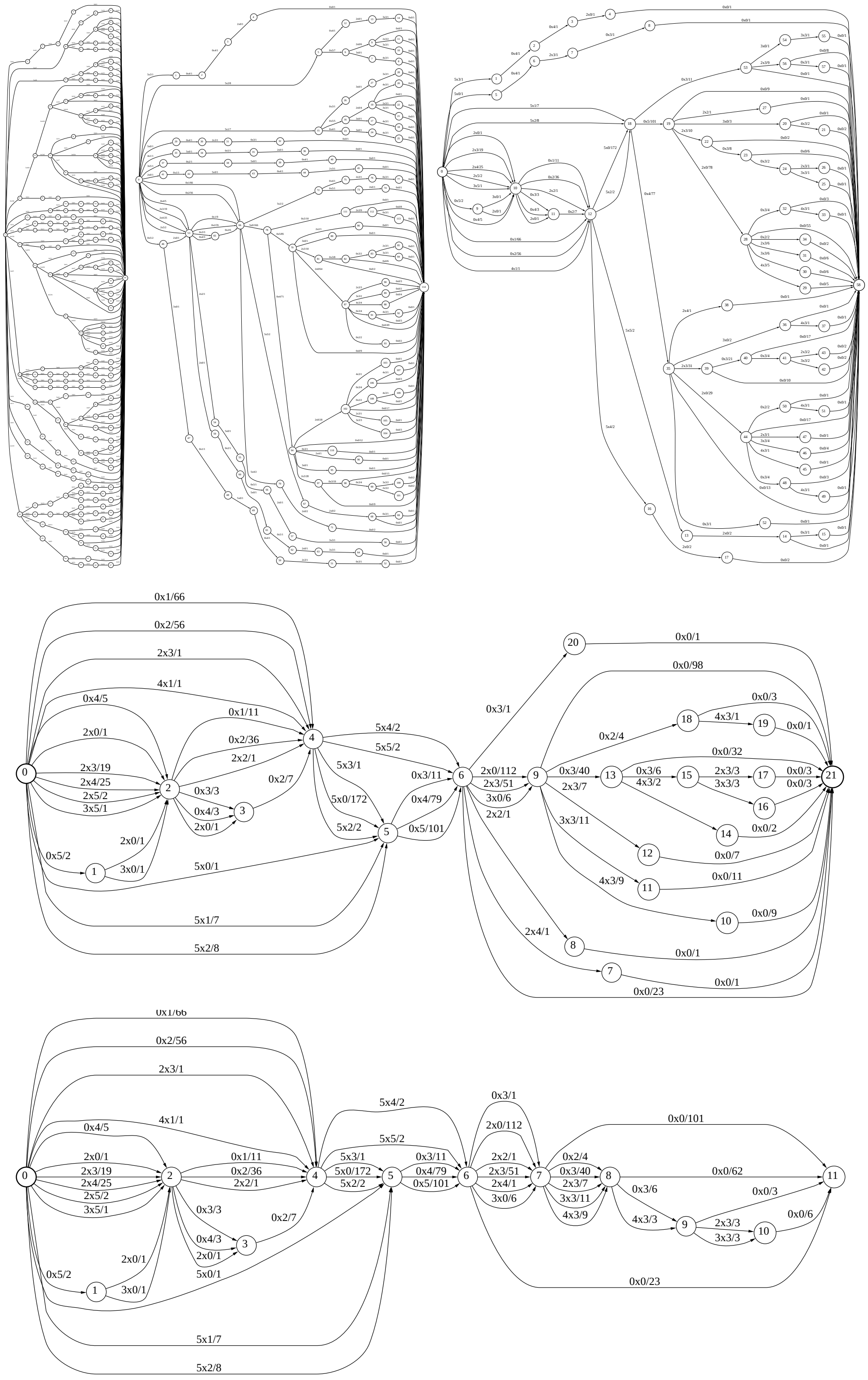


Figure 5.3: Several of the intermediate automata built along the run of the APFA learning algorithm.

given the transcription of the word. The automatically segmented letters can then be used to retrain or update the models.

A segmentation partitions the motor control commands into non-overlapping segments, where each segment corresponds to a different letter. Given a transcription of a cursively written word is given, the most likely segmentation of that word is found as follows. Denote the input sequence of motor control commands by $s_1, s_2, \ldots, s_L$ and the letters that constitute the transcription by $\sigma_1, \sigma_2, \ldots, \sigma_K$ ($\sigma_i \in \Sigma$). A segmentation is a monotonically increasing sequence of $K+1$ indices, denoted by $\mathcal{I} = i_0, i_1, \ldots, i_K$, such that $i_0 = 1$, $i_K = L+1$. As in the previous section, we associate with each cursive letter an APFA that approximates the distribution of the possible sequences of motor control commands representing that letter. Denote this set of APFAs by $\mathcal{A}$. Let the probability of a sequence s to be produced by a model corresponding to the letter σ be denoted by $P^{\sigma}(s)$. The likelihood of the sequence of motor control commands, given a transcription, a proposed segmentation, and a set of APFAs, is

$$P\left((s_1, \ldots, s_L) \,|\, \mathcal{I}\,, (\sigma_1, \sigma_2, \ldots, \sigma_K)\,, \mathcal{A}\right) = \prod_{k=1}^{K} P^{\sigma_k}(s_{i_{k-1}}, \ldots, s_{i_k - 1}, \zeta) \ , \tag{5.1}$$

where ζ is the final symbol $(0 \times 0 \times 0)$. If we assume that all possible segmentations are equally probable apriori, then the above is proportional to the probability of a segmentation given the input sequence, the set of APFAs, and the transcription. The most likely segmentation for a transcribed word can be found efficiently by using a dynamic programming scheme as follows. Let $Seg(n, k)$ be the likelihood of the prefix $s_1, \ldots, s_n$ given the most probable partial segmentation that consists of k letters. $Seg(n, k)$ is calculated recursively through,

$$Seg(n, k) = \max_{1 \leq n' < n} Seg(n', k-1) \times P^{\sigma_k}(s_{n'+1}, \ldots, s_n, \zeta) \ . \tag{5.2}$$

Initially, $Seg(k, n)$ is set to 0 for all $k \neq 0$, $n \neq 0$ and $Seg(0, 0)$ is set to 1. The above equation can be evaluated efficiently for all possible n and k by maintaining a table of size $L \times K$. The likelihood of the most probable segmentation is $Seg(L, K)$. The most probable segmentation itself is found by keeping the indices that maximize Equation (5.2), for all possible n and k, and backtracking these indices from $S(L, K)$ back to $S(0, 0)$. An example of the result of such a segmentation is depicted in Figure 5.4, where the cursive word `impossible`, reconstructed from the motor control commands, is shown with its most likely segmentation. Note that the segmentation is temporal, hence in the pictorial representation letters are sometimes cut in the 'middle' though the segmentation is correct.

The above segmentation procedure is incorporated into an online learning setting as follows. We start with an initial stage where a relatively reliable set of APFAs for the cursive letters is constructed from a small set of *segmented* data. We then continue with an online setting in which we employ the probabilities assigned by the automata to segment new unsegmented words, and 'feed' the segmented subsequences back as inputs to the corresponding APFAs. We use the APFAs online learning algorithm to update and refine the models of each cursive letter from the segmented subsequences. We iterate this process until all the trained (transcribed) data is scanned. The complete training scheme is described in Figure 5.5. When a new writer starts to use the system, the same scheme is applied using an initial reliable set of APFAs. After each input, which may constitute of several words, the writer may provide a transcription of the written text (in case it

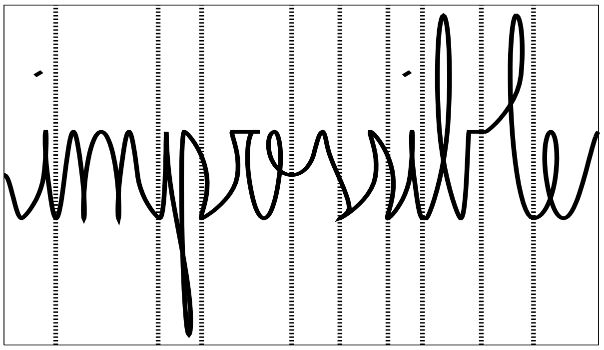


Figure 5.4: *Temporal* segmentation of the word `impossible`. The segmentation is performed by evaluating the probabilities of the APFAs which correspond to the letter constituents of the word. These probabilities are evaluated for each possible subsequence of the motor control commands. The most likely segmentation is then found using dynamic programming.

was incorrectly recognized). The transcribed input is then segmented and used to update the set of APFAs using the online learning mode.

5.4 Handling Noise in the Test Data

After a set of APFAs is built, we can calculate the probabilities of new subsequences of motor control commands and use these probabilities for recognizing cursive scripts. However, using the set of APFAs in a straight forward manner is not robust enough due the reasons described below. The main difficulty arises when a subsequence of motor control commands defines a state sequence (belonging to the APFA representing the letter that has been written) that crosses an edge which has not been observed in the learning stage. The algorithm presented in Chapter 3 assigns a small transition probability to such an edge and connects it to one of the slack states ($small(d)$). The rest of the subsequence is assigned a small probability, since the path proceeds with the states $small(d), small(d+1), \ldots, q_f$ whose transition probabilities are uniform. This construction of slack states, although robust enough when a segmentation of transcribed words is performed, is in practice too crude for recognition, since *one* substitution in the input sequence may result in a low probability assignment to the entire sequence. If we had collected more data for learning the set of APFAs, such a sequence might have been assigned a significantly higher probability. There are also other problems that result in the same difficulty, such as digitization errors of the pen motion capturing device (tablet) and incorrect model assumptions. Moreover, there may be estimation errors of the dynamical encoding scheme that may cause deletions and insertions of motor control commands. Again, such problems could be simply avoided by collecting and using more data in the training stage. However, in the recognition stage, we have to do the best we can with the models we have at hand. We treat all sources of errors as well as finite sample size effects on the same basis and devise a scheme that can tolerate a small number of insertions, deletions and substitutions of motor control commands. After a new sequence is recognized correctly (or its correct transcription is provided), we use the online learning mode to update the set of APFAs and obtain a refined set of models.

In order to tolerate a small number of errors, we leave edges with zero counts 'open', i.e., such edges are not connected to any state of the automata. When a new sequence is observed, these edges are *momentarily* connected to states with large counts (q s.t. $m_q \geq m_0$) for which the suffix of the sequence may be assigned a high probability. An illustrative example of this scheme is shown in Figure 5.6.

It remains to describe how to connect open edges and use this procedure to calculate the

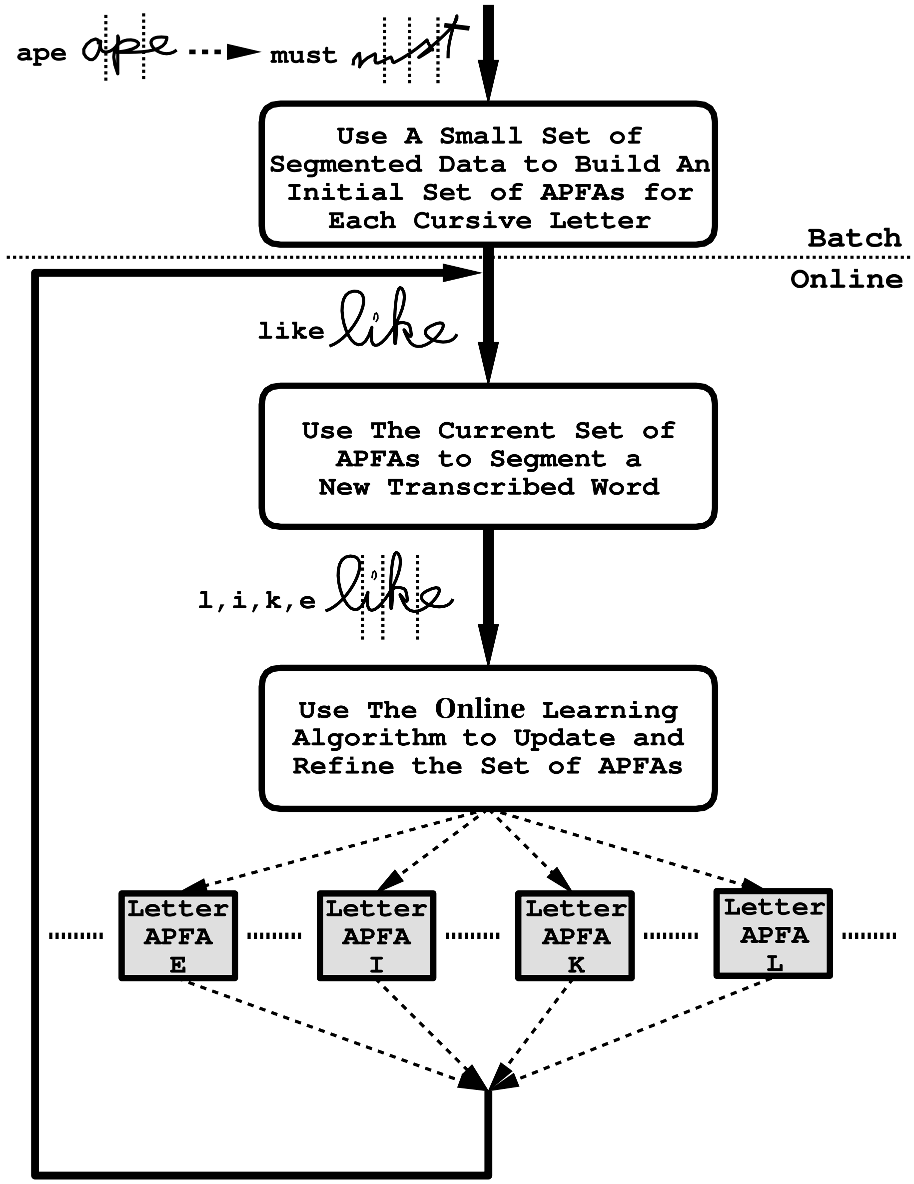


Figure 5.5: The training scheme for building a set of letter APFAs from *unsegmented* cursive words. In the online recognition mode, a similar scheme is used to recognize new scripts.

probabilities of new sequences. In many scientific areas, it is important to choose among various explanations of observed data. A general principle, governing scientific research, is to weigh each possible explanation by its 'complexity', and to choose the simplest (shortest) explanation that is consistent with the observed data. This type of argument is often called "Occam's Razor", and is related to William of Occam who said "Nunquam ponendest pluralitas sine necesitate", i.e., explanations should not be multiplied beyond necessity [131]. In our framework, Occam's Razor is equivalent to choosing the shortest description of a sequence of motor control commands aligned

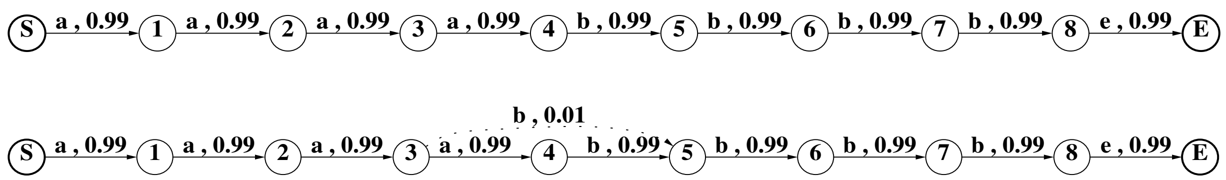


Figure 5.6: A toy example of the scheme for calculating the probability of a noisy sequence. In this example the alphabet is $\{a, b, e\}$ where e is the final symbol. Edges which have not been observed in the the training data are left 'open'. In this example, the symbol b was not observed at the states $S, 1, 2$ and 3, hence the edges labeled by b at these states are left 'open' and are not drawn. The automaton on the top assigns a high probability to the sequence $a, a, a, a, b, b, b, b, e$ and if the 'open' edges are connected to the slack states ($small(d)$) then the rest of the probability mass is almost uniformly distributed among all the other sequences from $\{a, b, e\}^\star$. Therefore, the sequence a, a, a, b, b, b, b, e, which differs from $a, a, a, a, b, b, b, b, e$ by only a single symbol, is assigned a low probability ($0.99^3 \times 0.01 \times (\frac{1}{3})^4$, where $(\frac{1}{3})^4$ is the probability assigned to the rest of the sequence by the slack states). However, if we momentarily connect the open edge labeled by b outgoing from state 3 to state 5 (bottom figure), the sequence is assigned a significantly higher probability ($0.99^7 \times 0.01 \times \frac{1}{11}$, where $\frac{1}{11} = \frac{1}{|Q|+1}$ is the extra cost incurred by connecting the edge to state 5).

to a given APFA. We use Rissanen's minimum description length[1] (MDL) principle to find the assignment of open edges that results in the shortest description of the data. We view the problem of finding the probability of a sequence as a communication problem. Suppose that a transmitter wants to send to a receiver a sequence of motor control commands $s_1, s_2, \ldots, s_L$, created by an APFA. Both the transmitter and the receiver keep track of the state of the automaton reached after observing a prefix of length n of the input sequence. Denote this state by q. Then, if the next symbol, s_{n+1}, corresponds to an edge with a count greater than zero, the transmitter encode the next symbol using the estimated transition probability $\tilde{\gamma}(q, s_{n+1})$. If the corresponding edge has zero count ($m_q(s_{n+1}) = 0$), i.e., the edge is not connected to any state, then the transmitter connects the edge to a state and sends the index of this state to the receiver. The number of possible states the edge may be connected to is bounded by the total number of states. This edge may also point at an entirely new state, hence at most $\log_2(|Q|+1)$ bits are required to encode the index of the next state. This is the additional logarithmic loss incurred by using an edge with zero count. To summarize, the number of bits transmitted for the next symbol s_{n+1} is:

- $-\log_2(\tilde{\gamma}(q, s_{n+1}))\ ,\quad$ if $m_q(s_{n+1}) > 0$.
- $-\log_2(\tilde{\gamma}(q, s_{n+1})) + \log_2(|Q|+1)\ ,\quad$ otherwise.

The probability of a sequence is defined to be 1/2 to the power of the number of bits transmitted. In order to find the shortest encoding, we can enumerate all possible assignments of open edges (whenever we need to traverse such an edge), calculate the code length of each possible state sequence, and choose the shortest encoding. A straightforward enumeration is clearly infeasible. However, using dynamic programming we can find the shortest code length in time proportional to $L \times (|Q|+1)^2$. We associate with each state q and each prefix of the input sequence of length n,

[1]Rissanen's work stems from the pioneering work of Kolmogorov [74, 75], Solomonoff [126] and Chaitin [21] who defined the algorithmic (descriptive) complexity of an object. For an in depth introduction to Kolmogorov complexity and its applications see [84].

a value which is the minimal code length of the prefix given that we reached state q after the n'th observation. These code lengths are stored in a table of size $L \times (|Q|+1)$, denoted by T. Therefore,

$$T(q,n) \stackrel{\text{def}}{=} \max_{\substack{q^0,q^1,\ldots,q^n \text{ s.t. } q^0=q_0,\, q^n=q, \\ q^{i+1}=\tau(q^i,s_i) \text{ or } \tau(q^i,s_i)=unassigned}} -\sum_{i=0}^{n-1} \log_2(\gamma(q^i,s_i)) \;+ \sum_{\tau(q^i,s_i)=unassigned} \log_2(|Q|+1) \ .$$

The table is updated for growing prefixes until an end of the input sequence is encountered. The log-likelihood of the sequence is the code length of the entire sequence at the final state, $T(q_f, L)$. A full description of the algorithm is given in Figure 5.7. The algorithm accommodates noise that insert symbols by adding to the automaton a virtual state, denoted by q_{new}, whose transition probabilities are all equal. This state is initially disconnected from all the states and along the run of the algorithm it can be momentarily connected to any state.

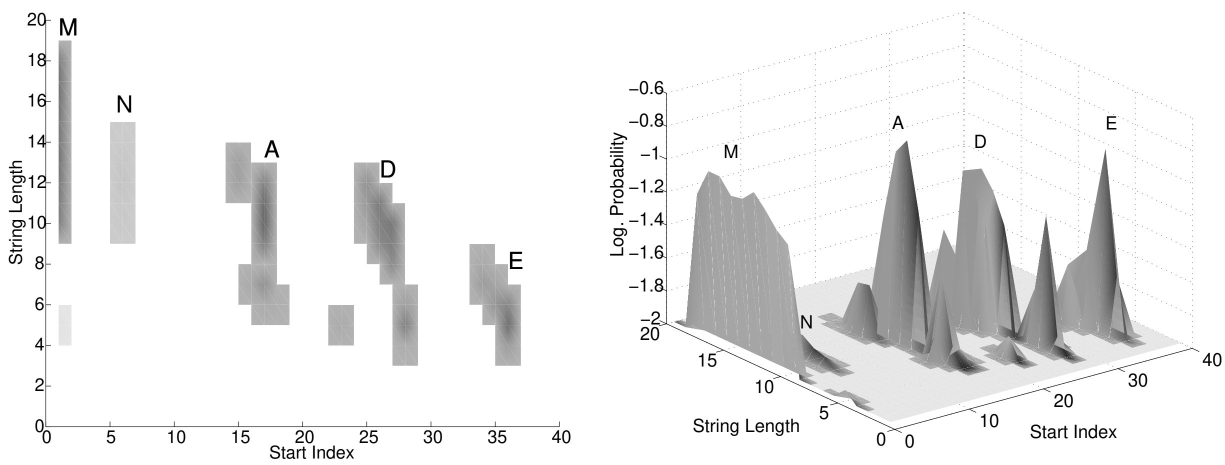
Input:
A sequence of motor control commands, $s_1, s_2, \ldots, s_L$ $(s_L = \zeta)$;
An APFA $A = (Q, q_0, q_f, \Sigma, \zeta, \tau, \gamma)$.

1. Set:
   (a) $\forall s \in \Sigma,\ q \in Q$ if $m_q(s) = 0$ then set $\tau(q,s) \leftarrow unassigned$.
   (b) $\forall s \in \Sigma,\ \tau(q_{new}, s) \leftarrow unassigned$.
2. Initialize: $\forall q \in Q \bigcup q_{new},\ 0 \le i \le L:\ T(q,i) = \infty$; $T(q_0, 0) = 0$.
3. Iterate for i from 0 to $L-1$ and for all $q \in Q \bigcup q_{new}$:
   (a) If $\tau(q, s_i) \ne unassigned$,
   $$T(\tau(q,s_i), i+1) \leftarrow \min \begin{cases} T(\tau(q,s_i), i+1) \\ T(q,i) - \log_2(\gamma_q(s_i)) \end{cases} .$$
   (b) If $\tau(q, s_i) = unassigned$, then for $q' \in Q \bigcup q_{new}$ do:
      i.
      $$T(q', i+1) \leftarrow \min \begin{cases} T(q', i+1) \\ T(q,i) + \log_2(|Q|+1) - \log_2(\gamma_q(s_i)) \end{cases} .$$
      ii. If $T(q', i+1)$ has changed set $\tau(q, s_i) = q'$.

Output: $P^A(s_1, s_2, \ldots, s_L) = \left(\frac{1}{2}\right)^{T(q_f,L)}$.

Figure 5.7: The algorithm for assigning probabilities to noisy sequences by finding the minimal message length.

Given a trained set of APFAs, denoted by $\mathcal{A}$, and the above scheme for calculating probabilities, we calculate the probabilities assigned by each automaton from the set for all possible subsequences of a new input sequence $s_1, s_2, \ldots, s_L$. The probability that the subsequence, $s_i, \ldots, s_j$ $(i \le j)$, was generated by an APFA $A \in \mathcal{A}$ is defined to be the probability that A generated the sequence and then moved to the final state, that is $P^A(s_i, \ldots, s_j, \zeta)$. We can represent these probabilities in three dimensions where the x axis is the start index i, the y axis is the subsequence length $(j-i+1)$ and the z axis is $P^A(s_i, \ldots, s_j, \zeta)$. If a subsequence $s_i, \ldots, s_j$ represents a cursive letter, then the probability induced by the corresponding APFA should be high and a 'bump' would appear around the index $i, j-i+1$ in this map. An example of such a map is given in Figure 5.8. A three dimensional plot and a topographic map that represents the highest value among the log-probabilities (likelihood values) induced by the set of APFAs are depicted. That is, the value at each point (i, j) in both plots is, $\max_{A \in \mathcal{A}} \log(P^A(s_i, \ldots, s_j, \zeta))$. Log-probabilities lower than -2 are clipped and not shown. An optimal setting is when the automata that correspond to the letter constituents of the word completely fill the space with (almost) non-overlapping 'tall bumps'. However, there might be false peaks. For example, in Figure 5.8, the automaton that corresponds to the letter `n` assigns a high probability for a subsequence that greatly overlaps with the subsequence that represents the letter `m`. Such ambiguities are resolved by incorporating linguistic knowledge, as described in the next section.



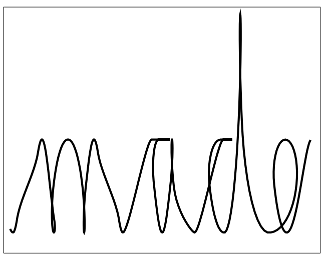


Figure 5.8: Visualization of the probabilities assigned by the set of letters APFAs for each possible subsequence of motor control commands representing the word `made`. The word, reconstructed from its motor control commands, is depicted on the bottom left. The log-probabilities are visualized through a topographic map (top left) and a three dimensional plot. A point (i, j) in both plots represents the maximal log-probability achieved among set of APFAs for the subsequence $s_{}, \ldots, s_j, \zeta$. Log-probabilities lower than -2 are clipped and not shown. Most of the space is covered by the automata that correspond to the letter constituents of the word, however there is a small 'bump' created by the APFA representing the letter `n`.

5.5 Incorporating Linguistic Knowledge

Cursive handwriting is one possible form of natural language communication. Language, whether written, spoken or even expressed in sign language, can be ambiguous, and cursive handwriting is no exception. In the previous section, an example of a simple ambiguous interpretation was given. Figure 5.9 demonstrates two less obvious examples of ambiguous cursive handwriting. As such ambiguities are inherent in any human generated handwriting, they cannot be solved without context. Therefore, some form of linguistic knowledge needs to be incorporated into the system. A common practice is to use a dictionary and search for the most likely transcription that appears in the dictionary. However, a straightforward evaluation of the likelihood of each word in the dictionary is infeasible in practice. Therefore, an approximated search is usually employed (*cf.* [54, 95, 125, 136]), which may result in a wrong transcription. Moreover, a dictionary based approach usually enforces an *isolated* word recognition scheme. Lastly, adding new words to the dictionary is a cumbersome task that frequently requires vast changes to the approximated dictionary search. We devise an alternative approach based on a Markov model with variable memory by building a model from natural texts. Texts containing daily conversations and common articles from [94] were used to build a prediction suffix tree. The alphabet includes all the lower case English letters and the blank character. Correlations across word boundaries may be found by the PSA learning algorithm using the blank character. Hence, sequences of motor control commands that include pen-up symbols $(0 \times 0 \times 0)$ may be broken into several words.

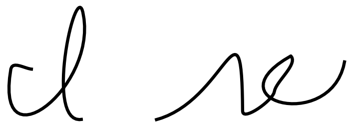


Figure 5.9: Examples of the ambiguity of cursive handwriting: the text on the left can be interpreted as either `d` or `cl` while the one on the right as `w` or `re`.

Denote by M the automaton that was built from the prediction suffix tree output by the learning algorithm described in Chapter 4. The construction of M from the resulting PST is described in Section 4.4 in that chapter. M is a PFA with a single start state, denoted by q_0. Let the blank character, whose role is to separate between words, be denoted by $\flat$. A transcription is a sequence of symbols from $\Sigma = \{a, b, c, \ldots, y, z, \flat\}$, denoted by $\sigma_1, \sigma_2, \ldots, \sigma_K$. Given a sequence of motor control commands, denoted as before by $s_1, \ldots, s_L$, we find the most likely transcription as follows. The probability that a subsequence $s_i, \ldots, s_j$ of motor control commands was generated by an APFA corresponding to a letter $\sigma \neq \flat$ is, $P^{\sigma}(s_i, \ldots, s_j, \zeta)$. That is, the automaton produced the subsequence and finished its operation by moving to the final state. If $\sigma = \flat$ we define,

$$P^{\flat}(s_i, \ldots, s_j, \zeta) = \begin{cases} 1 & \text{If } (s_i, \ldots, s_j) = \zeta^\star \\ 0 & \text{otherwise} \end{cases} \quad .$$

We can implement this probability measure using the automaton shown in Figure 5.10. The automaton generates only pen-up symbols or an empty sequence. The later occurs if the writer connected two consecutive words or if she lifted the pen for a very short period, too short to be captured by the digitizing device. Note that this automaton is not an APFA, since it is *not* acyclic and it has an edge outgoing from its final state. However, we can still use a dynamic programming based scheme since the notion of state space remains well defined. Using the same probabilistic representation for all the letters including blank enables a simple recognition scheme.

Denote by $P^M(\sigma_1, \ldots, \sigma_K)$ the probability that the PFA M generated the letter sequence $\sigma_1, \ldots, \sigma_K$. Recalling the definition from Chapter 4, this probability equals to

$$P^M(\sigma_1, \ldots, \sigma_K) = \prod_{k=1}^{K} \gamma_{\mathcal{M}}(q^{k-1}, \sigma_k) \ ,$$

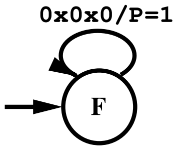


Figure 5.10: Words are separated by an automaton that outputs a (possibly empty) sequence of pen-up ($0 \times 0 \times 0$) symbols. The automaton has only one state – the final state.

where $q^k = \tau_M(q^{k-1}, \sigma_k)$ is the state reached after observing k letters and $q^0 = q_0$ is the start state of the automaton. The joint probability of a transcription $\sigma_1, \ldots, \sigma_K$ and a sequence $s_1, \ldots, s_L$ given the set $\mathcal{A}$ of APFAs and the PFA M, is found by enumerating all possible segmentations as follows,

$$P\left((\sigma_1, \ldots, \sigma_K), (s_1, \ldots, s_L) \mid \mathcal{A},\ M\right) = \\ P^M(\sigma_1, \ldots, \sigma_K) \left(\sum_{0=i_0<i_1<\ldots<i_{K-1}<i_K=L+1} \prod_{k=1}^{K} P^{\sigma_k}(s_{i_{k-1}}, \ldots, s_{i_k-1}, \zeta) \right) . \tag{5.3}$$

Although more involved than segmentation, finding the most likely transcription is again performed using a dynamic programming scheme. Let $Likl(n, k, q)$ be the joint probability of the most likely state sequence from M ending at state q and of the prefix of length n, $s_1, \ldots, s_n$. Also, let

$$Pred(q) \stackrel{\text{def}}{=} \{q' \mid \exists \sigma \text{ s.t. } \tau(q', \sigma) = q\}$$

be the set of states that have an outgoing edge that ends at q. $Likl(n, k, q)$ is calculated recursively through,

$$Likl(n, k, q) = \\ \max_{q' \in Pred(q)} \sum_{1 \le n' < n} Likl(n', k-1, q') \times P^{\sigma}(s_{n'+1}, \ldots, s_n, \zeta) \times \gamma_M(q', \sigma) \ , \tag{5.4}$$

where for each couple of states q and $q' \in Pred(q)$, σ is set such that $\tau(q', \sigma) = q$. $Likl(0, 0, q_0)$ is initially set to 1 and for all $q \neq q_0$ $Likl(0, 0, q)$ is set to 0. The probability of the most likely transcription is found by searching for the most likely state from M to end at, after observing the entire sequence of motor control commands. We also need to search the most likely transcription length. Hence, the probability of the most likely transcription is defined to be,

$$\max_{K,\, \sigma_1, \ldots, \sigma_K} P(\sigma_1, \ldots, \sigma_K | s_1, \ldots, s_L,\ \mathcal{A},\ M) \sim \\ \max_{K,\, \sigma_1, \ldots, \sigma_K} P\left((\sigma_1, \ldots, \sigma_K), (s_1, \ldots, s_L) \mid \mathcal{A},\ M\right) = \max_{q \in Q_M,\, K} Likl(L, K, q) \ . \tag{5.5}$$

The transcription itself is found by keeping the list of states that maximize Equation (5.5). Note that the list of states uniquely defines the transcription. If $q_i \to q_j$ then q_j is labeled by a string which is a suffix of $q_i \cdot \sigma$, $\sigma \in \Sigma$. Thus, σ is the letter resulting from this transition.

Since each APFA is acyclic, the sequences that can be generated by the automata are of a bounded length. We empirically found that the longest sequence, of length 24, is generated by the APFA corresponding to the letter `m`. To accommodate even longer sequences we set a bound on the maximal string production length of an APFA, denoted by B, to be 30. Using this bound, we can accelerate the computation by considering segmentations whose segments are of length at most B,

$$\begin{aligned} &Likl(n,k,q) = \\ &\qquad \max_{q' \in Pred(q)} \sum_{\mathbf{n-B \leq n' < n}} Likl(n', k-1, q') \times P^{\sigma}(s_{n'+1}, \ldots, s_n, \zeta) \times \gamma_M(q', \sigma) \ , \end{aligned} \tag{5.6}$$

$$\max_{K,\, \sigma_1,\ldots,\sigma_K} P\left((\sigma_1, \ldots, \sigma_K),\ (s_1, \ldots, s_L) \mid \mathcal{A},\ M\right) = \max_{q \in Q_M,\ \mathbf{\frac{L}{B} \leq K \leq L}} Likl(L, K, q) \ . \tag{5.7}$$

We devised an approximated scheme that further accelerates the above calculations. First, we replaced the sum over all possible segmentations in Equation (5.6) with a maximization. This approximation, termed *dominant sequence analysis*, is frequently used in HMM based speech analysis [104] and is well motivated, since most of the induced probability is captured by the most likely sequence [89]. Lastly, we further approximate the calculation by keeping for each n and k only promising states from the table $Likl(n,k,q)$. This approximation is also commonly used in evaluating the likelihood of a sequence by an HMM [68]. Given an approximation parameter δ, we keep a state q at time index n if $Likl(n,k,q) > \epsilon(\delta)$ where $\epsilon(\delta)$ is set such that,

$$\sum_{q \in Q_M,\, k, Likl(n,k,q) > \epsilon(\delta)} Likl(n,k,q) \ \ / \ \sum_{q \in Q_M,\, k} Likl(n,k,q) \ \geq \ 1 - \delta \ .$$

We experimentally found that for $\delta \leq 0.01$ the above approximations have almost no effect on the error rate and usually only a few states are actually kept and evaluated. By maintaining an adaptable minimal likelihood bound, $\epsilon(\delta)$, we can tolerate cases where the likelihood is rather evenly distributed. Such cases occur when *locally* there are several different transcriptions which are almost equally probable.

5.6 Evaluation and Discussion

We implemented the system on a Silicon Graphics workstation and used an external Wacom SD-501C tablet to record the pen motion during the writing process. The recordings include a pen-up/pen-down (proximity) indicator in addition to the X, Y location of the pen. The sampling rate of the tablet is 200 points per second. The recognition software package gets as input a stream of coordinates and proximity bits, a set of APFAs and a PFA as a simple language model. The set of APFAs and the PFA based language model are read from external files and are updated if the online adaptation mode is turned on. The system outputs a complete transcription as demonstrated in Figure 5.11.

Generally, one has to be careful comparing the recognition rates reported for different systems, as they are based on different data of different characteristics such as quality of handwriting, writing styles, and number of writers. To evaluate the performance of our system, we collected data from 10 different writers each writing around $300-400$ words from the same English texts used to build the language model. Achieving a low error rate with such a small amount of data is a challenging task. Most of the existing cursive handwriting recognition systems are trained in batch mode. Then, the performance of the system is evaluated using the model resulting from the training stage, with

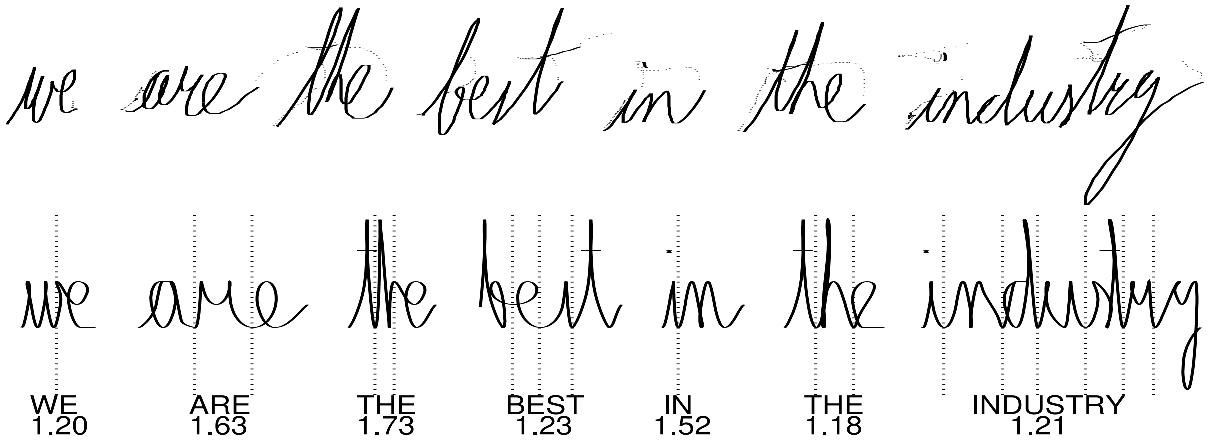


Figure 5.11: A demonstration of the recognition scheme. At the top the original handwriting is plotted. The original pen trajectory is composed of the pen movements on the paper as well as an approximation of the projection of pen movements onto the writing plane when the pen does not touch the writing plane. The reconstructed handwriting (synthesized from the motor control commands) is plotted at the bottom, together with the most likely transcription and segmentation. The segmentation is a byproduct of the recognition process and is not evaluated explicitly. Shown below each transcribed word are the average number of bits (log base 2 of the combined probabilities assigned by the set of APFAs and the PFA based language model) needed to encode the motor control commands that represent the word. In a typical successful recognition, less than 2 bits are on the average required to encode a motor control command.

no adaptation. However, we believe that adaptability is a key ingredient when analyzing highly ambiguous signals such as cursive scripts. Most people encounter great difficulties when trying to read handwriting they are unfamiliar with, due to the large variations of writing styles. Machines that recognize handwriting, in particular cursive scripts, encounter a similar problem when trying to recognize a new writer with a different writing style. We tested the performance of the system and its ability to adapt to new writing styles by using the online learning algorithm for APFAs. We used a small set (less than 250 letters) of segmented cursive letters from only one writer to bootstrap the whole learning process described in Section 5.3. We then tested each writer individually while adapting the set of APFAs. The PFA based language model was kept fixed in our experiments. We also tested the performance of the system without a language model using a uniform distribution over all letter sequences from the lower case English alphabet and the blank character. We used two error measures. The first is simply the number of words incorrectly recognized by the system. The second is the character error-rate, i.e., the fraction of insertions, deletions and substitutions (each counting one error) over the total number of characters. The results are summarized in Table 5.1.

	% char. error	% word error
No Language Model	19.9	74.3
With a Language Model	7.1	17.9

Table 5.1: Performance evaluation of the system, with and without a language model.

We tested the adaptability of the system by turning off the online learning mode and freezing the model for the rest of data. We turned off the adaptation at growing portions of the data for each writer. A portion of 100% is the full online mode. We also evaluated the log-likelihood of the most likely transcription, normalized by the length of the input sequence, at different portions of the data. The results are shown in Figure. 5.12. It is clear from the figure that the online adaptation plays an important role in achieving a low error rate. A challenging and profitable goal is to take the ideas presented here even further and build a fully adaptive system. In such a system new morphological and syntactic styles are treated on the same basis as new writing styles by adapting the language PSA.

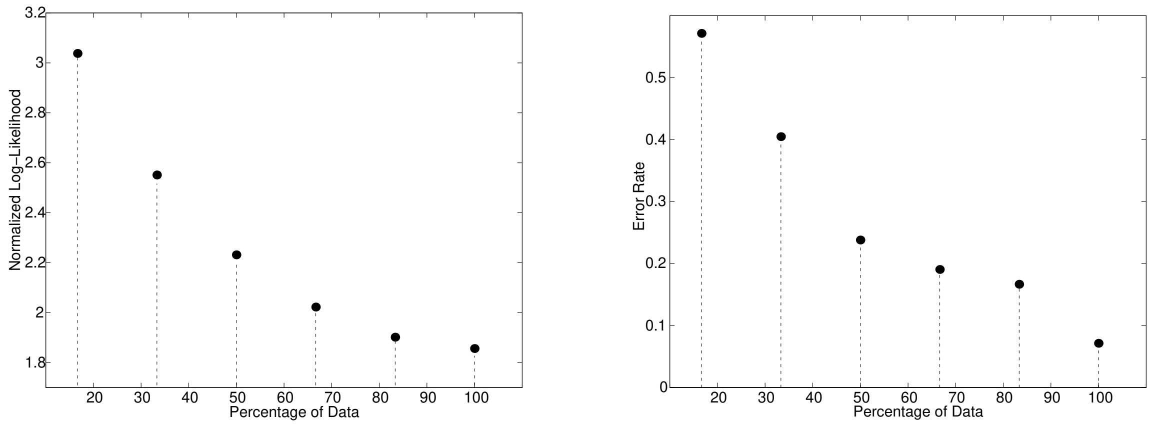


Figure 5.12: Evaluation of the importance of the online learning mode in our system. The performance of the system is tested by turning off the online mode and keeping the set of APFAs fixed after different portions of the data. A portion of 100% is the full online mode. Plotted on the left is the average log-likelihood, normalized by the length of the input sequence, and on the right the average error rate.

Chapter 6

Concluding Remarks

In the introduction we discussed recent approaches to the analysis of language while emphasizing their major drawbacks. We believe that the models and algorithms presented in this thesis overcome some of these drawbacks and will have a significant influence on the design and implementation of new language processing technologies. However, we wish to emphasize that the results presented in this thesis are only a small step towards a thorough understanding of language learning, acquisition and adaptation. The number of open problems far outnumber the solved ones, and most language analysis systems fall short of human capabilities. We would like to conclude with a short list of open problems and directions for future research.

Probabilistic Transducers The class of probabilistic transducers is an interesting and practical extension of the class of probabilistic automata. Probabilistic transducers are state machines associated with input and output alphabets, which transform an input sequence from the input alphabet to an output sequence. The current state of a transducer (stochastically) depends on the previous state, the current input symbol, and possibly on the previous output symbols. The output symbol may depend on the previous state and on the current input symbol. In [124], we investigated a subclass of probabilistic transducers for which the current state depends only on the previous state and the current input symbol. This subclass extends the structure of prediction suffix trees, presented in this thesis, to build suffix tree transducers. The case where the current state also depends on the output is more involved. It is not clear whether there are any direct extensions of the learning algorithms presented in this thesis for this case as well.

Prediction Over Unbounded Sets Throughout this thesis we have assumed that the result of the basic modeling stage, such as the dynamical encoding of cursive handwriting, is temporal sequences over a known and finite alphabet. However, there are cases where the alphabet is virtually unbounded. For instance, the set of all possible words in natural text has no explicit bound since new words and phrases constantly appear. Extending the learning algorithms to the case of unbounded alphabets is therefore a desired goal. A first step in this direction would be a precise definition of the problem, since it is not clear whether there is a simple extension of the distribution-free setting of the PAC model to this case. We have made a preliminary step in this direction by defining a Markov model with variable memory which outputs symbols drawn from an unbounded alphabet [98]. Implementation of such a model is rather complicated due the vast amounts of memory the model requires. The question whether there is a more compact representation that can be acquired automatically immediately arises when building such large models. Automatic identification and grouping of, for example, all verbs may help in building compact representations of suffix trees over an unbounded alphabet. Such problems have been addressed by several researchers in computational linguistics (*cf.* [22, 99]), using distributional clustering as the main analysis tool. We believe that devising a combined clustering and temporal modeling scheme will have a significant impact on the existing natural language processing methods.

Drifting Models Most of the existing learning algorithms for language analysis employ a tacit assumption that the source is stationary. However, in many modern texts, such as newspapers, new verbs, nouns, phrases and syntactic structures are used relatively frequently for some stretch of text only to drop to a much lower frequency after a while. Therefore, the assumption that the source is stationary is far from being correct and it would be interesting to look for models that can track a drifting distribution. Methods for tracking drifting concepts have been studied by several researchers (*cf.* [60]) yielding powerful algorithms such as the weighted majority [86]. A challenging research goal is to combine methods to track drifting concepts with the automata learning algorithms presented in this thesis. Devising such an approach might prove to be a powerful analysis tool when the environment is constantly changing.

Hierarchical Probabilistic Models Recently, there is an emergence of new models and algorithms, inspired by biological learning, which attempt to find a hierarchical structure in empirical data and use the hierarchical structure to better understand the underlying mechanisms of real systems (*cf.* [67]). The primary motivation for using a hierarchical structure is to enable better modeling of the different stochastic levels and length scales that are present in the natural language. Another important property of such models is the ability to infer correlated observations over long distances via the higher levels of the hierarchy. The positive results on learning subclasses of probabilistic automata, which have been successfully applied to "real-world" problems, give rise to the belief that hierarchical models based on probabilistic automata can be learned efficiently and can be useful in practical applications. Since HMMs have been used successfully many times in such applications, it would be interesting and profitable to study the learnability of restricted forms of hierarchical HMMs.

Active Learning Active learning may provide additional learning power and overcome intractability results that hold for passive learning. For example, actively learning a DFA is a much easier task than passive learning of the same DFA (e.g., [5, 112]). Whether active learning provides additional power for learning probabilistic models such as probabilistic automata is an intriguing question. A related question is whether the class of learnable automata can be extended if the learner is able to conduct active experiments by choosing the input to the automata. Active learning algorithms also have a practical motivation since they can provide an efficient method to obtain labeled data when such data is expensive.

Beyond Regular Languages Most language analysis techniques use a (probabilistic) regular language as their primary model. For example, most of the existing *part-of-speech* tagging systems rely on either a Markov model [26] or on a hidden Markov model [77]. It is nevertheless obvious that finite state models cannot capture the recursive nature of languages. There has been intensive research in learning stochastic context free grammars (*cf.* [81, 97]). However, most of the existing methods rely on the inside-outside algorithm, which is an extension of the forward-backward algorithm, and is derived using the EM formalism. Thus, it is a parameter estimation scheme that converges only to a local minimum. Moreover, almost all of the existing systems employ a manually defined grammar whose parsing rules are set by hand. An important question that arises is whether we need the full power of stochastic context free grammars. More than 20 years ago several sub-classes of context free languages and feed-back automata were studied by Bar-Hillel, Hartmanis, Perles, Shamir, Stearns and others [8, 58]. Several of the sub-classes were obtained by restricting the grammar. For example, we can restrict the forms of the rules responsible for creating cycles inherent in the phrase structure to $A \rightarrow \ldots A \rightarrow \ldots$ and disallow simultaneous occurrences of rules of the form $A \rightarrow \ldots B \ldots$ and $B \rightarrow \ldots A \ldots$ (where $A \neq B$). Such restrictions may yield sufficiently rich grammatical structures that are efficiently learnable under some restrictions on the input distribution. Such grammars will be applicable to language processing tasks that require automatic inference and identification of internal structures.

Learning Coupled Systems Most human communication systems can be viewed as a composition of two (or more) coupled systems, such as the articulatory–auditory system for speech production and perception, or the motor-control–visual system for mechanical object manipulation. The feedback between the systems is apparently crucial for their performance, and in many cases, defects in one of the systems cause malfunctioning in the other system. The difficulty in the design and analysis of learning algorithms for such systems arises from the need to decouple the systems from each other and to separate their self-dynamics from their (usually unknown) control signals. Though classical control theory provides several well analyzed tools such as the (Extended) Kalman filter [39], the applicability of these tools is mostly limited to systems with known ('almost' linear) dynamics. A new class of algorithms have recently emerged, known as *reinforcement learning*, which stem from stochastic dynamic programming [113]. These algorithms provide a general framework for learning systems where only a remote feedback of their performance is given. The theoretical basis for these learning algorithms is far from complete. Moreover, there are hardly any working practical applications that employ such algorithms. Learning coupled systems in the presence of only a distal teacher is therefore a challenging and profitable research goal.

Bibliography

[1] *Advances in Neural Information Processing System*, volume 1–7. Morgan Kaufmann, 1988–1994.

[2] N. Abe and M. Warmuth. On the computational complexity of approximating distributions by probabilistic automata. *Machine Learning*, 9:205–260, 1992.

[3] J.A. Anderson and E. Rosenfeld, editors. *Neurocomputing: Foundations of Research.* MIT Press, 1988.

[4] D. Angluin. On the complexity of minimum inference of regular sets. *Information and Control*, 39:337–350, 1978.

[5] D. Angluin. Learning regular sets from queries and counterexamples. *Information and Computation*, 75:87–106, 1987.

[6] D. Angluin and C.H. Smith. Inductive inference: Theory and methods. *Computing Surveys*, 15(3):237–269, September 1983.

[7] C. Antoniak. Mixture of Dirichlet processes with applications to Bayesian nonparametric problems. *Annals of Statistics*, 2:1152–174, 1974.

[8] Y. Bar-Hillel, M. Perles, and E. Shamir. On formal properties of simple phrase-structure grammars. *Zeitschrift fur Phonetik, Sprach. and Komm.*, 14(2):143–172, 1961.

[9] L. E. Baum. An inequality and associated maximization technique in statistical estimation for probabilistic functions of markov chains. *Inequalities*, 3:1–8, 1972.

[10] L. E. Baum, T. Petrie, G. Soules, and N. Weiss. A maximization technique occuring in the statistical analysis of probabilistic functions of markov chains. *Annals of Mathematical Statistics*, 41(1):164–171, 1970.

[11] L.E Baum and T. Petrie. Statistical inference for probabilistic functions of finite state markov chains. *Annals of Mathematical Statistics*, 37, 1966.

[12] E.J. Bellegarda, J.R Bellegarda, D. Nahamoo, and K.S. Nathan. A probabilistic framework for on-line handwriting recognition. In *The third Intl. Workshop on Frontiers in Handwriting Recognition, Buffalo NY*, pages 225–234, 1993.

[13] R. Bellman. *Dynamic Programming.* Princeton University Press, 1957.

[14] Y. Bengio, Y. le Cun, and D. Henderson. Globally trained handwritten word recognizer using spatial representation, convolutional neural networks, and hidden Markov models. In *Advances in Neural Information Processing Systems*, volume 6. Morgan Kaufmann, 1993.

[15] J. Berger. *Statistical Decision Theory and Bayesian Analysis.* Springer-Verlag, New York, 1985.

[16] H. Bergman, T. Wichmann, and M.R. DeLong. Reversal of experimental parkinsonism by lesions of the subthalamic nucleus. *Science*, 249:1436–1438, 1990.

[17] M. Berthod. On-line analysis of cursive writing. In C.Y. Suen and R. De Mori, editors, *Computer Analysis and Perception: Vol. 1 - Visual Signals*, pages 55–81. CRC Press, 1990.

[18] R.C. Berwick. *The acquisition of syntactic knowledge.* MIT Press, 1985.

[19] E. Brill. Automatic grammar induction and parsing free text: A transformation-based approach. In *Proc. of the ACL 31st*, pages 259–265, 1993.

[20] R. C. Carrasco and J. Oncina. Learning stochastic regular grammars by means of a state merging method. In *The 2nd Intl. Collo. on Grammatical Inference and Applications*, pages 139–152, 1994.

[21] G.J. Chaitin. On the length of programs for computing binary sequences. *J. Assoc. Comp. Mach.*, 13:547–569, 1966.

[22] E. Charniak. *Statistical Language Learning.* MIT Press, Cambridge, MA, 1993.

[23] E. Charniak, C. Hendrickson, N. Jacobson, and M. Perkowitz. Equations for Part-of-Speech tagging. In *Proc. of the Eleventh National Conf. on Artificial Intelligence*, pages 784–789, 1993.

[24] F.R. Chen. Identification of contextual factos for pronounciation networks. In *Proc. of IEEE Conf. on Acoustics, Speech and Signal Processing*, pages 753–756, 1990.

[25] H. Chernoff. A measure of asymptotic efficiency for tests of a hypothesis based on the sum of observations. *Annals of Math. Stat.*, 23:493–507, 1952.

[26] K. Church. An automatic parts program and noun phrase parser for unrestricted text. In *Proc. of ANLP 2nd*, pages 136–143, 1988.

[27] K.W. Church. A stochastic parts program and noun phrase parser for unrestricted text. In *Proc. of Intl. Conf. on Acoustics Speech and Signal Processing*, 1989.

[28] K.W. Church and W.A. Gale. A comparison of the enhanced Good-Turing and deleted estimation methods for estimating probabilities of English bigrams. *Computer Speech and Language*, 5:19–54, 1991.

[29] R.A. Cole, A.I. Rudincky, V.W. Zue, and D.R. Reddy. Speech as patterns on paper. In R.A. Cole, editor, *Perception and Production of Fluent Speech.* Lawrence Erlbaum Associates, 1980.

[30] R.A. Cole, R.M. Stern, M.S. Phillips, S.M. Brill, P. Specker, and A.P. Pilant. Feature based speaker independent recognition of English letters. In *IEEE Intl. Conf. on Acoustics, Speech, and Signal Processing*, 1983.

[31] T.M. Cover and J.A. Thomas. *Elements of Information Theory.* Wiley, 1991.

[32] R.H. Davis and J. Lyall. Recognition of handwritten characters - A review. In *Image Vision Comput.*, pages 208–218, 1986.

[33] A. Dempster, N. Laird, and D. Rubin. Maximum likelihood estimation from incomplete data via the EM algorithm. *J. Roy. Statist. Soc.*, 39(B):1–38, 1977.

[34] A. DeSantis, G. Markowsky, and M. N. Wegman. Learning probabilistic prediction functions. In *Proceedings of the Twenty-Ninth Annual Symposium on Foundations of Computer Science*, pages 110–119, 1988.

[35] L. Devroye. Automatic patten recognition: a study of the probability of error. *IEEE Trans. on Pattern Analysis and Machine Intelligence*, 10(4):530–543, 1988.

[36] T.G. Dietterich. Machine learning. In J.F. Traub, B.J. Grosz, B.W. Lampson, and N.J. Nilsson, editors, *Annual Review of Computer Science*, volume 4, pages 255–306. MIT Press, 1990.

[37] R.O. Duda and P.E. Hart. *Pattern Classification and Scene Analysis.* Wiley, 1973.

[38] R.M. Dudley. Central limit theorems for empirical measures. *The Annals of Probability*, 6(6):899–929, 1978.

[39] A. Gelb (ed.). *Applied Optimal Estimation.* MIT press, 1979.

[40] J.A. Fill. Eigenvalue bounds on convergence to stationary for nonreversible Markov chains, with an application to exclusion process. *Annals of Applied Probability*, 1:62–87, 1991.

[41] W.M. Fisher, V.W. Zue, J. Berbstein, and D. Pallett. An acoustic-phonetic data base. In *The 113th meeting of the ASA*, 1987.

[42] N. Flann and S. Shekhar. Recognizing on-line cursive handwriting using a mixture of cooperating pyramid-style neural networks. In *World Congres on Neural Networks*, 1993.

[43] W.N. Francis and F. Kučera. *Frequency Analysis of English Usage.* Houghton Mifflin, Boston MA, 1982.

[44] J.R. Frederiksen and J.F. Kroll. Spelling and sound: Approaches to the internal lexicon. *Journal of Experimental Psychology: Human Perception and Performance*, 2(3):361–379, 1976.

[45] Y. Freund, M. Kearns, D. Ron, R. Rubinfeld, R.E. Schapire, and L. Sellie. Efficient learning of typical finite automata from random walks. In *Proceedings of the 24th Annual ACM Symp. on Theory of Computing*, pages 315–324, 1993.

[46] L.S. Frishkopf and L.D. Harmon. machine reading of cursive script. In C. Cherry, editor, *Information Theory (4th London Symp.)*, pages 300–316, 1961.

[47] T. Fujisaki, K.S. Nathan, W. Cho, and H. Beigi. On-line unconstrained handwriting recognition by a probabilistic method. In *The third Intl. Workshop on Frontiers in Handwriting Recognition, Buffalo NY*, pages 235–241, 1993.

[48] I. Gat and N. Tishby. Statistical modeling of cell-assemblies activities in associative cortex of behaving monkeys. *Advances in Neural Information Processing Systems*, 5:945–953, 1993.

[49] D. Gillman and M. Sipser. inference and minimization of hidden markov chains. In *Proceedings of the Seventh Annual Workshop on Computational Learning Theory*, pages 147–158, 1994.

[50] M. E. Gold. System identification via state characterization. *Automatica*, 8:621–636, 1972.

[51] M. E. Gold. Complexity of automaton identification from given data. *Information and Control*, 37:302–320, 1978.

[52] G.I. Good. Statistics of language: Introduction. In A.R. Meetham and R.A. Hudson, editors, *Encyclopedia of Linguistics, Information and Control*, pages 567–581. Pergamon Press, Oxford, England, 1969.

[53] B.J. Grosz, K.S. Jones, and B.L. Webber, editors. *Readings in natural language processing.* Morgan Kaufmann, 1986.

[54] V.N. Gupta, M. Lennig, and P.Mermelstein. Fast search strategy in a large vocabulary word recognizer. *J. Acoust. Soc. Amer.*, 84(6):2007–2017, 1988.

[55] I. Guyon, P. Albercht, Y. Le Cun, J. Denker, and W. Hubbard. Design of a neural network character recognizer for touch terminal. *Pattern Recognition*, 24(2), 1991.

[56] G. Gyorgyi and N. Tishby. Statistical theory of learning a rule. In K. Thuemann and R. Koeberle, editors, *Neural Networks and Spin Glasses.* World Scientific, 1990.

[57] S. Hanakai and T. Yamazaki. On-line recognition of handprinted Kanji characters. *Pattern Recognition*, 12:421–429, 1980.

[58] J. Hartmanis, P.M. Lewis II, and R.E. Stearns. Hierarchies of memory limited computations. In *Proc. of 6th IEEE Symp. on SCTLD*, pages 179–190, 1965.

[59] J.-P Haton. Knowledge-based and expert systems in automatic speech recognition. In R. De-Mori, editor, *New Systems and Architectures for Automatic Speech Recognition and Synthesis.* Dorderchtm, Reidel, Netherlands, 1984.

[60] D.P. Helmbold and P.M. Long. Tracking drifting concepts by minimizing disagreements. *Machine Leanring*, 14(1):27–45, 1994.

[61] J. Hertz and A. Krogh abd R.G. Plamer. *Introduction to the Theory if Neural Computation.* Addison-Wesley, 1991.

[62] W. Hoeffding. Probability inequalities for sums of bounded random variables. *American Statistical Association Journal*, 58:13–30, 1963.

[63] K.-U. Höffgen. Learning and robust learning of product distributions. In *Proceedings of the Sixth Annual Workshop on Computational Learning Theory*, pages 97–106, 1993.

[64] N. Hogan and T. Flash. Moving gracefully: quantitative theories of motor coordination. *Trends in Neuro Science*, 10(4):170–174, 1987.

[65] J.H. Holland. *Adaptation in natural and artificial systems: An introductory analysis with applications to biology, control and artificial intelligence.* MIT Press, 1992.

[66] J.M. Hollerbach. An oscillation theory of handwriting. *Biological Cybernetics*, 39:139–156, 1981.

[67] R.A. Jacobs, M.I. Jordan, S.J.Nowlan, and G.E. Hinton. Adaptive mixture of local experts. *Neural Computation*, 3:79–87, 1991.

[68] F. Jelinek. A fast sequential decoding algorithm using a stack. *IBM J. Res. Develop.*, 13:675–685, 1969.

[69] F. Jelinek. Markov source modeling of text generation. Technical report, IBM T.J. Watson Research Center, 1983.

[70] F. Jelinek. Robust part-of-speech tagging using a hidden Markov model. Technical report, IBM T.J. Watson Research Center, 1983.

[71] F. Jelinek. Self-organized language modeling for speech recognition. Technical report, IBM T.J. Watson Research Center, 1985.

[72] M. Kearns, Y.Mansour, D. Ron, R. Rubinfeld, R.E. Schapire, and L. Sellie. On the learnability of discrete distributions. In *The 25th Annual ACM Symp. on Theory of Computing*, 1994.

[73] M.J. Kearns and U.V. Vazirani. *An introduction to computational learning theory.* MIT Press, 1994.

[74] A.N. Kolmogorov. Three approaches to the quantitative definition of information. *Problems of Information Transmission*, 1:4–7, 1965.

[75] A.N. Kolmogorov. Logical basis for information theory and probability theory. *IEEE Transactions on Information Theory*, IT-14(5):662–664, 1968.

[76] A. Krogh, S.I. Mian, and D. Haussler. A hidden markov model that finds genes in E. coli DNA. Technical Report UCSC-CRL-93-16, University of California at Santa-Cruz, 1993.

[77] J. Kupiec. Robust part-of-speech tagging using a hidden markov model. *Computer Speech and Language*, 6:225–242, 1992.

[78] E. Kushilevitz and Y. Mansour. Learning decision trees using the Fourier spectrum. *SIAM Journal on Computing*, 22(6):1331–1348, 1993.

[79] F. Lacquniti. Central representations of human limb movement as revealed by studies of drawing and handwriting. *Trends in Neuro Science*, 12(8):287–291, 1989.

[80] K. J. Lang. Random DFA's can be approximately learned from sparse uniform examples. In *Proc. of the Fifth Annual ACM Workshop on Computational Learning Theory*, pages 45–52, 1992.

[81] K. Lari and S.J.Young. Applications of stochastic context-free grammars using the inside-outside algorithm. *Computer Speech and Language*, 5:237–257, 1991.

[82] S.E. Levinson, L.R. Rabiner, and M.M. Sondhi. An introduction to the application of the theory of probabilistic functions of a markov process to automatic speech recognition. *Bell Syst. Tech*, 62(4):1983, 1035-1074.

[83] M. Li and U. Vazirani. On the learnability of finite automata. In *Proc. of the 1988 Workshop on Computational Learning Theory*, pages 359–370. Morgan Kaufmann, 1988.

[84] M. Li and P. Vitanyi. *An Introduction to Kolmogorov Complexity and its Applications.* Springer, New-York, 1993.

[85] N. Lindgen. Machine recognition of human language, part iii - cursive script recognition. *IEEE Spectrum*, pages 104–116, May 1965.

[86] Nick Littlestone and Manfred K. Warmuth. The weighted majority algorithm. In *30th Annual IEEE Symp. on Foundations of Computer Science*, pages 256–261, 1989.

[87] J.D. Markel and A.H. Gray. *Linear Prediction of Speech.* Springer-Verlag, 1976.

[88] N. Merhav and Y. Ephraim. Maximum likelihood hidden markov modeling using a dominant sequence of states. *IEEE trans. on signal processing*, ASSP-39(9):2111–2115, 1991.

[89] N. Merhav and Y. Ephraim. Maximum likelihood hidden Markov modeling using a dominant sequence of states. *IEEE Trans. on ASSP*, 39(9):2111–2115, 1991.

[90] M. Mihail. Conductance and convergence of Markov chains - A combinatorial treatment of expanders. In *Proceedings 30th Annual Conference on Foundations of Computer Science*, 1989.

[91] P. Morrase, L. Barberis, S. Pagliano, and D. Vernago. Recognition experiments of cursive dynamic handwriting with self-organizing networks. *Pattern Recognition*, 26(3):451–460, 1993.

[92] A. Nadas. Estimation of probabilities in the language model of the IBM speech recognition system. *IEEE Trans. on ASSP*, 32(4):859–861, 1984.

[93] R. Nag, K.H. Wong, and F. Fallside. Script recognition using hidden Markov models. In *Proc. IEEE Intl. Conf. Acoust. Speech Signal Proc., Tokyo Japan*, pages 2071–2074, 1986.

[94] C.K. Ogden. *Basic English.* K. Paul, Trench, Trubner publishers, 1944.

[95] T. Okuda, E. Tanaka, and K. Tamotsu. A method for the correction of garbled words based on the Levenshtein metric. *IEEE Transactions on Computers*, 25(2):172–177, 1976.

[96] A.V. Oppenheim and R.W. Schafer. *Digital Signal Processing.* Prentice-Hall, 1975.

[97] F.C. Pereira and Y. Schabes. Inside-outside reestimation from partially bracketed corpora. In *Proc. of ACL 30th*, 1992.

[98] F.C. Pereira, Y. Singer, and N. Tishby. Beyond n-grams. In *Thrid workshop on very large corpora*, 995.

[99] F.C. Pereira, N. Tishby, and L. Lee. Distributional clustering of english words. In *Proc. of the ACL 31st*, 1993.

[100] L. Pitt and M. K. Warmuth. The minimum consistent DFA problem cannot be approximated within any polynomial. *Journal of the Association for Computing Machinery*, 40(1):95–142, 1993.

[101] R. Plamondon and C.G Leedham, editors. *Computer Processing of Handwriting.* World Scientific, 1990.

[102] R. Plamondon, C.Y Suen, and M.L. Simner, editors. *Computer Recognition and Human Production of Handwriting.* World Scientific, 1989.

[103] D. Pollard. *Convergence of Stochastic Processes.* Springer-Verlag, 1984.

[104] L.R. Rabiner. A tutorial on hidden markov models and selected applications in speech recognition. *Proc. of the IEEE*, 1989.

[105] L.R. Rabiner and B. Gold. *Theory and application of digital signal processing.* Prentice-Hall, NJ, 1975.

[106] L.R. Rabiner and B. H. Juang. An introduction to hidden markov models. *IEEE ASSP Magazine*, 3(1):4–16, January 1986.

[107] L.R. Rabiner and B.H. Juang. *Fundamentals of Speech Recognition.* Prentice-Hall, 1993.

[108] L.R. Rabiner, J.P. Wilson, and B.H. Juang. A segmental k-means training procedure for connected word recognition. *AT&T Tech*, pages 21–40, 1986.

[109] M.D. Riley. A statistical model for generating pronounication networks. In *Proc. of IEEE Conf. on Acoustics, Speech and Signal Processing*, pages 737–740, 1991.

[110] J. Rissanen. A universal data compression system. *IEEE Trans. Inform. Theory*, 29(5):656–664, 1983.

[111] J. Rissanen. Complexity of strings in the class of Markov sources. *IEEE Trans. Inform. Theory*, 32(4):526–532, 1986.

[112] D. Ron and R. Rubinfeld. Learning fallible finite state automata. *Machine Learning*, 18:149–185, 1995.

[113] S. Ross. *Introduction to Stochastic Dynamic Programming.* Academic Press, 1983.

[114] K.E. Rudd. Maps, genes, sequences, and computers: An Escherichia coli case study. *ASM News*, 59:335–341, 1993.

[115] S. Rudich. Inferring the structure of a Markov chain from its output. In *Proceedings of the Twenty-Sixth Annual Symposium on Foundations of Computer Science*, pages 321–326, 1985.

[116] David E. Rumelhart. Theory to practice: A case study - recognizing cursive handwriting. Proc. of 1992 NEC Conf. on Computation and Cognition, 1992.

[117] D.E. Rumelhart and J.L. McClelland, editors. *Parallel Distributed Processing.* MIT Press, 1986.

[118] D. Sankoff and J.B. Kruskal. *Time warps, string edits and macromolecules: the theory and practice of sequence comparison.* Addison-Wesley, Reading Mass, 1983.

[119] L. Schomaker. Using stroke- or character-based self-organizing maps in the recognition of on-line cursive connected cursive script. *Pattern Recognition*, 26(3):443–450, 1993.

[120] H.S. Seung, H. Sampolinsky, and N. Tishby. Statistical mechanics of learning from examples. *Physical Review A*, 45:6056–6091, 1992.

[121] C.E. Shannon. Prediction and entropy of printed english. *Bell Sys. Tech. Jour.*, 30(1):50–64, 1951.

[122] J.W. Shavlik and T.G. Dietterich, editors. *Readings in Machine Learning.* Morgan Kaufman, 1990.

[123] H.T. Siegelmann and E.D. Sontag. On the computational power of neural nets. In *Proc. of the Fifth Annual ACM Workshop on Computational Learning Theory*, pages 440–449, 1992.

[124] Y. Singer. Adaptive mixture of probabilistic transducers, 1995. Submitted for publication.

[125] R.M.K. Sinha. On partitioning a dictionary for visual text recognition. *Pattern Recognition*, 23(5):497–500, 1990.

[126] R. J. Solomonoff. A formal theory of inductive inference. *Information and Control*, 7:1–22,224–254, 1964.

[127] A. Stolcke and S. Omohundro. Hidden Markov model induction by Bayesian model merging. In *Advances in Neural Information Processing Systems*, volume 5. Morgan Kaufmann, 1992.

[128] C.C. Tappert, C.Y. Suen, and T. Wakahara. The state of art in on-line handwriting recognition. *IEEE Trans. on Pattern Analysis and Machine Intelligence*, 12(8):787–808, 1990.

[129] H.L. Teulings, A.J.W.M. Thomassen, and G.P van Galen. Invariants in handwriting: the information contained in a motor program. In H.S.R kao, G.P van Galen, and R. Hoosain, editors, *Graphonomics: Contemporary Research in Handwriting*, 1986.

[130] A.J.W.M. Thomassen and H.L. Teulings. Time size and shape in handwriting: Exploring spatio-temporal relationships at different levels. In J.A. Michon and J.L. Jackson, editors, *Time, Mind , and Behavior*, pages 253–263. Springer-Verlag, 1986.

[131] S.C. Tornay. *Ockham: Studies and Selections.* Open Court Publishersm, La Salle, IL, 1938.

[132] B. A. Trakhtenbrot and Ya. M. Brazdin'. *Finite Automata: Behavior and Synthesis.* North-Holland, 1973.

[133] L. G. Valiant. A theory of the learnable. *Communications of the ACM*, 27(11):1134–1142, November 1984.

[134] V.N. Vapnik. *Estimation of Dependences Based on Empirical Data.* Springer-Verlag, 1982.

[135] V.N. Vapnik and A.Y. Chervonenkis. On the uniform convergence of relative frequencies of events to their probabilities. *Theory of Probability and its applications*, 17(2):264–280, 1971.

[136] R.A. Wagner and M.J. Fischer. The string-to-string correction problem. *J. ACM*, 21, 1974.

[137] A. Waibel, T. Hanazwa, G. Hinton, K. Shikano, and K. Lang. Phoneme recognition using time delay neural networks. *IEEE Trans. on Acoustics Speech and Signal Processing*, 37(3), 1989.

[138] A. Wald. Fitting of straight lines if both variables are subject to error. *Annals of Mathematical Statistics*, 11:284–300, 1940.

[139] M.J. Weinberger, A. Lempel, and J. Ziv. A sequential algorithm for the universal coding of finite-memory sources. *IEEE Trans. Inform. Theory*, 38:1002–1014, May 1982.

[140] R. Weischedel, M. Meteer, R. Schwartz, L. Ramshaw, and J. Palmucci. Coping with ambiguity and unknown words through probabilistic models. *Computational Linguistics*, 19(2):359–382, 1993.

[141] F.M.J. Willems, Y.M. Shtarkov, and T.J. Tjalkens. The context tree weighting method: Basic properties. *IEEE Trans. Inform. Theory*, 1993. Submitted for publication.

[142] C.F.J. Wu. On the convergence properties of the em algorithm. *Annals of Statistics*, 11(1):95–103, 1983.

[143] V.W. Zue. The use of speech knowledge in automatic speech recognitions. *Proc. of the IEEE*, 73(11):1602–1615, 1985.

Appendix A

Supplement for Chapter 2

Vertical Amplitude Modulation Discretization using *EM*

We assume that there is a virtual center for the vertical movements and that the amplitudes are symmetric about this center. The problem becomes similar to a mixture density estimation, but it is more involved since the parameters are tied via the symmetry constraints. The five levels correspond to five normal distributions with unknown means and a common variance. Initially, each level is chosen by the a priori probability P_i. We need to estimate the parameters H_i and find the most probable level indices I_t, when the available observations are the noisy vertical positions at the zero-crossings.

Let $\mu_i \stackrel{\Delta}{=} H_i$ and denote the stochastic levels by $Y_i \sim N(\mu_i, \sigma)$ $(i \in (\{1, \ldots, 5\})$. At each of the zero-crossings one of the levels is chosen with probability P_i $(\sum_{i=1}^{5} P_i = 1)$. The observed information is a noisy sample of the chosen level. We would like to estimate concurrently the vertical amplitude parameters and the levels obtained at the zero-crossings. Denote the parameter set by $\Theta = \{\{\Theta_i\}, \sigma\} = \{\{P_i\}, \{\mu_i\}, \sigma\}$. The joint distribution of the levels Y is $Z \sim \sum_{i=1}^{5} P_i N(\mu_i; \sigma)$. The symmetry constraints imply that $\mu_5 = 2\mu_3 - \mu_1$ and $\mu_4 = 2\mu_3 - \mu_2$. The complete data are denoted by $(\bar{Y}, \bar{I}) = (\{Y_t\}, \{I_t\})$ where I_t is the index of the chosen level at time t, and Y_t is the observed level value at that time. Let $I_t(i)$ be the levels indicator vector due to the index I_t, i.e., $I_t(i) = 1$ if $I_t = i$ and $I_t(i) = 0$ otherwise. The likelihood of an observation sequence $\{Y_t\}_{t=1}^{T}$ is

$$\log L_{\Theta}(\bar{Y}) \;=\; \log \sum_{t=1}^{T} P_{I_t} N(Y_t; \mu_{I_t}, \sigma) \;=\; \sum_{t=1}^{T} \sum_{i=1}^{5} I_t(i) \log P_i N(Y_t; \mu_i, \sigma) \quad . \qquad (A.1)$$

The first step in each *EM* iteration is to find the expectation of (A.1) using the current estimation of the parameter set denoted by $\Theta_1 = \{\{P_i^1\}, \{\mu_i^1\}, \sigma^1\}$. The following weights are calculated using the current parameters

$$W_t(i) \stackrel{\Delta}{=} E(I_t(i) \mid Y_t, \Theta_1) \;=\; \left(P_i^1 e^{-\frac{1}{2}\left(\frac{Y_t - \mu_i^1}{\sigma^1}\right)^2} \right) \Big/ \left(\sum_{i=1}^{5} P_i^1 e^{-\frac{1}{2}\left(\frac{Y_t - \mu_i^1}{\sigma^1}\right)^2} \right) \quad . \qquad (A.2)$$

The second stage of each *EM* iteration maximizes the current set of parameters, denoted by $Q(\Theta; \Theta_1)$, using the expectation of (A.1)

$$\max_{\Theta} Q(\Theta; \Theta_1) = \max_{P_i, \mu_i, \sigma} \sum_t \sum_i W_t(i) \cdot \left(\log P_i - \log \sigma - \frac{1}{2} \left(\frac{Y_t - \mu_i}{\sigma} \right)^2 \right) + \text{Const} \quad . \qquad (A.3)$$

Taking the partial derivative of (A.3) with respect to P_i under the constraint that $\sum_{i=1}^{5} P_i = 1$ and equating it to zero results in the following estimator, $P_i = \frac{\sum_t W_t(i)}{\sum_i \sum_t W_t(i)}$. The estimation of the current optimal level averages μ_i is more complicated due to the symmetry constraints. We rewrite Equ. (A.3) by substituting the symmetry constraints. Therefore, the explicit form for Q is

$$\begin{aligned} Q(\Theta;\Theta_1) =& \text{Const} + \sum_t \sum_{i=1}^{5} W_t(i) (\log P_i - \log \sigma) - \\ & \sum_t \sum_{i=1}^{3} \frac{1}{2} W_t(i) \left(\frac{Y_t - \mu_i}{\sigma} \right)^2 - \\ & \sum_t \sum_{i=1}^{2} \frac{1}{2} W_t(6-i) \left(\frac{Y_t - (2\mu_2 - \mu_i)}{\sigma} \right)^2 \quad . \end{aligned} \tag{A.4}$$

Define $\omega_i \triangleq \sum_t W_i(t)$ and $\chi_i \triangleq \sum_t W_i(t) Y_t$. Minimizing (A.4) with respect to μ_0, μ_1, μ_2 yields the following set of linear equations

$$\begin{cases} \mu_0\omega_0 - \chi_0 - (2\mu_2 - \mu_0)\omega_4 + \chi_4 = 0 \\ \mu_1\omega_1 - \chi_1 - (2\mu_2 - \mu_1)\omega_3 + \chi_3 = 0 \\ \mu_2\omega_2 - \chi_2 + 2(2\mu_2 - \mu_0)\omega_4 - 2\chi_4 + 2(2\mu_2 - \mu_1)\omega_3 - 2\chi_3 = 0 \quad . \end{cases}$$

These equations are explicitly solved using the symmetry constraints, to obtain the new values for μ_i as follows

$$\begin{aligned} D \triangleq\ & 4\,\omega_4\,\omega_0\,\omega_1 + 4\,\omega_4\,\omega_3\,\omega_1 + 4\,\omega_3\,\omega_0\,\omega_1 + 4\,\omega_4\,\omega_0\,\omega_3 + \\ & \omega_2\,\omega_0\,\omega_1 + \omega_4\,\omega_2\,\omega_1 + \omega_4\,\omega_2\,\omega_3 + \omega_2\,\omega_0\,\omega_3 \\ \mu_0 =\ & D^{-1} (4\,\omega_4\,\omega_3\,\chi_1 + 2\,\omega 4\,\omega_3\,\chi_2 + 2\,\omega_4\,\omega_1\,\chi_2 - 4\,\omega_1\,\omega_3\,\chi_4 - \\ & \chi_4\,\omega_2\,\omega_1 - \omega_3\,\omega_2\,\chi_4 + 4\,\omega_1\,\omega_4\,\chi_3 + 4\,\chi_0\,\omega_4\,\omega_1 + \\ & 4\,\omega_3\,\chi_0\,\omega_1 + 4\,\omega_4\,\omega_3\,\chi_0 + \chi_0\,\omega_2\,\omega_1 + \omega_3\,\omega_2\,\chi_0) \\ \mu_1 =\ & D^{-1} (2\,\omega_4\,\omega_3\,\chi_2 + 4\,\omega_4\,\omega_3\,\chi_1 + 4\,\omega_4\,\omega_0\,\chi_1 - \omega_4\,\omega_2\,\chi_3 + \\ & 4\,\omega_0\,\omega_3\,\chi_4 - \omega_2\,\omega_0\,\chi_3 + 4\,\omega_4\,\omega_3\,\chi_0 + \omega_4\,\omega_2\,\chi_1 + \\ & \omega_2\,\omega_0\,\chi_1 + 4\,\omega_3\,\omega_0\,\chi_1 - 4\,\omega_4\,\omega_0\,\chi_3 + 2\,\omega_3\,\omega_0\,\chi_2) \\ \mu_2 =\ & D^{-1} (\omega_4\,\omega_3\,\chi_2 + 2\,\omega_4\,\omega_3\,\chi_0 + 2\,\omega_4\,\omega_3\,\chi_1 + \omega_3\,\omega_0\,\chi_2 + \\ & 2\,\omega_3\,\omega_0\,\chi_1 + 2\,\omega_0\,\omega_3\,\chi_4 + 2\,\omega_1\,\omega_0\,\chi_3 + \omega_4\,\omega_1\,\chi_2 + \\ & 2\,\chi_0\,\omega_4\,\omega_1 + \omega_1\,\omega_0\,\chi_2 + 2\,\omega_1\,\omega_4\,\chi_3 + 2\,\omega_1\,\omega_0\,\chi_4) \\ & \mu_3 = 2\mu_2 - \mu_1\ ;\ \mu_4 = 2\mu_2 - \mu_0 \quad . \end{aligned}$$

Finally, the new variance is estimated using the new means, $\sigma^2 = \frac{\sum_{t,i} W_t(i)(Y_t - \mu_i)^2}{\sum_{t,i} W_t(i)}$. This process is iterated until convergence, which normally occurs within a few iterations. The final weights $W_t(i)$ correspond to the posterior probability that at time t the pen was at the vertical position H_i. Choosing the maximal value as the indicator of the level is the *maximum a posteriori* decision. This process can be performed on-line on a word basis or off-line for several words. In the latter case, the estimated a priori probabilities P_i reflect the stationary probability to be at position H_i. These probabilities are influenced by the motor characteristics of the handwriting as well as by the linguistic characteristics.

Appendix B

Supplement for Chapter 4

Proofs of Technical Lemmas

Lemma 4.6.1

1. *There exists a polynomial m'_0 in L, n, $|\Sigma|$, $\frac{1}{\epsilon}$, and $\frac{1}{\delta}$, such that the probability that a sample of $m' \geq m'_0(L, n, |\Sigma|, \frac{1}{\epsilon}, \frac{1}{\delta})$ strings each of length at least $L+1$ generated according to M is typical is at least $1-\delta$.*

2. *There exists a polynomial m_0 in L, n, $|\Sigma|$, $\frac{1}{\epsilon}$, $\frac{1}{\delta}$, and $1/(1-\lambda_2(U_M))$, such that the probability that a single sample string of length $m \geq m_0(L, n, |\Sigma|, \frac{1}{\epsilon}, \frac{1}{\delta}, 1/(1-\lambda_2(U_M)))$ generated according to M is typical is at least $1-\delta$.*

Proof: Before proving the lemma we would like to recall that the parameters ϵ_0, ϵ_1, ϵ_2, and γ_{min}, are all polynomial functions of $1/\epsilon$, n, L, and $|\Sigma|$, and were defined in Section 4.5.

<u>Several sample strings</u> We start with obtaining a lower bound for m', so that the first property of a typical sample holds. Since the sample strings are generated independently, we may view $\tilde{P}(s)$, for a given state s, as the average value of m' independent random variables. Each of these variables is in the range $[0,1]$ and its expected value is $\pi(s)$. Using a variant of Hoeffding's inequality (Appendix C) we get that if $m' \geq \frac{1}{2\epsilon_1^2\epsilon_0^2}\ln\frac{4n}{\delta}$, then with probability at least $1-\frac{\delta}{2n}$, $|\tilde{P}(s) - \pi(s)| \leq \epsilon_1\epsilon_0$. The probability that this inequality holds for every state is hence at least $1-\frac{\delta}{2}$.

We would like to point out that since our only assumptions on the sample strings are that they are generated independently, and that their length is at least $L+1$, we use only the independence between the different strings when bounding our error. We do not assume anything about the random variables related to $\tilde{P}(s)$ when restricted to any one sample string, other than that their expected value is $\pi(s)$. If the strings are known to be longer, then a more careful analysis can be applied as described subsequently for the case of a single sample string.

We now show that for an appropriate m' the second property holds with probability at least $1-\frac{\delta}{2}$ as well. Let s be a string in $\Sigma^{\leq L}$. In the following lines, when we refer to *appearances* of s in the sample we mean in the sense defined by $\tilde{P}$. That is, we count only appearances of s which end at the Lth or greater symbol of a sample string. For the ith appearance of s in the sample and for every symbol σ, let $X_i(\sigma|s)$ be a random variable which is 1 if σ appears after the ith appearance of s and 0 otherwise. If s is either a state or a suffix extension of a state, then for every σ, the random variables $\{X_i(\sigma|s)\}$ are independent 0/1 random variables with expected value $P(\sigma|s)$. Let N_s be the total number of times s appears in the sample, and let $N_{min} = \frac{2}{\epsilon_2^2\gamma_{min}^2}\ln\frac{4|\Sigma|n}{\epsilon_0\delta}$. If $N_s \geq N_{min}$, then with probability at least $1 - \frac{\delta\epsilon_0}{2n}$, for every symbol σ, $|\tilde{P}(\sigma|s) - P(\sigma|s)| \leq \frac{1}{2}\epsilon_2\gamma_{min}$. If s is a suffix of several states $s^1, \ldots, s^k$, then for every symbol σ,

$$P(\sigma|s) \;=\; \sum_{i=1}^{k} \frac{\pi(s^i)}{P(s)} P(\sigma|s^i)\;, \tag{B.1}$$

(where $P(s) = \sum_{i=1}^{k} \pi(s^i)$) and

$$\tilde{P}(\sigma|s) \;=\; \sum_{i=1}^{k} \frac{\tilde{P}(s^i)}{\tilde{P}(s)} \tilde{P}(\sigma|s^i)\;\;. \tag{B.2}$$

Recall that $\epsilon_1 = (\epsilon_2\gamma_{min})/(8n\epsilon_0)$. If:

1. For every state s^i, $|\tilde{P}(s^i) - \pi(s^i)| \leq \epsilon_1\epsilon_0$;
2. For each s^i satisfying $\pi(s^i) \geq 2\epsilon_1\epsilon_0$, $|\tilde{P}(\sigma|s^i) - P(\sigma|s^i)| \leq \frac{1}{2}\epsilon_2\gamma_{min}$ for every σ;

then $|\tilde{P}(\sigma|s) - P(\sigma|s)| \leq \epsilon_2\gamma_{min}$, as required.

If the sample has the first property required of a typical sample (i.e., $\forall s \in Q$, $|\tilde{P}(s) - P(s)| \leq \epsilon_1\epsilon_0$), and for every state s such that $\tilde{P}(s) \geq \epsilon_1\epsilon_0$, $N_s \geq N_{min}$, then with probability at least $1 - \frac{\delta}{4}$ the second property of a typical sample holds for all strings which are either states or suffixes of states. If for every string s which is a suffix extension a state such that $\tilde{P}(s) \geq (1 - \epsilon_1)\epsilon_0$, $N_s \geq N_{min}$, then for all such strings the second property holds with probability at least $1 - \frac{\delta}{4}$ as well. Putting together all the bounds above, if $m' \geq \frac{1}{2\epsilon_1^2\epsilon_0^2} \ln \frac{4n}{\delta} + N_{min}/(\epsilon_1\epsilon_0)$, then with probability at least $1 - \delta$ the sample is typical.

<u>*A single sample string*</u> In this case the analysis is somewhat more involved. We view our sample string generated according to M as a walk on the markov chain described by R_M (defined in Section 4.3). We may assume that the starting state is visible as well since its contribution to $\tilde{P}(\cdot)$ is negligible. We shall need the following theorem from [40] which gives bounds on the convergence rate to the stationary distribution of general ergodic Markov chains. This theorem is partially based on a work by Mihail [90], who gives bounds on the convergence in terms of combinatorial properties of the chain.

Markov Chain Convergence Theorem [40] *For any state s_0 in the Markov chain R_M, let $R_M^t(s_0, \cdot)$ denote the probability distribution over the states in R_M, after taking a walk of length t starting from state s_0. Then*

$$\left(\sum_{s \in Q} |R_M^t(s_0, s) - \pi(s)|\right)^2 \;\leq\; \frac{(\lambda_2(U_M))^t}{\pi(s_0)}\;\;.$$

First note that by simply applying Markov's inequality, we get that with probability at least $1 - \frac{\delta}{2n}$, $|\tilde{P}(s) - \pi(s)| \leq \epsilon_1\epsilon_0$, for each state s such that $\pi(s) < (\delta\epsilon_1\epsilon_0)/(2n)$. It thus remains to obtain a lower bound on m, so that the same is true for each s such that $\pi(s) \geq (\delta\epsilon_1\epsilon_0)/(2n)$. We do this by bounding the variance of the random variable related with $\tilde{P}(s)$, and applying Chebishev's Inequality.

Let

$$t_0 = \frac{\ln\left(n^3/\,32\delta^3\epsilon_0^5\epsilon_1^5\right)}{\ln\left(1/\lambda_2(U_M)\right)} \quad . \tag{B.3}$$

We next show that for every s satisfying $\pi(s) \geq (\delta\epsilon_1\epsilon_0)/(2n)$, $|R_M^{t_0}(s,s) - \pi(s)| \leq \frac{\delta}{4n}\epsilon_1^2\epsilon_0^2$. By the theorem above and our assumption on $\pi(s)$,

$$\begin{aligned}
\left(R_M^{t_0}(s,s) - \pi(s)\right)^2 &\leq \left(\sum_{s'\in Q} |R_M^{t_0}(s,s') - \pi(s')|\right)^2 \text{a} && \text{(B.4)}\\
&\leq \frac{(\lambda_2(U_M))^{t_0}}{\pi(s)}\text{b} && \text{(B.5)}\\
&\leq \frac{2n}{\delta\epsilon_0\epsilon_1}(\lambda_2(U_M))^{t_0}\text{c} && \text{(B.6)}\\
&= \frac{2n}{\delta\epsilon_0\epsilon_1}e^{-t_0\ln(1/\lambda_2(U_M))}\text{d} && \text{(B.7)}\\
&= \frac{\delta^2\epsilon_1^4\epsilon_0^4}{16n^2}\ .\text{e} && \text{(B.8)}
\end{aligned}$$

Therefore, $|R_M^t(s,s) - \pi(s)| \leq \frac{\delta}{4n}\epsilon_1^2\epsilon_0^2$.

Intuitively, this means that for every two integers, $t > t_0$, and $i \leq t - t_0$, the event that s is the $(i + t_0)$th state passed on a walk of length t, is 'almost independent' of the event that s is the ith state passed on the same walk.

For a given state s, satisfying $\pi(s) \geq (\delta\epsilon_1\epsilon_0)/(2n)$, let X_i be a 0/1 random variable which is 1 *iff* s is the ith state on a walk of length t, and $Y = \sum_{i=1}^t X_i$. By our definition of $\tilde{P}$, in the case of a single sample string, $\tilde{P}(s) = Y/t$, where $t = m - L - 1$. Clearly $E(Y/t) = \pi(s)$, and for every i, $Var(X_i) = \pi(s) - \pi^2(s)$. We next bound $Var(Y/t)$.

$$\begin{aligned}
Var\left(\frac{Y}{t}\right) &= \frac{1}{t^2}Var\left(\sum_{i=1}^t X_i\right) && \text{(B.9)}\\
&= \frac{1}{t^2}\left(\sum_{i,j}E(X_iX_j) - \sum_{i,j}E(X_i)E(X_j)\right)\text{b} && \text{(B.10)}\\
&= \frac{1}{t^2}\left(\sum_{i,j \text{ s.t. } |i-j|<t_0} E(X_iX_j) \ + \sum_{i,j \text{ s.t. } |i-j|\geq t_0} E(X_iX_j)\right) \ - \ \pi^2(s)\text{c} && \text{(B.11)}\\
&\leq \frac{2t_0}{t}\pi(s) + \frac{\delta}{4n}\epsilon_1^2\epsilon_0^2\pi(s) \ - \ \pi^2(s)\ \ .\text{d} && \text{(B.12)}
\end{aligned}$$

If we pick t to be greater than $(4nt_0)/(\delta\epsilon_1^2\epsilon_0^2)$, then $Var(Y/t) < \frac{\delta}{2n}\epsilon_1^2\epsilon_0^2$, and using Chebishev's Inequality $Pr[|Y/t - \pi(s)| > \epsilon_1\epsilon_0] < \frac{\delta}{2n}$. The probability the above holds for any s is at most $\frac{\delta}{2}$. The analysis of the second property required of a typical sample is identical to that described in the case of a sample consisting of many strings. ∎

Lemma 4.6.2 *If Learn-PSA is given a typical sample then:*

1. *For every string s in T, if $P(s) \geq \epsilon_0$ then $\frac{\gamma_s(\sigma)}{\hat{\gamma}_{s'}(\sigma)} \leq 1 + \epsilon/2$, where s' is the longest suffix of s corresponding to a node in $\hat{T}$.*
2. *$|\hat{T}| \leq (|\Sigma| - 1) \cdot |T|$.*

Proof:

1st Claim Assume contrary to the claim that there exists a string labeling a node s in T such that $P(s) \geq \epsilon_0$ and for some $\sigma \in \Sigma$

$$\frac{\gamma_s(\sigma)}{\hat{\gamma}_{s'}(\sigma)} > 1 + \epsilon/2, \tag{B.13}$$

where s' is the longest suffix of s in $\hat{T}$. For simplicity of the presentation, let us assume that there is a node labeled by s' in $\bar{T}$. If this is not the case ((s') is an internal node in $\bar{T}$, whose son s' is missing), the analysis is very similar. If $s \equiv s'$ then we easily show below that our counter assumption is false. If s' is a proper suffix of s then we prove the following. If the counter assumption is true, then we added to $\bar{T}$ a (not necessarily proper) suffix of s which is longer than s'. This contradicts the fact that s' is the longest suffix of s in $\hat{T}$.

We first achieve a lower bound on the ratio between the two true next symbol probabilities, $\gamma_s(\sigma)$ and $\gamma_{s'}(\sigma)$. According to our definition of $\hat{\gamma}_{s'}(\cdot)$,

$$\hat{\gamma}_{s'}(\sigma) \geq (1 - |\Sigma|\gamma_{min})\tilde{P}(\sigma|s') \ . \tag{B.14}$$

We analyze separately the case in which $\gamma_{s'}(\sigma) \geq \gamma_{min}$, and the case in which $\gamma_{s'}(\sigma) < \gamma_{min}$. Recall that $\gamma_{min} = \epsilon_2/|\Sigma|$. If $\gamma_{s'}(\sigma) \geq \gamma_{min}$, then

$$\frac{\gamma_s(\sigma)}{\gamma_{s'}(\sigma)} \geq \frac{\gamma_s(\sigma)}{\tilde{P}(\sigma|s')} \cdot (1 - \epsilon_2)\mathrm{a} \tag{B.15}$$

$$\geq \frac{\gamma_s(\sigma)}{\hat{\gamma}_{s'}(\sigma)} \cdot (1 - \epsilon_2)(1 - |\Sigma|\gamma_{min})\mathrm{b} \tag{B.16}$$

$$> (1 + \frac{\epsilon}{2})(1 - \epsilon_2)^2 \ ,\mathrm{c} \tag{B.17}$$

where Inequality (B.15) follows from our assumption that the sample is typical, Inequality (B.16) follows from our definition of $\hat{\gamma}_{s'}(\sigma)$, and Inequality (B.17) follows from the counter assumption (B.13), and our choice of γ_{min}. Since $\epsilon_2 < \epsilon/12$, and $\epsilon < 1$ then we get that

$$\frac{\gamma_s(\sigma)}{\gamma_{s'}(\sigma)} > 1 + \frac{\epsilon}{4} \ . \tag{B.18}$$

If $\gamma_{s'}(\sigma) < \gamma_{min}$, then $\hat{\gamma}_{s'}(\sigma) \geq \gamma_{s'}(\sigma)$, since $\hat{\gamma}_{s'}(\sigma)$ is defined to be at least γ_{min}. Therefore,

$$\frac{\gamma_s(\sigma)}{\gamma_{s'}(\sigma)} \geq \frac{\gamma_s(\sigma)}{\hat{\gamma}_{s'}(\sigma)} > 1 + \frac{\epsilon}{2} > 1 + \frac{\epsilon}{4} \tag{B.19}$$

as well. If $s \equiv s'$ then the counter assumption (B.13) is evidently false, and we must only address the case in which $s \neq s'$, i.e., s' is a proper suffix of s.

Let $s = s_1 s_2 \dots s_l$, and let s' be $s_i \dots s_l$, for some $2 \le i \le l$. We now show that if the counter assumption (B.13) is true, then there exists an index $1 \le j < i$ such that $s_j \dots s_l$ was added to $\bar{T}$. Let $2 \le r \le i$ be the first index for which $\gamma_{s_r \dots s_l}(\sigma) < (1 + 7\epsilon_2)\gamma_{min}$. If there is no such index then let $r = i$. The reason we need to deal with the prior case is clarified subsequently. In either case, since $\epsilon_2 < \epsilon/48$, and $\epsilon < 1$, then

$$\frac{\gamma_s(\sigma)}{\gamma_{s_r \dots s_l}(\sigma)} \; > \; 1 + \frac{\epsilon}{4}. \tag{B.20}$$

In other words

$$\frac{\gamma_s(\sigma)}{\gamma_{s_2 \dots s_l}(\sigma)} \cdot \frac{\gamma_{s_2 \dots s_l}(\sigma)}{\gamma_{s_3 \dots s_l}(\sigma)} \cdot \ldots \cdot \frac{\gamma_{s_{r-1} \dots s_l}(\sigma)}{\gamma_{s_r \dots s_l}(\sigma)} \; > \; 1 + \frac{\epsilon}{4} \; . \tag{B.21}$$

This last inequality implies that there must exist an index $1 \le j \le i - 1$, for which

$$\frac{\gamma_{s_j \dots s_l}(\sigma)}{\gamma_{s_{j+1} \dots s_l}(\sigma)} \; > \; 1 + \frac{\epsilon}{8L} \; . \tag{B.22}$$

We next show that Inequality (B.22) implies that $s_j \dots s_l$ was added to $\bar{T}$. We do this by showing that $s_j \dots s_l$ was added to $\bar{S}$, that we compared $\tilde{P}(\sigma|s_j \dots s_l)$ to $\tilde{P}(\sigma|s_{j+1} \dots s_l)$, and that the ratio between these two values is at least $(1 + 3\epsilon_2)$. Since $P(s) \ge \epsilon_0$ then necessarily

$$\tilde{P}(s_j \dots s_l) \ge (1 - \epsilon_1)\epsilon_0 \;\; , \tag{B.23}$$

and $s_j \dots s_l$ must have been added to $\bar{S}$. Based on our choice of the index r, and since $j < r$,

$$\gamma_{s_j \dots s_l}(\sigma) \ge (1 + 7\epsilon_2)\gamma_{min}. \tag{B.24}$$

Since we assume that the sample is typical,

$$\tilde{P}(\sigma|s_j \dots s_l) \ge (1 + 6\epsilon_2)\gamma_{min} > (1 + \epsilon_2)\gamma_{min} \;\; , \tag{B.25}$$

which means that we must have compared $\tilde{P}(\sigma|s_j \dots s_l)$ to $\tilde{P}(\sigma|s_{j+1} \dots s_l)$.

We now separate the case in which $\gamma_{s_{j+1} \dots s_l}(\sigma) < \gamma_{min}$, from the case in which $\gamma_{s_{j+1} \dots s_l}(\sigma) \ge \gamma_{min}$. If $\gamma_{s_{j+1} \dots s_l}(\sigma) < \gamma_{min}$ then

$$\tilde{P}(\sigma|s_{j+1} \dots s_l) \le (1 + \epsilon_2)\gamma_{min} \;\; . \tag{B.26}$$

Therefore,

$$\frac{\tilde{P}(\sigma|s_j \dots s_l)}{\tilde{P}(\sigma|s_{j+1} \dots s_l)} \ge \frac{(1 + 6\epsilon_2)\gamma_{min}}{(1 + \epsilon_2)\gamma_{min}} \ge (1 + 3\epsilon_2) \;\; , \tag{B.27}$$

and $s_j \dots s_l$ would have been added to $\bar{T}$. On the other hand, if $\gamma_{s_{j+1} \dots s_l}(\sigma) \ge \gamma_{min}$, the same would hold since

$$\begin{aligned}
\frac{\tilde{P}(\sigma|s_j \dots s_l)}{\tilde{P}(\sigma|s_{j+1} \dots s_l)} \;\; &\ge \;\; \frac{(1 - \epsilon_2)\gamma_{s_j \dots s_l}(\sigma)}{(1 + \epsilon_2)\gamma_{s_{j+1} \dots s_l}(\sigma)}\mathrm{a} && \text{(B.28)} \\
&> \;\; \frac{(1 - \epsilon_2)(1 + \frac{\epsilon}{8L})}{(1 + \epsilon_2)}\mathrm{b} && \text{(B.29)} \\
&\ge \;\; \frac{(1 - \epsilon_2)(1 + 6\epsilon_2)}{(1 - \epsilon_2)}\mathrm{c} && \text{(B.30)} \\
&> \;\; 1 + 3\epsilon_2 \;\; , \mathrm{d} && \text{(B.31)}
\end{aligned}$$

where Inequality B.30 follows from our choice of ϵ_2 ($\epsilon_2 = \frac{\epsilon}{48L}$). This contradicts our initial assumption that s' is the longest suffix of s added to $\bar{T}$.

2nd Claim: We prove below that $\bar{T}$ is a subtree of T. The claim then follows directly, since when transforming $\bar{T}$ into $\hat{T}$, we add at most all $|\Sigma| - 1$ siblings of every node in $\bar{T}$. Therefore it suffices to show that we did not add to $\bar{T}$ any node which is not in T. Assume to the contrary that we add to $\bar{T}$ a node s which is not in T. According to the algorithm, the reason we add s to $\bar{T}$, is that there exists a symbol σ such that $\tilde{P}(\sigma|s) \geq (1+\epsilon_2)\gamma_{min}$, and $\tilde{P}(\sigma|s)/\tilde{P}(\sigma|(\mathsf{s})) > 1 + 3\epsilon_2$, while both $\tilde{P}(s)$ and $\tilde{P}((\mathsf{s}))$ are greater than $(1-\epsilon_1)\epsilon_0$. If the sample string is typical then

$$P(\sigma|s) \geq \gamma_{min} \ , \ \ \tilde{P}(\sigma|s) \ \leq \ P(\sigma|s) + \epsilon_2\gamma_{min} \ \leq \ (1+\epsilon_2)P(\sigma|s) \ , \tag{B.32}$$

and

$$\tilde{P}(\sigma|(\mathsf{s})) \ \geq \ \mathsf{P}(\sigma|(\mathsf{s})) - \epsilon_2\gamma_{\mathsf{min}} \ . \tag{B.33}$$

If $P(\sigma|(\mathsf{s})) \geq \gamma_{\mathsf{min}}$ then $\tilde{P}(\sigma|(\mathsf{s})) \ \geq \ (1-\epsilon_2)\mathsf{P}(\sigma|(\mathsf{s}))$, and thus

$$\frac{P(\sigma|s)}{P(\sigma|(\mathsf{s}))} \ \geq \ \frac{(1-\epsilon_2)}{(1+\epsilon_2)}(1+3\epsilon_2) \ , \tag{B.34}$$

which is greater than 1 since $\epsilon_2 < 1/3$. If $P(\sigma|(\mathsf{s})) < \gamma_{\mathsf{min}}$, since

$$P(\sigma|s) \geq \gamma_{min} \ ,$$

then

$$P(\sigma|s)/P(\sigma|(\mathsf{s})) > 1$$

as well. In both cases this ratio cannot be greater than 1 if s is not in the tree, contradicting our assumption. ■

Appendix C

Chernoff Bounds

In this brief appendix we state two useful inequalities that we use repeatedly in this thesis.

For $m > 0$, let $X_1, X_2, ...X_m$ be m independent 0/1 random variables were $Pr[X_i = 1] = p_i$, and $0 < p_i < 1$. Let $p = \sum_i p_i/m$.

Inequality 1 (Multiplicative Form) *For* $0 < \alpha \leq 1$,

$$\Pr\left[\frac{\sum_{i=1}^{m} X_i}{m} - p > \alpha\right] < e^{-2\alpha^2 m}$$

and

$$\Pr\left[p - \frac{\sum_{i=1}^{m} X_i}{m} > \alpha\right] < e^{-2\alpha^2 m}$$

Inequality 2 (Additive Form) *For* $0 < \alpha \leq 1$,

$$\Pr\left[\frac{\sum_{i=1}^{m} X_i}{m} > (1+\alpha)p\right] < e^{-\frac{1}{3}\alpha^2 pm}$$

and

$$\Pr\left[\frac{\sum_{i=1}^{m} X_i}{m} < (1-\alpha)p\right] < e^{-\frac{1}{2}\alpha^2 pm}$$

The Additive Form of the bound is usually credited to Hoeffding [62] and the Multiplicative Form to Chernoff [25]. In the computer science literature both forms are often referred to by the name *Chernoff bounds.*